

Mitsuhisa Sato *Editor*

Advanced Software Technologies for Post-Peta Scale Computing

The Japanese Post-Peta CREST Research
Project



Springer

Advanced Software Technologies for Post-Peta Scale Computing

Mitsuhisa Sato

Editor

Advanced Software Technologies for Post-Peta Scale Computing

The Japanese Post-Peta CREST Research
Project



Springer

Editor
Mitsuhisa Sato
RIKEN Center for Computational Science
Kobe, Japan

ISBN 978-981-13-1923-5 ISBN 978-981-13-1924-2 (eBook)
<https://doi.org/10.1007/978-981-13-1924-2>

Library of Congress Control Number: 2018959846

© Springer Nature Singapore Pte Ltd. 2019

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Preface

Computational Science, which enables us to explore uncharted fields of science through applications of high performance computing, is a third paradigm of scientific research which has become indispensable for the development of science and technology of the twenty-first century.

Computational Science has been making great progress rapidly. The computational capability of supercomputers is now reaching on the verge of surpassing a Peta-flops (10^{15} floating operations per second). This advance allows us in making it possible to explore a wide range of phenomena through computer simulations which was impossible in the past, for example, the creation and evolution of the universe, the quantum origin of the functions of nano- and biomaterials and its implication to life, and global climate changes. At the same time, the development of intelligent information processing technologies is beginning to enable a handling and analysis of enormous amount of data, which are accelerating discoveries in many science disciplines, such as genome analyses, high energy accelerator experiments, astronomical observations, and satellite observation of geo-environments. In short, computational science is rapidly developing into a unified framework in which large-scale modeling and simulation, large-scale data analysis, and experiments/observation are integrated together to solve grand challenge issues in various branches of science. As such, computational science has now become an indispensable tool to better understand nature, life, and environment in order to create a better future for mankind.

In Japan, the importance of computational sciences was explicitly noted in the Japanese Government's Third Basic Plan of Science and Technology (2006–2010), and a national project for the development of "Next Generation Supercomputer" has been carried out from 2006 as one of the key technologies of national importance. The K computer has been produced as a major result of the project, achieving world's best performance in TOP500 list in 2011. Currently, the FLAGSHIP 2020 project for the development of the next Japanese flagship supercomputer has been launched in 2014, and the development is under way. The operation for the public service will be scheduled around 2020.

In 2010, the Japan Science and Technology Agency (JST) has initiated a research area, titled “Development of System Software Technologies for Post-Peta Scale High Performance Computing,” as a part of its Strategic Basic Research Program (CREST). The project was named “JST CREST Post-Petascale software project.” The research area of the project aimed at developing system software technologies as well as related systems to be used for high performance computing systems including the next generations of the Japanese flagship system, the K computer, which is under development. Several researches and developments were conducted for system software enabling us to exploit maximum efficiency and performance from supercomputers composed of general purpose many-core processors as well as accelerators such as GPUs and FPGA. From 2010 to 2012, 14 research teams were selected, and 5-year research has been being conducted by each research team. Many Japanese researchers and graduate students related to HPC have been participating in these research teams.

This book describes the major outcomes obtained by research teams of the JST CREST post-petascale software project.

Advanced system software is the key technology for post-petascale and exascale high performance computing systems which will be developed in next decade. I hope that the technologies developed in the JST CREST post-petascale software project will play a role bridging to exascale computing and beyond through system software technologies and advance the future computational science.

Kobe, Japan
May 2018

Mitsuhisa Sato

Contents

1	JST CREST Post-petascale Software Project Bridging to Exascale Computing	1
	Mitsuhisa Sato	
2	ppOpen-HPC/pK-Open-HPC: Application Development Framework with Automatic Tuning (AT)	11
	Kengo Nakajima, Masaharu Matsumoto, Masatoshi Kawai, Takahiro Katagiri, Takashi Arakawa, Hisashi Yashiro, and Akihiro Ida	
3	Scalable Eigen-Analysis Engine for Large-Scale Eigenvalue Problems	37
	Tetsuya Sakurai, Yasunori Futamura, Akira Imakura, and Toshiyuki Imamura	
4	System Software for Many-Core and Multi-core Architecture	59
	Atsushi Hori, Yuichi Tsujita, Akio Shimada, Kazumi Yoshinaga, Namiki Mitaro, Go Fukazawa, Mikiko Sato, George Bosilca, Aurélien Bouteiller, and Thomas Herault	
5	Highly Productive, High Performance Application Frameworks for Post Petascale Computing	77
	Naoya Maruyama, Takayuki Aoki, Kenjiro Taura, Rio Yokota, Mohamed Wahib, Motohiko Matsuda, Keisuke Fukuda, Takashi Shimokawabe, Naoyuki Onodera, Michel Müller, and Shintaro Iwasaki	
6	System Software for Data-Intensive Science	99
	Osamu Tatebe, Yoshihiro Oyama, Masahiro Tanaka, Hiroki Ohtsuji, Fuyumasa Takatsu, and Xieming Li	
7	Approaches for Memory-Efficient Communication Library and Runtime Communication Optimization	121
	Takeshi Nanri	

8 A Development Platform for Embedded Domain-Specific Languages 139
 Shigeru Chiba, YungYu Zhuang, and Thanh-Chung Dao

9 Xevolver: A User-Defined Code Transformation Approach to Streamlining Legacy Code Migration 163
 Hiroyuki Takizawa, Reiji Suda, Daisuke Takahashi, and Ryusuke Egawa

10 Numerical Library Based on Hierarchical Domain Decomposition .. 183
 Ryuji Shioya, Masao Ogino, Yoshitaka Wada, Kohei Murotani, Seiichi Koshizuka, Hiroshi Kawai, Shin-ichiro Sugimoto, and Amane Takei

11 Advanced Computing and Optimization Infrastructure for Extremely Large-Scale Graphs on Post-peta-scale Supercomputers..... 207
 Katsuki Fujisawa, Toyotaro Suzumura, Hitoshi Sato, Koji Ueno, Satoshi Imamura, Ryo Mizote, Akira Tanaka, Nozomi Hata, and Toshio Endo

12 Software Technology That Deals with Deeper Memory Hierarchy in Post-petascale Era 227
 Toshio Endo, Hiroko Midorikawa, and Yukinori Sato

13 Power Management Framework for Post-petascale Supercomputers 249
 Masaaki Kondo, Ikuo Miyoshi, Koji Inoue, and Shinobu Miwa

14 Project CASSIA —Framework for Exhaustive and Large-Scale Social Simulation— 271
 Itsuki Noda, Yohsuke Murase, Nobuyasu Ito, Kiyoshi Izumi, Hiromitsu Hattori, Tomio Kamada, Hideyuki Mizuta, and Mikio Takeuchi

15 GPU Accelerated Language and Communication Support by FPGA 301
 Taisuke Boku, Toshihiro Hanawa, Hitoshi Murai, Masahiro Nakao, Yohei Miki, Hideharu Amano, and Masayuki Umemura

Chapter 1

JST CREST Post-petascale Software Project Bridging to Exascale Computing



Mitsuhisa Sato

Abstract JST CREST post-petascale software project aimed to establish software technologies to explore extreme performance computing beyond petascale computing, on the road to exascale computing. Several research and development has been conducted for system software enabling us to exploit maximum efficiency and reliability from high-performance computing systems composed of general-purpose many-core processors as well as accelerators including GPGPU from the second half of the 2010s to 2020s. The research topics cover from system software such as programming languages, compilers, runtime systems, operating systems, communication middleware, and file systems to application development support software and ultra-large data processing software. As well as conventional technologies for large-scale numerical computation, the project was also able to address the storage technology required for big data processing, the complexity of memory hierarchy, and the power problem. Exploration for the direction of future high-performance computing is also an urgent and significant agenda in our research area. This chapter presents the outline of JST CREST post-petascale software project with brief description of the research topics, followed by summary of results and achievements.

1.1 Trends of High-Performance Computing

High-performance computing systems used for cutting edge of advanced computational have reached several petaflops (a million billion calculations per second) performance and will be targeted to the next generation of exascale systems as a post-petascale system. Scientific applications require increasing performance for industrial and societal general improvements.

M. Sato (✉)
RIKEN Center for Computational Science, Kobe, Japan
e-mail: msato@riken.jp

© Springer Nature Singapore Pte Ltd. 2019
M. Sato (ed.), *Advanced Software Technologies for Post-Peta Scale Computing*,
https://doi.org/10.1007/978-981-13-1924-2_1

Japan already has installed several petascale computers including the K computer in RIKEN and now explores the evolution toward future exascale systems. Following the end of the existing Moore law, the number of core per chip increases and specialized hardware has been used to accelerate specific type of applications. The number of processors and the interconnecting network increase also and we have then to face new programming problems. Post-petascale systems and future exascale computers are expected to have an ultra-large-scale and high-performance architecture with nodes of many-core processors and accelerators. To manage these ultra-large-scale parallel systems, we require new sophisticated system software technologies, allowing to manage complex parallel computations with huge distributed data, minimizing the energy consumption, and with fault-resilient properties.

JST CREST post-petascale software project has been launched to establish software technologies to explore extreme performance computing beyond petascale computing, on the road to exascale computing. The ability to manage and program these future high-performance systems efficiently is considered by all research national agencies all along the world as a strategic and important issue.

1.2 Outline of JST CREST Post-petascale Software Project

CREST (Core Research for Evolutional Science and Technology) is a funding program supported by JST (Japan Science and Technology Agency), which is an independent public body of the Ministry of Education, Culture, Sports, Science and Technology (MEXT), Japan. CREST is a funding program for team-based research expected to produce outstanding results to lead scientific and technological innovation. JST CREST post-petascale software project is one of CREST programs, starting from 2012.

Our research area was funded under the title of “Development of System Software Technologies for Post-petascale High-Performance Computing.” It was launched by the first program research supervisor, Prof. Akinori Yonezawa, RIKEN, from 2012 to 2014, and it was taken over to the second program research supervisor, Prof. Mitsuhsa Sato, RIKEN, from 2015 to 2018.

The research area aimed at developing software technologies as well as related systems to be used for high-performance computing in the post generations of the Japanese national supercomputer, the K computer. Several research and development has been conducted for system software enabling us to exploit maximum efficiency and reliability from high-performance computing systems composed of general-purpose many-core processors as well as accelerators including GPGPU from the second half of the 2010s to 2020s. In addition to the system software such as programming languages, compilers, runtime systems, operating systems, communication middleware, and file systems, application development support software and ultra-large data processing systems are also the targets for our research and development.

The calls for project proposals were issued at every year from 2010 to 2012, and finally the 14 projects were adopted: 5 projects in the first year, 5 projects in the second year, and 4 projects in the third year as a result of peer reviews by advisory committee for the project proposals submitted as responses to the calls. The duration of each project is 5.5 years. The total budget from 2010 to 2017 was about 60 M USD. Table 1.1 shows the adopted teams of JST CREST post-petascale software project.

The advisory committee was organized to advice the research direction of the project teams by following members:

- Mutsumi Aoyagi, Professor, Research Institute for Information Technology, Kyushu University (assumption of office period: Oct. 2010–Dec. 2014)
- Yutaka Ishikawa, Project Leader, Flagship 2020 Project, RIKEN Advanced Institute for Computational Science
- Kouichi Kumon, Member of the board, Fujitsu Laboratories Ltd.
- Kenji Kono, Professor, Keio University
- Hiroaki Kobayashi, Director, Cyberscience Center, Tohoku University
- Mitsuhisa Sato, Professor, Department of Computer Science, University of Tsukuba (assumption of office period: Oct. 2010–Mar. 2015)
- Shinji Shimojo, Professor, Cybermedia Center, Osaka University
- Keiko Takahashi, Director, Center for Earth Information Science and Technology, Japan Agency for Marine-Earth Science and Technology
- Yaoko Nakagawa, Senior Project Manager, Center for Technology Innovation-Information and Telecommunications, Research & Development Group, Hitachi Ltd.
- Hiroshi Nakashima, Professor and Director, Academic Center for Computing and Media Studies, Kyoto University
- Junichiro Makino, Professor, Department of Planetology, Graduate School of Science, Kobe University
- Satoshi Matsuoka, Professor, Global Scientific Information and Computing Center, Tokyo Institute of Technology

To carry the projects out strategically, the policies and goals of management in our research area were defined as follows:

- (1) Research and development of highly functional and reliable system software for sustainable high-performance computing technologies to solve social and scientific problems

Numerical simulation and data analysis utilizing ultra-large-scale computational resources and storage have dramatically been increasing the importance of its role for modern science and technology. In response to this fact, the USA, Europe, China, and Japan are racing to develop the next generation of supercomputer – exascale systems – capable of a million trillion calculations a second by around 2020. In our country, the project, FLAGSHIP2020, has been launched to develop the next flagship system following the K computer.

Table 1.1 Research teams of JST CREST post-petascale software project

PI, title/affiliation	Research theme
1. Research teams adopted in 2010	
Tetsuya Sakurai Professor, University of Tsukuba	Development of an Eigen-Supercomputing Engine Using a Post-Petascale Hierarchical Model
Osamu Tatebe Professor, University of Tsukuba	System Software for Post-Petascale Data-Intensive Science
Kengo Nakajima Professor, University of Tokyo	ppOpen-HPC: Open-Source Infrastructure for Development and Execution of Large-Scale Scientific Applications on Post-Petascale Supercomputers with Automatic Tuning (AT)
Atsushi Hori Senior Researcher, RIKEN	Parallel System Software for Multi-core and Many-core
Naoya Maruyama Research Team Leader, RIKEN	Highly Productive, High-Performance Application Frameworks for Post-petascale Computing
2. Research teams adopted in 2011	
Ryuji Shioya Professor, Toyo University	Development of a Numerical Library Based on Hierarchical Domain Decomposition for Post-petascale Simulation
Hiroyuki Takizawa Associate Professor, Tohoku University	An Evolutionary Approach to Construction of a Software Development Environment for Massively Parallel Heterogeneous Systems
Shigeru Chiba Professor, The University of Tokyo	Software Development for Post-petascale Supercomputing: Modularity for Supercomputing
Takeshi Nanri Associate Professor, Kyushu University	Development of Scalable Communication Library with Technologies for Memory Saving and Runtime Optimization
Katsuki Fujisawa Professor, Kyushu University	Advanced Computing and Optimization Infrastructure for Extremely Large-Scale Graphs on Post-petascale Supercomputers
3. Research teams adopted in 2012	
Toshio Endo Associate Professor, Tokyo Institute of Technology	Software Technology that Deals with Deeper Memory Hierarchy in Post-petascale Era
Masaaki Kondo Associate Professor, The University of Tokyo	Power Management Framework for Post-petascale Supercomputers
Itsuki Noda Principal Research Manager, AIST	Framework for Administration of Social Simulations on Massively Parallel Computers
Taisuke Boku Professor, University of Tsukuba	Research and Development on Unified Environment of Accelerated Computing and Interconnection for Post-petascale Era

Under such circumstances, in order to keep the ability to solve several social and scientific problems by making full use of the supercomputer, it is extremely significant not only to have technologies to execute a large-scale programs (simulation programs, data analysis programs, etc.) efficiently on current supercomputers but

also to perform researches on software technologies to bring out the full potential of the next generation of high-performance systems and technologies in the future. The sustainable progress of high-performance computing software is essential for the sustainable contribution to science and technology advancement and innovation by the high-performance computing.

In our research area, we are developing a high-performance, highly functional, and high-reliability system software including programming language, compiler, runtime system, operating system, communication middleware, file system, numerical calculation library, job management system, and ultra-large-scale data processing system software. In addition, from the viewpoint of sustainable contribution by the high-performance computing mentioned above, it should be not only academic research to demonstrate simple novel idea and its feasibility but the development of actual usable software. It is requested for the teams to make the developed software used in related community and is also to be emphasized in the evaluation.

(2) Exploration for the direction of future high-performance computing research and development

At the time of calls for the project proposal, it was not clear what kind of technologies would be used for the future next-generation high-performance computing systems, general-purpose many-core processor, or specialize hardware such as GPU. So, we requested to indicate the target system of the research and how to make the proposed software executed efficiently in the proposed target systems. The accepted research project teams were supposed to publish their research results as open-source software.

As the duration of the adopted project was 5 years, in intermediate evaluation at the third year, the team was requested to demonstrate that the developed software will be usable realized at the end of the project. By this request, we expected that the developed software would show the direction of system architecture of the future high-performance computing systems and be used actually in these systems including the next-generation system of the K computer or GPU-based accelerator systems. For this purpose, we aimed for international collaboration and industry-academia collaboration while sharing information with overseas researchers and companies.

Furthermore, when each research team is about to be finished (from 2016), we expected that the practical and usable software developed by the teams as a result of the project is used to enable advanced large-scale simulation and valuable prediction using large-scale data in a wide range of science and technology fields.

(3) Fostering of the next-generation leaders in the field of high-performance computing

In order to keep the ability of sustainable development of high-performance computing systems and promote the direction of future high-performance computing research and development, human resource who can bear it is indispensable. We actively appointed young researchers who were expected to be in the future at the selection of the projects and took into account the development of young research

leaders who will be responsible for research and development in the next generation of high-performance computing technology in Japan, by emphasizing autonomy in the planning, making teams and management.

1.3 Research Topics of the Project

In this section, the research topics and some highlights of the adopted projects are described briefly. As mentioned above, the research topics of our research area covers from system software such as programming languages, compilers, runtime systems, operating systems, communication middleware, and file systems to application development support software and ultra-large data processing systems.

Several teams carried researches on programming models and frameworks for post-petascale systems.

Maruyama's team (Chap. 5) has been working on several programming frameworks and libraries to make programming easy for the next generation of high-performance systems. They developed Daino, a high-level framework for parallel and efficient AMR on GPUs, and investigated effectiveness of high-level programming techniques such as Gridtools for global climate model simulation.

Endo's team (Chap. 12) was working on problems for recent trends on deeper memory hierarchy. For exascale high-performance systems, the "Memory Wall" problem will become even more severe. His team promotes research toward this problem via co-design approach among application algorithms, system software, and architecture. They have developed several libraries to make it easy to use the system of deeper memory hierarchy.

Chiba's team (Chap. 8) focused on productive programming models for post-petascale systems. A single general programming language or framework that covers all subjects will not be feasible for post-petascale supercomputing. Their goal is to apply modern techniques for software engineering and theoretical foundations of programming languages, such as software modularization, to high-performance computing.

Takizawa's team (Chap. 9) proposed the Xevolver framework which takes an evolutionary approach to incremental migration of existing software resources to new systems. The goal is to establish an effective migration path to new algorithms, implementation schemes, and programming environments for massively parallel and heterogeneous systems in an upcoming extreme-scale computing era.

Boku's team (Chap. 15) has been working on TCA (tightly coupled accelerators) by short-latency communication among GPUs over nodes to achieve strong scalability on next-generation accelerated computing and its programming model. They developed prototype system named PEACH2 which implemented by FPGA for flexible design and suitability for PCIe interface. As its extension, they proposed aggressive solution named AiS (Accelerator in Switch) to exploit high potential of recent FPGA. To solve the low productivity on programming by MPI plus CUDA

style, we are developing new language framework XcalableACC (XACC) where OpenACC description is involved to XcalableMP PGAS language.

The teams of system software cover researches from storage technology to communication middleware.

Tatebe's team (Chap. 6) carried researches on the next-generation storage technologies for post-petascale computing. The performance gap between CPU and storage is growing wider and wider. Distributed file system using compute-node local storage is promising to fill the gap. His team promotes node-local distributed file system and parallel and distributed execution framework for the file system.

Hori's team (Chap. 4) proposed a new process model for many-core. As in MPI program, the multiprocess model allows each process to own a private address space, though processes can allocate explicit shared memory regions. On the other hand, the multi-threaded model shares all address space by default, though threads can explicitly move data to thread-private storage. His team proposes a third model called Process-in-Process (PiP), where multiple processes are mapped into a single virtual address space, which may make use of many-core processor efficiently.

Nanri's team (Chap. 7) has been developing a PGAS-based communication library, ACP, and its applications. Memory efficiency of communication libraries is becoming important issue in large-scale parallel systems, where the number of processes is expected to be tens of millions. ACP enables both high-performance and low memory consumption for post-petascale systems.

Power consumption is now becoming a first class design constraint for developing future post-petascale computing systems. To achieve exa-flops-level performance with realistic power provisioning of 20–30 megawatts, significant power efficiency improvement over today's supercomputers is necessary.

Kondo's team (Chap. 13) has been working on this power issue in the next generation of high-performance computing system. In order to maximize effective performance within a power constraint, they proposed the power-constraint adaptive system design (P-CAS), which allows the system's peak power to exceed maximum power provisioning with adaptively controlling power-knows equipped in hardware components so that effective power consumption at runtime is under the power constraint.

Researches on numerical algorithms and libraries and software which support high-performance scientific applications also have been carried out.

Sakurai's team (Chap. 3) aimed to develop a massively parallel eigenvalue solver, eigen-supercomputing engine for post-petascale systems. Eigenvalue solver is one of the most important algorithms in several computational science applications. The eigen-engine is to be developed based on newly designed algorithms that are suited to the hierarchical architecture in post-petascale systems.

Nakajima's team (Chap. 2) has been developing "ppOpen-HPC," an open-source infrastructure for development and execution of optimized and reliable simulation code on post-petascale parallel computers based on many-core architectures, and it consists of various types of libraries, which cover general procedures for scientific computation.

Shioya's team (Chap. 10) has been developing an open-source software called ADVENTURE system. The ADVENTURE system is a general-purpose parallel finite element analysis system and can simulate a large-scale analysis model with supercomputer like the Earth Simulator or K computer. In the ADVENTURE system, HDDM (hierarchical domain decomposition method), a very effective technique for large-scale analysis, was developed. They aimed to develop a numerical library based on HDDM that is extended to pre- and post-processing parts, including mesh generation and visualization of large-scale data, for the post-petascale simulation.

Fujisawa's team (Chap. 11) has been developing advanced computing and optimization infrastructures for extremely large-scale graphs on post-petascale supercomputers. The large-scale graph analysis has attracted significant attention as a new application of the next-generation supercomputer. It is, however, extremely difficult to realize a high-speed graph processing in various application fields by utilizing previous methods. They aimed to develop advanced computing and optimization infrastructures for extremely large-scale graphs on the next-generation supercomputers.

Noda's team (Chap. 14) has been working on Project CASSIA (Comprehensive Architecture of Social Simulation for Inclusive Analysis) which aims to develop a framework to administer to execute large-scale multiagent simulations exhaustively to analyze socially interactive systems. The framework realizes engineering environment to design and synthesize social systems like traffics, economy, and politics.

1.4 Results and Achievements

The JST CREST post-petascale project has ended at the end of March 2018. At the end, the final evaluation was done, and the overall results were evaluated as "excellent" by the evaluation committee.

As planned at the beginning of the project, the goals described in Sect. 1.2 have been achieved.

Regarding research and development of highly functional and reliable system software for sustainable high-performance computing technologies, at first of all, the researchers of each team have published many excellent technical papers and valuable software as well as research presentations at prominent academic societies. Furthermore, making use of the developed software, large-scale applications are executed in many practical application fields from tsunami simulation, weather simulation to graph analysis, economic simulation, resulting in valuable contribution to solving social and scientific problems. In particular, it was showing not only the conventional large-scale numerical simulation but also the application of high-performance computing to new and important field such as big data analysis and social simulation. And, in the team having close to the real application, collaboration and joint research with the industries were actively carried out.

Exploration for the future direction of future high-performance computing research and development is an urgent and significant agenda in our research area. Although the subjects of this research area were mainly focused on software, at the time of setting research area, there was discussion on whether or not to conduct hardware research. During the project, the research done by many researchers has caught the trend of large-scale systems such as with many-core processors or accelerators from software's point of view, and consequently the researches is applied to the current systems, indicating the direction to the future high-performance computing systems. As well as conventional technologies for large-scale numerical computation, the project was also able to address the storage technology required for big data processing, the complexity of memory hierarchy, and the power problem. Furthermore, we have handled quickly to new trends such as FPGA after the start of the project.

As mentioned in Sect. 1.2 the project for the development of the next Japanese flagship supercomputer, the Post-K (Flagship 2020 project), was started from FY2014, and the development is under way. The operation for the public service will be scheduled around 2020. Many software developed in our project, such as numerical libraries and power control software, will be actually deployed and used for the post-K.

From the viewpoint of fostering human resources of the leaders in the next generation of high-performance computing, many excellent young researchers were produced as many young researchers in the project are awarded such as the "Young Encouragement Prize" and the "Research Award" in the community. It is thought as a big achievement that the research team took the initiative to organize numerous international and domestic symposiums, workshops, and seminars related to the research area, resulting in improvement of international awareness of leaders and researchers of the teams.

As a major international collaboration, the project collaborated with SPPEXA with Germany and France. Under the framework of the "Software for Exascale Computing (SPPEXA)" program which is implemented by DFG (Germany), JST (Japan) and ANR (France) agreed to support trilateral projects for high-performance computing. Some of teams have been awarded as a SPPEXA partner supported by JST. The research duration of the awarded teams was extended by 1 or 2 years, and the research results could be further improved through the international collaborative research.

The JST CREST post-petascale software project was expected to play a role bridging to post-petascale and exascale computing through system software. Currently, research and development of supercomputers aiming at the exascale computing is going on in several countries. It is important to continue several dissemination activities as well as for post-K. We hope that the technologies and software developed in the project should be used in several applications even after the research period and further drive the direction of research for the future high-performance computing.

Chapter 2

ppOpen-HPC/pK-Open-HPC: Application Development Framework with Automatic Tuning (AT)



Kengo Nakajima, Masaharu Matsumoto, Masatoshi Kawai,
Takahiro Katagiri, Takashi Arakawa, Hisashi Yashiro, and Akihiro Ida

Abstract ppOpen-HPC and pK-Open-HPC are open source infrastructures for development and execution of large-scale scientific applications on post-petascale (pp) supercomputers with automatic tuning (AT). Both of ppOpen-HPC and pK-Open-HPC focus on parallel computers based on many-core architectures and consist of various types of libraries covering general procedures for scientific computations. The source code, developed on a PC with a single processor, is linked with these libraries, and the parallel code generated is optimized for post-petascale systems. In this article, recent achievements and progress of the ppOpen-HPC and pK-Open-HPC project are summarized.

2.1 Overview of ppOpen-HPC

“ppOpen-HPC [1, 2]” is an open source infrastructure for development and execution of optimized and reliable simulation code on post-petascale (pp) parallel computers based on many-core architectures, and it consists of various types of libraries, which cover general procedures for scientific computation. Source code

K. Nakajima (✉) · M. Matsumoto · M. Kawai · A. Ida
The University of Tokyo, Tokyo, Japan
e-mail: nakajima@cc.u-tokyo.ac.jp; matsumoto@is.s.u-tokyo.ac.jp; masatoshi.kawai@riken.jp;
ida@cc.u-tokyo.ac.jp

T. Katagiri
Nagoya University, Nagoya, Aichi, Japan
e-mail: katagiri@cc.nagoya-u.ac.jp

T. Arakawa
Research Organization for Information Science and Technology (RIST), Tokyo, Japan
e-mail: arakawa@rist.jp

H. Yashiro
RIKEN Advanced Institute for Computational Science (AICS), Kobe, Hyogo, Japan
e-mail: h.yashiro@riken.jp

developed on a PC with a single processor is linked with these libraries, and the parallel code generated is optimized for post-petascale systems. The target post-petascale system is many-core-based systems, such as the Oakforest-PACS system operated by JCAHPC [3]. ppOpen-HPC is part of a 5-year project (FY.2011–2015) spawned by the “Development of System Software Technologies for Post-petascale High-Performance Computing [4]” funded by JST-CREST.

The framework covers various types of procedures for scientific computations, such as parallel I/O of datasets, matrix assembly, linear solvers with practical and scalable preconditioners, visualization, adaptive mesh refinement, and dynamic load balancing, in various types of computational models, such as FEM (finite element method), FDM (finite difference method), FVM (finite volume method), BEM (boundary element method), and DEM (discrete element method). Automatic tuning (AT) technology enables automatic generation of optimized libraries and applications under various types of environments. We release the most updated version of ppOpen-HPC as open source software every year in November (2012–2015), which is available at the home page of ppOpen-HPC [2].

In 2016, the team of ppOpen-HPC joined ESSEX-II (Equipping Sparse Solvers for Exascale) project (Leading P.I. Professor Gerhard Wellein (University of Erlangen-Nuremberg)) [5], which is funded by JST-CREST and the German DFG Priority Programme 1648 “Software for Exascale Computing” (SPPEXA) [6] under Japan (JST)-Germany (DFG) collaboration until FY2018. In ESSEX-II, we develop pK-Open-HPC (extended version of ppOpen-HPC, framework for ex-feasible applications), preconditioned iterative solvers for quantum sciences, and a framework for automatic tuning (AT) with performance model.

Original ppOpen-HPC includes the following four components (Fig. 2.1):

- ppOpen-APPL Frameworks for development of applications by FEM, FDM, FVM, BEM, and DEM (Fig. 2.2) [1, 2]

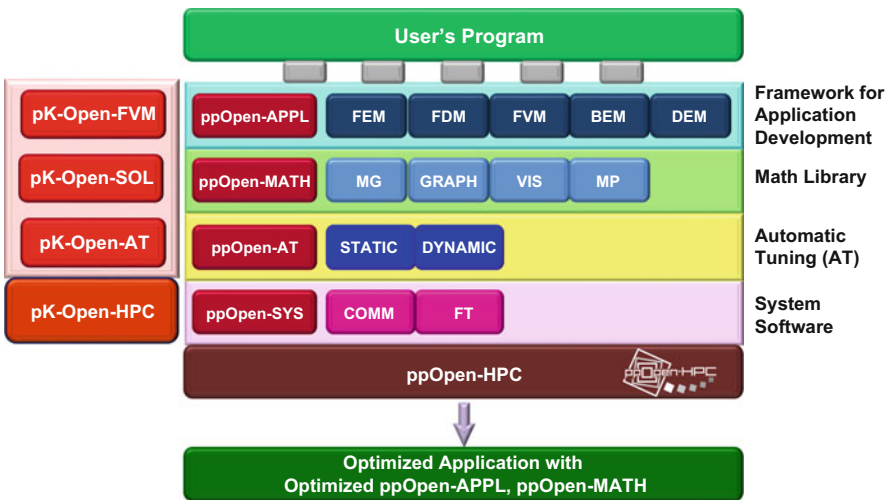


Fig. 2.1 Overview of ppOpen-HPC and pK-Open-HPC [1, 2]

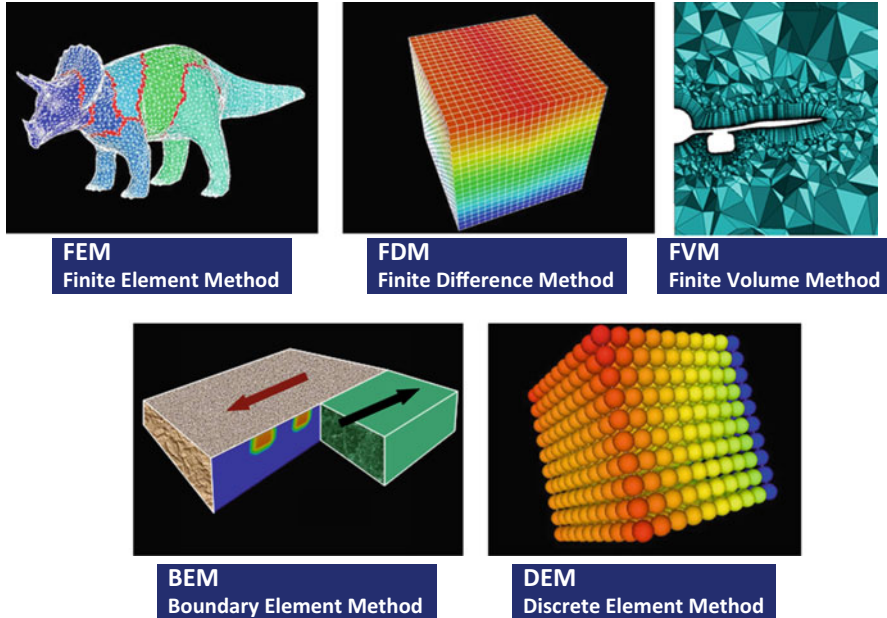


Fig. 2.2 Target applications of ppOpen-HPC and pK-Open-HPC

- ppOpen-MATH Math library
- ppOpen-AT Capability of automatic tuning
- ppOpen-SYS System software

In, pK-Open-HPC, we are focusing on the following issues and developed new libraries (Fig. 2.2)

- pK-Open-FVM Frameworks for development of applications by FVM with AMR (adaptive mesh refinement)
- pK-Open-SOL Robust and efficient preconditioned iterative solvers
- pK-Open-AT Capability of automatic tuning

In this article, we introduce recent developments and activities in ppOpen-HPC and pK-Open-HPC. The structure of this article with information of authors is as follows:

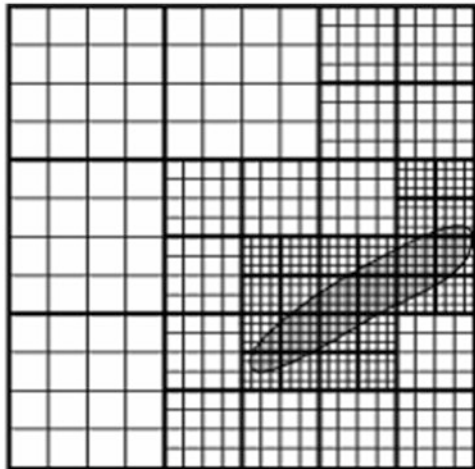
1. Overview of ppOpen-HPC (Kengo Nakajima)
2. Development of pK-Open-FVM (Masaharu Matsumoto)
3. Robust and massively parallelized preconditioner for quantum systems (Masatoshi Kawai) (related to pK-Open-SOL)
4. Automatic tuning (AT) in ppOpen-HPC and pK-Open-HPC (Takahiro Katagiri)
5. Development of a multi-physics coupler ppOpen-MATH/MP (Takashi Arakawa, Hisashi Yashiro)
6. Efficient structures of H-matrices on distributed memory computer systems (Akihiro Ida) (related to ppOpen-APPL/BEM and pK-Open-SOL)
7. Summary (Kengo Nakajima)

2.2 Development of pK-Open-FVM

Adaptive mesh refinement (AMR) technique [7, 8] can provide efficient numerical calculation by generating hierarchical layers with different cell sizes at the local regions where high resolution is needed. It is, however, generally difficult to implement the AMR treatment in conventional simulation codes discretized by finite volume method, finite difference method, and so on. To overcome this problem, a block-based AMR framework, pK-Open-FVM, with which the AMR technique can be relatively easily ported to generic simulation programs which hire the uniform cell system has been developed. In this framework, the AMR technique is applied to a block-structured region consisting of the fixed number of cells, as shown in Fig. 2.3. A generic simulation program using uniform cell size can be implemented in each block in the AMR framework [9]. Once a situation occurs where high resolution is needed in a local region, the corresponding block-structured region is divided into eight for three-dimensional case and new block-structured regions with uniform cell with half size of the original one are generated. In the AMR framework, the simulation domain is divided into multiple sub-domains, and they are assigned to a number of processes for parallel computing using MPI. A sub-domain is composed of multiple block-structured regions each of which has the fixed number of grids. When high resolution is required at a certain region in the sub-domain, a block-structured region with refined cells, which is called child block, is locally created.

For the application of the AMR framework, (1) Vlasov-Poisson plasma simulations, which are composed of 1D uniform direction and 1D AMR direction, (2) AMR plasma particle simulations for the development of reactive plasma deposition equipment, and (3) particle-based sugarscape model simulations are introduced as follows.

Fig. 2.3 Example of block-AMR



2.2.1 Application to Vlasov-Poisson Simulation

On large-scale computing, high-efficient calculation by AMR technique is one of the most attractive features. In particular, high-dimensional (≥ 4 dimension) simulation is suitable for the AMR simulation. Numerical simulation of Vlasov equation is one of the famous high-dimensional simulations in the plasma physics field. In Vlasov equation, a collision term on the right hand side of Boltzmann equation is eliminated.

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \frac{\partial f}{\partial \mathbf{x}} + \frac{\mathbf{F}}{m} \cdot \frac{\partial f}{\partial \mathbf{v}} = 0$$

Here, $f = f(\mathbf{x}, \mathbf{v}, t)$ is velocity distribution function, x, v are position, velocity, m is particle mass, and F is external force, respectively. This equation includes position coordinate, 3 dimension, + velocity coordinate, 3 dimension = total 6 dimension. It is difficult to solve this equation under the condition with six-dimensional coordinate because very large amount of memory is needed. Therefore, the function of AMR framework is extended to combine uniform mesh direction and AMR direction, and the extended function is applied to two-dimensional Vlasov-Poisson system (one-dimensional space and one-dimensional velocity). In this simulation, spatial direction is set to uniform 1D mesh because Poisson equation is solved to estimate electric field as an external force by fast Fourier transform, and velocity direction is set to adaptive mesh. Figure 2.4 shows the contour figure of distribution function at $t = 0, 128, 176$ s under the typical simulation conditions. The model calculated here is called “beam instability.” In this model, twin electron beams with

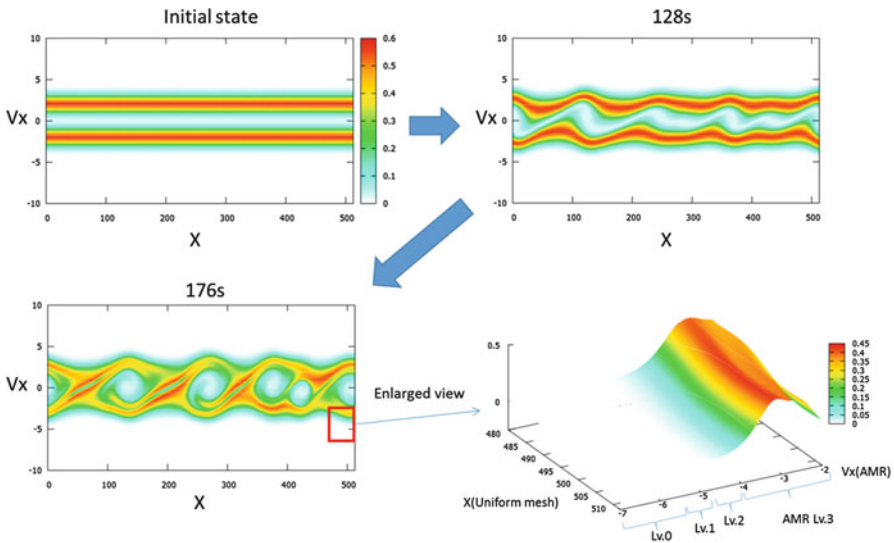


Fig. 2.4 Application of AMR to 1D-1D Vlasov-Poisson simulation

different components of velocity generate a vortex and combine into single electron beam as time proceeds. As criteria for mesh refinement, the gradient of velocity is set. When adaptive mesh direction (namely, velocity) have large gradient, mesh refinement occurs, and high-resolution simulation can be conducted. The lower right of Fig. 2.4 shows enlarged view of vortex. Lv. 3 block is generated where the gradient of velocity is large. The execution time of the AMR simulation can reduce 30% compared to the simulation with uniform mesh at velocity direction.

2.2.2 *AMR Particle Simulation for the Development of Reactive Plasma Deposition Equipment*

Plasma particle/fluid hybrid simulation code for the development of reactive plasma deposition (RPD) equipment (Fig. 2.5) has been developed by using pK-Open-FVM [10]. In this simulation, the interaction of electromagnetic fields and plasma particles can be simulated by calculating electromagnetic fields defined on the computational mesh and mesh-free plasma particles, simultaneously. In such a plasma particle simulation, in general, several hundreds of particles per one mesh are needed because the influence of statistical error depending on the number of particles is reduced.

In a conventional simulation with uniform mesh, the statistical error increases with increasing the difference of plasma density in the computational domain. In order to reduce statistical error, much more particles are needed and the execution time of simulation is also increased. On the other hand, by using the AMR

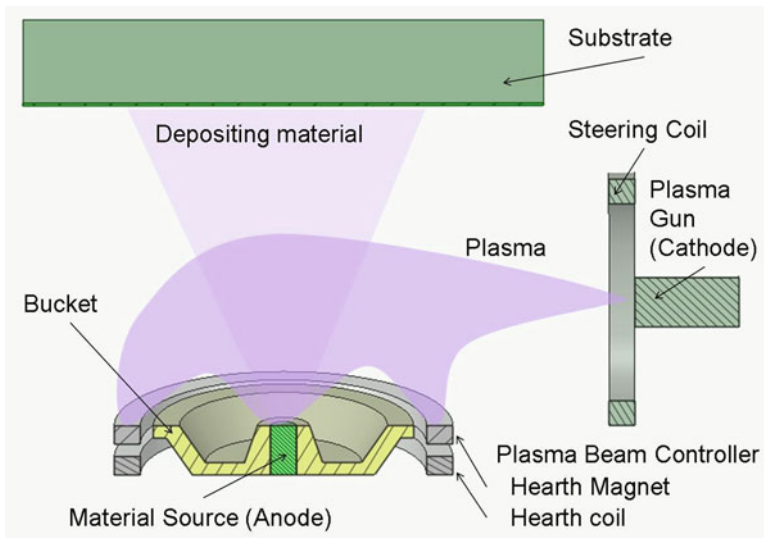


Fig. 2.5 RPD equipment

Fig. 2.6 Distribution of ion density and computational mesh

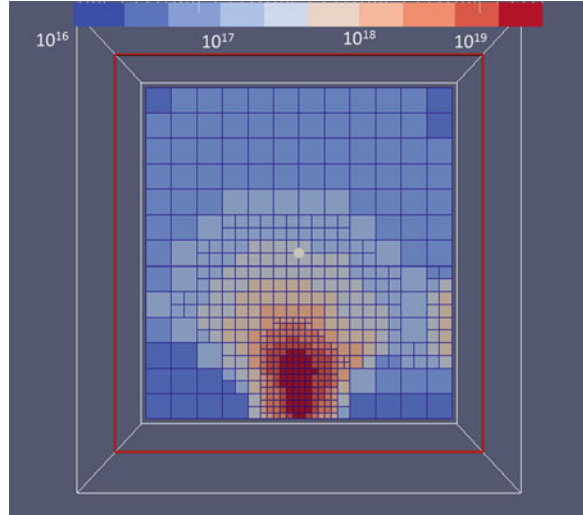
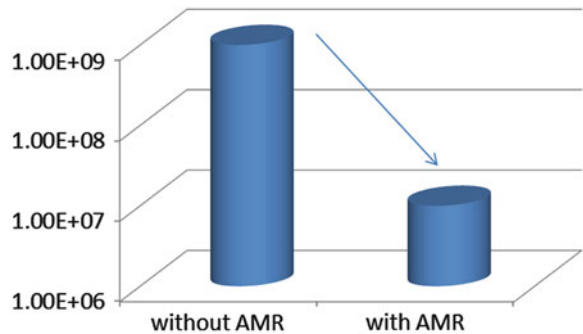


Fig. 2.7 The number of particles used in the simulation with/without AMR technique

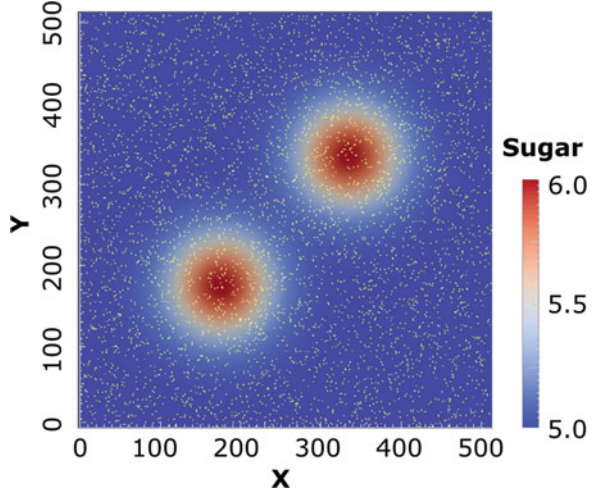


framework, cell size can be adjusted according to the number of particles (plasma density) (Fig. 2.6). That is, cell size becomes fine at high plasma density region or becomes coarse at low plasma density region. Furthermore, Fig. 2.7 shows the comparison of the number of particles used in the simulation with/without AMR technique. As a result, the total number of particles and execution time can be reduced by using AMR technique. The simulation using indium tin oxide as a material source which can't be conducted in conventional code can evaluate an experimental result (dependency of coil current), quantitatively.

2.2.3 Application to Particle-Based Sugarscape Model

Test simulations by adopting the sugarscape model which is proposed for the simulation for an artificial society by using many agents representing inhabitants in a certain area are conducted [11]. The inhabitants are treated as a bunch of particles and the sugar amount are assigned at each grid as the environment in a

Fig. 2.8 Initial distribution of sugar amount in the color map and inhabitants with dots



two-dimensional simulation domain. In the simulation, initially, two peaks of sugar are placed and randomly distribute the inhabitants in Fig. 2.8. In the PSS model, the inhabitant agents can move freely in the computational domain. The following are the equations of motion to be solved:

$$\begin{cases} \frac{\partial \mathbf{x}}{\partial t} = \mathbf{v} \\ \frac{\partial \mathbf{v}}{\partial t} = m\mathbf{F}_s - n\mathbf{F}_d \end{cases}$$

Here, \mathbf{F}_s is the spatial gradient of the sugar amount obtained at inhabitant position, and \mathbf{F}_d is the spatial gradient of the agent density obtained at inhabitant position. Therefore, from these equations, the inhabitant agents receive forces to the direction of high-sugar density region and low agent density region, as shown in Fig. 2.9.

2.3 Robust and Massively Parallelized Preconditioner for Quantum Systems

2.3.1 Objective

The objective of this research is to develop an iterative solver with robustness and exascale parallelism.

In the field of quantum systems, researchers are very interested in electrical, structural, and chemical properties of materials such as graphene and topological insulators. To clarify the physical properties of the target materials, we must solve the generalized eigenvalue problems. A collaborating project of ppOpen-HPC

Fig. 2.9 Distribution of inhabitants number density and refinement block

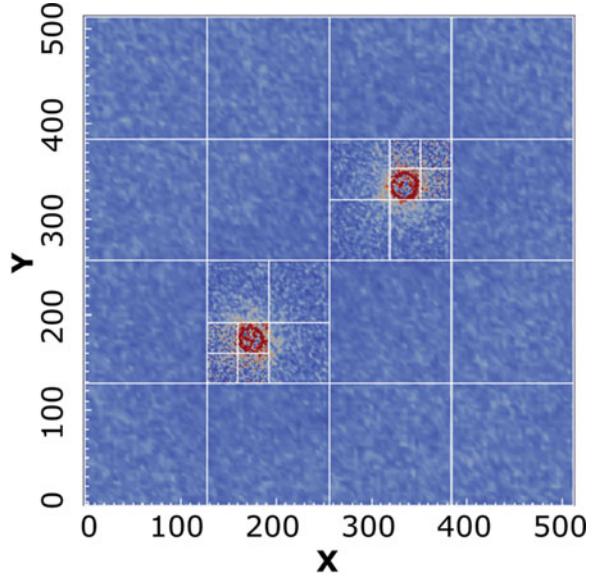
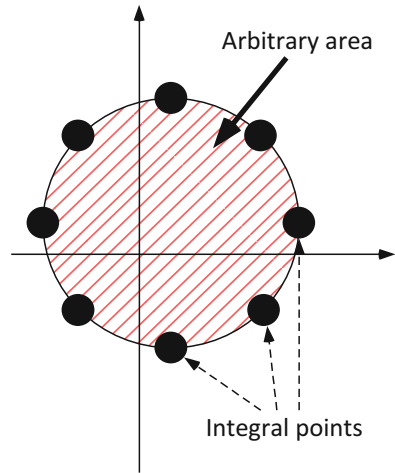


Fig. 2.10 Example of SS/FEAST method



(founded by CREST) and ESSEX-II (founded by SPPEXA) focuses on solving the problems using the Sakurai-Sugiura (SS) [12] or FEAST [13] method. By using these methods, we can calculate eigenvalue-eigenvector pairs that lie in an arbitrary area (Fig. 2.10). The area is described using a liner integral, and we must solve a system of simultaneous linear equations (SLEs) on integral points. Coefficient matrices A_z of the SLEs on each integral point are calculated as $A_z = zB - A$. Then, matrices A and B are defined by an application. These matrices derived from our target applications are sparse and large scale. The values of z are determined from the coordinate of each integral point. There is a possibility that ill-conditioned coefficient matrices are provided. Therefore, the requirements of the iterative solver are:

- Robustness
- Massive parallelism

In this study, we select an ILU preconditioned Krylov subspace method as the solver. In addition, we applied regularization methods to incomplete LU (ILU) preconditioner for the robustness [14]. For the massive parallelism, we proposed hierarchical parallelization for multicoloring algorithms [15].

2.3.2 Regularizations for Robustness

For robustness of the ILU preconditioned Krylov subspace method, we applied two regularization methods. The first method is a blocking technique, and the second is diagonal shifting. If conventional ILU factorization methods are applied to real-world applications, problems such as accumulation of rounding error or a breakdown of factorization are known to occur. The applied regularization methods increase the robustness for overcoming such problems.

2.3.2.1 Blocking Technique

The ILU preconditioner using a blocking technique is a well-known approach to improving the convergence. In our approach, we focus on the increasing robustness as well. The breakdown of ILU factorization occurs if diagonal entries of the target matrix are small. This is because the rounding errors accumulate by dividing the off-diagonal entries by diagonal entries. By applying a blocking technique, we obtain larger diagonal submatrices. The diagonal blocks include off-diagonal entries, as shown in Fig. 2.11.

2.3.2.2 Diagonal Shifting

This is a strong and direct method to make diagonal dominant matrices. If we apply diagonal shifting, the matrix \tilde{A}_Z , which should be incompletely factorized, is calculated as:

$$\tilde{A}_Z = A_Z + \alpha I \quad I = \text{identical matrix}$$

Then, the constant value α is decided by the user. α is then added to the diagonal entries of the target matrix A_Z , making it the diagonal dominant matrix \tilde{A}_Z . The larger value of α makes a more diagonal dominant matrix. However, the effect of ILU preconditioner gets smaller because of larger difference between A_Z and \tilde{A}_Z . The best parameter α depends on the target application and shift value z .

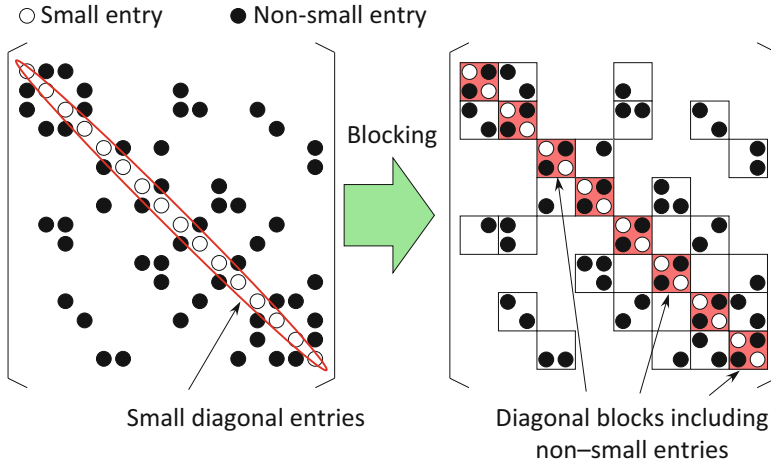


Fig. 2.11 The effect of the blocking technique

2.3.2.3 Numerical Evaluation

For numerical evaluations, we prepared 128 datasets. There are two types of different model. One model simulates the graphene, and the other simulates the topological insulator. Each model has four kinds of degrees of freedom (DoF) data, and each set of data has 16 data shifts. These models are symmetrical and complex values. For solving them, we implemented the block IC preconditioned conjugate orthogonal conjugate gradient (COCG) method. The BIC-COCG solver converged when the relative residual norm was less than 10^{-7} . If we applied only a blocking technique with a small size, we would solve only 64 datasets. However, by applying a larger block size (64) and (0.0, 1.0) data shifts, we solved all datasets (Fig. 2.12).

2.3.3 Hierarchical Parallelization of Multicoloring Algorithms for Massive Parallelism

The multicoloring is often used for parallelization of ILU factorizations and forward-backward substitutions. However, the coloring algorithms themselves are not parallelized in most reported research. To support the exascale system, we must parallelize them. In addition, the coloring algorithms have a significant impact on the convergence rate and performance of the ILU preconditioned Krylov subspace method. If local colorings are applied to parallelizing the ILU preconditioners for each MPI process, the convergence rate degrades as the number of processes increases. The multicoloring with hierarchical parallelization has a small influence on the convergence rate, because the proposed method provides global coloring.

Fig. 2.12 Numerical evaluation of the regularized IC-COCG

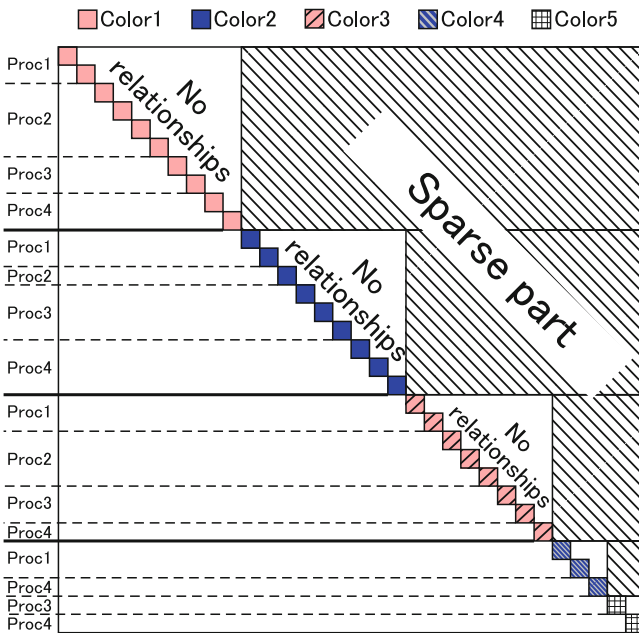
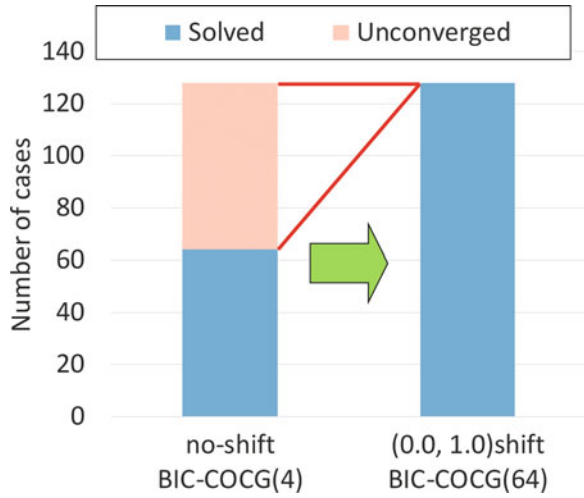


Fig. 2.13 Example of five-colored matrix for four processes

2.3.3.1 Parallelization of the ILU Preconditioner with Multicoloring

Generally, the ILU factorization and forward-backward substitution have sequentiality. To parallelize them, the multicoloring algorithms are widely applied. Figure 2.13 shows the example of a “three-colored matrix for four processes” parallelization. The diagonal submatrices colored with the same color have no relationships.

All processes calculate these diagonal submatrices and off-diagonal submatrices in the same row, parallelly.

2.3.3.2 Hierarchical Parallelization

As mentioned above, the result of the multicoloring has a strong impact on convergence and computational time per iteration. As a result, there are many kinds of coloring algorithms, such as Cuthill-McKee (CM), greedy [16], algebraic multicoloring (AMC) [17], and so on. The best algorithm depends on the application. Therefore, we need a versatile method that can parallelize existing algorithms.

To parallelize these algorithms, a hierarchical parallelization method that supplies colored groups is proposed. In the initial condition, each process stores a part of a coefficient matrix. First, each process separates handling elements into several groups (Fig. 2.14: Step 1). Second, each process makes a new graph that shows the relationships between separated groups, and a master process gathers the graph. After that, a master process colored the new graph (Fig. 2.14: Step 2). Third,

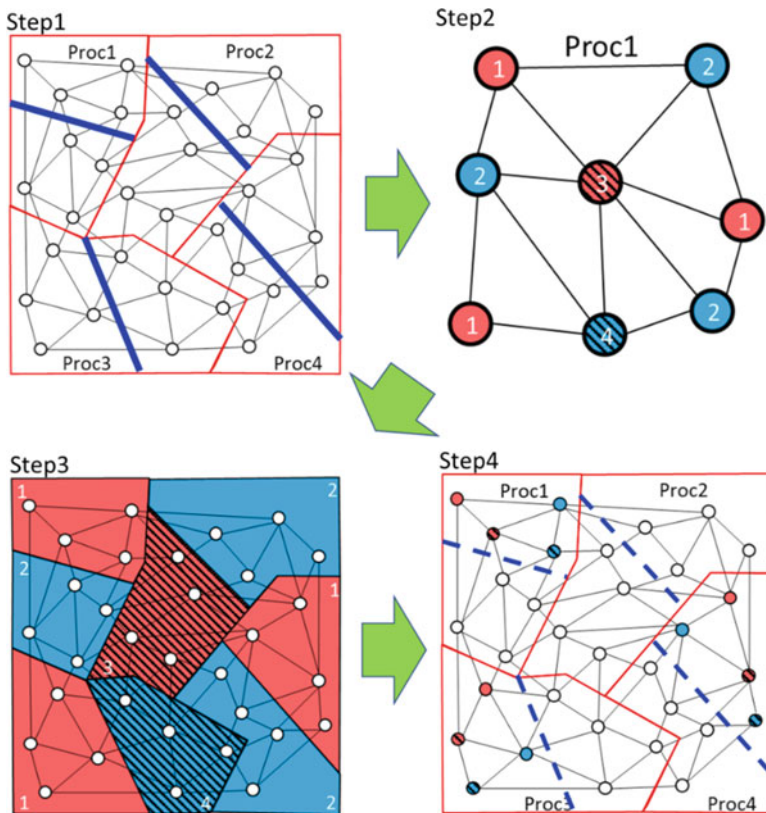


Fig. 2.14 Coloring process with hierarchical parallelization

the master process scatters the coloring result to the other processes. Using this approach, all processes get colored groups (Fig. 2.14: Step3). Finally, each process colored the handling elements in accordance with the colored groups (Fig. 2.14: Step 4).

2.3.3.3 Numerical Evaluation

In this section, we show that there is not a large difference of the convergence and performance between sequential hierarchical parallelized multicoloring. In addition, we show that the convergence of the ILU preconditioned Krylov subspace method parallelized with the proposed multicoloring does not influence the convergence. The target problems in this section are symmetric and have real values. Then, we use the block IC preconditioned CG (BICCG) method for numerical evaluations. The size of the block is four, and the constant value for the diagonal shifting is 100. The BICCG solver converged when the relative residual norm was less than 10^{-7} . We show numerical evaluations of the performance and convergence on several models. Numerical tests were conducted on the Reedbush-U supercomputer at the Information Technology Center, University of Tokyo, Japan.

The test problems for the numerical experiments were ParabolicFEM, Thermal2, Flan1565 [18], and Poisson (seven-points stencil, $128 \times 128 \times 256$ DOF). For the coloring, we used greedy and AMC methods with lexicographic and CM ordering. Figure 2.15 shows the comparison between the sequential and the hierarchical

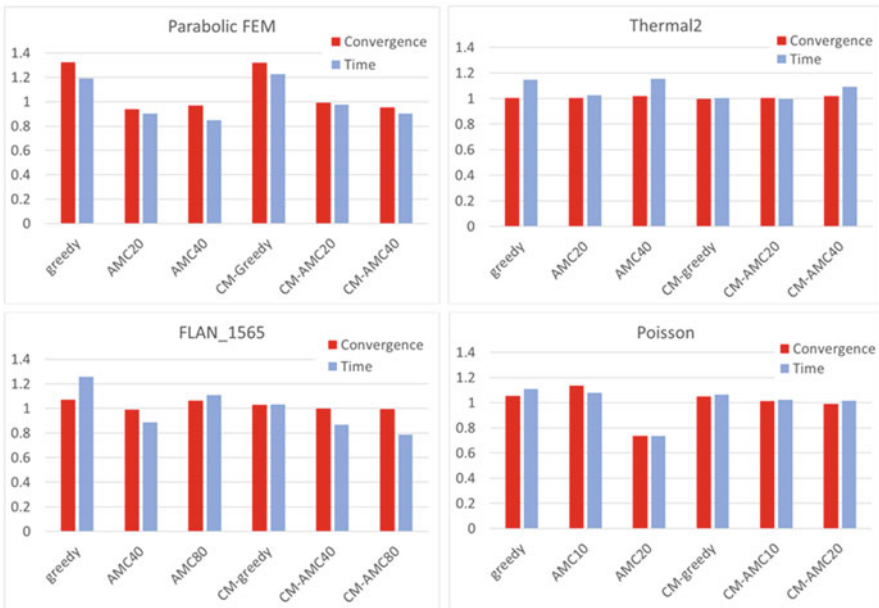


Fig. 2.15 Comparison between the sequential and the hierarchical parallelized multicoloring

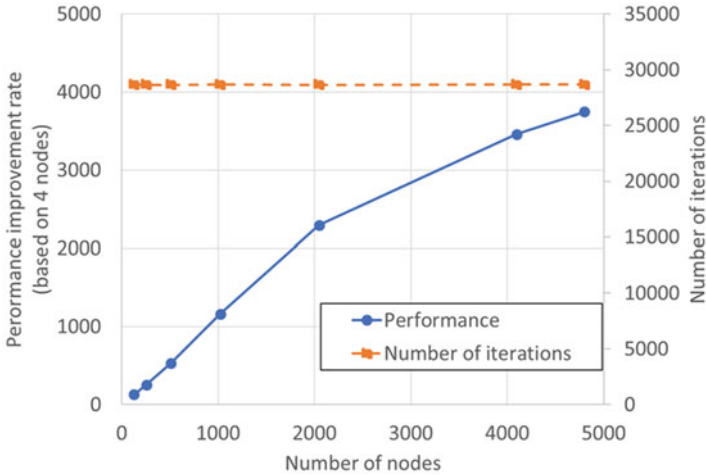


Fig. 2.16 Numerical evaluation of proposed method on Oakleaf-FX

parallelized coloring on 32 nodes. This figure shows the ratio of the number of iterations and the computational time required by the hierarchical parallel coloring method to those required by the sequential coloring method. The differences in the convergence and computational time between the sequential and hierarchical parallelized coloring are small. The most significant differences in the convergence and computational time are 32.3% and 26.6%, respectively.

Next, we show a numerical evaluation on a larger model with 4800 nodes cluster. Numerical tests were conducted on the Oakleaf-FX (SPARC64™ IXfx) supercomputer at the Information Technology Center, University of Tokyo, Japan. The program was parallelized with both OpenMP and MPI. The number of threads was 16 per process. The test problem for the numerical experiment was the graphene model. The DoF was 536,870,912. The coloring algorithm, which is parallelized with the hierarchical approach, is the ten-color AMC.

By applying the hierarchical parallelized algebraic multicoloring, the convergences for each number of processes are similar (Fig. 2.16, dashed line with square marks). Regarding the effect of constant convergence, the performance of the BICCG method is good.

2.3.4 Publication of Deliverable

To publicize the developing ILU preconditioner, we implemented our code in PHIST (Fig. 2.17) [5]. PHIST has an eigenvalue solver, such as the SS and FEAST method, and supports a change library such as GHOST (developed by ESSEX-II), MKL, CuBLAS, and MAGMA. The SLEs derived from FEAST in the PHIST are solved

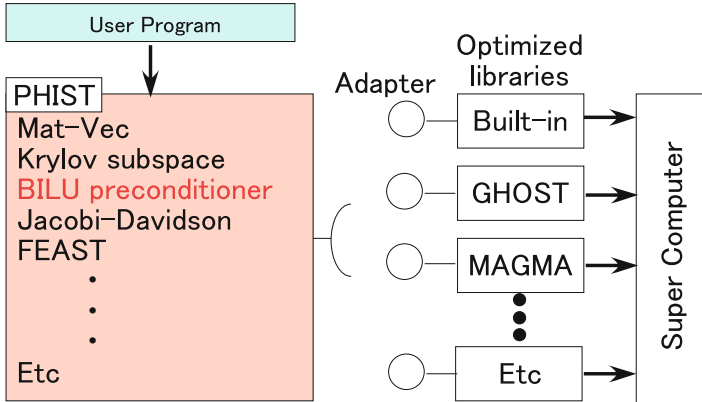


Fig. 2.17 Example of PHIST with the block ILU preconditioner [5]

by GMRES, BiCGStab, or any other iterative algorithms. By implementing our preconditioner in PHIST, we try to accelerate the iterative solver for the SS and FEAST.

2.4 Automatic Tuning (AT) in ppOpen-HPC and pK-Open-HPC

2.4.1 Functions of ppOpen-AT

ppOpen-MATH/MP is a coupling software applicable to the models employing various discretization methods such as FDM, finite volume method (FVM), and finite element method (FEM).

For functions of auto-tuning (AT) by ppOpen-AT, the following loop transformations as AT functions are proposed in our previous research [19].

- (a) Loop split and replacement of statements to optimize register optimizations
- (b) Loop collapse to obtain longer length of loops to optimize thread-level optimizations
- (c) Code selection with computational kernels between low Byte per Flops (B/F) and high B/F

With respect to the above three AT functions, in particular, (c) code selection with kernels between low B/F and high B/F is original AT facility for ppOpen-AT/FDM. The AT of code selection is aimed to optimize the code by lower level of memory optimization with respect to hierarchical memory configurations on current computer architectures. This optimization is categorized for “lower memory optimizations” to cache memories, such as to level 1 cache, level 2 cache, and

registers by adapting the above loop split, for example. On the other hand, we categorize “upper memory optimizations” to under the lowest cache, such as last level cache (LLC), and including memory, such as 3D stacked memory with highly bandwidth.

We suppose code optimization for the upper memory optimizations as like algorithm-level optimizations. Hence, we call the code optimization for upper memory “code selection.”

A *select* construct can be specified by ppOpen-AT, which is one of AT languages for the above code optimizations to upper memory optimizations, while lower memory optimizations, such as register optimization, can be specified by inner processes on the target function specified by ppOpen-AT. Hence the AT by ppOpen-AT can provide the higher and lower code optimizations.

2.4.2 Target Users

We focus on the following users for ppOpen-AT.

- (a) Users who are using ppOpen-APPL/FDM as a black-box tool
- (b) Users who want to know effective tuning strategy for newly released computers
- (c) Researchers of AT methodology

For the users (a), they receive a merit for execution speedup with ppOpen-AT. For the users (b), they receive actual tuned codes and effect of tuning for the newly released computers, such as fast method of loop collapse by referring history of AT by ppOpen-APPL/FDM.

For the researches (c), they can compare effect of proposal method by them with compared to tuned execution time by ppOpen-APPL/FDM.

Hence there are many potential users for ppOpen-AT.

2.4.3 Performance Evaluation

The following three kinds of processes are implemented in this performance evaluation:

- (a) *Baseline code*: Ratio of B/F is high (approximately 1.7). Experimentally, the code is the best code for vector machines.
- (b) *Loop collapse code* (including IF-sentences): (i) Fourth-order central difference scheme, (ii) process of boundary area, and (iii) time-step expansion for leapfrog scheme are included into a conventional loop; then make it one loop. To apply this modification, ratio of B/F can be reduced to approximately 0.4. Hence this makes a loop with low B/F ratio.

- (c) *Loop collapse code* (IF-sentences free): (i) A loop with IF-sentences free and (ii) a loop with dedicated process of boundary area are made in the code of (b). For the (i), it is expected to obtain better code to the loop in (b) by compilers, because of simple code without IF-sentences.

We evaluate proposal AT methodology by implementing the code selection function based on the above three codes. Performance evaluation of ppOpen-APPL/FDM are evaluated with the Intel Xeon Phi (Knights Landing, KNL) by using a node of the Oakforest-PACS, which is installed in JCHPCH (The University of Tokyo and Tsukuba University, Japan), and the Intel Broadwell EP (BDW) by using a node of the Reedbush-U, which is installed in Information Technology Center, The University of Tokyo, and the Sparc64 Xlfx (FX100) by a node of the Fujitsu PRIMEHPC FX100, and the Haswell-EP by a node of the Fujitsu CX400, which is installed in Information Technology Center, Nagoya University.

It is implemented an advanced memory with 3D stacked memory for memory on KNL and FX100. Both effective performance of memory bandwidth is approximately 300 GB/s. Effect of AT is presented in [20]. The obtained knowledge from [20] is summarized as follows:

- With FLAT-QUADRANT mode for memory and core job allocation for the KNL, factor of speedups by AT is reached to 1.33x at maximum.
- Selected the best implementation for kernel of *update_stress*, (b) loop collapse code (including IF-sentences) is the best for the KNL, while (c) loop collapse code (IF-sentences free) is the best for the others (the FX100 and the Haswell-EP). The best implementation depends on computer architectures, hence the proposed AT methodology effects well for current advanced CPU environments.

Figure 2.18a, b show execution time between the BDW and the KNL by AT and normalized factors of speedups for the KNL based on execution time in the BDW. Fig. 2.18a shows that maximum 2.4x speedup is established for the KNL



Fig. 2.18 Effect of AT and comparison of execution time. It shows aggregated execution time of kernel *update_stress* up to 2000 time steps. (a) The best execution time between BDW and KNL between original and with AT. (b) Normalized speedup factors of the KNL based on execution time in the BDW

to the BDW, and the BRD obtains much speedups to the KNL between original and with AT. It has almost 3x bandwidth to the BDW with respect to memory bandwidth for the KNL. Hence it has a room to optimize codes for the KNL, but it is reasonable performance for the KNL in viewpoint of hardware ability. For summary of this performance evaluation, we show speedup by the AT we proposed with an application of FDM. This also indicates that crucial speedups can be obtained by AT we have developed in this project.

2.5 Development of a Multi-physics Coupler ppOpen-MATH/MP

2.5.1 Background

Recent progress on high-end supercomputers has enabled larger scale, more complex simulations. Owing to this progress, the phenomena currently possible to simulate have become more complex, and software applications seen as a projection of actual natural phenomena have also become larger and larger. As an example, we take a climate model that is one of the most typical complex system simulation models in use today. According to [21], Climate model NCAR CCSM was actually composed of two component models, atmosphere and ocean, at the beginning of its development. In the 1980s, a sea ice model was added, and an aerosol model was added in the 1990s. Over time, interactive vegetation and carbon cycle models have also been added. These component models are usually developed independently by researchers in the field of the respective research and have a grid space and grid structure suitable to the phenomena that the model represents. Therefore, on a multi-physics simulation, a coupling program that overcomes the difference of code structure and grid system has acquired an important role.

Because of the background described above, we have been developing a coupler called ppOpen-MATH/MP. ppOpen-MATH/MP is designed to be applicable to models employing various discretization methods supported by ppOpen-HPC software suites such as FDM, FVM, and FEM. This wide applicability is achieved by being independent from the grid structure and is designed so that users can implement their own interpolation code.

2.5.2 NICAM-COCO Coupling

To demonstrate the applicability of ppOpen-MATH/MP, we utilized it to achieve the coupling of an atmospheric model and an ocean model. The atmospheric model selected for this purpose is NICAM, a non-hydrostatic global model employing an icosahedral grid system and the FVM discretization method [24]. The CCSR Ocean

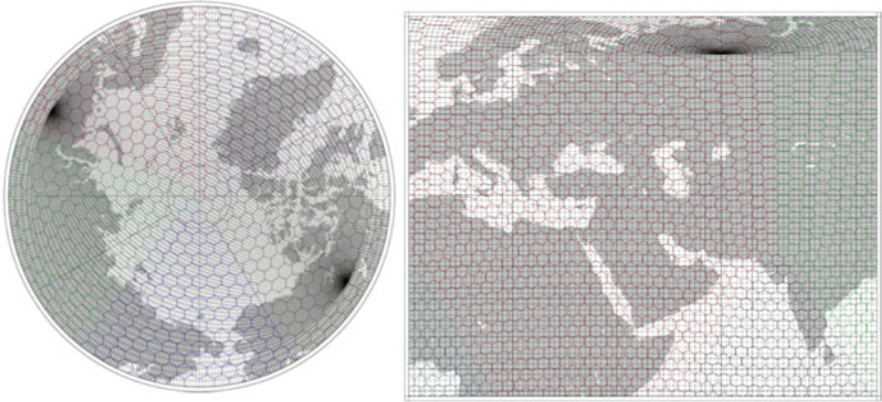


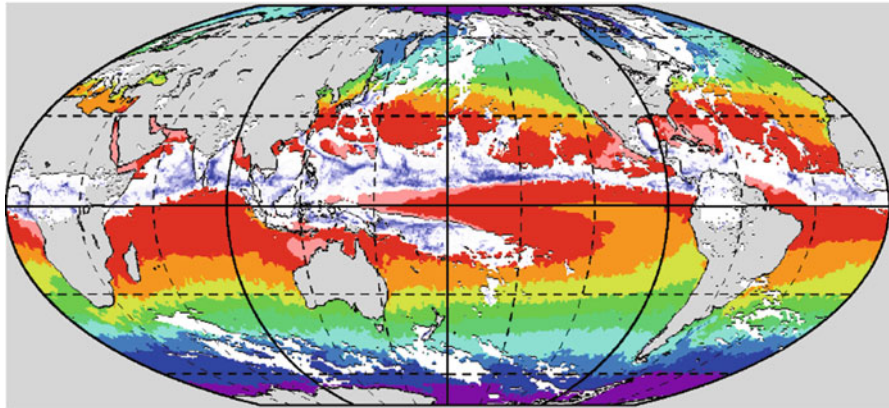
Fig. 2.19 Grid structure of NICAM and COCO. The colored hexagonal grid represents NICAM grid and the black rectangular grid represents COCO grid

Component Model, COCO [24], is used as the ocean model to be coupled with NICAM. COCO adopts a tripolar grid, in which the northern polar region grid points do not follow latitude-longitude, and the discretization method is FDM. Figure 2.19 shows the grid structure. The colored hexagonal grid represents the NICAM grid, and the black rectangular grid represents the COCO grid. As stated previously, for achieving wide applicability, ppOpen-MATH/MP is designed so that users can implement their own interpolation code. The interpolation algorithm utilized in this study is based on the first-order conservative remapping scheme in [22]. Physical quantities exchanged from NICAM to COCO are comprised of 13 variables such as wind speed, heat flux, and precipitation. And the number of quantities exchanged from COCO to NICAM is comprised of six variables such as SST and sea ice thickness.

2.5.3 Coupling Results

Coupled simulation was performed on the Fujitsu Oakforest-PACS of the University of Tokyo. Figure 2.20 shows the result, total precipitation, and monthly mean sea surface temperature of October. Horizontal resolution is about 14 km on NICAM and 0.1 degree on COCO. We can see that the realistic distribution is successfully represented. In more detail, it can be seen that several precipitation band extends from the western Pacific equatorial region to the Japanese archipelago. This represents the course of the typhoon that approached in October. In addition to high-resolution calculations as shown above, low-resolution NICAM-COCO coupling by ppOpen-MATH/MP has been also used for realistic meteorological simulation studies such as Madden-Julian oscillation reproduction experiments [25].

Total precipitation and sea surface temperature
Monthly average of October



NICAM 14km x COCO 0.1deg

Fig. 2.20 Total precipitation (blue scale) and average sea surface temperature (color scale) of October on NICAM-COCO coupled simulation. Horizontal resolution is 14 km on NICAM and 0.1 degree on COCO

2.5.4 NICAM-COCO and I/O Component

In addition to NICAM-COCO coupling, we have implemented NICAM and I/O component coupling. The reason for this coupling is that the icosahedral grid employed by NICAM is not suitable for analyzing the results. For example, the calculation of the zonal mean values is not straightforward, and the visualization tools assume a latitude-longitude grid (lat-lon grid) in many cases. So, we developed an IO program that converts the icosahedral grid to the lat-lon grid and is executed in parallel with NICAM. We implemented three types of conversion schemes, a bilinear interpolation, a control volume weighted average, and the nearest neighbor method. The I/O component is designed so that the user can select one of these schemes for each set of output data by using a configuration file. Figure 2.21 is a schematic of the coupling system described above. The coupling system is designed so that NICAM automatically detects the coupling pattern at runtime without any other configuration by the user. For example, when COCO is executed in parallel with NICAM, subroutines for NICAM-COCO coupling are used, and if not, subroutines of the mixed layer ocean model are called. IO is also the same as in the case of COCO, in which NICAM automatically sends output data to I/O only when the I/O component is executed.

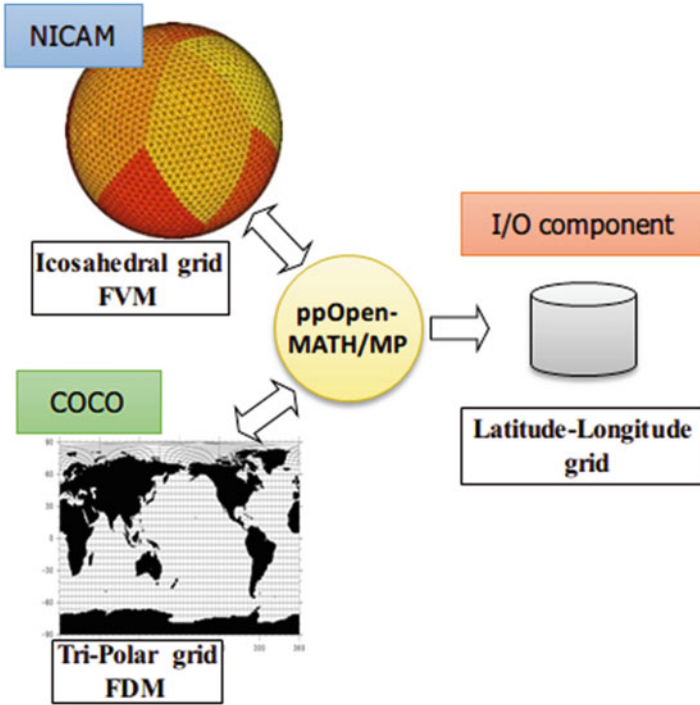


Fig. 2.21 A schematic of NICAM coupling system. For simplicity ppOpen-MATH/MP is illustrated as an independent component, but actually it works as a library called from each component

2.6 Efficient Structures of H-Matrices on Distributed Memory Computer Systems

The ppOpen-APPL/BEM, which is a software for BEM analyses, consists of BEM-BB framework and \mathcal{H} ACApK library. All the components are parallelized based on the hybrid MPI + OpenMP programming model. By using the BEM-BB framework, the users can easily develop a parallel BEM code for their own application by adding a user function describing the integral operation to the framework [26]. To overcome a disadvantage of BEM that the complexity of at least $O(N^2)$ is required, we also provide the \mathcal{H} ACApK library which adopts hierarchical matrices (\mathcal{H} -matrices) as low-rank structured matrices. The idea of \mathcal{H} -matrices is based on the fact that submatrices corresponding to remote interactions become numerically low-rank. \mathcal{H} -matrices reduce the complexity from $O(N^2)$ to $O(N \log N)$. For parallelization of \mathcal{H} -matrices on distributed memory computer systems, we have proposed a set of algorithms for \mathcal{H} -matrices construction and performing \mathcal{H} -matrix-vector multiplication [27], which employed in \mathcal{H} ACApK. In performance tests using electric field analyses, the \mathcal{H} ACApK exhibits a parallel speedup for the \mathcal{H} -matrix-vector multiplication up to about a few hundred MPI processes.

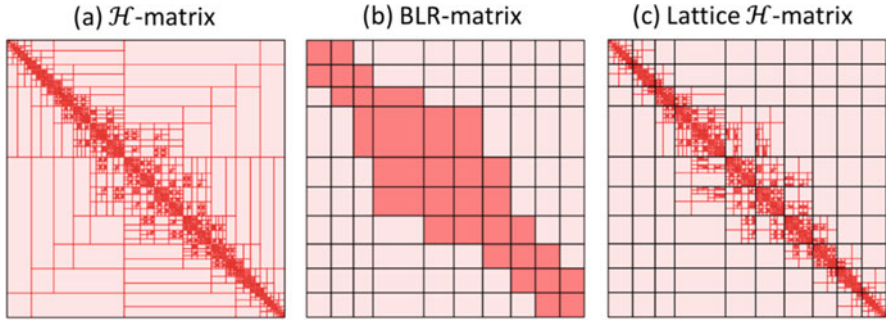


Fig. 2.22 An \mathcal{H} -matrix structure coming from normal \mathcal{H} -matrices (a) and its conversion to a BLR (b) and a lattice \mathcal{H} -matrix (c). Blocks painted in deep red show dense submatrices and blocks in light red indicate low-rank submatrices

For massively parallel processing (MPP), key points are to achieve good load balancing and to construct efficient communication pattern among MPI processes. Unfortunately, the partition structures of \mathcal{H} -matrices are too complicated as shown in Fig. 2.22a to meet these requirements. Simplifying the partition structures would be one possible approach to meet requirements for MPP. Some simplified formats are proposed in the literature and they can be regarded as special cases of \mathcal{H} -matrices. The block low-rank (BLR) matrices have the simpler structure as shown in Fig. 2.22b, and the structure is one of the most convenient ones for the approach above. However, as a trade-off, the memory usage increases from \mathcal{H} -matrices $O(N \log N)$ to $O(N^{1.5})$. In [28], we discussed the applicability of BLR-matrices instead of \mathcal{H} -matrices for large-scaled BEM analyses on a distributed memory computer system. The implementation of the BLR using our proposed exhibits a parallel speedup up to about 10,000 MPI processes for the test problems that H-matrices suffered from a saturation of speedup around 100 MPI processes. We confirm that the BLR version is significantly faster than the normal \mathcal{H} -matrix version, given a large number of CPU cores.

In [29], we proposed a new variant of low-rank structured matrices, named “lattice \mathcal{H} -matrices.” As shown in Fig. 2.22, the lattice \mathcal{H} -matrices appear to be a hybrid of normal \mathcal{H} -matrices and BLR-matrices. Lattice \mathcal{H} -matrices are defined by introducing the lattice structure into normal H-matrices. In other words, the lattice \mathcal{H} -matrices are constructed by utilizing \mathcal{H} -matrices as submatrices in blocks of the lattice structures observed in BLR-matrices. When assigning the lattice blocks to MPI processes, we expect the lattice \mathcal{H} -matrices to maintain the advantages of both \mathcal{H} -matrices and BLR-matrices. The advantages are the high compressibility of \mathcal{H} -matrices, which reduces the memory usage, and the convenience of matrix arithmetic with BLR-matrices. We can restrict ourselves to performing complex arithmetic originating from the \mathcal{H} -matrices structure only in serial computing or thread processing on a CPU node, while we can utilize sophisticated existing parallel algorithms for dense matrices used in ScaLAPACK based on efficient

communication patterns between MPI processes on distributed memory systems. Thanks to the complex structure in each block of the lattice, lattice \mathcal{H} -matrices reduce the memory complexity from BLR $O(N^{1.5})$ to $O(N \log N)$. We confirmed the memory complexity both in theory and in practical experiments. In numerical experiments of electric field analyses, we confirmed that a relatively good load balance is maintained in the case of lattice \mathcal{H} -matrices even if we use a large number of MPI processes. The execution time of lattice \mathcal{H} -matrices is slightly larger when normal \mathcal{H} -matrices continue to demonstrate a speedup. In contrast to the speedup saturation with only a few dozen processors for normal \mathcal{H} -matrices, the implementation of the lattice \mathcal{H} -matrices exhibits a parallel speedup that reaches as high as 4000 MPI processes. It is confirmed that the implementation of lattice \mathcal{H} -matrix version is significantly faster than of normal \mathcal{H} -matrix version if we use more than several hundred MPI processes.

2.7 Summary

In this article, recent achievements and progress of the ppOpen-HPC and pK-Open-HPC have been presented. The libraries developed for ppOpen-HPC and pK-Open-HPC are open for public use under MIT license and can be downloaded at the website of the project [1]. ppOpen-HPC and pK-Open-HPC have been installed on various types of super computers, and are utilized for research and development that requires large-scale supercomputer systems. Moreover, ppOpen-HPC and pK-Open-HPC are introduced in graduate and undergraduate classes at universities. Some of the development of pK-Open-SOL is implemented to PHIS [5]. Currently, we are preparing for further research and development toward ppOpen-HPC and pK-Open-HPC on exascale and Post Moore systems.

Acknowledgments This work is supported by Core Research for Evolutional Science and Technology (CREST), the Japan Science and Technology Agency (JST), Japan, and the German Priority Programme 1648 Software for Exascale Computing (SPPEXA-II). Authors would like to thank Professor Gerhard Wellein (Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany) and members of the ESSEX-II project.

References

1. ppOpen-HPC.: <http://ppopenhpc.cc.u-tokyo.ac.jp/>. Accessed 23 Mar 2018
2. Nakajima, K., Satoh, M., Furumura, T., Okuda, H., Iwashita, T., Sakaguchi, H., Katagiri, T., Matsumoto, M., Ohshima, M., Jitsumoto, H., Arakawa, T., Mori, F., Kitayama, T., Ida, A., Matsuo, M.Y.: ppOpen-HPC: open source infrastructure for development and execution of large-scale scientific applications on post-peta-scale supercomputers with automatic tuning (AT). In: Optimization in the Real World -Towards Solving Real-Worlds Optimization Problems, Mathematics for Industry, vol. 13, pp. 15–35. Springer, Beijing (2015)

3. Joint Center for Advanced High Performance Computing (JCAHPC): <http://jcahpc.jp/>. Accessed 23 Mar 2018
4. Post-Peta CREST.: <http://postpeta.jst.go.jp/en/>. Accessed 23 Mar 2018
5. ESSEX.: <http://blogs.fau.de/esssex/>. Accessed 23 Mar 2018
6. SPPEXA.: <http://www.sppexa.de/>. Accessed 23 Mar 2018
7. Dezeeuw, D., Powell, K.G.: An adaptively refined Cartesian mesh solver for the Euler equations. *J. Comput. Phys.* **104**, 56–68 (1993)
8. Matsumoto, M., Mori, F., Ohshima, S., Jitsumoto, H., Katagiri, T., Nakajima, K.: Implementation and evaluation of an AMR framework for FDM applications. *Proc. Comput. Sci.* **29**, 936–946 (2014)
9. Usui, H., Nagara, A., Nunami, M., Matsumoto, M.: Development of a computational framework for block-based AMR simulations. *Proc. Comput. Sci.* **29**, 2351–2359 (2014)
10. Miyashita, M., Matsumoto, M., Kubota, K.: The development of adaptive mesh refinement technique for hybrid kinetic/fluid plasma simulation. In: Proceedings of 2015 International Conference on Numerical Simulation of Plasmas (ICNSP2015) (2015)
11. Usui, H., Kito, S., Nunami, M., Matsumoto, M.: Application of block-structured adaptive mesh refinement to particle simulation. *Proc. Comput. Sci.* **108**, 2527–2536 (2017)
12. Sakurai, T., Sugiura, H.: A projection method for generalized eigenvalue problems using numerical integration. *J. Comput. Appl. Math.* **159**(1), 119–128 (2003)
13. Galgon, M., Lukas, K., Lang, B.: The FEAST algorithm for large eigenvalue problems. *Proc. Parallel Appl. Math. Mech. (PAMM)*. **11**(1), 747–748 (2011)
14. Kawai, M., Ida, A., Nakajima, K.: Modified IC Preconditioner of CG method for ill-conditioned problems. IPSJ SIG, Technical Report 2017-HPC-158-9 (2017.) (in Japanese)
15. Kawai, M., Ida, A., Nakajima, K.: Hierarchical parallelization of multi-coloring algorithms for block IC, preconditioners. In: IEEE 19th International Conference on High Performance Computing and Communications (HPCC), pp. 138–145 (2017)
16. Saad, Y.: *Iterative Methods for Sparse Linear Systems*, 2nd edn. SIAM, Philadelphia (2003)
17. Iwashita, T., Shimasaki, M.: Algebraic multicolor ordering for parallelized ICCG solver in finite-element analyses. *IEEE Trans. Magn.* **38**(2), 429–432 (2002)
18. Davis, T.A., Hu, Y.: The University of Florida sparse matrix collection. *ACM Trans. Math. Softw. (TOMS)*. **38**(1), 1–25 (2011)
19. Katagiri, T., Ohshima, S., Matsumoto, M.: Directive-based auto-tuning for the finite difference method on the Xeon Phi. In: IEEE Proceedings of IPDPSW2015, pp. 1221–1230 (2015)
20. Katagiri, T., Ohshima, S., Matsumoto, M.: Auto-tuning on NUMA and many-core environments with an FDM code. In: IEEE Proceedings of IPDPSW2017, pp. 1399–1407 (2017)
21. Washington, W.M., Buja, L., Craig, A.: The computational future for climate and earth system models: on the path to petaflop and beyond. *Phil. Trans. R. Soc. A.* **367**, 833–846 (2009)
22. Satoh, M., Tomita, H., Yoshiro, H., Miura, H., Kodama, C., Seiki, T., Noda, A., Yamada, Y., Goto, D., Sawada, M., Miyoshi, T., Niwa, Y., Hara, M., Ohno, T., Iga, S., Arakawa, T., Inoue, T., Kubokawa, H.: The non-hydrostatic icosahedral atmospheric model: description and development. *Prog Earth Planet Sci.* **1**, (2014). <https://doi.org/10.1186/s40645-014-0018-1>
23. Hasumi, H.: CCSR Ocean Component Model (COCO) Version 4.0. <http://ccsr.aori.utokyo.ac.jp/hasumi/COCO/coco4.pdf>. Accessed 23 Mar 2018
24. Jones, P.H.: First and second order conservative remapping schemes for grids in spherical coordinates. *Mon. Weather Rev.* **127**, 2204–2210 (1999)
25. Miyakawa, T., Yoshiro, H., Suzuki, T., Tatebe, H., Satoh, M.: A Madden-Julian oscillation-event remotely accelerates ocean upwelling to abruptly terminate the 1997/1998 super El Nino. *Geophys. Res. Lett.* **44**, 9489 (2017). <https://doi.org/10.1002/2017GL074683>
26. Iwashita, T., Ida, A., Mifune, T., Takahashi, Y.: Software framework for parallel BEM analyses with H-matrices using MPI and OpenMP. *Proc. Comput. Sci.* **108C**, 2200–2209 (2017)
27. Ida, A., Iwashita, T., Mifune, T., Takahashi, Y.: Parallel hierarchical matrices with adaptive cross approximation on symmetric multiprocessing clusters. *J. Inf. Process.* **22**(4), 642–650 (2014)

28. Ida, A., Nakashima, H., Kawai, M.: Parallel hierarchical matrices with block low-rank representation on distributed memory computer systems. In: ACM Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2018), pp. 232–240 (2018)
29. Ida, A.: Lattice H-matrices on distributed-memory systems. In: Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS) (2018). in press

Chapter 3

Scalable Eigen-Analysis Engine for Large-Scale Eigenvalue Problems



Tetsuya Sakurai, Yasunori Futamura, Akira Imakura,
and Toshiyuki Imamura

Abstract Our project aims to develop a massively parallel Eigen-Supercomputing Engine for post-petascale systems. Our Eigen-Engines are based on newly designed algorithms that are suited to the hierarchical architecture in post-petascale systems and show very good performance on petascale systems including K computer. In this paper, we introduce our Eigen-Supercomputing Engines: z-Pares and EigenExa and their performance.

3.1 Introduction

Our project aims to develop a massively parallel Eigen-Supercomputing Engine for post-petascale systems. This Eigen-Engine is to be developed based on newly designed algorithms that are suited to the hierarchical architecture in post-petascale systems. Our Eigen-Engine is expected to overcome issues of scalability and fault tolerance in conventional eigensolvers. To achieve the aim and provide a high-performance Eigen-Engine that can contribute to practical applications, six research groups are organized, and a variety of studies and developments are conducted through collaboration with researchers in applied mathematics HPC and application fields. Our Eigen-Engine will make it possible to do extensive scale scientific computations that are impossible today and open the door to innovation in various fields of science and industry.

We have collaboratively developed two Eigen-Engines: z-Pares for sparse eigenvalue problems and EigenExa for dense eigenvalue problems. These engines consist of building blocks developed in the project, and their performance have been evaluated in actual applications.

T. Sakurai (✉) · Y. Futamura · A. Imakura
University of Tsukuba, Tsukuba, Ibaraki, Japan
e-mail: sakurai@cs.tsukuba.ac.jp; futamura@cs.tsukuba.ac.jp; imakura@cs.tsukuba.ac.jp

T. Imamura
RIKEN Center for Computational Science, Kobe, Hyogo, Japan
e-mail: imamura.toshiyuki@riken.jp

3.2 Sparse Eigen-Super Computing Engine

Here, we consider complex moment-based eigensolvers and their high-performance software: z-Pares for solving the following generalized eigenvalue problem

$$A\mathbf{x}_i = \lambda_i B\mathbf{x}_i, \quad A, B \in \mathbb{C}^{n \times n}, \quad \mathbf{x}_i \in \mathbb{C}^n \setminus \{\mathbf{0}\}, \quad \lambda_i \in \Omega \subset \mathbb{C}, \quad (3.1)$$

with sparse matrices A and B , where $zB - A$ is non-singular in a boundary Γ of the target region Ω .

3.2.1 Complex Moment-Based Eigensolvers

3.2.1.1 Basic Concepts

As one of the powerful algorithms for solving (3.1), a complex moment-based eigensolver has been proposed by Sakurai and Sugiura in 2003 [37]. The basic concept is to introduce the rational function

$$r(z) := \tilde{\mathbf{v}}^H (zB - A)^{-1} B\mathbf{v}, \quad \mathbf{v}, \tilde{\mathbf{v}} \in \mathbb{C}^n,$$

whose poles are the eigenvalues of the generalized eigenvalue problem: $A\mathbf{x}_i = \lambda_i B\mathbf{x}_i$, and compute all poles located in Ω by solving Hankel generalized eigenvalue problem with complex moments

$$\mu_k := \frac{1}{2\pi i} \oint_{\Gamma} z^k r(z) dz$$

using the method proposed by Kravanja et al. [33]. Now, there are several improvements and variants including direct extensions of Sakurai and Sugiura's approach [20–22, 25, 27, 39] and the FEAST eigensolver developed by Polizzi [36] and its improvements [15, 32, 44, 49, 50].

Let $L, M \in \mathbb{N}$ be the input parameters and $V \in \mathbb{C}^{n \times L}$ be an input matrix. We define $S_k \in \mathbb{C}^{n \times L}$ ($k = 0, 1, \dots, M - 1$) as follows:

$$S_k := \frac{1}{2\pi i} \oint_{\Gamma} z^k (zB - A)^{-1} B V dz. \quad (3.2)$$

Complex moment-based eigensolvers are mathematically designed based on the properties of the matrices S_k . Then, practical algorithms are derived by approximating the contour integral (3.2) using the numerical integration rule:

$$\widehat{S}_k := \sum_{j=1}^N \omega_j z_j^k (z_j B - A)^{-1} B V dz, \quad (3.3)$$

where z_j is a quadrature point and ω_j is its corresponding weight.

The algorithms of complex moment-based eigensolvers comprise the following three steps:

Step 1. Solve N linear systems with L right-hand sides:

$$(z_j B - A)W_j = BV, \quad j = 1, 2, \dots, N. \quad (3.4)$$

Step 2. Construct complex moment matrices $\widehat{S}_k (k = 0, 1, \dots, M - 1)$ and others, from $W_j (j = 1, 2, \dots, N)$.

Step 3. Extract the target eigenpairs from the complex moment matrices.

The most time-consuming part of the complex moment-based eigensolvers is Step 1 that is solving the linear systems (3.4). For solving the linear systems, these eigensolvers have hierarchical parallelism.

Layer 1. Contour paths can be independently performed.

Layer 2. Each linear system can be solved in parallel.

Layer 3. The linear systems can be independently solved.

Because of the hierarchical structure of the algorithms, these methods are expected to achieve high scalability [13, 19, 27, 31, 32, 43, 48]. The algorithm on GridRPC systems is also considered [38, 40].

These methods have been implemented in the form of the high-performance parallel software: z-Pares [52] and FEAST [10], respectively.

3.2.1.2 Theoretical Aspect

The complex moment-based eigensolvers can be regarded as projection methods using a subspace constructed by the contour integral (3.3). The property of the subspace is well analyzed using the so-called filter function:

$$f(\lambda_i) := \sum_{j=1}^N \frac{\omega_j}{z_j - \lambda_i},$$

which approximates a band-path filter for the target region Ω . Using the filter function, error analyses of the complex moment-based eigensolvers were given in [15, 23, 24, 44]. An error resilience technique and an accuracy deterioration technique have also been discussed in [16, 28] using the results of the error analyses.

The relationship among typical complex moment-based eigensolvers was also analyzed in [24] focusing on the subspace. The block SS-RR method [21] and the FEAST eigensolver [44] are projection methods directory for solving the target eigenvalue problem (3.1), whereas the block SS-Hankel method [20], Beyn [3], and the block SS-Arnoldi methods [22] are projection methods for solving an implicitly constructed standard eigenvalue problem (see [24] for the details). A map of the relationship among the contour integral-based eigensolvers is presented in Fig. 3.1.

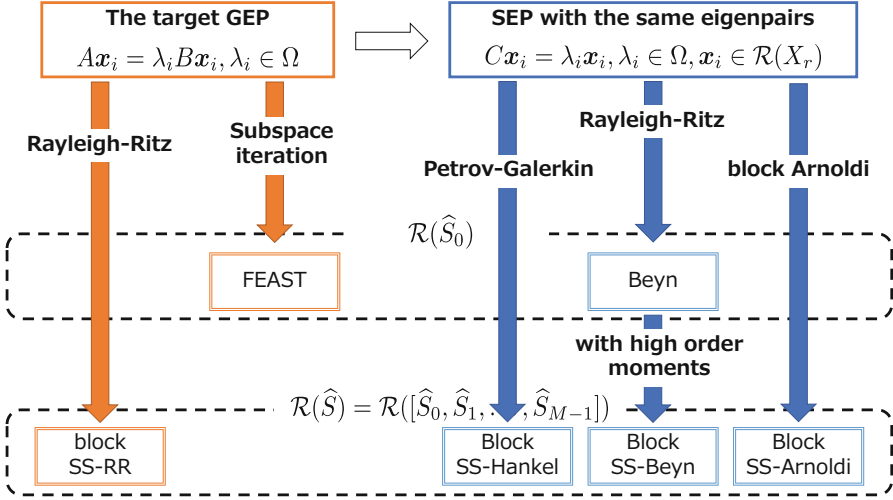


Fig. 3.1 A map of the relationship among the contour integral-based eigensolvers

3.2.1.3 Extension to Nonlinear Eigenvalue Problems

The complex moment-based eigensolvers were extended to solve nonlinear eigenvalue problems (NEPs):

$$T(\lambda_i)x_i = \mathbf{0}, \quad x_i \in \mathbb{C}^n \setminus \{\mathbf{0}\}, \quad \lambda_i \in \Omega \subset \mathbb{C},$$

where the matrix-valued function $T : \Omega \rightarrow \mathbb{C}^{n \times n}$ is holomorphic in some open domain Ω .

The block SS-Hankel [1, 2], block SS-RR [51], and block SS-CAA methods [26] are simple extensions of the GEP solvers. Improving technique of the numerical stability of the block SS-RR method for solving NEP was also studied in [7, 8].

As another type of complex moment-based nonlinear eigensolvers, Beyn proposed a method based on Keldysh's theorem and the singular value decomposition [3]. Also, van Barel and Kravanja proposed an improvement of the Beyn method using the canonical polyadic (CP) decomposition [46].

3.2.2 Distributed Parallel Sparse Eigensolver Package z-Pares

3.2.2.1 Introduction

z-Pares is a package for solving generalized eigenvalue problems (3.1). The symmetries and definitenesses of the matrices can be exploited suitably. z-Pares

computes eigenvalues inside a user-specified contour path and the corresponding eigenvectors. The most important feature of z-Pares is two-level message passing interface (MPI)-distributed parallelism.

3.2.2.2 Features

The main features of z-Pares are described below.

- Implemented in Fortran 90/95
- Solves standard eigenvalue problems $A\mathbf{x} = \lambda\mathbf{x}$ and generalized eigenvalue problems $A\mathbf{x} = \lambda B\mathbf{x}$
- Computes eigenvalues located in an interval or a circle and the corresponding (right) eigenvectors
- Both real and complex types are supported
- Single precision and double precision are supported
- Both sequential and distributed parallel MPI builds are available
- Two-level distributed parallelism can be employed by using a pair of MPI communicators
- Reverse communication mechanism is used to ensure the package accept any matrix data structure
- Interfaces for dense and sparse CSR format are available (only with one-level-distributed parallelism)

3.2.2.3 Dependences

z-Pares depends on following packages:

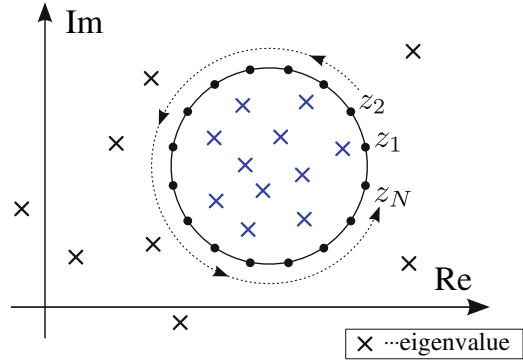
- BLAS/LAPACK
- Message passing interface (MPI-2 standard)
- MUMPS 4.10.0 (optional)

BLAS/LAPACK should be installed, and MPI is needed for the parallel version of z-Pares. MUMPS is required to use the sparse CSR interface.

3.2.2.4 Basic Concepts of z-Pares

Here we show the schematic illustration of a numerical contour integration (3.3) in Fig. 3.2. As described in Fig. 3.2, numerical quadrature with N quadrature points is used to approximate the contour integral. The basis of the subspace which is used for extracting eigenpairs is computed by solving linear systems with multiple right-hand sides. In this section, matrix V is called a source matrix, and its column vector is called a source vector.

Fig. 3.2 z-Pares computes eigenvalues located inside a contour path on the complex plane (Blue cross)



Because the linear systems can be solved independently, the computations can be embarrassingly parallelized. Additionally, each linear system can be solved in parallel.

3.2.2.5 Two-Level MPI Parallelism

Above the parallelism of quadrature points, there is independent parallelism if multiple contour paths are given. Here we define three levels of parallelism:

- **Top level:** Parallelism of computations on contour paths
- **Middle level:** Parallelism of computations on quadrature points
- **Bottom level:** Parallelism of computations for solving linear systems

z-Pares uses the middle- and bottom-level parallelism by employing a pair of MPI communicators. The MPI communicators that manage the middle level and the bottom level are called *the higher-level communicator* and *the lower-level communicator*, respectively. Because the top-level parallelism can be implemented completely without communications, we have not added the implementation of this level to the feature of z-Pares. Users should manage the top-level parallelism by themselves if necessary.

The above descriptions are shown in Fig. 3.3.

For the Rayleigh-Ritz procedure and the residual calculations, matrix-vector multiplications (mat-vec) of A and B must be done for multiple vectors. The higher-level communicator manages the parallelization in performing mat-vec for different vectors. The lower-level communicator manages the parallelization for one mat-vec.

3.2.2.6 `zparep_prm` Derived Type

The derived type `zparep_prm` plays a central role in the use of z-Pares. `zparep_prm` consists of components that represent several input and output

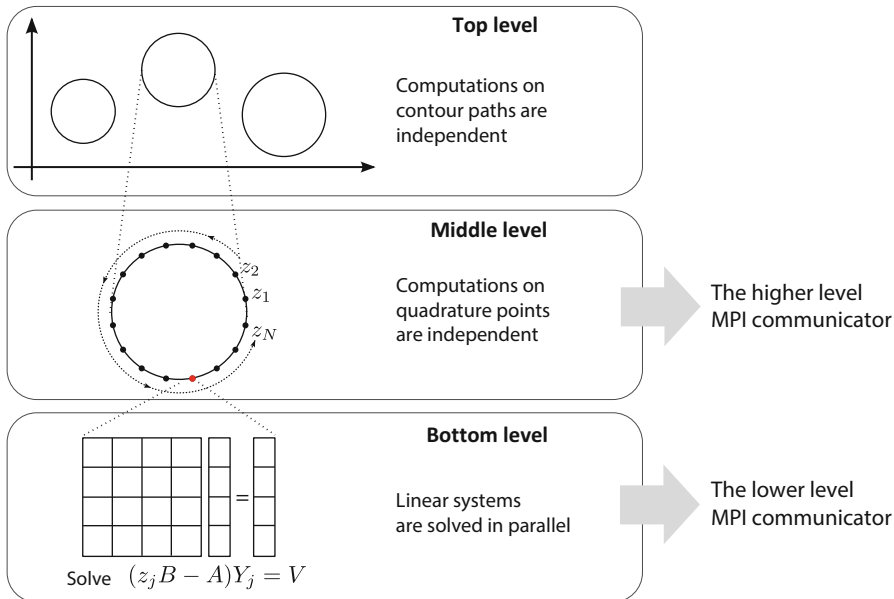


Fig. 3.3 Three levels of parallelism and two-level MPI communicator

parameters and inner variables. See z-Pares users' guide for more details. In the rest of this section, an entry of `zpares_prm` is referred to as `prm`.

3.2.2.7 Reverse Communication Interface

z-Pares basically delegates tasks of

- Solving linear systems with multiple right-hand sides $(z_j B - A)Y_j = BV$
- Performing matrix-vector multiplications of A and B

to the user, because an efficient algorithm and matrix data structure are seriously problem dependent.

z-Pares delegates the tasks by using the reverse communication interface (RCI) rather than the modern procedure pointer or an external subroutine.

When using RCI, the user code communicates with the z-Pares subroutine in the following manner:

1. Reverse communication flag `prm%itask` is initialized with `zpares_init` before entering the loop of 2.
2. The z-Pares subroutine is repeatedly called until `prm%itask == ZPARES_TASK_FINISH`
3. In the loop of 2., the tasks indicated by `prm%itask` are completed with the user's implementation

When using RCI, the user need not define global, COMMON, or module variables to share information (such as matrix data) with the subroutine given to the package, in contrast to manners using the procedure pointer or external subroutine. RCI is also used in eigensolver packages such as ARPACK and FEAST.

To briefly describe a user code using RCI, a skeleton code for solving complex non-Hermitian problem is given below.

Listing 3.1 Usage of reverse communication interface

```
do while ( prm%itask /= ZPARES_TASK_FINISH )
  call zpares_zrcigev &
    (prm, nrow_local, z, mwork, cwork, left, right, num.ev, eigval, X, res, info)

  select case (prm%itask)
  case (ZPARES_TASK_FACTO)

    ! Here, the user factorizes (z*B - A)
    ! At the next return from zpares_zrcigev,
    ! prm%itask==ZPARES_TASK_SOLVE with the same z is returned

  case (ZPARES_TASK_SOLVE)

    ! i = prm%ws; j = prm%ws+prm%nc-1
    ! Here, user solves (z*B - A) X = cwork(:,i:j)
    ! The solution X should be stored in cwork(:,i:j)

  case (ZPARES_TASK_MULT_A)

    ! iw = prm%ws; jw = prm%ws+prm%nc-1
    ! ix = prm%xs; jx = prm%xs+prm%nc-1
    ! Here, the user performs matrix-vector multiplications:
    ! mwork(:,iw:jw) = A*X(:,ix:jx)

  case (ZPARES_TASK_MULT_B)

    ! iw = prm%ws; jw = prm%ws+prm%nc-1
    ! ix = prm%xs; jx = prm%xs+prm%nc-1
    ! Here, the user performs matrix-vector multiplications:
    ! mwork(:,iw:jw) = B*X(:,ix:jx)

  end select
end do
```

ZPARES_TASK_FINISH, ZPARES_TASK_FACTO, ZPARES_TASK_SOLVE, ZPARES_TASK_MULT_A, and ZPARES_TASK_MULT_B are defined as module variables of the type integer, parameter of the zpares module. Tasks delegated to the user are indicated by these values.

Implementing a linear solver is a heavy task for users. To allow users to get started with z-Pares easily, we provide two interfaces for a specific matrix data structure:

- Dense interface using LAPACK
- Sparse CSR interface using MUMPS

3.2.2.8 Efficient Implementations for Specific Problems

In the above descriptions, we have used the subroutines for complex non-Hermitian problems. In z-Pares, efficient implementations are given for exploiting specific features of the problem. The following features are considered:

- Symmetry or hermiticity of matrices A and B
- Positive definiteness of B
- $B = I$ (standard eigenvalue problem)

We recommend the user to let z-Pares consider these features by setting appropriate parameters to obtain maximum efficiency.

A stochastic estimation method of eigenvalue distribution in a given domain is proposed in [12, 34]. This method is used to evaluate appropriate parameters. Some properties of the contour integral-type methods are considered to determine efficient parameters in [41, 42].

3.3 EigenExa: Development of a Dense Solver

3.3.1 Introduction

Eigenvalue calculation is a significant tool for scientific numerical simulation and engineering analysis. High-performance and highly reliable software must be available. As the size of problems to be solved and available computer resources become substantial, the practical eigenvalue solver is expected to change accordingly.

When this post-petascale CREST project was initiated, we reviewed the trend of the future hardware technology, microprocessor, accelerator unit, memory module, and interconnect network. We supposed that the following must be an essential requirement to build a next-generation, so-called exascale, supercomputer system:

- CPU socket
8CPUs+1000FPUs,
On-chip shared memory,
1.25TFLOP/s
- Compute node
8sockets,
64GB shared memory,
20TFLOP/s
- System
 10^5 nodes,
6.4PB memory,
2EFLOP/s

In a rough sketch of the design mentioned above, the system has a heterogeneous and hierarchical combination of CPU modules and memories interconnected. We also predicted that such a complication of hierarchical hardware results in a new hierarchical parallel programming style. In fact, two programming languages were critical issues at the beginning of the project: MPI for a representative tool of distributed parallelism and OpenMP for thread parallelism among computational cores in a shared memory fashion.

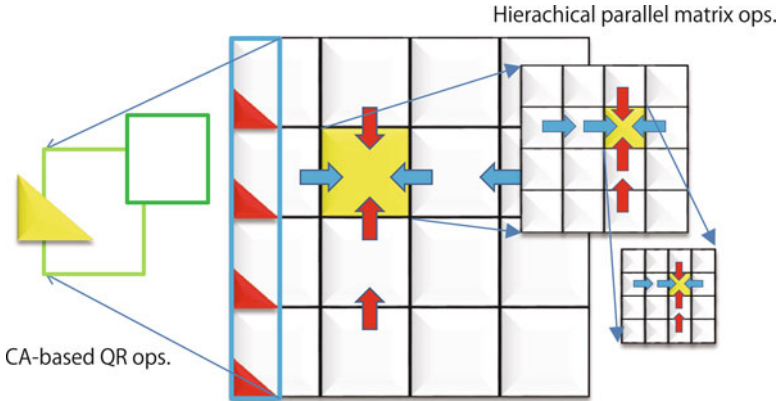


Fig. 3.4 Concept of hierarchical and multi-layered approach in dense linear algebra

As parallel numerical linear algebra software, not only eigenvalue calculation, the configuration of ScaLAPACK + LAPACK + BLAS has been known as de facto standard not changed significantly for nearly 20 years since the 1990s. In the above hierarchization, vertical (upper and lower) parallelism could be handled by a combination of the existing numerical linear algebra software, but it was quite hard to perform higher parallel control by combining flexible parallelism and parallel execution in the same horizontal layer.

What is more, in 2010, a pragmatic innovation was required from not only the software environment but also the parallel algorithm. For one thing, it was necessary to collaborate with a highly parallel/concurrent language to support parallel/concurrent processing over multiple parallel layers and software runtime. Naturally, the emergence of numerical algorithms having multilevel parallelism in every hierarchy was demanded. In fact, there is a need to respond to the emergence of times when hardware has parallelism ranging from hundreds to tens of thousands. In our implementation of algorithms crossing over multiple layers of a dense matrix representation, we decided to realize by the configuration method being aware of any hierarchy as shown in the figure. In particular, we decided to develop mainly block algorithms corresponding to multiple memory hierarchies found in (i) typical high-speed implementation of matrix-matrix products and (ii) local and global communication avoidance, for example, by the CAQR-type approach (Fig. 3.4).

Here, we would like to summarize a brief review of the solver development project, which was promoted during the 2010s. As mentioned previously, ScaLAPACK released in the 1990s keeps still in the position as the de facto standard in distributed parallel environments. Meanwhile, thread parallelism has also to be taken into consideration at the same time by the multicore processor that appeared around 2010. On the other hand, in the HPC community, the development of numerical libraries specialized for thread parallelism for many-core environment equipped with dozens to thousands of cores, the MAGMA project, and the PLASMA project, has been started.

Focused on the trend of numerical eigenvalue libraries, it had been known that xSYEVD of ScaLAPACK based on Cuppen's divide and conquer method had a significant advantage in the distributed parallel environment. Nowadays, a successor routine of xSYEVR, which adopts a different mathematical algorithm of MR3 (multiple relatively robust representations), is still naive implementation. However, the new algorithm has $O(N^2)$ complexity and a promising way (has another name of "the holy grail") to reduce the computational cost regarding flops. As still ScaLAPACK as old design in 1990s, the developer needs to improve thread parallelism. Since 2010, there have been several signs of progress in parallel eigenvalue solvers, such as ELPA by the German team, Elemental, DPLASM by the University of Tennessee, our EigenExa library, and QDWH-based library by KAUST. Each eigensolver library has a unique aspect in the numerical algorithm and parallel implementations. For example, the ELPA library insists significance of a so-called two-stage algorithm, which is a promising way to resolve the matters of narrow memory bandwidth and non-negligible network latency. Also, Elemental employs a brand-new parallel implementation of the MR3 algorithm.

In the rest of the section, we report the status summary of the EigenExa library, which was developed in the CREST project.

3.3.2 *Brief History of the Dense Solver Project*

Since 2010, the H4ES (=H⁴ES, high performance, high scalability, high portability, and high-reliability Eigen-Supercomputing engine) project conducted by Prof. Tetsuya Sakurai has been kicked off supported by one of the national grant CREST JST. We organized a mini-research group to devote to the development of a dense eigenvalue solver for a post-petascale supercomputer system at the University of Electro-Communications and later at RIKEN Advanced Institute for Computational Science (AICS) from 2012 to the present.

The activities initiated by T. Imamura had not been started since the H4ES project but another CREST project led by Dr. Masahiko Machida, Japan Atomic Energy Agency (2005–2010). The present dense solver project inherited the primary results from the Earth Simulator system, which was the world's largest vector supercomputer [47]. The library was optimized with a combination of a very conservative long-vector-oriented technique and a modern cache-oriented technique, so-called vectorization and efficient cache reuse, respectively. After the Earth Simulator age, we comprehensively scrapped and built up the vector code to multi-threading processing code [30], which was deployed on each system of T2K cluster. The T2K cluster was designed as a commodity supercomputer. Then, the code was ported to the K computer with the help of Fujitsu, and we named the eigensolver library **EigenExa**. The K computer was the first 10 petaFLOPS system housed at RIKEN AICS and is still keeping rank 10 in the top 500 benchmark (Nov. 2017). The K computer has a unique interconnect, namely, Tofu interconnect, and more than 80,000 SPARC64 VIIIfx processors consist of the heart of the system. We observed

a big impact of high-performance eigensolver on the RSDFT code, which won the Gordon Bell prize in SC2011 [17], even though the solver was very early version and not optimized so well. Currently, the EigenExa library version 2.4p1 is the latest release [9].

Since 2014, RIKEN AICS has started a national project to develop a flagship system, so-called the post-K system. For that, we continue to improve the EigenExa library toward emerging supercomputer systems, such as Oakforest-PACS, which is the rank 9 system in the top 500 benchmark (Nov. 2017) and is hosted at joint center between the University of Tokyo and University of Tsukuba. Since another aspect on the extreme computing implies not only capability computing but capacity computing, we recognized the necessity of the high-performance eigensolver with a broad variety of parallelism and parallel scaling. It is an entirely challenging work for applied mathematics, computer science, and engineering.

3.3.3 Our Approaches in Parallel Algorithm

As shown, we have been developing an eigenvalue library EigenExa, which consists of multiple eigensolvers, toward next-generation distributed memory parallel supercomputers [30]. For performance improvement of the solvers, it is critical to identify the significant performance bottleneck and remove it on the highly parallel environment [17]. We exploit several algorithmic and implementation techniques to remove bottleneck which comes from data communication among or through a great number of computing nodes.

Main components of EigenExa are two driver routines and one reducer routine from generalized eigenvalue problem (GEVP) to standard eigenvalue problem (SEVP). Two driver routines are:

- `eigen_s`: a conventional scheme, and
- `eigen_sx`: a novel one-stage scheme.

We exploited a novel one-stage scheme for the implementation of `eigen_sx`, and its outline is summarized as follows (see also Fig. 3.5).

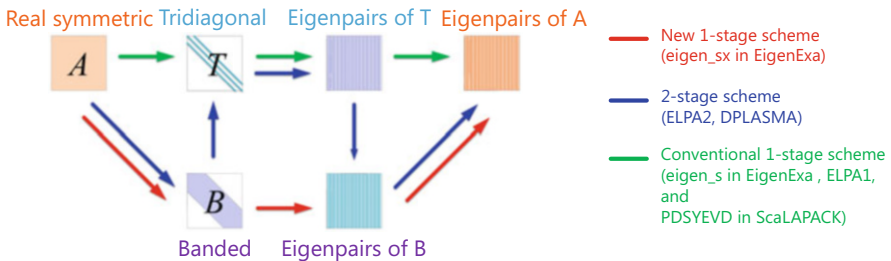


Fig. 3.5 Schematics of Numerical schemes for dense symmetric eigenvalue problem

1. Transform a matrix A to a pentadiagonal matrix P by similarity transformation with an orthogonal matrix V (forward transformation, which consists of multiple Householder transformation): $V^\top AV = P$,
2. Compute the eigenpairs of P by the divide-and-conquer (DC) algorithm for a banded matrix: $P = U\Lambda U^\top$ (where U is an orthogonal matrix and Λ is a diagonal matrix),
3. Compute the eigenvectors of A (back transformation): $Q = VU$.

3.3.3.1 Householder Band Reduction

The first and the third steps are implemented based on Bischof's band reduction algorithm, so-called successively band reduction (SBR) [4, 5]. The parallelization is performed in an MPI/OpenMP hybrid fashion. In `eigen_sx`, a pentadiagonal form is used as the intermediate matrix to reduce the internode communication cost, while the tridiagonal form is used in most of the high-performance eigensolvers. Core manipulations are based on block Householder transformation.

1. Compute a block reflector u corresponding to w and an associated lower triangular matrix C , somehow, such that they hold $(I - uCu^\top)w = ER$. Here, E is a unit matrix, and R is an upper triangular matrix.
2. Compute $v_0 := Au$, and $S := Cu^\top v_0 C^\top$.
3. Compute $v := (vC^\top - \frac{1}{2}uS)$.
4. Update $A := A - uv^\top - vu^\top$.

3.3.3.2 Divide and Conquer for a Banded Matrix

The routine for the second step is implemented and modified for the pentadiagonal matrix based on the routine PDSTEDC in ScaLAPACK [6, 45], which is for solving an eigenvalue problem of a tridiagonal matrix. The principle of the algorithm is "single perturbation of a diagonal matrix" defined as $M = D + \rho uu^\top$. For a banded matrix, we can summarize the algorithm as follow. As you can see, Step 2 can be done recursively.

1. Divide $P := P_1 \oplus P_2 + U\Sigma U^\top$, here Σ is a diagonal matrix, and K refers to the half value of the bandwidth of matrix P .
2. Compute eigenproblems P_1 and P_2 , somehow.
3. Transform $P_1 \oplus P_2 + U\Sigma U^\top \rightarrow D_1 \oplus D_2 + V\Sigma V^\top$ by similarity transformation.
4. Set $D = D_1 \oplus D_2$.
5. for $i=1, \dots, K$
6. Solve a single perturbation problem $F := D + \sigma_i v_i v_i^\top$.
7. Set $D := Q^\top F Q$. Here, Q is corresponding eigenvectors of matrix F .
8. Set $V_{i+1:K} := Q[v_{i+1}, \dots, v_K]$
9. Return the eigenvalues in the diagonal of D and the corresponding eigenvectors in matrix Q .

3.3.3.3 Back Transformation

The third step employs the compact WY representation to accelerate the computational performance as most of the modern solvers do. A core part of the block Householder transformation with the WY representation is as follows.

1. Construct a triangular matrix C corresponding to block reflector from $u = [u_1, u_2, \dots, u_b]$, such that $I - uCu^T = (I - u_bu_b^T) \cdots (I - u_2u_2^T)(I - u_1u_1^T)$.
2. Update $X := (I - uCu^T)X = X - uC(u^T X)$.

Since all the reflector vectors $u = [u_1, u_2, \dots]$ were already computed in the forward transformation, data redistribution (or broadcasting) of them has many variations. We introduced one of the communication hiding techniques (CH), the overlap of communication, and computation to reduce the communication overhead hiding behind computation.

3.3.4 Performance

In the project, we evaluated and analyzed the feasibility of the algorithm and parallel implementation. Also, performance of `eigen_s` and `eigen_sx` driver routines is investigated using the K computer [29] housed in RIKEN AICS (Project number hp120170 and ra000005). The benchmark presented in this article was done by using EigenExa version 2.5 Release Candidate (development code “c4”), which was developed in February 2018 under the support of KAKENHI grand-in-aid (15H02709).

3.3.4.1 Performance on the K Computer

Figure 3.6 shows a strong scaling benchmark results demonstrated on the K computer. We consider problems of dimension 10,000 to 130,000, and we selected a Frank matrix defined by $(A)_{ij} = \max(i, j)$ as a test matrix, which has eigenvalues represented analytically as $\lambda_k = 1/2(1 - \cos(\pi(2k + 1)/(2N + 1)))$. EigenExa performs on the K computer in a hybrid MPI/OpenMP parallel fashion 8threads/1process deployed on a node with varying the number of processes from 32 to 4096 processes. Our EigenExa libraries yielded excellent performance than the ELPA2 solver does. However, in case of a middle-sized problem, $N=10,000$, performance improvement stacked up when more than 200 processes were used because the problem size and the amount of required computational counts were not enough for such large number of processes. However, in the cases of more larger dimensional matrices, $N=50,000$ and 130,000, we observed the gradual but acceptable performance improvement up to 4096 processes.

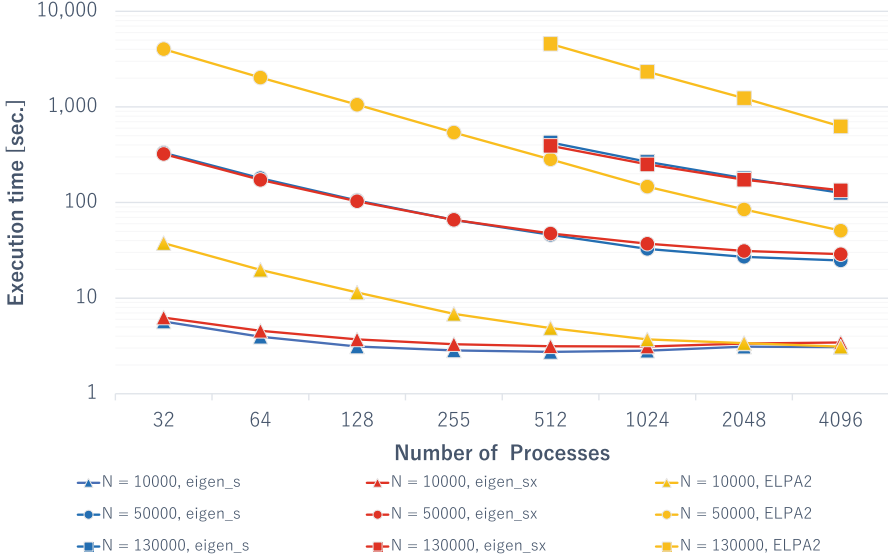


Fig. 3.6 Strong scaling benchmark on the K computer

3.3.4.2 Ultra-Scale Benchmark

The most outstanding result on the development of EigenExa was the success of a full diagonalization of a one million-dimensional matrix by using the whole system of the K computer. The actual log is shown in Fig. 3.7. This ultra-scale benchmark was done under the conditions:

1. EigenExa version 1.0 (`eigen_sx` driver routine).
2. The test matrix was generated as a random matrix symmetrized, $(A + A^T)/2$.
3. Fujitsu software environment was K-1.2.0-14.
4. Full node of the K computer, i.e., 82944 nodes, was occupied during the job.
5. MPI/OpenMP hybrid parallelism. 1MPI process/node, 8threads/1MPI process.
6. Job time stamp was “Wed Aug 14 23:16:05 JST 2013.”

Another experiment on Jan 16, 2014, revealed an accuracy of such a huge-sized eigenvalue problem. The relative residual was $\max_i \|Ax_i - \lambda_i x_i\|_1 / N \|A\|_1 = 5.99 \times 10^{-16}$, and the orthogonal error was $\|X^T X - I\|_F / N = 2.16 \times 10^{-16}$. These observations exhibit that the parallel algorithm adopted in the current implementation works feasibly when even matrix size and parallelism grow in super-scale. Furthermore, the algorithm and parallel implementation are trustable in the exascale era.

From the results through our current and past benchmark tests, we clarified the parallel performance improvement and the performance bottleneck of the solvers on the highly parallel environment. These achievements provide us with deep insight and perspectives of a high-performance numerical library toward the future supercomputer systems such as the post-K computer.

```

NUM.OF.PROCESS= 82944 ( 288 288 )
NUM.OF.THREADS= 8
calc (u,beta)      503.0970594882965
mat-vec (Au)      1007.285000801086  661845.1244051798
2update (A-uv-vu) 117.4089198112488  5678160.294281102
calc v            0.000000000000000
v=v-(UV+VU)u     328.3385872840881
UV post reduction 0.6406571865081787
COMM_STAT
  BCAST  ::  424.3022489547729
  REDUCE  ::  928.1299135684967
  REDIST  ::  0.000000000000000
  GATHER  ::  78.28400993347168
TRD-BLK 1000000 1968.435860157013  677356.7583893638  GFLOPS
TRD-BLK-INFO 1000000 48
before PDSTEDC 0.1448299884796143
PDSTEDC 905.2210271358490
MY-REDIST1 1.544256925582886
MY-REDIST2 14.75343394279480
RERE1 4.861211776733398E-02
COMM_STAT
  BCAST  ::  4.860305786132812E-02
  REDUCE  ::  2.155399322509766E-02
  REDIST  ::  0.000000000000000
  GATHER  ::  0.000000000000000
PDGEMM 532.6731402873993  5417097.565200453  GFLOPS
D&C 921.8044028282166  3130319.580211733  GFLOPS
TRBAK= 573.9026420116425  COMM= 533.7601048946381
      573.9026420116425  3484911.644577213  GFLOPS
      182.3303561210632  5484550.248648792  GFLOPS
      152.0370917320251  6577342.335399065  GFLOPS
      0.1022961139678955  7.379654884338379
COMM_STAT
  BCAST  ::  229.3666801452637
  REDUCE  ::  234.4477448463440
  REDIST  ::  0.000000000000000
  GATHER  ::  0.000000000000000
TRBAKWY 573.9029450416565
TRDBAK 1000000 573.9216639995575  3484796.141101135  GFLOPS
Total 3464.162075996399  1795203.448396145  GFLOPS
Matrix dimension = 1000000
Internally required memory = 480502032 [Byte]
Elapsed time = 3464.187163788010 [sec]

```

Fig. 3.7 Console output of the full-diagonalization benchmark of a one million-dimensional matrix

3.3.5 Related Sub-projects

To develop a dense eigenvalue solver, mathematical and computational innovations are required. Several topics were conducted in the project and interacted with other projects.

3.3.5.1 CholeskyQR2

Orthogonalization by QR factorization is one of the critical issues for the eigen decomposition or internal Householder transformation. Fukaya and Yamamoto et al. proposed the CholeskyQR2 algorithm, which factorizes a tall-skinny matrix in a QR representation, where matrix Q holds $Q^\top Q = I$ and R is an upper triangular matrix.

1. $R_0 := \text{Chol}(A^\top A)$, st. $A^\top A = R_0^\top R_0$.
2. $Q_0 := AR_0^{-1}$.
3. $R_1 := \text{Chol}(Q_0^\top Q_0)$.
4. $Q := Q_0 R_1^{-1}$, $R := R_1 R_0$.

They elucidated that CholeskyQR2, which is an algorithm performing CholeskyQR twice, gives excellent accuracy and computing speed in most practical cases. In their experiments using 16,384 nodes of the K computer, CholeskyQR2 outperformed TSQR nearly threefold in computing time for a $4,194,304 \times 64$ matrix [14].

3.3.5.2 Performance Modeling

It is inevitable to establish a methodology of the performance prediction model for emerging large-scale systems. In the development of EigenExa, we investigated two styles of performance modeling: (i) an empirical base-function approximation [11] and (ii) the LP method, which introduced suppress of overfitting with nonnegative constraint [35].

In the above empirical study, we reported the evaluation on a Fujitsu PRIME-HPC FX10, which was mainly focused on investigating the differences between the two driver routines. The obtained results were expected to be useful for not only for a future EigenExa library but other parallel dense matrix computations. In contrast to the empirical way, we examined that the LP method predicted more accurately than the LASSO method, which is often used in the sparse modeling field. The LP method exhibited the valid base functions of the EigenExa library systematically from the products of $\{N^3, N^2, N\}$ and $\{1, 1/\sqrt{p}, 1/p\}$, and then we obtained a model function of the collective communication represented by a simple linear combination of the base functions:

$$c_1 \frac{N^2}{\sqrt{p}} + c_2 N^3 + c_3 N.$$

Since theoretical discussions do not provide the term of N^3 , it suggests that a significant scale eigenvalue problem might tend to incur severe performance degradation due to (i) non-parallelized parts and (ii) increasing communication overhead. Since similar arguments held for another empirical approximation modeling, we recognized the significance to continue the topics in the future.

3.3.5.3 High-Precision Calculation

The higher precision calculations often demanded in practical engineering and quantum chemistry to obtain more accurate eigenvalues or identify the algebraic multiplicity of a cluster of eigenmodes. For that, it is essential to take account of Bailey's double-double arithmetic as a quadruple precision format from the viewpoint of accuracy and performance. We evaluated the performance of the high-performance quadruple precision eigensolver libraries QPEigenK on the K computer. The latest version of QPEigenK performs exhibiting excellent scalability. We observed that the elapsed time to solve an eigenproblem with $n = 10,000$ was 118 seconds on 16384 nodes of the K computer, whereas it was 31 times longer than the case of a double precision solver [18].

Currently, the standardization of IEEE754 half precision format has induced other arguments of computing precision in numerical libraries. Not only high precision but flexible or selectable precision may become significant in future computing with the help of new hardware such as an FPGA device. Also, we expect that the topic could be enhanced to the new research world of reproducible computing, which guarantees identical computing result on any circumstances, anywhere and anytime.

3.4 Conclusion

In this paper, we introduced massively parallel Eigen-Supercomputing Engines: z-Pares and EigenExa, for post-petascale systems. Our Eigen-Engines were based on newly designed algorithms that are suited to the hierarchical architecture in post-petascale systems and showed very good performance on petascale systems including K computer.

References

1. Asakura, J., Sakurai, T., Tadano, H., Ikegami, T., Kimura, K.: A numerical method for nonlinear eigenvalue problems using contour integrals. *JSIAM Lett.* **1**, 52–55 (2009)
2. Asakura, J., Sakurai, T., Tadano, H., Ikegami, T., Kimura, K.: A numerical method for polynomial eigenvalue problems using contour integral. *Jpn. J. Indust. Appl. Math.* **27**, 73–90 (2010)
3. Beyn, W.-J.: An integral method for solving nonlinear eigenvalue problems. *Linear Algebra Appl.* **436**, 3839–3863 (2012)
4. Bischof, C., et al.: A framework for symmetric band reduction. *ACM Trans. Math. Softw. (TOMS)* **26**, 581–601 (2000)
5. Bischof, C., et al.: Algorithm 807: the SBR toolbox – software for successive band reduction. *ACM Trans. Math. Softw. (TOMS)* **26**, 602–616 (2000)
6. Blackford, L.S., et al.: *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia (1997)

7. Chen, H., Imakura, A., Sakurai, T.: Improving backward stability of Sakurai-Sugiura method with balancing technique in polynomial eigenvalue problem. *Appl. Math.* **62**, 357–375 (2017)
8. Chen, H., Maeda, Y., Imakura, A., Sakurai, T., Tisseur, F.: Improving the numerical stability of the Sakurai-Sugiura method for quadratic eigenvalue problems. *JSIAM Lett.* **9**, 17–20 (2017)
9. EigenExa Homepage: <http://www.aics.riken.jp/labs/lpnctr/en/projects/eigenexa/>
10. FEAST Eigenvalue Solver: <http://www.ecs.umass.edu/~polizzi/feast/>
11. Fukaya, T., Imamura, T.: Performance evaluation of the EigenExa Eigensolver on Oakleaf-FX: Tridiagonalization Versus Pentadiagonalization. In: *Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) (PDSEC 2015)*, pp. 960–969 (2015)
12. Futamura, Y., Tadano, H., Sakurai, T.: Parallel stochastic estimation method of eigenvalue distribution. *JSIAM Lett.* **2**, 127–130 (2010)
13. Futamura, Y., Sakurai, T., Furuya, S., Iwata, J.-I.: Efficient algorithm for linear systems arising in solutions of eigenproblems and its application to electronic-structure calculations. In: *Proceedings of the 10th International Meeting on High-Performance Computing for Computational Science (VECPAR 2012)*, pp. 226–235 (2013)
14. Fukaya, T., Nakatsukasa, Y., Yanagisawa, Y., Yamamoto, Y.: CholeskyQR2: a simple and communication-avoiding algorithm for computing a tall-skinny QR factorization on a large-scale parallel system. In: *Proceedings of the 5th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA14)*, pp. 31–38 (2014)
15. Güttel, S., Polizzi, E., Tang, T., Viaud, G.: Zolotarev quadrature rules and load balancing for the FEAST eigensolver. *SIAM J. Sci. Comput.* **37**, A2100–A2122 (2015)
16. Hasegawa, T., Imakura, A., Sakurai, T.: Recovering from accuracy deterioration in the contour integral-based eigensolver. *JSIAM Lett.* **8**, 1–4 (2016)
17. Hasegawa, Y., et al.: First-principles calculations of electron states of a silicon nanowire with 100,000 atoms on the K computer. In: *Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, Article No. 1 (2011)
18. Hirota, Y., Yamada, S., Imamura, T., Sasa, N., Machida, M.: Performance of quadruple precision eigenvalue solver libraries QPEigenK & QPEigenG on the K computer. In: *Proceedings of the International Supercomputing Conference (ISC'16). HPC in Asia Poster Session* (2016)
19. Ide, T., Toda, K., Futamura, Y., Sakurai, T.: Highly parallel computation of eigenvalue analysis in vibration for automatic transmission using Sakurai-Sugiura method and K computer. *SAE Technical Paper*, 2016-01-1378 (2016)
20. Ikegami, T., Sakurai, T., Nagashima, U.: A filter diagonalization for generalized eigenvalue problems based on the Sakurai-Sugiura projection method. *J. Comput. Appl. Math.* **233**, 1927–1936 (2010)
21. Ikegami, T., Sakurai, T.: Contour integral eigensolver for non-Hermitian systems: a Rayleigh-Ritz-type approach. *Taiwan. J. Math.* **14**, 825–837 (2010)
22. Imakura, A., Du, L., Sakurai, T.: A block Arnoldi-type contour integral spectral projection method for solving generalized eigenvalue problems. *Appl. Math. Lett.* **32**, 22–27 (2014)
23. Imakura, A., Du, L., Sakurai, T.: Error bounds of Rayleigh–Ritz type contour integral-based eigensolver for solving generalized eigenvalue problems. *Numer. Algorithms* **71**, 103–120 (2016)
24. Imakura, A., Du, L., Sakurai, T.: Relationships among contour integral-based methods for solving generalized eigenvalue problems. *Jpn. J. Ind. Appl. Math.* **33**, 721–750 (2016)
25. Imakura, A., Sakurai, T.: Block Krylov-type complex moment-based eigensolvers for solving generalized eigenvalue problems. *Numer. Algorithms* **75**, 413–433 (2017)
26. Imakura, A., Sakurai, T.: Block SS–CAA: a complex moment-based parallel nonlinear eigensolver using the block communication-avoiding Arnoldi procedure. *Parallel Comput.* **74**, 34–48 (2018)
27. Imakura, A., Futamura, Y., Sakurai, T.: Structure-preserving block SS–Hankel method for solving Hermitian generalized eigenvalue problems. In *Proceedings of 12th International Conference on Parallel Processing and Applied Mathematics (PPAM2017)* (2017, accepted)

28. Imakura, A., Futamura, Y., Sakurai, T.: Structure-preserving technique in the block SS–Hankel method for solving Hermitian generalized eigenvalue problems. In: Wyrzykowski, R., Dongarra, J., Deelman, E., Karczewski, K. (eds.) *Parallel Processing and Applied Mathematics. PPAM 2017. Lecture Notes in Computer Science*, vol. 10777, pp. 600–611. Springer, Cham (2017)
29. Imamura, T., et al.: Current status of EigenExa, high-performance parallel dense eigensolver. In: EPASA2018 (2018)
30. Imamura, T., et al.: Development of a high performance eigensolver on the Peta-Scale next generation supercomputer system. *Prog. Nucl. Sci. Technol.* **2**, 643–650 (2011)
31. Iwase, S., Futamura, Y., Imakura, A., Sakurai, T., Ono, T.: Efficient and Scalable Calculation of Complex Band Structure using Sakurai-Sugiura Method, In SC’17 proceeding of the International Conference for High Performance Computing, Networking, Storage and Analysis, 17, 2017 (accepted).
32. Kestyn, J., Kalantzis, V., Polizzi, E., Saad, Y.: PFEAST: a high performance sparse eigenvalue solver using distributed-memory linear solvers, In SC’16 proceeding of the International Conference for High Performance Computing, Networking, Storage and Analysis, vol. 16 (2016)
33. Kravanja, P., Sakurai, T., van Barel, M.: On locating clusters of zeros of analytic functions. *BIT* **39**, 646–682 (1999)
34. Maeda, Y., Futamura, Y., Sakurai, T.: Stochastic estimation method of eigenvalue density for nonlinear eigenvalue problem on the complex plane. *JSIAM Lett.* **3**, 61–64 (2011)
35. Orii, S., Imamura, T., Yamamoto, Y.: Performance Prediction of Large-Scale Parallel Computing by Regression Model with Non-Negative Model Parameters, *IPJS SIG Technical Reports (High Performance Computing)*, Vol. 2016-HPC-155, No. 9, pp. 1–9 (2016). (in Japanese)
36. Polizzi, E.: A density matrix-based algorithm for solving eigenvalue problems, *Phys. Rev. B* **79**, 115112 (2009)
37. Sakurai, T., Sugiura, H.: A projection method for generalized eigenvalue problems using numerical integration. *J. Comput. Appl. Math.* **159**, 119–128 (2003)
38. Sakurai, T., Hayakawa, K., Sato, M., Takahashi, D.: A parallel method for large sparse generalized eigenvalue problems by OmniRPC in a grid environment. *Lect. Notes Comput. Sci.* **3732**, 1151–1158 (2005)
39. Sakurai, T., Tadano, H.: CIRR: a Rayleigh-Ritz type method with counter integral for generalized eigenvalue problems. *Hokkaido Math. J.* **36**, 745–757 (2007)
40. Sakurai, T., Kodaki, Y., Tadano, H., Takahashi, D., Sato, M., Nagashima, U.: A parallel method for large sparse generalized eigenvalue problems using a grid RPC system. *Fut. Gen. Comput. Syst. Appl. Distrib. Grid Comput.* **24**, 613–619 (2008)
41. Sakurai, T., Asakura, J., Tadano, H., Ikegami, T.: Error analysis for a matrix pencil of Hankel matrices with perturbed complex moments. *JSIAM Lett.* **1**, 76–79 (2009)
42. Sakurai, T., Futamura, Y., Tadano, H.: Efficient parameter estimation and implementation of a contour integral-based eigensolver. *J. Algorithms Comput. Tech.* **7**, 249–269 (2013)
43. Shimizu, N., Utsuno, Y., Futamura, Y., Sakurai, T.: Stochastic estimation of nuclear level density in the nuclear shell model: an application to parity-dependent level density in ^{58}Ni . *Phys. Lett. B* **753**, 13–17 (2016)
44. Tang, P.T.P., Polizzi, E.: FEAST as a subspace iteration eigensolver accelerated by approximate spectral projection. *SIAM J. Matrix Anal. Appl.* **35**, 354–390 (2014)
45. Tisseur, F., et al.: A Parallel Divide and Conquer Algorithm for the Symmetric Eigenvalue Problem on Distributed Memory Architectures. *SIAM J. Sci. Comput.* **20**, 2223–2236 (1999)
46. van Barel, M., Kravanja, P.: Nonlinear eigenvalue problems and contour integrals. *J. Comput. Appl. Math.* **292**, 526–540 (2016)
47. Yamada, S., et al.: High-Performance Computing for Exact Numerical Approaches to Quantum Many-Body Problems on the Earth Simulator (SC06) (2006)
48. Yamazaki, I., Ikegami, T., Tadano, H., Sakurai, T.: Performance comparison of parallel eigensolvers based on a contour integral method and a Lanczos method. *Parallel Comput.* **39**, 280–290 (2013)

49. Yin, G.: A randomized FEAST algorithm for generalized eigenvalue problems, arXiv:1612.03300 [math.NA] (2016)
50. Yin, G., Chan, R.H., Yeung, M.-C.: A FEAST algorithm for generalized non-Hermitian eigenvalue problems. *Numer. Linear Algebra Appl.* **24**, e2092 (2017)
51. Yokota, S., Sakurai, T.: A projection method for nonlinear eigenvalue problems using contour integrals. *JSIAM Lett.* **5**, 41–44 (2013)
52. z-Pares: Parallel Eigenvalue Solver. <http://zpare.cs.tsukuba.ac.jp/>

Chapter 4

System Software for Many-Core and Multi-core Architecture



Atsushi Hori, Yuichi Tsujita, Akio Shimada, Kazumi Yoshinaga, Namiki Mitaro, Go Fukazawa, Mikiko Sato, George Bosilca, Aurélien Bouteiller, and Thomas Herault

Abstract In this project, the software technologies for the post-peta scale computing were explored. More specifically, OS technologies for heterogeneous architectures, lightweight thread, scalable I/O, and fault mitigation were investigated. As for the OS technologies, a new parallel execution model, *Partitioned Virtual Address Space (PVAS)*, for the many-core CPU was proposed. For the heterogeneous architectures, where multi-core CPU and many-core CPU are connected with an I/O bus, an extension of PVAS, *Multiple-PVAS*, to have a unified virtual address

A. Hori (✉)

RIKEN, R-CCS, Minato-ku, Tokyo, Japan

e-mail: ahori@riken.jp

Y. Tsujita

RIKEN, R-CCS, Kobe, Hyogo, Japan

e-mail: yuichi.tsujita@riken.jp

A. Shimada

Research and Development Group, Hitachi, Ltd., Yokohama, Kanagawa, Japan

e-mail: akio.shimada.ht@hitachi.com

K. Yoshinaga

eF-4 Co., Ltd., Meguro-ku, Tokyo, Japan

e-mail: k.yoshinaga@ef-4.co.jp

N. Mitaro

Tokyo University of Agriculture and Technology, Koganei-shi, Tokyo, Japan

e-mail: namiki@cc.tuat.ac.jp

M. Sato

Department of Embedded Technology, School of Information and Telecommunication Engineering, Tokai University, Minato-ku, Tokyo, Japan

e-mail: mikiko.sato@tokai.ac.jp

G. Fukazawa

Yamaha Corp., Naka-ku, Hamamatsu, Japan

e-mail: go.fukazawa@fzawa.net

G. Bosilca · A. Bouteiller · T. Herault

Innovative Computing Laboratory, University of Tennessee, Knoxville, TN, USA

e-mail: bosilca@icl.utk.edu; bouteill@icl.utk.edu; herault@icl.utk.edu

© Springer Nature Singapore Pte Ltd. 2019

M. Sato (ed.), *Advanced Software Technologies for Post-Peta Scale Computing*,

https://doi.org/10.1007/978-981-13-1924-2_4

space of multi-core and many-core CPUs was proposed. The proposed PVAS was also enhanced to have multiple processes where process context switch can take place at the user level (named *User-Level Process: ULP*). As for the scalable I/O, *EARTH*, optimization techniques for MPI collective I/O, was proposed. Lastly, for the fault mitigation, *User Level Fault Mitigation, ULFM* was improved to have faster agreement process, and *sliding methods* to substitute failed nodes with spare nodes was proposed. The funding of this project was ended in 2016; however, many proposed technologies are still being propelled.

4.1 Overview and Background

The goal of this project was to develop OS technologies needed for post-peta scale machines in the future under the assumption of heterogeneous architecture would have dominated. The goal of this project was to develop key software technologies required for the post-peta scale computers. There were four research topics in this project:

- OS technologies for heterogeneous architectures,
- Lightweight thread,
- Scalable I/O,
- Fault mitigation.

This 5-year project was started from April 2011 and ended March 2016. When this project was started, the first commercial many-core CPU, Intel Knights Corner (known as KNC) [12], became available, and many people believed it was the dawn of many-core era. KNC was designed as a coprocessor and it needed another many-core CPU to be attached with. From the view point of OS research, this heterogeneous CPU architecture was new because an OS ran on each CPU, unlike the other coprocessor architectures such as GP-GPU or vector processing unit (e.g., ClearSpeed [9]) where no (explicit) OS runs on those coprocessors. Our focus was to provide a single system image for such heterogeneous architecture.

The following research topics in this project were higher-level than the above OS research. The performance improvement of applications having dynamic workload characteristics is a big concern for the coming post-scale era. The oversubscribed lightweight multithread is thought to be the answer for this kind of applications. We proposed a novel technique for this topic.

I/O performance has been a headache in the long history of HPC. We focused on the collective I/O in MPI and proposed several optimization techniques to improve I/O performance.

Fault mitigation has been a big concern as supercomputers are scaling up. In this project, we proposed two techniques; one is a parallel execution environment in which a process failure can be mitigated, and another is the technique how failed nodes must be replaced with spare nodes.

4.2 OS Technologies for Heterogeneous Architecture

The widely used in-node parallel execution models are multi-process model (e.g., MPI) and multi-thread model (e.g., OpenMP). The multi-process model has the problem when to communicate with the others because each process has its own virtual address space and shared memory is often used for inter-process communication. The widely used shared memory technique cannot avoid the 2-copy to exchange data between processes. On the other hand, the multi-thread model has the non-negligible overhead of mutual exclusion on shared variables. These overheads become more striking when the number of cores increases.

To have the best of the two worlds, multi-process and multi-thread, the third execution model had been proposed and implemented. SMARTMAP is to pack processes into one virtual address space, but SMARTMAP only runs on Kitten OS and relies on x86 architecture [4]. MPC allows threads to have privatized variables, but MPC heavily depends on a language processing system, and it requires special language processing systems [11]. We decided to develop our own system to implement the third execution model. This is called *Partitioned Virtual Address Space (PVAS)* implemented by the patched Linux kernel. Its details are explained in the next subsection.

PVAS provides the efficient parallel execution mode on many-core architectures. However, our final target architecture is heterogeneous architecture where many-core CPU and multi-core CPU are connected by an I/O bus. Each CPU has its own physical memory. An OS runs independently on each CPU. On KNC, Intel provides the SCIF to communicate between the KNC process and host process. Unfortunately, its communication performance was not so good (shown in Sect. 4.2.2). Our proposed solution to this is the extension of PVAS to pack processes running on KNC and processes running on the host processor into one virtual address space so that every process can access the data of the other processes regardless which process is running on which processor [15, 22]. This scheme was called *Multiple-PVAS (MPVAS)*, and more efficient communication than Intel provided SCIF could be achieved.

4.2.1 Partitioned Virtual Address Space

Basically, a process has an isolated virtual address space, and it cannot access the data in the other process. Shared memory technique is widely used to implement inter-process communication in HPC. Shared memory is a special memory region in which physical memory region is shared between processes and the data in the shared memory region can be accessed from the processes sharing the memory region. However, during communication between two processes, sender process writes a data into the shared memory segment and receiver process reads the data in

the shared memory segment. Thus, two memory copies take place, and this memory copy overhead hinders the parallel execution.

Each process has its own virtual address space so that the process is guaranteed not to be interfered by the other processes. Let's assume that there are two processes connected by a pipe. If a sender process dies for some reason, then the receiver process is killed by the SIGPIPE signal. Even if the SIGPIPE signal is ignored, then the receiver process might be blocked forever for receiving data, since there is no process to send data to the receiver process. Thus, communicating processes can be said to share the same fate. The protection mechanism for processes in modern OS incurs the overhead of the inter-process communication. In the coming many-core era, the intra-node communication will be more frequent. Therefore the development of efficient intra-node communication is important.

If all communication processes share the same virtual address space, then the data owned by a process can be accessed by the other processes without incurring the overhead of 2-copy. This is the idea of PVAS [17–21]. There is no protection between the processes sharing the same virtual address space. However, as mentioned above, the communicating processes share the same fate, and the protection mechanism has no meaning (Fig. 4.1).

In PVAS, the virtual address space is firstly partitioned, and process occupies one of the partitions. To implement PVAS, the Linux OS kernel is patched. Figure 4.2 shows the Open MPI intra-node communication performance comparisons among shared memory, denoted as “SM BTL,” KNEM (a kernel assisted 1-copy messaging), denoted as “SM-KNEM BTL,” XPMEM (allows process to expose a memory region to be accessed by the other process), denoted as “Vader BTL,” and PVAS, denoted as “PVAS BTL.” This evaluation was carried out on KNC (Intel Xeon Phi, 5110P 1.053 GHz). The left graph shows the PingPong latency measured by using IMB benchmark, and the right graph shows the system-wide memory usage of IMB AlltoAll communication. As shown in those graphs, the intra-node communication using PVAS exhibits the lowest latency and the least memory usage.

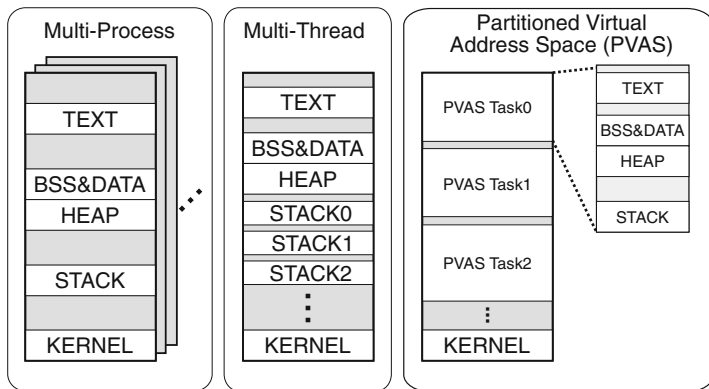


Fig. 4.1 Address maps of multi-process, multi-thread and PVAS

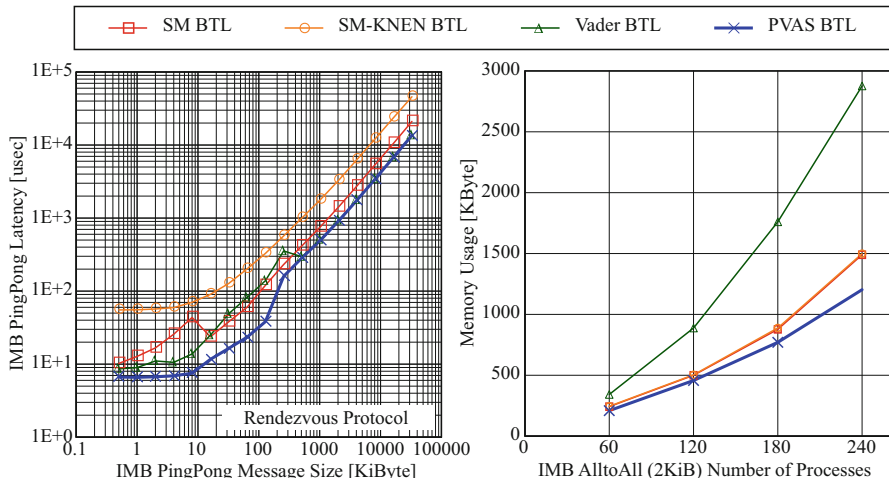


Fig. 4.2 IMB PingPong latency (left) and memory consumption (right)

In the MPI point-to-point communication, MPI datatypes which is an expression of non-contiguous data layout can be specified in the send function and the receive function. Since datatype information is local, the sender process has no knowledge on the datatype on receiver process. In the current MPI implementations, if a sender wants to send a non-contiguous data to a receiver, firstly the sender packs the non-contiguous data and then send the packed data to the receiver. And on the receiver process, the packed data is unpacked. Thus, costly message 2-copy for packing and unpacking must take place.

With PVAS, however, the sender process can take a look at the datatype information on the receiver process. This is because the sender process and the receiver process share the same virtual address space in PVAS. And non-contiguous messages can be sent with 1-copy according to the datatypes of sender and receiver. Shimada [17] reported 20% improvement with the `ft2d_datatype` benchmark [16].

4.2.2 Heterogeneous Extension of PVAS

On KNC, there are two sets of CPU and memory, and they are connected by the PCIe bus. Independent OS runs on each CPU. The multi-core CPU has a fewer number of fast cores and the many-core CPU has a large number of slow cores. Applications having large parallelism can exploit the power of many-core. When an application running on many-core CPU tries to send a message using MPI, for example, then MPI’s complex protocol which is very hard to parallelize must be handled by the

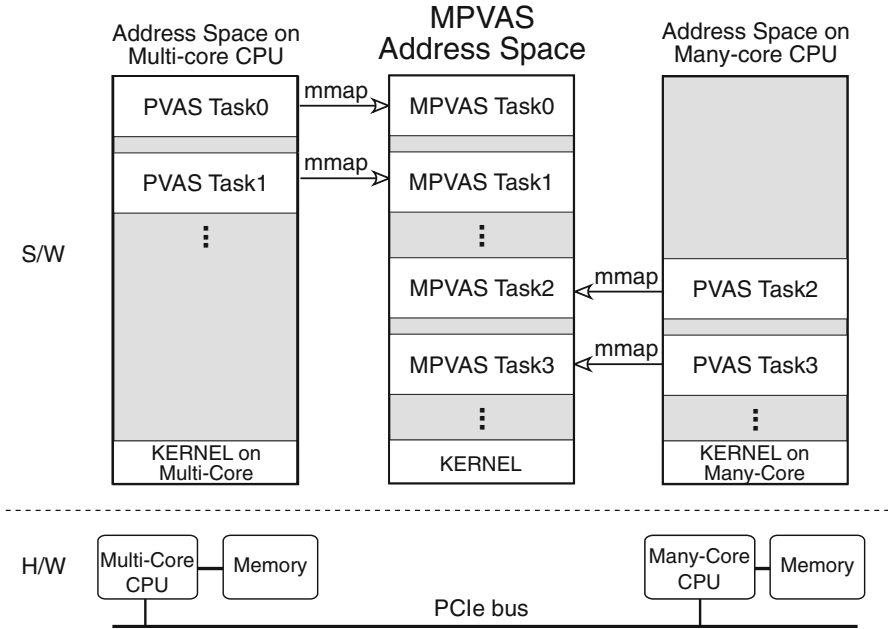


Fig. 4.3 Multiple PVAS

slow many-core CPU. This can add extra latency when compared with the case of using multi-core CPU.

The idea here is to delegate the heavy protocol handling from many-core CPU to multi-core CPU so that the complex MPI protocol can be handled by the faster multi-core CPU. The key technology here is the low-latency delegation mechanism from many-core CPU to multi-core CPU and vice versa. Generally speaking, this idea can be applied not only MPI but many situations.

Multiple-PVAS (MPVAS) was designed for such purpose. The processes running on many-core CPU and the processes running on multi-core CPU are mapped into one virtual address space (Fig. 4.3) [5]. In the MPVAS-aware MPI, based on NMVAPICH, to send a message from a process on many-core CPU, the send request is passed to the dedicated process running on the multi-core CPU, and then the request is processed on the multi-core CPU. Since the data to be sent can be accessed by the multi-core process, there is no need of copying it.

Figure 4.4 shows the latency of RPC between many-core (E5-2650v2, 2.60 GHz) and multi-core (Xeon Phi 7120P). X-axis shows the payload length and Y-axis shows the latency. As shown in this figure, the latency of using the offloading mechanism provided by Intel is several order of magnitude slower. Fukazawa

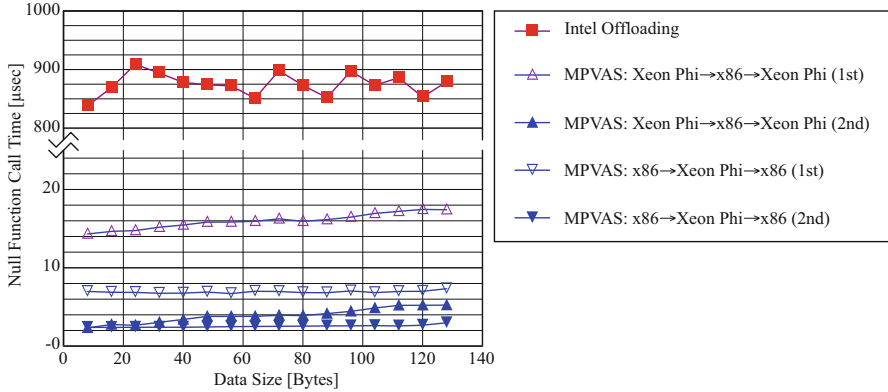


Fig. 4.4 Inter-CPU offloading latency (KNC)

reported that the MPI latency with this delegation mechanism is almost halved compared with the one using only many-core CPU [5, 14].

4.3 Lightweight Thread

So far, each PVAS process has an associated OS kernel thread to run in parallel. There can be PVAS processes having OS kernel thread and there can be PVAS processes having no OS kernel thread. Assume PVAS process *A* has no OS kernel thread and PVAS process *B* has OS kernel thread. The OS kernel thread of *B* can switch to *A* at user level by calling the `swapcontext()` or `setcontext()` Glibc function. This mechanism is very similar to the one of user-level thread. However, each PAVS process may have different program and do have privatized variables, unlike (user-level) thread. We named this *User-Level Process (ULP)* because PVAS [21].

Figure 4.5 shows the context switching times (Y-axis) comparing Linux processes, Pthreads, MassiveThreads [10] (a user-level thread implementation), and ULP over the number of processes or threads (X-axis). The context switching times of ULP is very close to the ones of MassiveThreads.

If some MPI processes in a node are implemented by using ULP and they are oversubscribed by having more number of MPI processes than the number of CPU cores, then those MPI processes can be switched from one to the other in a fast way. This would result in leveling the load imbalance between CPU cores. Thus, the load imbalance of dynamic workload can run more efficiently.

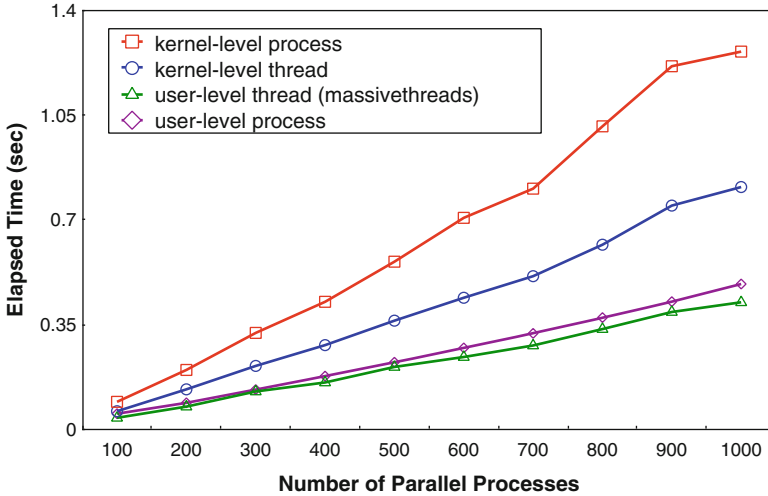


Fig. 4.5 Comparison of context switching time

4.4 Scalable I/O

The current optimization technique of collective MPI-IO [23] is based on two-phase I/O [13], where the scattered I/O requests on compute nodes are sorted according to the parallel file offset, and then actual I/O requests are sent to I/O servers. This technique is very effective because the smaller I/O requests on compute nodes are gathered on aggregator processes so that larger I/O requests can be sent to I/O servers.

In general, not limited to HPC, it is the file servers' nature that a number of simultaneous I/O requests can degrade its performance. To prevent this, I/O throttling technique was proposed. In the two-phase I/O, a number of I/O requests are also sent to I/O servers. Our first idea is to throttle the number of request to the IO servers to get higher performance. There is another bottleneck in two-phase I/O. Heavy all-to-all-like communication between user processes and aggregator processes to sort and merge the user's I/O requests takes place. If the I/O requests to the server are throttled, then this all-to-all-like communication can also be throttled. Consequently, the communication bottleneck can be relaxed. Thus, the flow of the I/O requests from user process to I/O servers can be done in a way of step-wise pipeline to level the loads of the communication network and I/O servers.

This optimization technique was named *EARTH* which includes the above step-wise communication, throttling to the I/O servers, striping-aware data blocking and distribution, and optimized allocation of the aggregator processes to decrease the message congestion in communication network [24, 25]. Figure 4.6 shows the write bandwidth on the K computer [26] comparing the original MPI-IO and *EARTH* optimized MPI-IO. In the *EARTH* cases, the number of requests at each step of the

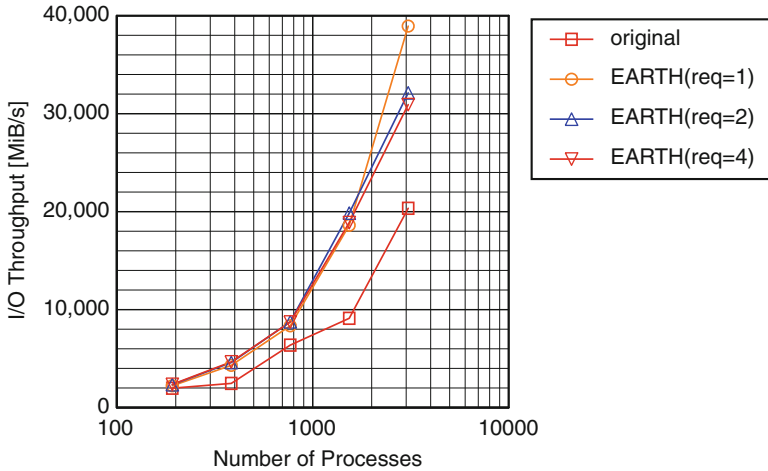


Fig. 4.6 HP-IO write performance (the K computer)

step-wise pipeline is also varied (denoted as “req = N ”). As shown in this figure, EARTH performs more better when the number of processes increases.

4.5 Fault Mitigation

Checkpointing is a well-known technique to recover from a failure. However, its cost can be very high, and the fault mitigation becomes the hot topic in the fault-tolerant research. User Level Fault Mitigation (ULFM) which is a user-level library for a parallel program to survive from a process failure is gathering the attentions as a practical technique of fault mitigation. In this project, a technique to improve the performance of ULFM was developed. The implementation of ULFM is not easy. The signal of a process failure must be propagated to the other processes to reach a consensus on the failure among the healthy processes. Another process failure may happen during the propagation and consensus protocols. ULFM must survive from any process failure that can happen at any time. In the next subsection, an optimization technique for ULFM to be more efficient is explained.

Since ULFM provides the way where user program can handle process failure events. It is up to the user program how to handle such situation. Having spare nodes to replace failed node is considered to be effective; however, study on the spare nodes is not active so far. We also focused on this issue and propose how to allocate spare nodes and how failed nodes should be replace by the spare nodes.

4.5.1 *User-Level Fault Mitigation (ULFM)*

4.5.1.1 Conceptual Design

The main goal of the *User Level Fault Mitigation ULFM* [2, 3, 6] approach remained unchanged over the period of this project: define a minimal set of features to notify application processes about failures, and allow the user to interrupt the normal behavior of his application and to restore communication capability after process failures. Exchanges with developers of MPI applications who envision to introduce resilience in their codes, or who have tried to do so using the ongoing ULFM specification, as well as fruitful discussion in the MPI Forum, mainly inside the Fault Tolerance Working Group and in plenary sessions, have highlighted few opportunities for improvements in the ULFM specification. Ambiguities in the proposed text have been removed and replaced by a clearer description of the required behavior. The RMA chapter has been entirely rewritten, to account for the changes in the RMA chapter introduced in the version 3 of the MPI standard. The `MPI_Comm_shrink()` and `MPI_Comm_agree()` routines, which are part of the Fault Tolerance extensions, have been simplified, allowing for a more diverse usage. Non-blocking versions of some of these constructs have been also added, to allow for more flexible recovery strategies.

4.5.1.2 Implementation

In same time as improving the concepts design, we wanted to provide a decent implementation that can be used by interested parties for their own research and developments. The first part of the award period was dedicated to providing correct, but non necessary efficient and scalable, support for all concepts. In parallel we advanced on the theoretical side by developing all the necessary tools and algorithms to design and implement more scalable versions of the needed concepts.

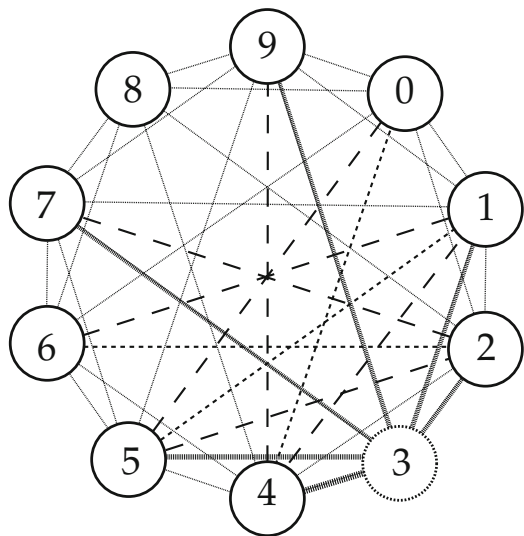
More specifically, the last year of the project has been almost entirely focused on advancing on the performance side of the two major operations proposed for the handling of the faults (revocation and agreement) and on promulgating the extension to the MPI standard to the user communities. In same time, a significant effort has been done to improve the code quality of the available implementation, in order to provide a stable, efficient, and portable implementation to the user communities.

Similarly to past years, exchanges with developers of MPI applications who envision to introduce resilience in their codes, or who have tried to do so using the ongoing ULFM specification, as well as fruitful discussion in the MPI Forum, inside the Fault Tolerance Working Group and in plenary sessions, have highlighted a few places for improvements in the ULFM specification. The main goal of the ULFM approach remains unchanged: define a minimal set of features to notify application processes about failures, and allow the user to interrupt the normal behavior of his application and to restore communication capability after process

failures. Ambiguities in the proposed text have been removed and replaced by a clearer description of the required behavior. The RMA chapter has been entirely rewritten, to account for the changes in the RMA chapter introduced in the version 3 of the MPI standard. The `MPI_Comm_shrink()` and `MPI_Comm_agree()` routines, which are part of the Fault Tolerance extensions, have been simplified, allowing for a more diverse usage.

On the implementation front, the `MPI_Comm_revoke()` and `MPI_Comm_agree()` operations have been the focus of our efforts for this year. These two routines are critical to the efficiency of a resilient code and are by nature costly because they need to be fault-tolerant themselves. `MPI_Comm_revoke()` is provided to the user to notify (asynchronously) all processes belonging to a communicator that an exception happened and to stop the normal execution flow (e.g., to trigger a collective repair of the communicator). At the conceptual level, it implements a form of reliable broadcast: if one process receives such revoke notification, all processes of the communicator must receive one. Because failures happening during the revoke operation can introduce messages loss, the protocol that implements the revoke operation must be fault-tolerant and ensure that if one notification is delivered, all notifications are delivered. The first implementation of the revocation was done at the Runtime Environment (RTE) level, over the Out-Of-Band (OOB) network. This implementation was not fault tolerant and required a large number of messages ($O(N^2)$). We redesigned the algorithms integrating research in resilient graph topologies and successfully improved not only the resilience and performance of the revocation algorithm but also the impact on the network infrastructure and the number of messages. The final algorithm that we will be delivered with ULFM is based on the Binomial Graph (BMG) topology, a specialize form of circulant graph topologies [1] (Fig. 4.7). This graph minimizes

Fig. 4.7 Binomial Graph (BMG) topology

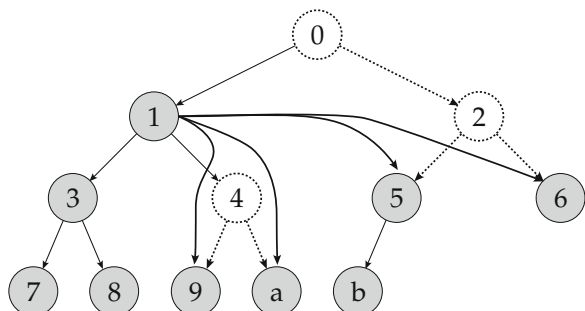


the number of messages and the degree of the graph while providing a strongly resilient topology that can only be bipartite by the sudden disappearance of more than half of the processes. Moreover, the new algorithm is implemented at the BTL level and takes advantage of the fast network interconnect available on most of the HPC clusters.

The `MPI_Comm_agree()` routine was also relying, for parts of its services (namely, the all-reduce tree maintenance), on messages at the OOB level. The overhead imposed by the OOB mechanism has been hindering the performance and scalability of the agreement. We completely reworked the agreement to use directly the fast network communication infrastructure available in Open MPI (namely, the BTL). This step provided a significant boost in terms of performance of the agreement algorithms. However, this highlighted few implementation issues with quadratic search involved in the original algorithms (due mainly to the MPI semantics of translating ranks between communicators). As a result, we decided to redesign the agreement implementation from the ground up. Dynamic topologies have been introduced, topologies that remain optimal as long as no new processes disappear and that are updated upon a successful completion of an agreement (due to the semantic of the agreement itself, a consensus on the dead processes is established during the agreement, allowing for a consistent global view of the alive processes). This topology is described in the Fig. 4.8. The important change is the coarsest dashed lines, which represent the new connections that are established to cope with the discovery of dead processes (signaled by the next coarsest dashed line). These lines connect a process to the leftmost ancestor still alive and count for one of the most costly, but critical, operations during the agreement. Without going in too much details, we have designed a new consensus algorithm named Early Returning Agreement, with a worst case cost of $\sum f = 1 \dots F(\log(P - f))$ where F is the number of fault discovered during a single agreement and P is the total number of processes.

Moreover, users reports signaled recurring corner cases in the previous agreement implementations, where the agreement routine was unable to provide its service because of the presence of failures. We wrote a reference implementation of the agreement, based on an existing Early Termination Agreement protocol, which work in asynchronous all-to-all phases, ensuring a correct result in any possible failure scenario. In addition to this reference implementation, we initiated works

Fig. 4.8 Example of dynamic topology



on more efficient implementations, which will take advantage of the high resilience property of the BMG, to reduce the number of messages while still providing the insurance of safe agreement despite failures in most situations.

4.5.2 Spare Node Substitution

This subsection considers the questions of how spare nodes should be allocated, how to substitute them with faulty nodes, and how much the communication performance is affected by the substitution. The third question stems from the modification of the rank mapping by node substitutions, which can incur additional message collisions. In a stencil computation, rank mapping can be done in a straightforward way on a Cartesian network without incurring any message collisions. However, once a substitution has occurred, such collision-free node-rank mapping can be destroyed. Figure 4.9 shows an example of message collisions (right figure). In this case, the failed node substitution results in having at most five message collisions. When two messages collide, their latency increases, up to at most doubling. Thus, the number of message collisions should be as low as possible.

The number of message collisions depends on the spare node allocation and how the failed node is substituted. Although the substitution idea proposed here can be applied to Cartesian networks having any order, the explanations are mostly using 2D network topology, due to the simplicity.

Figure 4.10 shows how the failed node is substituted in this proposed way. In this example, spare nodes are allocated at the edges of the 2D node space. The *OD sliding* method is the most simple one: just replace the failed node with a spare

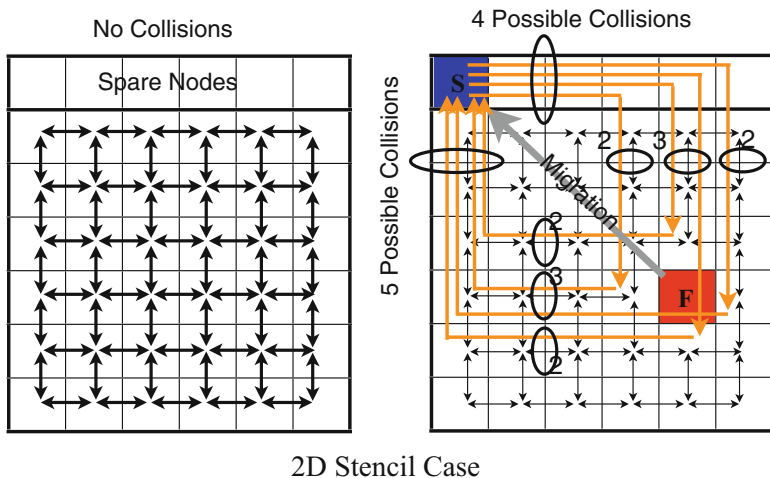


Fig. 4.9 Message collisions on 5P 2D stencil on a Cartesian network (X-Y routing)

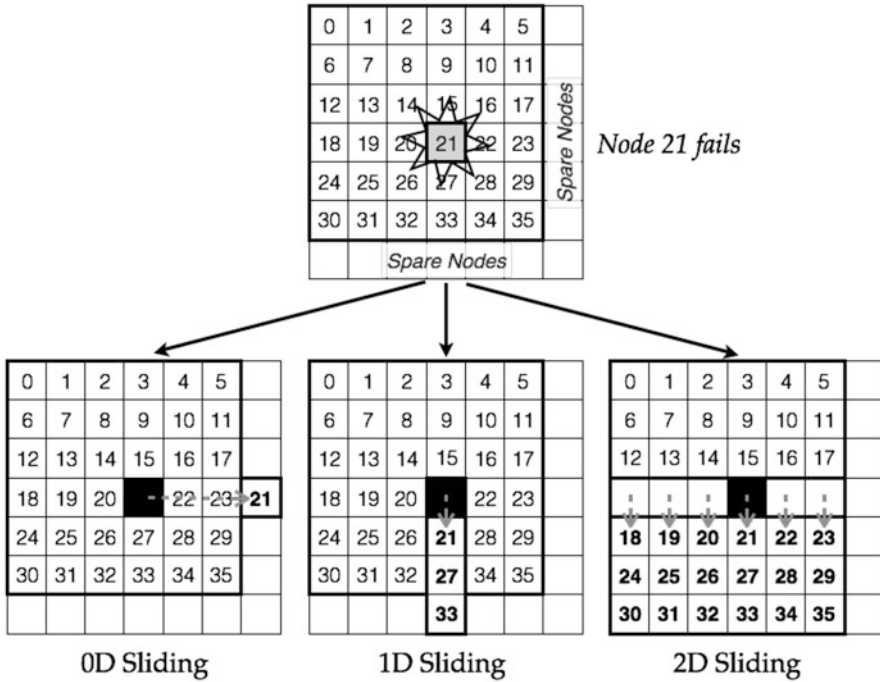


Fig. 4.10 0D, 1D, and @d sliding methods on 2D Cartesian network

node which is located to the nearest Manhattan distance from the failed node. In the *1D sliding* method, firstly the spare node having the same X-axis or Y-axis is chosen and the nodes between the failed node and the spare node are shifted. In *2D sliding* method, all nodes having the larger or equal to X-axis or Y-axis location are shifted. In general, the sliding method can be extended up to *ND sliding*, where *N* is the dimension order of a Cartesian topology network. When a substitution of a failed node happens, the inter-node communication of a stencil computation takes place between the neighbor nodes of the spare node. And the message collisions can happen on the paths between the neighbor nodes and the spare node. This is the reason why the nearest spare node is chosen so that the paths can be minimized.

The highest-order sliding method can be applied to only the number of the order of the network, assuming spare nodes are allocated on every single edge of the network. And this highest-order sliding method increases one hop count, but no collision happens.

Those substitution methods can be combined, and this is named *hybrid sliding* method as shown in Fig. 4.11. In 2D sliding method, the nodes in the gap which is created by the 2D sliding method can be used as a new spare node set. In this hybrid sliding method, the method having the highest order must be applied. This is

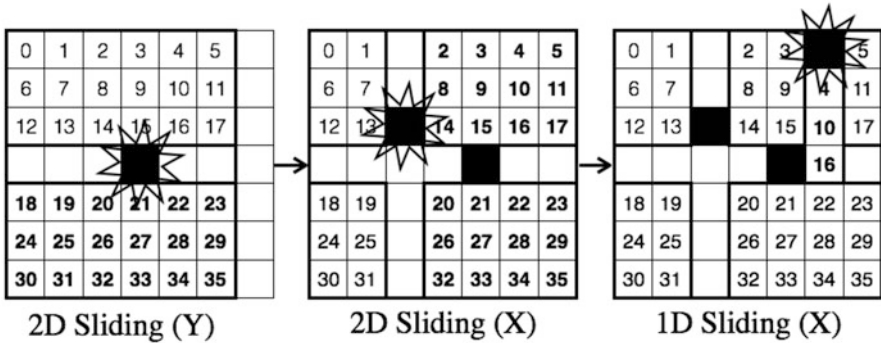


Fig. 4.11 Hybrid sliding methods on 2D network

because the highest-order sliding method does not impact the network performance so much. And if a sliding method cannot be applied, then the lower-order sliding method is applied. This procedure is repeated until the 1D sliding method fails. Since the 0D sliding method has no limitation as long as spare nodes exists, it can be used as a last resort.

We developed a simulator of the proposed sliding methods to count the maximum message collisions and compare the performance (message collisions) of the proposed sliding methods. We also evaluated the sliding methods on the K computer, BG/Q, and Tsubame2.5 [8].

4.6 Subsequent Development

Although this project was started with the prediction of heterogeneous CPU architecture would be a common many-core architecture, this does not come true. Intel announced stand-alone many-core CPU, Knights Landing (KNL). Although MPVAS software developed in this project became useless for the stand-alone many-core CPUs, the idea of packing multiple processes running on heterogeneous CPUs into one virtual address space is still effective.

The PVAS project is still active. One of the most drawbacks of PVAS was the implementation by the patched Linux, and PVAS was hard to port to the other OSES and architectures. To overcome this, a new user-level implementation has been being developed, named *Process-in-Process (PiP)* [7]. ULP on PVAS is also planned to implement on top of PiP. The other techniques developed in this project excepting MPVAS are applicable for the exa-scale computing. The research on scalable MPI-IO, EARTH, is also going on. The ULFM development team is trying to push ULFM into the MPI standard.

4.7 Summary

We proposed PVAS as a new parallel execution model especially for many-core CPU, MPVAS for heterogeneous CPU architectures, ULP to have fast context switching between user-level processes, EARTH for efficient MPI collective I/O, ULFM enhancement, and sliding substitution methods to replace failed nodes with spare nodes. Successor projects are contributing to the exa-scale computing.

References

1. Angskun, T., Bosilca, G., Dongarra, J.J.: Binomial graph: a scalable and fault-tolerant logical network topology. In: *Parallel and Distributed Processing and Applications, 5th International Symposium, ISPA 2007, Niagara Falls, 29–31 Aug 2007*, pp. 471–482 (2007)
2. Bland, W., Bouteiller, A., Herault, T., Bosilca, G., Dongarra, J.: Post-failure recovery of MPI communication capability: Design and rationale. *Int. J. High Perform. Comput. Appl.* **27**(3), 244–254 (2013). <https://doi.org/10.1177/1094342013488238>
3. Bouteiller, A., Bosilca, G., Dongarra, J.J.: Plan b: Interruption of ongoing MPI operations to support failure recovery. In: *Proceedings of the 22Nd European MPI Users' Group Meeting, EuroMPI '15*, pp. 11:1–11:9. ACM, New York (2015). <https://doi.org/10.1145/2802658.2802668>
4. Brightwell, R., Pedretti, K., Hudson, T.: SMARTMAP: operating system support for efficient data sharing among processes on a multi-core processor. In: *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, SC '08*, pp. 25:1–25:12. IEEE Press, Piscataway (2008). <http://dl.acm.org/citation.cfm?id=1413370.1413396>
5. Fukazawa, G.: Multiple PVAS: a systems software for HPC application programs on multi-core and many-core. Master Thesis, in Japanese (2014)
6. Herault, T., Bouteiller, A., Bosilca, G., Gamell, M., Teranishi, K., Parashar, M., Dongarra, J.: Practical scalable consensus for pseudo-synchronous distributed systems. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '15*, pp. 31:1–31:12. ACM, New York (2015) <https://doi.org/10.1145/2807591.2807665>
7. Hori, A., Si, M., Gerofi, B., Takagi, M., Dayal, J., Balaji, P., Ishikawa, Y.: Process-in-process: techniques for practical address-space sharing. In: *The 27th International Symposium on High-Performance Parallel and Distributed Computing (HPDC'18)*. ACM (2018)
8. Hori, A., Yoshinaga, K., Herault, T., Bouteiller, A., Bosilca, G., Ishikawa, Y.: Sliding Substitution of Failed Nodes. In: *Proceedings of the 22Nd European MPI Users' Group Meeting, EuroMPI '15*, pp. 14:1–14:10. ACM, New York (2015). <https://doi.org/10.1145/2802658.2802670>
9. Matsuoka, S.: The road to tsubame and beyond. In: Resch, M., Roller, S., Lammers, P., Furui, T., Galle, M., Bez, W. (eds.) *High Performance Computing on Vector Systems 2007*, pp. 265–267. Springer, Berlin/Heidelberg (2008)
10. Nakashima, J., Taura, K.: MassiveThreads: A Thread Library for High Productivity Languages, pp. 222–238. Springer, Berlin/Heidelberg (2014). https://doi.org/10.1007/978-3-662-44471-9_10
11. Pérache, M., Jourden, H., Namyst, R.: MPC: a unified parallel runtime for clusters of NUMA machines. In: *Proceedings of the 14th International Euro-Par Conference on Parallel Processing, Euro-Par'08*, pp. 78–88. Springer, Berlin/Heidelberg (2008). http://doi.org/10.1007/978-3-540-85451-7_9
12. Reinders, J.: *An Overview of Programming for Intel Xeon processors and Intel Xeon Phi coprocessors* (2012)

13. del Rosario, J.M., Bordawekar, R., Choudhary, A.: Improved parallel i/o via a two-phase run-time access strategy. *SIGARCH Comput. Archit. News* **21**(5), 31–38 (1993). <https://doi.org/http://doi.acm.org/10.1145/165660.165667>
14. Sato, M., Fukazawa, G., Shimada, A., Hori, A., Ishikawa, Y., Namiki, M.: Design of multiple pvas on infiniband cluster system consisting of many-core and multi-core. In: *Proceedings of the 21st European MPI Users' Group Meeting, EuroMPI/ASIA '14*, pp. 133:133–133:138. ACM, New York (2014). <https://doi.org/10.1145/2642769.2642795>.
15. Sato, M., Fukazawa, G., Yoshinaga, K., Tsujita, Y., Hori, A., Namiki, M.: A hybrid operating system for a computing node with multi-core and many-core processors. In: *International Journal Advanced mputer Science (IJACSci)*, vol. 3, pp. 368–377 (2013)
16. Schneider, T., Gerstenberger, R., Hoefler, T.: Micro-Applications for Communication Data Access Patterns and MPI Datatypes. In: *Recent Advances in the Message Passing Interface – 19th European MPI Users' Group Meeting, EuroMPI 2012, Vienna, Austria, 23–26 Sept 2012. Proceedings*, vol. 7490, pp. 121–131. Springer (2012)
17. Shimada, A.: A study on task models for high-performance and efficient intra-node communication in many-core environments. Ph.D. thesis, Keio University (2017). (in Japanese)
18. Shimada, A., Gerofi, B., Hori, A., Ishikawa, Y.: Pgas intra-node communication towards many-core architecture. In: *In PGAS 2012: 6th Conference on Partitioned Global Address Space Programing Model, PGAS'12* (2012)
19. Shimada, A., Gerofi, B., Hori, A., Ishikawa, Y.: Proposing a new task model towards many-core architecture. In: *Proceedings of the First International Workshop on Many-core Embedded Systems, MES '13*, pp. 45–48. ACM, New York (2013). <https://doi.org/10.1145/2489068.2489075>. <http://doi.acm.org/10.1145/2489068.2489075>
20. Shimada, A., Hori, A., Ishikawa, Y.: Eliminating costs for crossing process boundary from mpi intra-node communication. In: *Proceedings of the 21st European MPI Users' Group Meeting, EuroMPI/ASIA '14*, pp. 119:119–119:120. ACM, New York (2014). <https://doi.org/10.1145/2642769.2642790>
21. Shimada, A., Hori, A., Ishikawa, Y., Balaji, P.: User-level process towards exascale systems **2014**(22), 1–7 (2014). <http://ci.nii.ac.jp/naid/110009850784/>
22. Shimosawa, T., Gerofi, B., Takagi, M., Shirasawa, T., Shimizu, M., Hori, A., Ishikawa, Y.: Interface for Heterogeneous Kernels: A Framework to Enable Hybrid OS Designs Targeting High Performance Computing. In: *IEEE International Conference on High Performance Computing (HiPC)*. IEEE (2014)
23. Thakur, R., Lusk, E., Gropp, W.: *Users Guide for ROMIO: A High-Performance, Portable MPI-IO Implementation*. Technical Memorandum ANL/MCS-TM-234, Argonne National Laboratory (2004)
24. Tsujita, Y., Hori, A., Ishikawa, Y.: Locality-aware process mapping for high performance collective MPI-IO on FEFS with Tofu interconnect. In: *Proceedings of the 21th European MPI Users' Group Meeting*, pp. 157–162. ACM (2014). *Challenges in Data-Centric Computing*. <https://doi.org/10.1145/2642769.2642799>
25. Tsujita, Y., Hori, A., Kameyama, T., Uno, A., Shoji, F., Ishikawa, Y.: Improving collective MPI-IO using topology-aware stepwise data aggregation with I/O throttling. In: *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, pp. 12–23. ACM (2018). <https://doi.org/10.1145/3149457.3149464>
26. Yokokawa, M., Shoji, F., Uno, A., Kurokawa, M., Watanabe, T.: The K Computer: Japanese Next-Generation Supercomputer Development Project. In: *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design, ISLPED '11*, pp. 371–372. IEEE Press, Piscataway (2011). <http://dl.acm.org/citation.cfm?id=2016802.2016889>

Chapter 5

Highly Productive, High-Performance Application Frameworks for Post-Petascale Computing



**Naoya Maruyama, Takayuki Aoki, Kenjiro Taura, Rio Yokota,
Mohamed Wahib, Motohiko Matsuda, Keisuke Fukuda,
Takashi Shimokawabe, Naoyuki Onodera, Michel Müller,
and Shintaro Iwasaki**

Abstract We present an overview of our project that aimed to achieve both high performance and high productivity. In order to achieve our aim, we designed and developed high-level domain-specific frameworks that can automate many of tedious and complicated program optimizations for certain computation patterns. This article walks through some of our research results and highlights how we achieved both high performance and high productivity.

5.1 Overview

The main challenge of our project was to achieve both high performance and high productivity. While it is generally intractable to attain both of them simultaneously, we observe that existing software stacks for HPC applications often compromise either of them as they strive to be flexible, general-purpose software. While generality is obviously important in general, we envisioned that specialized software stacks could provide high performance without compromising productivity. More specifically, we focused on computational fluid dynamics and fast multipole method and first developed high-performance reference implementations that allowed us to learn key optimization techniques for large-scale heterogeneous systems (Sects. 5.2

N. Maruyama (✉) · M. Wahib · M. Matsuda
RIKEN AICS, Kobe, Hyogo, Japan
e-mail: nmaruyama@riken.jp

T. Aoki · R. Yokota · K. Fukuda · N. Onodera · M. Müller
Tokyo Institute of Technology, Meguro-ku, Tokyo, Japan

K. Taura · T. Shimokawabe · S. Iwasaki
University of Tokyo, Bunkyo-ku, Tokyo, Japan

and 5.3). We then designed several domain-specific frameworks (Sect. 5.5) that automate the learned optimization techniques for the target application domains by exploiting scalable runtime system software (Sect. 5.4).

5.2 High-Performance Computational Fluid Dynamics Simulations

5.2.1 *Large-Scale LES Wind Simulation Using Lattice Boltzmann Method for a 10 km × 10 km Area in Metropolitan Tokyo*

Pedestrians often feel strong winds around tall buildings in the metropolitan Tokyo area. The concentration of tall buildings makes the air flow turbulent. In order to understand the details of the airflow there, it is necessary to carry out large-scale computational fluid dynamics (CFD) simulations, but thanks to recent progress in supercomputers, we can now execute large-scale computation using a billion mesh points.

The lattice Boltzmann method (LBM) constitutes a class of CFD methods that solve the discrete-velocity Boltzmann equation. Since the LBM is based on a weak compressible formulation, the time integration is explicit, and we do not need to solve the pressure Poisson equation. This makes the LBM scalable and, thus, suitable for large-scale computation. As an example, researchers performing large-scale calculation using LBM [40] won the Gordon Bell prize in SC10. Moreover, fluid phenomena taking into consideration complex shapes have also been studied [6, 61]. However, LBM has not been applied to turbulent flows at high Reynolds numbers. So far, there has been no large-scale wind flow simulation accounting for real building shapes.

A large-eddy simulation (LES) is an approach that can deal with unsteady turbulent flows. It is expected that the application of LES to LBM would allow one to make a stable calculation of turbulent flows with high Reynolds numbers [65]. The dynamic Smagorinsky model [19] is a prominent subgrid-scale model based on the concept of eddy viscosity. However, to determine the model parameter, a spatial average has to be carried out over the global domain. This makes the computation too inefficient for large-scale simulations. The coherent structure model (CSM) [24] is a remedy to these problems; it enables the model parameter to be locally calculated without taking any averages. CSM is suitable for a petascale LES wind simulation.

We measured the performance of our LBM code on TSUBAME 2.0 (NVIDIA TESLA M2050) and TSUBAME 2.5. (NVIDIA TESLA K20X). The code was written in the GPU programming framework CUDA. The overlapping technique, which hide the communication time with the computation time, is also applied to achieve high performance on parallel computation.

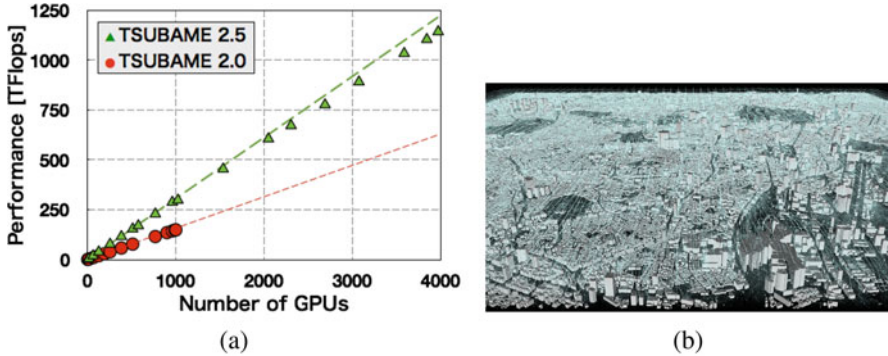


Fig. 5.1 Performance and simulation results. (a) Weak scalability in single precision (b) Snapshot of wind flow (north is up)

Figure 5.1a presents the weak scalabilities of the LBM performance on TSUBAME 2.0 and 2.5. Each GPU handled a domain with a $192 \times 256 \times 256$ mesh. We achieved 149 TFLOPS on 1,000 GPUs of TSUBAME 2.0 and 1.14 PFLOPS on 3968 GPUs of TSUBAME 2.5 in single precision. Since the model parameter of CSM could be locally determined without averaging, we obtained fairly good performance for weak scalability.

The large-scale wind simulation of a $10 \text{ km} \times 10 \text{ km}$ area had a resolution of 1 m, and it included real building data of the city of Tokyo. The computational domain covered the major wards of Tokyo, including Shinjuku, Chiyoda, Chuo, Minato, and Meguro. We used 4,032 GPUs and a $10,080 \times 10,240 \times 512$ mesh. The inflow and outflow conditions were applied in the streamwise direction (from north to south). Periodic boundary conditions were assumed in the spanwise direction (east and west). The ground had a nonslip condition. The inlet velocity was set to be $U_{in} = A \cdot \log_{10} z / 2$, where wind velocity was 10 m/s at a height of 100 m. Figure 5.1b shows a snapshot of the wind flows visualized by using mass-less particles.

We concluded that the present scheme is one of most promising approaches to simulating a wide area of turbulence. The LES computation for the $10 \text{ km} \times 10 \text{ km}$ area with a 1 m resolution is the most extensive of its kind.

5.3 High-Performance Fast Multipole Method

This article describes aspects of our fast multipole method code – *exaFMM* –, which could otherwise have been difficult to publish in a scientific journal because they were too software oriented. What we mention here has greatly helped simplify our code and has kept the pace of development from slowing down even when the code became bigger. These techniques can be thought of as subtle improvements in the code implementation and following best practices in software engineering.

5.3.1 Dual Tree Traversal

In conventional FMM codes, the interaction lists for the M2L kernel are determined from the parent’s neighbor’s children that are non-neighbors, with extensions to adaptive tree structures by considering well-separated descendants of neighbors of leaf cells [8]. There are also methods to reduce the translation stencil from 189 to 119 [21] or even down to 40 [66]. In some cases the definition of the neighbor region is extended from $3 \times 3 \times 3$ to $5 \times 5 \times 5$ [2].

exaFMM was designed to combine the strengths of treecodes and FMMs and has adopted the dual tree traversal as a mechanism to calculate the optimal interaction list for any tree structure. The idea of a dual tree traversal was first mentioned by Warren and Salmon [59] and in more detail along with an optimized implementation by Dehnen [12]. It has been parallelized using task-based models independently by Yokota [62] and by Lange and Fortin [26]. It’s concept is extremely simple, where the source tree and target tree are simultaneously traversed while applying the MAC to each pair of cells.

Figure 5.2 shows the flow of calculation for the dual tree traversal. We start from the pair of root cells and traverse both the source tree and target tree simultaneously. If the pair of cells satisfy the MAC, the M2L kernel is executed/queued, and the traversal is continued recursively otherwise. When the traversal hits the bottom of both trees and it still doesn’t satisfy the MAC, the P2P kernel is executed/queued.

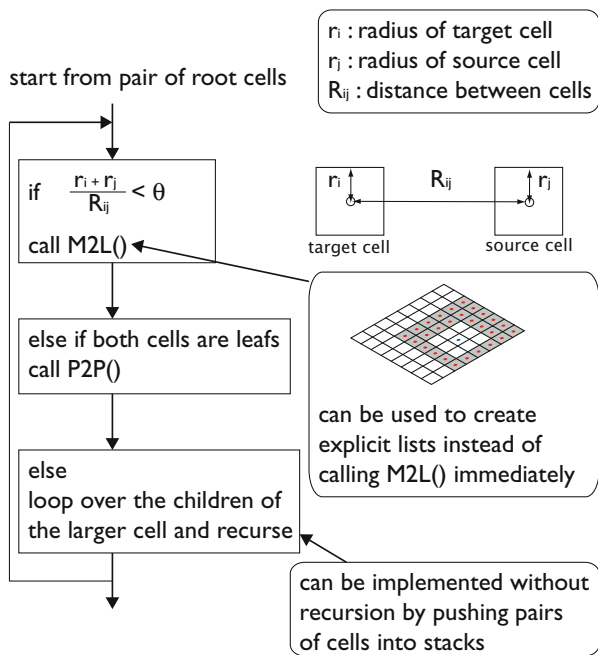


Fig. 5.2 Dual tree traversal

The concept of traversing two trees to obtain the interaction lists goes back to the original work by Appel [4], though implementing it in its original form was proven to be suboptimal [29].

5.3.2 *Inner Kernels*

5.3.2.1 **Operator Overloading**

In order to achieve performance portability without saving different implementations of each FMM kernel for every single architecture, we resort to C++ operator overloading of a SIMD vector type. For example, a “+” operator for a SIMD vector of floats will be overloaded with a `_mm512_add_ps` if the architecture has AVX512, `_mm256_add_ps` for AVX, and `_mm_add_ps` for SSE equipped hardware. With this approach, the kernels need to be written only once to run on hardware of different SIMD generations. The performance is equivalent to coding in intrinsics because the operators are overloaded with intrinsics. Using this technology, the following inner kernels can be implemented efficiently.

5.3.2.2 **Implementation Differences**

There is also a large difference in performance among the various implementations of the same type of expansion. For example, the spherical harmonics with rotation is a popular choice, and various implementations exist, with different ways to calculate the recurrence relation [9, 10, 25, 28], reduce the interaction list size by exploiting symmetry [11, 21, 52, 66], or use BLAS to process the translation operations in batches [15, 16]. Spherical harmonics and Legendre polynomials involve the calculation of factorials and could result in loss of precision if not implemented carefully. This is also true for the Hankel and Bessel functions that are needed for the Helmholtz kernels [37].

5.3.3 *Partitioning (Load-Balancing)*

exaFMM supports all of the partitioning schemes mentioned in this subsection. There are two main categories for partitioning schemes for hierarchical N -body methods. One is the orthogonal recursive bisection (ORB) and the other is the hashed octree (HOT).

5.3.3.1 **Orthogonal Recursive Bisection**

In ORB the domain is bisected at a position where the workloads on both sides are equal, where the workload is determined from the previous step [43]. If the

number of processes is not a power of two, we can use multi-sections where the domain is partitioned into either two or three subdomains [30]. The ORB works well with the hypercube type all-to-all communication scheme [27] if we map every level of the bisection to a hypercube dimension. In this scheme, every process will only communicate with one other process to exchange the bodies at each bisection, and the end result will still be an all-to-all communication. In the `exaFMM` implementation of ORB, we split the MPI communicator hierarchically to map it to the binary tree structure. Therefore, it is trivial to find the pair of processes that exchange bodies by simply using the split communicator at each level of the multi-section tree.

5.3.3.2 Hashed Octree

In HOT the Morton/Hilbert ordered space filling curves are split into equal segments based on the workload and assigned to each process, where the workload is determined from the previous step [58]. A similar method called costzones was proposed for shared memory [50], and HOT can be thought of as a distributed memory extension of it [20]. At the partitioning stage, it is not necessary to sort within the partitions. Therefore, applying a global parallel sort on all bodies is suboptimal, considering that there are usually millions of bodies per partition. Parallel sampling-based techniques have proven to be useful for both finding the bisectors in ORB [30] and finding the splitting keys in HOT [51]. In `exaFMM` we have our own implementation of a sampling-based parallel histogram sort, which sorts only among the partitions and not within the partitions. `exaFMM` allows the user to choose between Morton and Hilbert ordering, though for most distributions we have found that it makes little difference.

5.3.4 Local Essential Tree

5.3.4.1 Communication

In a distributed memory implementation of the FMM, a unique subset of the tree from every process must be communicated to every other process. From this information, each process can construct the local essential tree (LET), which is a subset of the global tree that is required by each process [43]. Since the communication for the LET requires unique information to be sent from every process to every other process, its naive implementation relies on an `MPI_Alltoallv`. Furthermore, the distribution of the bodies and the resulting tree structure on remote processes is unknown, so each process does not actually know what information to ask for. This problem can be solved by guessing from the local tree what needs to be sent to every other process. Such sender-initiated LET communication schemes are described for both ORB [14] and HOT [41]. `exaFMM` uses this sender-initiated LET

communication for both the ORB and HOT. The MPI communicator splitting used for the partition is also used here to facilitate a hypercube type communication.

5.3.4.2 Grafting

Typically, code for merging the LET would take a large portion of a parallel FMM code, and this has made it difficult to implement new features such as periodic boundary conditions, mutual interaction, more efficient translation stencils, and dual tree traversals in most FMM codes. `exaFMM` is able to incorporate all these extended features and still maintain a fast pace of development because of its simplification in how the global tree structure is geometrically separated from the local tree structure. This separation of the global and local tree structure is also a requirement for achieving a theoretical communication complexity of $\mathcal{O}(\log P + (N/P)^{2/3})$ [64].

5.3.4.3 Thread Scalability

The thread scalability is shown for up to 16 threads for various tol and N values in Fig. 5.3a. In general, higher accuracy and larger problem size lead to much more computation and therefore have much more concurrency and are easier to scale. These experiments are conducted on a node with 12 physical cores, so it is not expected to scale to 16 threads. `exaFMM` has compute-bound kernels and hyper-threading will not help in this case. We see that `exaFMM` scales up to 8 cores if the computational load is large enough.

5.3.4.4 MPI Scalability

For the strong and weak scalability results, we use a Cray XC40 with 6174 nodes. The code used for the distributed memory scalability runs is in `exafmm/uniform`

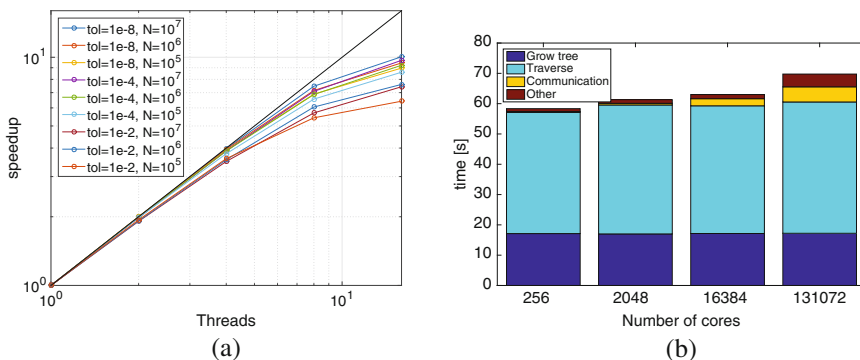


Fig. 5.3 Scalability of `exaFMM` code (a) Thread scalability of `exaFMM` for different numbers of bodies N and different orders of expansions P (b) Weak scalability of `exaFMM` for $N = 10^8$ per node, with 64 cores per node, and $P = 6$

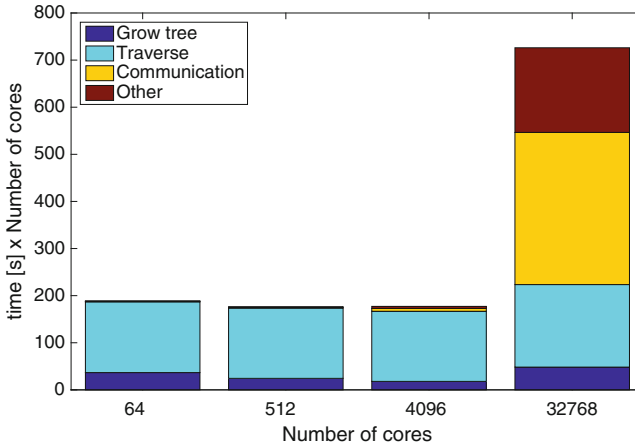


Fig. 5.4 Strong scalability of exaFMM for $N = 200,000,000$ with 64 cores per node, and $P = 6$

and compiled with `make parallel`. The weak scalability results are shown in Fig. 5.3b, where the number of bodies is set to $N = 100,000,000$ per node with 64 cores per node. The order of expansion is set to $P = 6$. In the breakdown, “Grow tree” represents the tree construction time, “Traverse” represents the kernel evaluation time, “Communication” represents the MPI communication time, and “Other” is the total of everything else. The communication becomes visible beyond 16,384 cores but is still quite small even for 131,072 cores. The strong scalability is shown in Fig. 5.4, where the number of bodies is set to $N = 200,000,000$ in total. We use 64 cores per node and an expansion order $P = 6$ similar to the weak scaling case. The computation time is multiplied by the number of cores for the sake of visibility of the breakdown. The communication starts to dominate at 32,768 cores, but note that the actual execution time (without multiplying by the number of cores) is still faster for 32,768 than the 4,096 case.

5.4 A Runtime System for Unified Task-Parallel Programming Models

5.4.1 What Are Task-Parallel Models?

A task-parallel programming model is a parallel programming model supporting dynamic creation of tasks at arbitrary point in the execution. It allows a running task to create another task, just like any serial programming language allows a function to call another function. It thus supports a nested tree of tasks. While details vary among systems, it generally allows a task to wait for its child tasks to finish. In task-parallel programming models, the programmer is responsible

for *extracting parallelism* by creating tasks, leaving the job of mapping them to available hardware for the system (a compiler, a runtime system, or a combination thereof). In particular, the programmer does not have to match the number of runnable (ready-to-execute) tasks with the number of available execution units (e.g., cores). In addition, implementation of task-parallel systems generally tries to minimize the overhead for creating and finishing a task. With the automatic mapping of tasks to hardware resources combined with the low overhead, the programmer is relieved from burdens of careful partitioning of the work to available resources.

Task-parallel systems differ in the scope in which a task can distribute. Most commonly, a task-parallel system distributes tasks to cores within a shared memory node (*local* task-parallel systems). Programming languages supporting local task parallelism include Cilk [17], OpenMP tasks [5], Intel Threading Building Block [42], and Chapel [67]. Libraries that can readily be used for implementing such languages include MassiveThreads [36], Qthreads [60], and Argobots [44]. There are systems that distribute tasks globally across nodes (*global* task-parallel systems), e.g., [13, 22, 32] among others. It is also possible and logical to map tasks to SIMD lanes, another dimension along which the hardware exposes parallelism to the programmer. Mapping what is conceived as a task or a thread from the programmer's perspective to SIMD lanes entails a significant compiler effort. Our research efforts range across all types of task parallelism.

5.4.2 *MassiveThreads Lightweight Thread Library*

MassiveThreads [36] is a lightweight thread library that exposes an ordinary thread-like API but implements it much more efficiently in the user level. It also uses the work-stealing algorithm [7, 33] for task scheduling and distribution, so that it is suitable for divide-and-conquer parallel algorithms. On top of MassiveThreads API, we have implemented a layer compatible with `taskgroup` in the Intel Threading Building Block. This demonstrates that one can easily build a high-level task-parallel interface on top of MassiveThreads. We also implemented a binding for Standard ML# [38] and integrated it into Cray Chapel [67] as an optional tasking layer.

5.4.3 *Task-Parallel Fast Multipole Methods (exaFMM)*

We have implemented a task-parallel formulation the exaFMM library [63], a highly optimized implementation of fast multipole methods. Its central data structure is an octree that decomposes the three-dimensional space. The shape of the octree is determined by particle distributions and can be highly imbalanced when the distribution is skewed.

We have shown a task parallelism plays a great role in parallelizing every phase of exaFMM, including octree construction, calculation of interaction, and upward and downward propagation [53]. In particular, exaFMM employs a dual tree traversal [12], which takes two trees and descends along both of them in a data-dependent fashion. MassiveThreads greatly helps its parallelization, by allowing the programmer to simply spawn a new task when it descends a tree.

5.4.4 *UniAddress: A Library-Based Global Work Stealing with RDMA*

Compared to the local task parallelism, which has become readily available as a means to distribute tasks within a single shared memory node, distributing tasks across nodes is far more challenging.

We developed a global work stealing *library*, MassiveThreads/DM [1], which can distribute tasks compiled with ordinary C/C++ compilers. Being library is an important property, as it allows the user programs to be compiled and optimized by mature production compilers. Most global task-parallel systems as of today do not support recursive (arbitrarily nested) task creations/synchronizations, which are essential for describing divide-and-conquer algorithms. To our best knowledge, MassiveThreads/DM is the first system that supports recursive task creations/synchronizations with ordinary C/C++ compilers.

Toward realizing this goal, a challenging issue arises due to the fact that each node in a distributed memory supercomputer has its own distinct address space. In this setting, moving a task from one node to another requires an explicit movement (migration) of the task state, which is essentially a stack of activation frames. After a migration, the stack will be copied into an address available in the destination node, which is, naturally, different from the address it occupied in the originating node. If this is the case, the runtime system must modify pointers pointing to an address within the moving stack. This fix-up operation requires precisely locating all pointers in a stack, which is difficult to achieve in non-type-safe languages such as C/C++. Since our goal is to take advantage of production C/C++ compilers and its mature optimizations, our only choice seems to ensure a migrating stack is copied into exactly the same address in the destination node. The scheme was invented by Antoniu et al. and named IsoAddress [3]. There is a scalability issue in IsoAddress scheme, however; it requires a huge logical address space in each node, so that *any* node can potentially accommodate *any* live stack in the entire system. Since the number of live stacks is proportional to the number of cores, there is a real risk that we run out of logical address space in future massively parallel systems. Furthermore, this luxury use of virtual addresses has a bad memory locality and causes extra TLB misses and page faults. We proposed a solution to these problems, UniAddress [1], which ensures that a stack is on the identical address only when it is actually running.

5.4.5 *Static Optimizations and Vectorization of Tasks*

To obtain good peak performance and energy efficiency, recent processors increasingly rely on wide single instruction, multiple data (SIMD) instructions, instructions that apply a single operation (e.g., multiply, add, etc.) to multiple data. Programmers typically use SIMD instructions either through low-level intrinsic APIs (functions that directly map to SIMD instructions) or regular loops that compilers are able to use SIMD instructions for. In the context of task-parallel programs, this means that programmers typically have to program multicore and SIMD separately; they use tasks for utilizing multiple cores and loops or intrinsics within each task for utilizing SIMD parallelism within a core.

This doubles the burden on the programmer. Furthermore, considering that the idea of task-parallel programming models is to create lots of logically concurrent tasks to be mapped by the system to hardware resources, it is natural to explore the possibility to distribute tasks to SIMD lanes too. In this ideal model, the programmer creates lots of tasks without explicitly vectorizing the work within individual tasks, and the system utilizes SIMD instructions to execute *multiple* tasks. This is conceptually similar to the single instruction, multiple threads (SIMT) execution model of CUDA, in which programmers only need to create lots of CUDA threads, to be executed by GPU hardware. Internally, GPU has a mechanism similar to SIMD, called *warp*, with which all threads within a single warp can execute an instruction in the same clock cycle, as long as they are executing the same instruction. SIMT execution model unifies parallelism inside and across warps in the programming model, albeit only for the flat parallelism expressible in the SIMT model. It is also similar to Intel SPMD compiler (ISPC), which compiles SPMD programs into SIMD execution [39].

Both SIMT and ISPC rely on the fact that the program is written in the SPMD model; the program describes *the action taken by each logical thread* and launches many instances of them. Programs expressed in this manner is amenable to SIMD execution, as the compiler's job is essentially to generate instructions that execute the described action in all SIMD lanes. Conditional executions are handled through predicated execution and/or branches. All in all, the compiler's job is similar to vectorizing a loop whose execution count is known.

Compiling task-parallel programs into SIMD is much more challenging for a number of reasons. First, tasks are dynamically created, so the number of tasks is not apparent in the program. Second, tasks may have dependencies, so all created threads are not necessarily concurrently executable. Third, there are no single piece of code that describe the action of all concurrent threads. The compiler must undertake significant static analysis to identify concurrent tasks that will execute the same instructions and transform them into a representation more amenable to vectorization.

We developed such a transformation based on the LLVM compiler framework [23]. It performs static elision of task creation near leaves of the task tree (static cut-off) and applies a number of optimizations to coalesced leaves. For regular

programs, it is able to transform them into loops, which may then be vectorized by the backend compiler. We observed a significant speedup (up to two orders of magnitude in an extreme case) for a number of programs.

5.5 Domain-Specific High-Level Frameworks

In this project, we have developed several tools and frameworks for computational fluid dynamics and fast multipole method, including the Physis stencil framework [31], a kernel fusion/fission optimization framework [54, 55], the Daino AMR framework [57], and the Tapas framework for FMM [18]. Below, we give a brief overview of some of the frameworks.

5.5.1 High-Level Framework for High-Performance AMR

In many scientific and engineering simulations, partial differential equations (PDEs) are solved in a uniform mesh arrangement by using finite-difference schemes, referred to as iterative stencils. Typically, the resolution of the mesh is uniformly set to the highest resolution to provide accurate solutions. For meshes that require only high resolution for some portions of the mesh, an alternative method, known as adaptive mesh refinement (AMR), can be used instead of the uniform mesh. The AMR method solves the problem on a relatively coarse grid and dynamically refines it in regions requiring higher resolution. However, AMR codes tend to be far more complicated than their uniform mesh counterparts due to the software infrastructure necessary to dynamically manage the hierarchical mesh framework. Despite this complexity, it is generally believed that future applications will increasingly rely on adaptive methods to study problems at unprecedented scale and resolution.

We design a high-level programming framework that provides a highly productive programming environment for AMR [56, 57]. The framework is transparent and requires minimal involvement from the programmer while generating efficient and scalable AMR code. The framework consists of a compiler and runtime components. A set of directives allows the programmer to identify stencils of a uniform mesh in an architecture-neutral way. The uniform mesh code is then translated to GPU-optimized parallel AMR code, which is then compiled to an executable. The runtime component encapsulates the AMR hierarchy and provides an interface for the mesh management operations. We demonstrate the scalability of auto-generated AMR code using three production applications. We compare the speedup and scalability with handwritten AMR of all three applications.

Applications: (a) Phase-field simulation: This application simulates 3D dendrite growth during binary alloy solidification. (b) Hydrodynamics solver: This solver models a second-order directionally split hyperbolic schemes to solve Euler equations. (c) Shallow water: Modeling shallow water by depth-averaging Navier-Stokes equations.

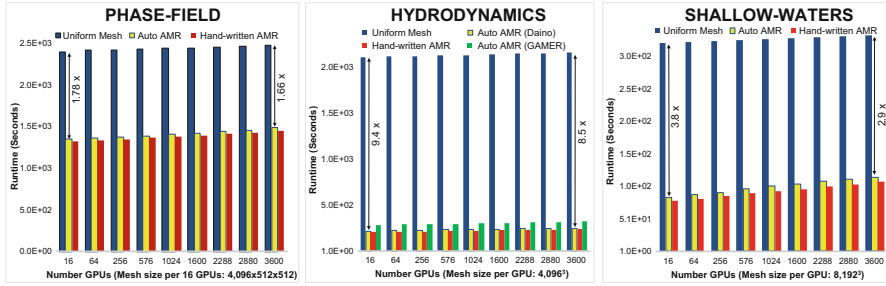


Fig. 5.5 Weak scaling of uniform mesh and handwritten and automated AMR

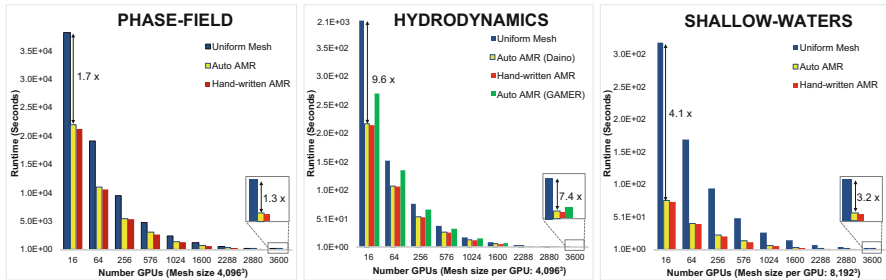


Fig. 5.6 Strong scaling of uniform mesh and handwritten and automated AMR

Results: In a weak scaling experiment, shown in Fig. 5.5, the runtime for uniform mesh, handwritten AMR, and auto-generated AMR are compared. The following points are important to note. First, more than $1.7\times$ speedup is achieved using Daino using the full TSUBAME machine, 3,640 GPUs, for the phase-field simulation. This is a considerable improvement considering that the uniform mesh implementation is a Gordon Bell prize winner for time-to-solution. Second, Daino achieves good scaling that is comparable to the scalability of the handwritten AMR code. Figure 5.6 shows a strong scaling comparison for handwritten and auto-generated AMR against uniform mesh implementation. The code generated by Daino achieves speedups and scalability comparable to handwritten implementations. However, when using more GPUs, reduction in speedup starts to occur as the management of AMR starts to dominate the simulation runtime.

5.5.2 Automatically Fusing Hundreds of Stencil Kernels for Better Performance and Productivity

Kernel fusion is an advanced optimization at which codes of different memory-bound kernels are aggregated to expose inter-kernel data localities. Performance can then potentially improve if opportunities of data reuse in the transformed code are

exploited effectively. Applying kernel fusion to real-world large-scale applications, however, is difficult. A main challenge is the need for a scalable method to search for the optimal choice of which kernels to fuse among the enumeration of feasible fusions. Another main challenge is to design an architecture-aware method for collectively applying fusion as a transformation: exposing locality does not guarantee actual performance improvement unless the architecture-related features were taken into consideration when optimizing for data reuse. Finally, a transformation method for applying kernel fusion that is automated and applicable to a diversity of applications is a nontrivial challenge.

This work introduces GPU kernel fusion for a class of memory-bound applications [54, 55]. In particular, PDEs solvers comprise a significant fraction of scientific applications in a variety of areas. Those applications are often implemented using memory-bound finite-difference techniques referred to as stencils. Optimized stencil codes in GPU typically depend on the on-chip memory to enhance performance. Kernel fusion further optimizes the use of on-chip memory in stencils and can be increasingly important in the future of performance optimization driven by the increased dependence on on-chip memory.

The work conducted in this project targets the following: make the process of applying kernel fusion to improve performance a practical choice for real-world applications. The main proposal is an end-to-end automated kernel transformation framework. The end-to-end solution includes the following steps: first, gathering of metadata via instrumentation and static analysis to understand the dependencies among kernels; second, searching for the optimal fusion of kernels using a customized optimization algorithm; and third, auto-generating new optimized kernels to replace the original kernels. A positive consequence of the automated approach is an opportunity for improving the optimization algorithm. More specifically, we propose the use of kernel fission to improve the search space exploration by increasing the number of feasible solutions through relaxation of the on-chip memory capacity constraint. This in turn increases the chances of finding kernel fusions that could expose higher locality.

We evaluate the proposed end-to-end solution by applying automated kernel transformations to enhance the performance of six real-world scientific applications in such diverse areas as weather modeling, seismic simulations, oceanic circulation modeling, electromagnetics, and fluid dynamics. The transformation resulted in speedups ranging between $1.12\times$ and $1.76\times$ compared to the original CUDA codebases. Further details of the performance results can be found in [55].

5.5.3 High-Productivity Framework on GPU-Rich Supercomputers for Weather Prediction Code

Numerical weather prediction is one of the major applications in high-performance computing and is accelerated on GPU supercomputers. The Japan Meteorological

Agency (JMA) is developing a high-resolution mesoscale weather prediction code ASUCA. Multi-GPU computation of mesh-based applications, including ASUCA, has the potential to achieve high performance. However, it requires relatively high cost of implementation [45, 46, 49].

We have developed high-productivity and high-portability framework for multi-GPU computation of mesh-based applications and implemented ASUCA based on this proposed framework from scratch [47, 48]. The proposed framework is designed to provide highly productive programming environment for stencil applications with explicit time integration running on regular structured grids. The framework updates the physical variables defined on grid points and stored in arrays in user programs. Our framework is implemented in C++ and CUDA languages. It automatically translates user-written stencil functions that update a grid point and generates both GPU and CPU codes. The programmers write user code just in C++ and it can be executed on multiple GPUs.

In this framework, stencils must be defined as C++ functors called stencil functions. The stencil function for three-dimensional diffusion equation is defined as follows:

Listing 5.1 Stencil functions as C++ functors

```
struct Diffusion3d {
    __host__ __device__
    void operator()(const ArrayIndex3D &idx ,
                   float ce, float cw, float cn, float cs,
                   float ct, float cb, float cc,
                   const float *f, float *fn) {
        fn[idx.ix()] =
            cc*f[idx.ix()]
        + ce*f[idx.ix<1,0,0>()] + cw*f[idx.ix<-1,0,0>()]
        + cn*f[idx.ix<0,1,0>()] + cs*f[idx.ix<0,-1,0>()]
        + ct*f[idx.ix<0,0,1>()] + cb*f[idx.ix<0,0,-1>()];
    }
};
```

Stencil access patterns on three-dimensional grids are described by using `ArrayIndex3D`, which is provided by the framework. `ArrayIndex3D` holds the size of each dimension of a grid and represents a certain grid point (i, j, k) , which is the coordinate of the point where this function is applied. To apply user-written stencil functions to grids, the framework provides the `Loop3D` class, which is used to invoke the diffusion equation on the three-dimensional grid by passing to it the stencil functions and their respective parameters.

In the multi-GPU computation, since stencil computation that updates to a point of grid needs to access its neighbor points, the data exchanges of boundary regions between subdomains are performed frequently. The framework provides

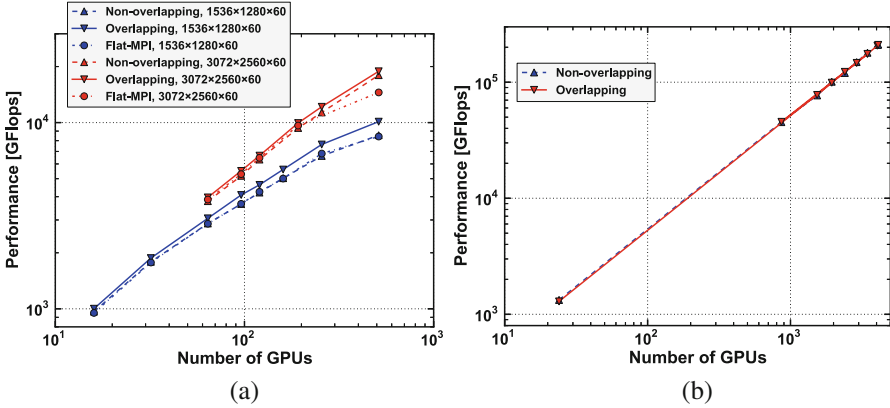


Fig. 5.7 Performance results for ASUCA. (a) Strong scaling. (b) Weak scaling

the BoundaryExchange class to write this communication. This class utilizes appropriate GPU-GPU communications such as GPUDirect peer-to-peer access with MPI.

The data communication time between GPUs is not ignored in the total execution time in the case of large-scale computation. The overlapping technique to hide communication overhead with computation can contribute to performance improvement. This framework provides kernel-division overlapping method reported in our previous work [45, 46]. This method exploits data independency within a single variable.

Figure 5.7a shows the performance of the framework-based ASUCA running on Tesla K20X GPUs on TSUBAME 2.5 and compares the strong scalability of the non-overlapping version and the overlapping version of ASUCA. We observe that the overlapping method works effectively to hide communication cost for both mesh sizes we expected, resulting in performance improvement. Using a $3,072 \times 2,560 \times 60$ mesh on 512 GPUs, the overlapping method achieves 18.9 TFlops. In weak scaling (Figure 5.7b), we achieve an extremely high performance of 209.6 TFlops using 4,108 GPUs with the overlapping method in single precision.

5.5.4 A Framework for High-Productivity GPU Acceleration Through Code Transformation Applied to Operational Weather Prediction Model in Japan

Numerical weather prediction (NWP) is steadily advancing toward increasingly high resolutions in order to allow accurate modeling of cloud-forming processes,

creating increasing demand for memory bandwidth among other hardware aspects. GPUs are very promising in this regard – their internal memory is highly performant compared to conventional approaches. On the other hand, numerical weather prediction is a very demanding environment with regard to operational requirements for the codebases in use – maintainability and ease-of-use for domain scientists are paramount for a successful deployment of new hardware architectures. While the performance of GPUs for earth system software has been studied extensively in related work, there is a gap in studies and frameworks where productivity plays a central role. Two technical challenges stand out in terms of their potential impact on productivity and code integrity, as many NWP codes have been developed mainly with CPUs in mind:

1. The granularity of physical processes tends to be too coarse for GPU.
2. The memory layout of data structures often needs to be reordered.

Existing approaches to solve these issues have led to varying combinations of limited performance, poor maintainability, and invasive rewrites. For ASUCA, a production regional-scale weather model used in operation at the Japan Meteorological Agency, an approach has been sought that comes with none of these drawbacks.

To meet these challenges, we introduce “Hybrid Fortran,” a new approach that allows high-performance GPGPU porting for data-parallel Fortran codes. This technique requires only minimal changes for a CPU targeted codebase, a significant advancement in terms of productivity. Parallel loops are abstracted through the means of a new language construct (parallel region), allowing to specify multiple granularity levels with the same code, depending on the target architecture (thus keeping support for CPU). Figure 5.8 gives an overview of this approach for

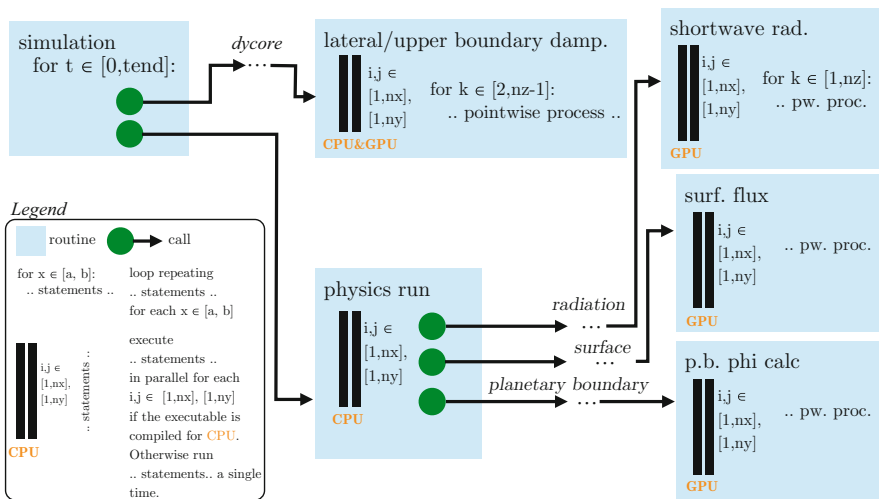


Fig. 5.8 ASUCA hybrid code structure with two granularity levels depending on target architecture

ASUCA. Memory layout is transformed at compile time, enabling the reuse of existing code with zero runtime overhead.

Hybrid Fortran has been successfully applied to both dynamical core and physical processes of ASUCA, a Japanese mesoscale weather prediction model with more than 150k lines of code. By using extensive code transformation techniques, more than 85% of the original codebase was left unchanged, and the overall code size has been extended by less than 4%. This shows very promising productivity, maintainability, and ease-of-adoption in an operational setting [34].

By means of a minimal weather application that resembles ASUCA's code structure, Hybrid Fortran is compared to both a performance model and today's commonly used method, OpenACC. As a result, the Hybrid Fortran implementation is shown to deliver the same or better performance than OpenACC, and its performance agrees with the model both on CPU and GPU. In a full-scale production run, using an ASUCA grid with $1581 \times 1301 \times 58$ cells and real-world weather data in 2km resolution, 24 NVIDIA Tesla P100 running the Hybrid Fortran based GPU port are shown to replace more than 50 18-core Intel Xeon Broadwell E5-2695 v4 running the reference implementation – an achievement comparable to more invasive GPGPU rewrites of other weather models. In terms of kernel performance, a speedup of more than $3\times$ is shown between latest-generation CPUs and GPUs (Reedbush-H), and a speedup of $4.9\times$ is shown on TSUBAME 2.5, comparing Kepler K20x GPUs to six-core Westmere CPUs [35].

5.6 Concluding Remarks

In this article, we presented an overview of our project that aimed to achieve both high performance and high productivity. In order to achieve our aim, we designed and developed high-level domain-specific frameworks that can automate many of tedious and complicated program optimizations for certain computation patterns. In particular, we focused on computational fluid dynamics and fast multipole method and first developed high-performance reference implementations that allowed us to learn key optimization techniques for large-scale heterogeneous systems. We then designed several domain-specific frameworks that automate the learned optimization techniques for the target application domains by exploiting scalable runtime system software.

We believe that we were able to demonstrate that achieving both high performance and high productivity is possible by specializing software stacks. It is becoming even more important given the uncertainty on future HPC hardware architecture beyond exascale computing. Obviously, however, it is not trivial to develop such domain-specific frameworks at all, although some of the common tasks can be accomplished by runtime software such as MassiveThreads. In order to realize high-performance and high-productive programming environments for a wider variety of application domains, the cost of developing frameworks must be reduced significantly. Another major challenge is to make frameworks into

professionally managed products. Domain-specific frameworks, by definition, have smaller user communities than general-purpose software, and they tend to end up as research demonstrations, including ours, rather than continuously maintained software products. We note that the two challenges are mutually related and think that addressing these challenges will become more important in the exascale and post-exascale era.

References

1. Akiyama, S., Taura, K.: Uni-address threads: scalable thread management for RDMA-based work stealing. In: Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing, HPDC'15, Portland, pp. 15–26 (2015)
2. Andoh, Y., Yoshii, N., Fujimoto, K., Mizutani, K., Kojima, H., Yamada, A., Okazaki, S., Kawaguchi, K., Nagao, H., Iwashita, K., Mizutani, F., Minami, K., Ichikawa, S., Komatsu, H., Ishizuki, S., Takeda, Y., Fukushima, M.: MODYLAS: a highly parallelized general-purpose molecular dynamics simulation program for large-scale systems with long-range forces calculated by fast multipole method (FMM) and highly scalable fine-grained new parallel processing algorithms. *J. Chem. Theory Comput.* **9**, 3201–3209 (2012)
3. Antoniu, G., Bougé, L., Namyst, R.: An efficient and transparent thread migration scheme in the PM2 runtime system. In: Proceedings of the 11 IPPS/SPDP'99 Workshops Held in Conjunction with the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing, San Juan, pp. 496–510 (1999)
4. Appel, A.W.: An efficient program for many-body simulation. *SIAM J. Sci. Stat. Comput.* **6**(1), 85–103 (1985)
5. Architecture Review Board: OpenMP application program interface version 3.0. Technical report (2008)
6. Bernaschi, M., Fatica, M., Melchionna, S., Succi, S., Kaxiras, E.: A flexible high-performance Lattice Boltzmann GPU code for the simulations of fluid flows in complex geometries. *Concurr. Comput. Pract. Exp.* **22**(1), 1–14 (2010)
7. Blumofe, R.D., Leiserson, C.E.: Scheduling multithreaded computations by work stealing. *J. ACM* **46**(5), 720–748 (1999)
8. Carrier, J., Greengard, L., Rokhlin, V.: A fast adaptive multipole algorithm for particle simulations. *SIAM J. Sci. Stat. Comput.* **9**(4), 669–686 (1988)
9. Choi, C.H., Ivanic, J., Gordon, M.S., Reudenberg, K.: Rapid and stable determination of rotation matrices between spherical harmonics by direct recursion. *J. Chem. Phys.* **111**(19), 8825–8831 (1999)
10. Dachsel, H.: Fast and accurate determination of the Wigner rotation matrices in the fast multipole method. *J. Chem. Phys.* **124**, 144115 (2006)
11. Darve, E., Cecka, C., Takahashi, T.: The fast multipole method on parallel clusters, multicore processors, and graphics processing units. *Comptes Rendus Mécanique* **339**, 185–193 (2011)
12. Dehnen, W.: A hierarchical $O(N)$ force calculation algorithm. *J. Comput. Phys.* **179**(1), 27–42 (2002)
13. Dinan, J., Brian Larkins, D., Sadayappan, P., Krishnamoorthy, S., Nieplocha, J.: Scalable work stealing. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC '09, Portland, pp. 53:1–53:11 (2009)
14. Dubinski, J.: A parallel tree code. *New Astron.* **1**, 133–147 (1996)
15. Fortin, P.: Multipole-to-local operator in the fast multipole method: comparison of FFT, rotations and BLAS improvements. Technical Report RR-5752, Rapports de recherche, et theses de l'Inria (2005)

16. Fortin, P.: High performance parallel hierarchical algorithm for N-body problems. Ph.D. thesis, Universite Bordeaux 1 (2007)
17. Frigo, M., Leiserson, C.E., Randall, K.H.: The implementation of the Cilk-5 multithreaded language. In: Proceedings of the ACM SIGPLAN 1998 Conference on Programming Language Design and Implementation, PLDI '98, Montreal, pp. 212–223 (1998)
18. Fukuda, K., Matsuda, M., Maruyama, N., Yokota, R., Taura, K., Matsuoka, S.: Tapas: an implicitly parallel programming framework for hierarchical n-body algorithms. In: 22nd IEEE International Conference on Parallel and Distributed Systems, ICPADS 2016, Wuhan, China, 13–16 Dec 2016, pp. 1100–1109 (2016)
19. Germano, M., Piomelli, U., Moin, P., Cabot, W.H.: A dynamic subgrid-scale eddy viscosity model. *Phys. Fluids A Fluid Dyn.* **3**(7), 1760–1765 (1991)
20. Grama, A.Y., Kumar, V., Sameh, A.: Scalable parallel formulations of the Barnes-Hut method for N-body simulations. In: Proceedings of the 1994 ACM/IEEE Conference on Supercomputing, Washington, DC, pp. 1–10 (1994)
21. Gumerov, N.A., Duraiswami, R.: Fast multipole methods on graphics processors. *J. Comput. Phys.* **227**, 8290–8313 (2008)
22. Hiraishi, T., Yasugi, M., Umatani, S., Yuasa, T.: Backtracking-based load balancing. In: Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP '09, Raleigh, pp. 55–64 (2009)
23. Iwasaki, S., Taura, K.: A static cut-off for task parallel programs. In: Proceedings of the 2016 International Conference on Parallel Architectures and Compilation, PACT '16, pp. 139–150. ACM, New York (2016)
24. Kobayashi, H., Ham, F., Wu, X.: Application of a local SGS model based on coherent structures to complex geometries. *Int. J. Heat Fluid Flow* **29**(3), 640–653 (2008). The Fifth International Symposium on Turbulence and Shear Flow Phenomena (TSFP5), Munich
25. Křivánek, J., Kontinen, J., Pattanaik, S., Bouatouch, K.: Fast approximation to spherical harmonic rotation. Technical Report 1728, Institut De Recherche En Informatique Et Systemes Aleatoires (2005)
26. Lange, B., Fortin, P.: Parallel dual tree traversal on multi-core and many-core architectures for astrophysical N-body simulations. Technical Report hal-00947130, Sorbonne Universités UPMC (2014)
27. Lashuk, I., Chandramowlishwaran, A., Langston, H., Nguyen, T.-A., Sampath, R., Shringarpure, A., Vuduc, R., Ying, L., Zorin, D., Biros, G.: A massively parallel adaptive fast multipole method on heterogeneous architectures. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, Portland (2009)
28. Lessig, C., de Witt, T., Fiume, E.: Efficient and accurate rotation of finite spherical harmonics expansions. *J. Comput. Phys.* **231**, 243–250 (2012)
29. Makino, J.: Comparison of two different tree algorithms. *J. Comput. Phys.* **88**, 393–408 (1990)
30. Makino, J.: A fast parallel treecode with GRAPE. *Publ. Astron. Soc. Jpn.* **56**, 521–531 (2004)
31. Maruyama, N., Nomura, T., Sato, K., Matsuoka, S.: Physics: an implicitly parallel programming model for stencil computations on large-scale GPU-accelerated supercomputers. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11, Seattle (2011)
32. Min, S.-J., Iancu, C., Yelick, K.: Hierarchical work stealing on manycore clusters. In: Fifth Conference on Partitioned Global Address Space Programming Models, PGAS '11, Galveston Island (2011)
33. Mohr, E., Kranz, D.A., Halstead, Jr. R. H.: Lazy task creation: a technique for increasing the granularity of parallel programs. *IEEE Trans. Parallel Distrib. Syst.* **2**, 264–280 (1991)
34. Müller, M., Aoki, T.: Hybrid fortran: high productivity GPU porting framework applied to Japanese weather prediction model. In: WACCPD: Accelerator Programming Using Directives 2017, pp. 20–41. Springer (2018)
35. Müller, M., Aoki, T.: New high performance GPGPU code transformation framework applied to large production weather prediction code (2018). Preprint as accepted for ACM TOPC

36. Nakashima, J., Nakatani, S., Taura, K.: Design and implementation of a customizable work stealing scheduler. In: Proceedings of the 3rd International Workshop on Runtime and Operating Systems for Supercomputers, ROSS '13, Eugene, pp. 9:1–9:8 (2013)
37. Ohnuki, S., Chewl, W.C.: Error minimization of multipole expansion. *SIAM J. Sci. Comput.* **26**(6), 2047–2065 (2005)
38. Ohori, A., Taura, K., Ueno, K.: Making SML# a general-purpose high-performance language. In: *ML Family Workshop*, Oxford (2017)
39. Pharr, M., Mark, W.R.: ISPC: a SPMD compiler for high-performance CPU programming. In: 2012 Innovative Parallel Computing (InPar), San Jose, pp. 1–13, May 2012.
40. Rahimian, A., Lashuk, I., Veerapaneni, S., Chandramowlishwaran, A., Malhotra, D., Moon, L., Sampath, R., Shringarpure, A., Vetter, J., Vuduc, R., Zorin, D., Biros, G.: Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures. In: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC '10, New Orleans, pp. 1–11 (2010)
41. Rankin, W.T.: Efficient parallel implementations of multipole based N-body algorithm. Ph.D. thesis, Duke University (1999)
42. Reinders, J.: *Intel Threading Building Blocks*, 1st edn. O'Reilly & Associates, Inc., Sebastopol (2007)
43. Salmon, J.K.: *Parallel Hierarchical N-Body Methods*. Ph.D. thesis, California Institute of Technology (1991)
44. Seo, S., Amer, A., Balaji, P., Bordage, C., Bosilca, G., Brooks, A., Carns, P., Castelló, A., Genet, D., Hérault, T., Iwasaki, S., Jindal, P., Kalé, L.V., Krishnamoorthy, S., Lifflander, J., Lu, H., Meneses, E., Snir, M., Sun, Y., Taura, K., Beckman, P.: Argobots: a lightweight low-level threading and tasking framework. *IEEE Trans. Parallel Distrib. Syst.* **29**(3), 512–526 (2018)
45. Shimokawabe, T., Aoki, T., Ishida, J., Kawano, K., Muroi, C.: 145 TFlops performance on 3990 GPUs of TSUBAME 2.0 supercomputer for an operational weather prediction. *Proc. Comput. Sci.* **4**, 1535–1544 (2011)
46. Shimokawabe, T., Aoki, T., Muroi, C., Ishida, J., Kawano, K., Endo, T., Nukada, A., Maruyama, N., Matsuoka, S.: An 80-fold speedup, 15.0 TFlops full GPU acceleration of non-hydrostatic weather model ASUCA production code. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, New Orleans, pp. 1–11 (2010)
47. Shimokawabe, T., Aoki, T., Onodera, N.: A high-productivity framework for multi-GPU computation of mesh-based applications. In: *HiStencils 2014*, Vienna, p. 23 (2014)
48. Shimokawabe, T., Aoki, T., Onodera, N.: High-productivity framework on GPU-rich supercomputers for operational weather prediction code ASUCA. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '14, New Orleans, pp. 251–261 (2014)
49. Shimokawabe, T., Takaki, T., Endo, T., Yamanaka, A., Maruyama, N., Aoki, T., Nukada, A., Matsuoka, S.: Peta-scale phase-field simulation for dendritic solidification on the TSUBAME 2.0 supercomputer. In: 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Seattle, pp. 1–11 (2011)
50. Singh, J.P., Holt, C., Hennessy, J.L., Gupta, A.: A parallel adaptive fast multipole method. In: Proceedings of the Supercomputing Conference 1993, Portland, pp. 54–65 (1993)
51. Solomonik, E., Kalé, L.V.: Highly scalable parallel sorting. In: *IEEE International Symposium on Parallel and Distributed Processing*, Rio de Janeiro, pp. 1–12 (2010)
52. Takahashi, T., Cecka, C., Fong, W., Darve, E.: Optimizing the multipole-to-local operator in the fast multipole method for graphical processing units. *Int. J. Numer. Methods Eng.* **89**, 105–133 (2012)
53. Taura, K., Nakashima, J., Yokota, R., Maruyama, N.: A task parallel implementation of fast multipole methods. In: *Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*, Salt Lake City (2012)

54. Wahib, M., Maruyama, N.: Scalable kernel fusion for memory-bound GPU applications. In: ACM/IEEE Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'14), New Orleans (2014)
55. Wahib, M., Maruyama, N.: Automated GPU kernel transformations in large-scale production stencil applications. In: ACM Conference on High Performance and Distributed Computing (HPDC'15), Portland (2015)
56. Wahib, M., Maruyama, N.: Data-centric GPU-based adaptive mesh refinement. In: Workshop on Irregular Applications: Architectures and Algorithms (IA3 2015), Austin (2015)
57. Wahib, M., Maruyama, N., Aoki, T.: Daino: a high-level framework for parallel and efficient AMR on GPUs. In: ACM/IEEE Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'16), Salt Lake City (2016)
58. Warren, M.S., Salmon, J.K.: A parallel hashed OCT-tree N-body algorithm. In: Proceedings of the 1993 ACM/IEEE Conference on Supercomputing, Portland, pp. 12–21 (1993)
59. Warren, M.S., Salmon, J.K.: A portable parallel particle program. *Comput. Phys. Commun.* **87**, 266–290 (1995)
60. Wheeler, K.B., Murphy, R.C., Thain, D.: Qthreads: an API for programming with millions of lightweight threads. In: 2008 IEEE International Symposium on Parallel and Distributed Processing, IPDPS '08, pp. 1–8 (2008)
61. Xian, W., Takayuki, A.: Multi-GPU performance of incompressible flow computation by Lattice Boltzmann method on GPU cluster. *Parallel Comput.* **37**(9), 521–535 (2011). *Emerging Programming Paradigms for Large-Scale Scientific Computing*
62. Yokota, R.: An FMM based on dual tree traversal for many-core architectures. *J. Algorithms Comput. Technol.* **7**(3), 301–324 (2013)
63. Yokota, R., Barba, L.A.: A tuned and scalable fast multipole method as a preeminent algorithm for Exascale systems. *Int. J. High Perform. Comput. Appl.* **26**(4), 337–346 (2012)
64. Yokota, R., Turkiyyah, G., Keyes, D.: Communication complexity of the fast multipole method and its algebraic variants. *Supercomput.* *Front. Innov.* **1**(1), 63–84 (2014)
65. Yu, H., Girimaji, S.S., Luo, L.-S.: DNS and LES of decaying isotropic turbulence with and without frame rotation using Lattice Boltzmann method. *J. Comput. Phys.* **209**(2), 599–616 (2005)
66. Zhang, B.: Asynchronous task scheduling of the fast multipole method using various runtime systems. In: Proceedings of the Forth Workshop on Data-Flow Execution Models for Extreme Scale Computing, Edmonton (2014)
67. Zima, H.P., Callahan, D., Chamberlain, B.L.: The cascade high productivity language. In: International Workshop on High-Level Programming Models and Supportive Environments, Santa Fe, pp. 52–60 (2004)

Chapter 6

System Software for Data-Intensive Science



Osamu Tatebe, Yoshihiro Oyama, Masahiro Tanaka, Hiroki Ohtsuji, Fuyumasa Takatsu, and Xieming Li

Abstract The storage performance is an issue for supercomputers to facilitate the data-intensive science. To improve the storage bandwidth according to the number of compute nodes, we assume a node-local scale-out storage architecture. The number of local storages increases according to the number of compute nodes, and the total storage bandwidth increases scalably. Our research target is a distributed file system in the node-local storage architecture, an operating system for compute node, and runtime systems for the distributed file system using node-local storages for workflow systems, MapReduce, MPI-IO, and batch job schedulers.

6.1 Distributed File System

CPU performance of supercomputers has been improved toward Exaflops. However, the speed of storage performance improvement is slow, and the performance gap between CPU and storage is growing wider and wider. The performance improvement of most applications is limited. To fill the gap, an innovation in computer systems is required. To improve the storage bandwidth according to the number of compute nodes, we assume a node-local scale-out storage architecture. The number of local storages increases according to the number of compute nodes,

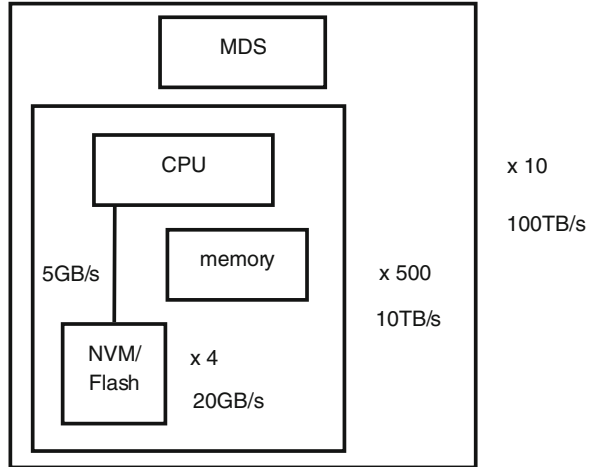
O. Tatebe (✉) · Y. Oyama
University of Tsukuba, Tsukuba, Ibaraki, Japan
e-mail: tatebe@cs.tsukuba.ac.jp; oyama@cs.tsukuba.ac.jp

M. Tanaka
Keio University, Fujisawa, Kanagawa, Japan
e-mail: masa16.tanaka@keio.jp

H. Ohtsuji
University of Tsukuba (Currently Fujitsu Laboratories Ltd.), Kawasaki, Kanagawa, Japan
e-mail: ohtsuji.hiroki@jp.fujitsu.com

F. Takatsu · X. Li
Yahoo Japan Corporation, Chiyoda-ku, Tokyo, Japan

Fig. 6.1 A node-local scale-out storage architecture



and the total storage bandwidth increases scalably. As a general rule of thumb, well-balanced 100 PFlops supercomputers require 100 TB/s of the storage bandwidth. An example of system configuration is shown in Fig. 6.1. Each node has 4 NVMe SSDs that provide 20 GB/s of storage bandwidth in total. A file system metadata server manages 500 nodes. Ten sets of them provide 100 TB/s of the storage performance.

In this storage architecture, file data is stored in a local storage. The file location and file metadata such as a modification date and an access control list are stored in a metadata server. Separate management of file metadata and file data makes it possible to directly access file data from a client in parallel, which helps to improve storage access performance.

Existing systems having the node-local scale-out storage architecture include Google file system [5], Hadoop distributed file system [6], and Gfarm file system [27]. Google file system and Hadoop distributed file system are for MapReduce [3] processing; on the other hand, Gfarm file system is a POSIX-compliant file system. There are several issues to realize a 100 TB/s node-local scale-out storage system. One issue is the performance and the availability of file system metadata servers. All existing systems utilize a single metadata server that has a limitation of the metadata performance, which can typically manage up to thousands of compute nodes. Regarding the availability, Gfarm file system has a slave metadata server to prepare for a failover when a master metadata server fails. The metadata server should not be a single point of failure.

Another issue is the availability of the file data. When using the node-local storage, the file data stored on that node cannot be accessed if it fails. In the worst case, the file data is lost. To prevent the data lost or increase the degree of the availability, files are often replicated among nodes. The file replication requires two times more capacity of storage when three copies are created. Erasure coding helps to reduce additional capacity, but there is overhead to encode, which may degrade the write performance. How to reduce this performance penalty is another problem.

New device innovation such as a nonvolatile memory and a flash device requires redesign of storage system software since how to improve the performance is different. To improve the performance, hard disk drives require sequential access of blocks, while a flash device and a nonvolatile memory require parallel accesses or non-blocking accesses to hide access latency.

In the node-local distributed storage systems, access performance depends on the location of files. A local storage provides better performance than a remote storage. Exploiting locality of storage access is critical to improve the storage performance. This is an issue for runtime systems to allocate processes on a node that stores the input files.

6.1.1 File System Metadata Server

To achieve scalable performance for file operations using multiple metadata servers, the metadata management, especially hierarchical namespace management, is critically important. Traditional local file system uses a directory entry that is an array or a tree of an entry name and the inode number to manage a hierarchical namespace. This design is not appropriate for concurrent operations since it is challenging to manage in distributed servers and to modify it in parallel.

To avoid this difficulty, we design a PPMDS metadata server. A directory entry does not manage a list of entries. Instead, it manages a server list that stores inode entries in the directory. Entries in the directory are distributed across multiple servers in the server list. Each entry can be accessed, updated, created, and removed in parallel unless these are conflicting operations for the same entry.

This design can eliminate unnecessary blocking regarding file operations when they are not conflicted. Directory operations that require to modify multiple entries consistently can be supported by a non-blocking transaction efficiently [7]. This design eliminates a global lock or a distributed lock in a directory level and ensures to process concurrent operations in parallel unless they are conflicting operations.

The traditional file system is managed by the inode data structure, which is optimized for a block device in a local machine. It is not suitable for distributed data management since a directory entry is managed by a single inode. File system metadata consists of the following two types of data:

- file metadata for each inode entry such as a list of block locations, permission and timestamps, and
- hierarchical namespace.

Hierarchical namespace is managed by directory entries in the traditional file system, while it has several problems in distributed management and parallel updates.

In PPMDS, the file system metadata is managed by key-value pairs. Each key-value pair is an inode entry. The file metadata is stored as a value of a key-value pair. Key is a pair of a parent inode number and an entry name. Inode entry is identified by a pair of a parent inode number and an entry name. This representation is also used by TableFS [17].

This key-value metadata data structure is easily distributed among multiple servers. Each metadata server has its own key-value store. Inode entries in the directory are distributed using a hash function among servers.

6.1.2 Object Storage

OpenNVM [4] is a proposed standard interface for a flash storage. It includes a sparse address space and atomic batch write operations, which can greatly improve the flash storage performance. The sparse space address is similar to a logical address in memory space. Only written blocks are physically allocated in the flash storage. The atomic batch write operations can write or trim several I/O blocks or I/O vectors atomically.

Using these operations, an object storage can be designed by a straightforward array of fixed-sized regions. Region size is large enough to fit the maximum size of objects. In this case, each object can be accessed directly using the region number. Besides, atomic batch operations can be used for batch initialization, which improves the object creation performance. Our PPOST implementation [22] shows 12 times speedup compared with the DirectFS [9] and achieves 740,000 file creations per second.

6.1.3 Active Storage for Cluster-Wide RAID

To increase the degree of availability, file replications are often used, while they waste the capacity. RAID or erasure coding can reduce the additional capacity, but there is an overhead to encode. RAID encoding is usually done using a RAID controller in a storage server, while it is not a case for a cluster-wide RAID spread among storage servers.

We proposed an active storage mechanism for a cluster-wide RAID to avoid the encoding overhead [13]. In a RAID controller model, a client calculates parity blocks and sends data and parity blocks to storage servers, which consume additional CPU cycles in a client and additional network traffic for parity blocks. Active storage mechanism calculates parity blocks at storage server side. The client sends only data blocks not parity blocks. A storage server sends a data block to a storage server to calculate a parity block. It is possible to construct a pipeline stage for this data transfer and the calculation of the parity block, which can hide the parity calculation overhead. A zero-overhead RAID-5 pipeline is demonstrated in [13].

6.1.4 PPFS Scale-Out Distributed File System

To demonstrate a scale-out distributed file system, we design a PPFS file system using PPMS distributed metadata server and PPOST object storage [23]. A

naive implementation suffers from overhead of communication among a client and a server and among servers. To reduce the overhead, a bulk creation and an object prefetching are proposed. The bulk creation creates N empty files at one time, which are cached at the metadata server. This reduces the number of file creation requests to $1/N$. The object prefetching hides the latency of communication between servers. It introduces a helper thread to create objects beforehand and hides the communication latency. Using these techniques, PPFS file creation performance achieves 138,000 file creations per second, which is 2.5 times better than the IndexFS [18].

6.2 Compute-Node Operating System for Data-Intensive Computing

System software for compute nodes is critical for maximizing the performance of a distributed file system in supercomputers. Even if the I/O servers of the distributed file system are fully optimized, runtime overheads on I/O operations on compute nodes can significantly decrease the overall performance of applications. Based on this observation, we developed fundamental technologies for compute-node operating systems that can maximize the performance of data-intensive applications running with distributed file systems. The distributed file system used in this research is Gfarm [27].

Various results have been obtained. First, we developed an RDMA-based mechanism to transfer file data between clients and servers of a distributed file system. The mechanism helps minimize the number of data copies and context switches occurring in I/O operations. Second, we developed technologies to efficiently manage cache of file data on compute nodes. The technologies include deduplication of file data and cooperative caching. Finally, we developed a method to reduce operating system jitter on compute nodes due to high load of I/O operations. The following part of this section presents the results.

6.2.1 *RDMA-Based File Data Transfer in Distributed File Systems*

We developed a mechanism that enables the client interface (kernel driver) of a distributed file system to quickly transfer file data to or from I/O servers. The key building block of the mechanism is the remote direct memory access (RDMA), which is provided by a high-speed communication mechanism InfiniBand. The RDMA allows a node to directly read or write the memory of other nodes connected with the InfiniBand. Nodes using RDMA can transfer data faster than those using packet-based communication such as Ethernet. A TCP/IP communication is still

possible on top of InfiniBand owing to the IPoIB (IP over IB) communication mechanism. However, the use of TCP/IP results in significant runtime overhead because of the operations of the TCP/IP protocol.

A conventional method of accelerating file accesses from the client interface of a distributed file system involves using a kernel driver specialized for the distributed file system. This helps reduce the number of context switches between the processes, context switches between the kernel level and user level, and memory copies of file data, when compared to methods using user-level client interface such as FUSE. In the past, the Gfarm project developed a kernel driver specialized for Gfarm, achieving faster file accesses than FUSE-based client interface, which has been conventionally used with Gfarm. However, even after the introduction of the kernel driver, the operating system kernel of the client nodes still executes redundant memory copies.

We attempted to eliminate the redundant memory copies using InfiniBand RDMA. The conventional method first receives file data from the server in its communication buffer, and then the operating system kernel copies the data to a page-cache area. A page-cache area is a kernel-level memory area on which the kernel stores the file data read from the storage. Figure 6.2 shows the data flow in the conventional method (blue dotted lines) and in the mechanism developed in this study (red dotted line). The developed mechanism directly transfers file data from the I/O server to a page-cache area of a client node without memory copy. We implemented the mechanism and integrated it into Gfarm. The elimination of memory copies in the mechanism reduces the consumption of CPU resource, which is extremely valuable for compute nodes. To the best of our knowledge, this research is the first to implement an RDMA-based mechanism whereby client nodes directly receive file data in page-cache areas.

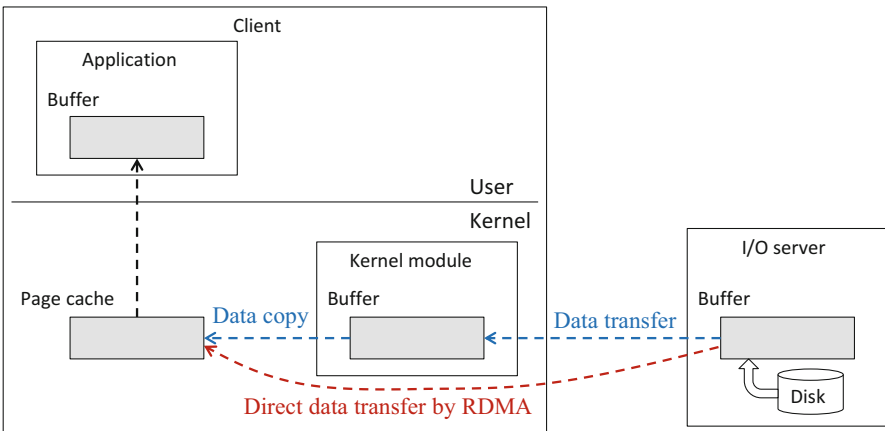


Fig. 6.2 Flows of data in a conventional method and with the developed mechanism

Table 6.1 Bandwidth of sequential read

Buffer size	User-level daemon	User-level daemon + RDMA	Proposed mechanism
64 KiB	306.6	511.4	907.8
1024 KiB	321.1	509.8	909.6

(MiB/s)

Table 6.2 Execution times of the Montage workflow application

User-level daemon	User-level daemon + RDMA	Proposed mechanism
983.4 s	984.6 s	883.3 s

We measured the basic performance of the mechanism using the IOR benchmark, which is widely used to evaluate I/O performance of parallel file systems. We measured the bandwidth of the sequential file read after loading the file data on the memory of the I/O server. Table 6.1 lists the result. The label “User-level daemon” indicates the use of the original FUSE-based client interface. The label “User-level daemon + RDMA” indicates the use of the client interface extended with a simple RDMA extension, with which file data are not transferred to a page-cache area but are rather received in an RDMA communication buffer and then copied to a buffer in the user-level daemon. The buffer sizes in the table indicates the buffer sizes of the file I/O operations. Under the condition of 1024 KiB buffer size, the developed mechanism achieved 78.4% improvement in the bandwidth when compared to the simply extended client interface.

In addition, we evaluated the improvements in the performances of several real-world scientific applications. Table 6.2 lists the result of the Montage workflow application, which synthesizes astronomical images with many file I/O operations. The developed mechanism achieved faster execution than the FUSE-based user-level client interface and the original kernel driver. The details of the experiments are presented in our previous paper [20].

6.2.2 Efficient File Cache Management Using Deduplication

We developed an efficient cache management mechanism for client nodes of a distributed file system. When introducing a caching mechanism, the resulting performance of not only the file I/O itself but also of the applications is crucial. Therefore, in general, caching mechanisms should refrain from consuming too much memory because memory resource is valuable for applications, and a high memory pressure from caching can have a negative impact on the performance of applications.

To enable the mechanism to achieve both high cache hit rate and moderate memory consumption, we employed the deduplication technique, which is used in many file systems to reduce the footprints of memory and storage.

Although the major application of deduplication is to back up large-scale storage, we believe that deduplication would be useful for data-intensive high-performance computing because it has significant potential to improve the performance of file I/O and applications. Many applications in data-intensive science involve reading and writing of extremely large files, the data of which are loaded on the memory of client nodes, thus significantly reducing the amount of free memory shared with applications. However, only a few studies have been published so far concerning the experimental evaluation of the effect of deduplication on file systems used in data-intensive high-performance computing.

We developed a cache management mechanism based on deduplication, thereby eliminating the operations and footprints for the redundant parts of files.

A straightforward approach of creating cache on client nodes involves saving an entire copy of an accessed file. However, this approach has two disadvantages. One is that the amount of memory or storage required for cache is largely equal to the total size of the accessed files and can become considerable. The other is that the cache management cost can increase for certain file I/O workload. In this approach, an entire copy must be updated or discarded each time the corresponding file is updated, irrespective of the number of updates. It should be noted that files in a distributed file system can be accessed simultaneously by multiple client nodes. The effect of caching in this approach becomes negligible in applications wherein a small part of a large file is frequently updated.

Another approach is to create cache for each block of a file. We employed this approach because it addresses the abovementioned disadvantages.

In the developed mechanism, a client divides file data into variable-length blocks (we refer to them as *chunks*) and creates a cache of chunks in the memory of the self-node. The cache is used in future accesses to the file data. The mechanism deduplicates the chunks and calculates the SHA-1 hash values of the created chunks and merges multiple chunks that have the same hash value into one. A single chunk can be a partial cache of different files or a partial cache of different parts of a file. Cache is also utilized in the transfer of files that a client accesses for the first time. If a file contains chunks, the contents of which are the same as those that have already been created on the client, the client reuses the chunks without transferring the file data corresponding to the chunks.

A main characteristic of the mechanism is the aggressive use of the content-defined chunking (CDC) method, using which the boundaries of the chunks are determined on the basis of their contents. In particular, the method is used to calculate the fingerprints of partial file data in a fixed-length window by sliding the window byte to byte from the beginning of a file to the end. If the fingerprint satisfies a certain condition, a boundary is placed just after the present position of the window. This method of boundary setting allows to deduplicate common byte sequences that appear in arbitrary file offsets. To the best of our knowledge, there has been no research in which a CDC-based deduplication is applied to the client-node cache of a distributed file system.

The developed mechanism reduces memory consumption and local storage for cache and the amount of data transferred between I/O servers and clients. In addition, it can decrease the number of cache updates or discards required when a file is updated, because the unit of caching is not the entire files but rather chunks.

Deduplication operations themselves incur runtime overheads, and the mechanism is not an exception. If the advantages of deduplication are more than the disadvantages, the application performance can be improved. However, otherwise, deduplication degrades the performance. Hence, we integrated two techniques to minimize deduplication overhead. One is to asynchronously update the table for managing chunks, and the other is the parallel execution of file data manipulation and hash-value computation. The details regarding the same are given in our previous paper [16].

We evaluated the effect of the mechanism through experiments using the IOR benchmark. The result shows that it achieved up to 9.2 times speedup in random read of file data. In addition, we evaluated the rate of data reduction due to deduplication. The result shows that it achieved over 30% data reduction in all cases of various average chunk sizes. The details of the experiments are presented in our previous paper [16]. We believe that the results are useful to understand, predict, and analyze the effect of deduplication on distributed file systems in data-intensive high-performance computing.

6.2.3 Cooperative Caching Using Memory of Compute Nodes

In addition to deduplication-based caching, we studied the effect of cooperative caching [2] for data-intensive high-performance computing. Cooperative caching is a technique with which client nodes cooperatively share the cache of files in a distributed file system. If a client fails to find cache in a local node, the technique attempts to obtain the cache from other client nodes as well as from I/O servers. Clients exchange cache between them by sending and receiving cache requests. Cooperative caching is advantageous if the communication between clients is fast and if the cache miss penalty, including the cost of storage accesses in an I/O server, is considerable.

Although cooperative caching has a long history, most previous studies assumed computing environments that are quite different from those of modern environments for data-intensive computing. For example, the assumed computers and network devices were much slower, not to mention the unavailability of InfiniBand RDMA at the time these studies were conducted. As a result, the effectiveness of cooperative caching in modern computing environments is quite unclear.

Based on the observation, we implemented a cooperative caching mechanism for Gfarm and clarified its effectiveness through experiments. We assumed an environment in which all the nodes of the compute nodes and I/O servers are

connected with InfiniBand. Clients using the mechanism share the cache of files in the Gfarm distributed file system. We extended Gfarm code so that clients could send cache requests to other clients and could accept and handle such requests. File data are transferred with RDMA between clients and between a client and an I/O server. If a client finds requested data in the local node, it transfers them directly from the memory of the self-node to the memory of another client node without redundant memory copies.

The cooperative caching mechanism associates the metadata of each file with cache placement information, which is a list of client nodes that is likely to maintain the required part of the file being accessed.

The possible methods of information management are classified into two. The first is to manage information in a distributed manner using distributed data structures such as distributed hash tables. However, methods in this class have a problem of increased access latencies because of many hops in communication, which cannot be neglected in high-performance computing. The other class is to manage information in a centralized manner, e.g., by using a central server managing all cache placement information for an entire distributed file system. The methods in this class have many advantages including high expected accuracy and low expected latency, though they have a disadvantage in that the server becomes a bottleneck when cache placement information is frequently requested or updated.

We chose the centralized method and extended the Gfarm metadata server with the function of managing cache placement information. When a client receives the cache of file data from another node, it provides the information to the central server. The central server integrates the information sent by clients to maintain the statistics on the number of cache requests sent by each client and the number of cache transfers that was actually performed. When a client inquires the central server of a client that is likely to have the desired cache, the central server answers the client that is most likely to have one.

We evaluated the cooperative caching mechanism through experiments using three scientific applications: Montage, SDFRED, and NGS Analyzer-MINI. Table 6.3 lists the result of Montage. The label “Kernel module + RDMA” in the table indicates the use of our RDMA-based kernel module for handling file I/O operations instead of a user-level daemon. The label “Kernel module + RDMA + Cooperative caching” indicates the use of the kernel module extended with cooperative caching. We confirmed that the mechanism achieved a performance improvement of 5.8% for Montage. The highest hit rate among all applications on all clients was 97.3%, and the average hit rates were always greater than 60%. The details of the experiments are presented in our previous paper [19].

Table 6.3 Execution times of the Montage workflow application

Original	Kernel module + RDMA	Kernel module + RDMA + cooperative caching
4745.5 s	4421.8 s	4156.2 s

6.2.4 Reduction in the Operating System Jitter Because of File Accesses

Operating system jitter (OS jitter) is a well-known factor that slows down applications running on a compute-node operating system. OS jitters are operations performed by the operating system kernel or system daemons that affect the application performance. It has been said from a long time in high-performance computing that reducing OS jitter is crucial for achieving higher performance.

To address the problem of OS jitter, we quantified the impact of OS jitter on high-performance scientific applications and developed a method to minimize performance degradation due to OS jitter. Although there are many types of OS jitters, we focus on a type of OS jitter that is caused by file system operations. First, we simultaneously ran a scientific computing application and a program that executes many file I/O operations and measured the performance of the application. The result shows that the performance was seriously degraded. This is because a huge amount of memory was consumed for file cache because of the file I/O operations. When physical memory runs out, a system daemon called *kswapd* starts to reclaim the memory from applications and file cache. The daemon is usually asleep but awakens when the amount of free memory decreases. We identified that the *kswapd* daemon and other daemons, such as the migration daemon, consumed a large amount of CPU time. This implies that a high load on a file system can significantly degrade the performance of scientific applications. This problem becomes particularly serious in data-intensive high-performance computing.

To address the abovementioned problem, we developed a mechanism that minimizes the runtime overhead due to the memory-reclaiming daemon *kswapd*. We solved the problem by extending the page reclaim mechanism in the operating system kernel with a kernel module. The kernel module starts to reclaim pages before the number of free pages decreases below a certain threshold used by the *kswapd* daemon. The mechanism is awakened before *kswapd* because its threshold is set to a value greater than that of the *kswapd*. Users can keep a small number of invocations of the mechanism by having it reclaim numerous pages in one invocation.

As mentioned above, the mechanism provides two critical configuration parameters: (1) the threshold of the number of free pages and (2) the maximum number of pages reclaimed in one invocation. Users can customize both the parameters according to their environments, conditions, or requirements. The mechanism is invoked typically when a large amount of memory is consumed for page cache, whereas it is not invoked when a large amount of memory is consumed for ordinary memory objects such as data structures created by applications. When memory is consumed mainly for memory objects, reclaiming many pages is likely to degrade the performance of applications that are concerned with memory objects. The mechanism determines the present status of memory usage based on the frequency of disk accesses. If the number of disk reads in unit time exceeds a threshold, the mechanism determines that the memory is consumed mainly for page cache.

The operating system kernel provides several parameters to control the behavior of kswapd. Users can reduce the impact of OS jitter by changing the parameters. However, it is quite unclear whether the kswapd or the developed mechanism is more effective. Moreover, it is difficult to identify optimum parameter values that minimize the impact of OS jitter. We surmise that inappropriate values of the parameters will have only a small effect on OS jitter. Experiments are needed to clear these ambiguous points.

We evaluated the mechanism through experiments, in which we simultaneously ran a weather forecast application WRF and a program that repeatedly invokes file I/O operations. We executed WRF on seven compute nodes in parallel, using 11 cores on each node. We measured the changes in the application performance considering the presence of the mechanism. In addition, we measured the performance changes observed when varying the parameter values of the kswapd daemon.

Figure 6.3 shows the execution times elapsed for each computation step of the WRF. The label “WRF” indicates that only WRF is run. The label “WRF + Jitter” indicates that WRF and jitter-generating programs without the mechanism are run. The label “WRF + Jitter + Proposed” indicates that the WRF and jitter-generating programs with the mechanism are run. The best performance was obtained when the proposed mechanism was used. The execution times significantly increased because of the OS jitter, whereas the increase in the execution time when using the proposed mechanism was negligible. Although varying the kswapd parameters improved the performance, it could not achieve as high a performance as that using mechanism. We also found that a certain level of skill and experience was required to provide optimum parameter values. The details of the experiments are described in our previous papers [14, 15].

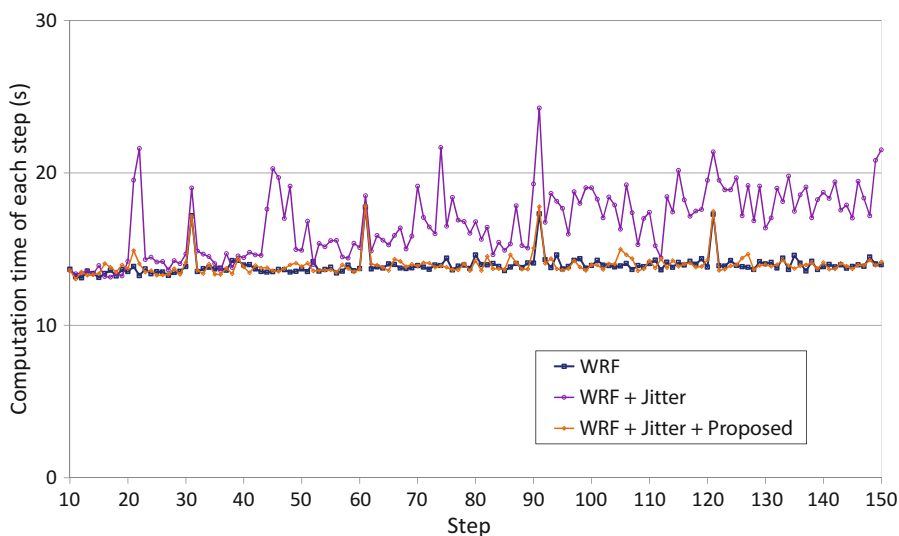


Fig. 6.3 Execution time required for each step of WRF

6.3 Pwraake Workflow System

Pwraake [24] is a parallel and distributed workflow system developed for data-intensive, many-task computing on computer clusters. Pwraake is implemented as an extension for on Rake (Ruby Make). Pwraake inherits a workflow language from Rake. Rake has a powerful workflow language since it has various useful features such as mapper rules and capability to write scripts in the Ruby language. Pwraake has features to utilize Gfarm file system [27] for parallel I/O performance. Gfarm file system has mechanisms to exploit local I/O performance, e.g., (a) select a close replica when reading a file replicated to multiple nodes, and (b) select local storage when creating a new file to maximize write performance. However, the performance of reading input files depends on the task scheduling. The following two scheduling methods are developed for Pwraake:

- Locality-aware scheduling using multi-constraint graph partitioning [25].
- Disk cache-aware scheduling mitigating trailing task problem [26].

These studies are briefly described in Sects. 6.3.1 and 6.3.2, respectively.

6.3.1 Locality-Aware Scheduling

In default behavior of Pwraake, tasks are scheduled based on the information of nodes where input files are stored. However, this method is not enough for a scientific workflow like Montage [8] where data have geometric relationships. The paper [25] proposed new locality-aware scheduling method using information from a directed acyclic graph (DAG) which represents a workflow. In a workflow graph, a vertex represents a task, and an edge represents a dependency. The *graph partitioning* is an algorithm to divide a graph with minimum edge cut. Figure 6.4a is the result of graph partitioning applied to a Montage workflow. This figure shows that graph partitioning does not reflect task parallelism.

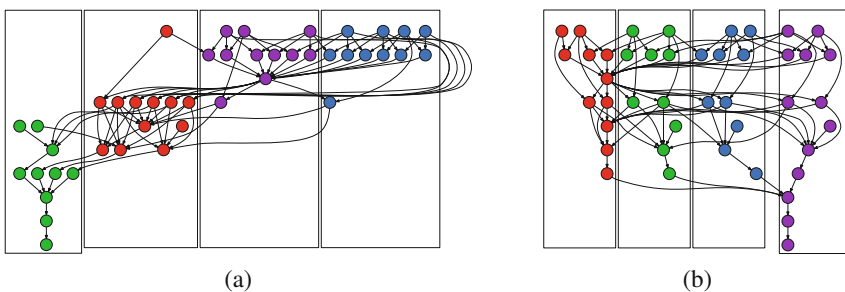


Fig. 6.4 The result of graph partitioning of Montage workflow DAG. (a) Graph partitioning without weight vector. (b) Proposed method based on MCGP

Fig. 6.5 Definition of weight vectors in the MCGP scheduling

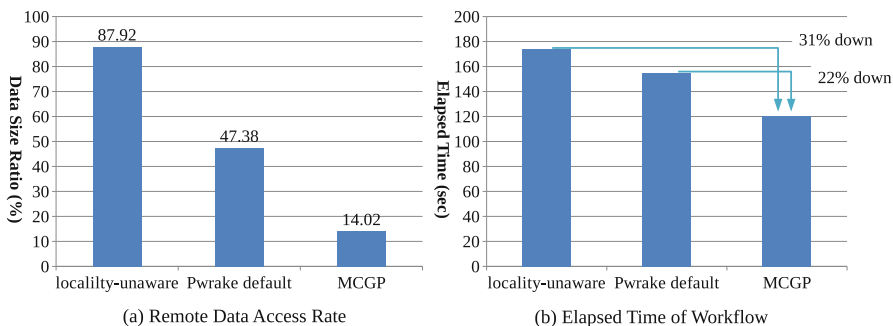
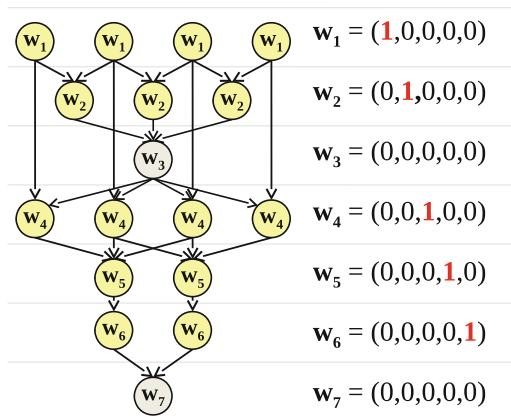


Fig. 6.6 Experimental results of Montage workflow comparing three scheduling methods

The paper [25] proposed a new task scheduling method using the multi-constraint graph partitioning (MCGP) [10]. The MCGP algorithm was studied and implemented in the METIS library [10, 21]. In the ordinary graph partitioning, a vertex may have a scalar weight. Meanwhile, in MCGP, a vertex is weighted by a vector which consists of multiple values. The MCGP algorithm balances the summation of weight values in each dimension. Figure 6.5 illustrates the proposed method in [25] to define weight vectors for the Montage workflow. In this method, each dimension of the weight vector is assigned to each phase of parallel tasks.

In the experiment, the Montage workflow is executed on an eight-node cluster with four cores per node and evaluated three scheduling methods. Figure 6.6a shows that the rate of file access to a remote node is reduced to 14% by the MCGP scheduling from locality-unaware scheduling (88%) and Pwrake default scheduling (47%). Figure 6.6b shows that the MCGP scheduling reduces the elapsed time by 31% from locality-unaware scheduling and by 22% from Pwrake default scheduling. Note that the elapsed time is reduced by 53 s in return for spending 0.03 s for graph partitioning. This result shows that the proposed method using the MCGP is powerful to improve the performance of workflow execution.

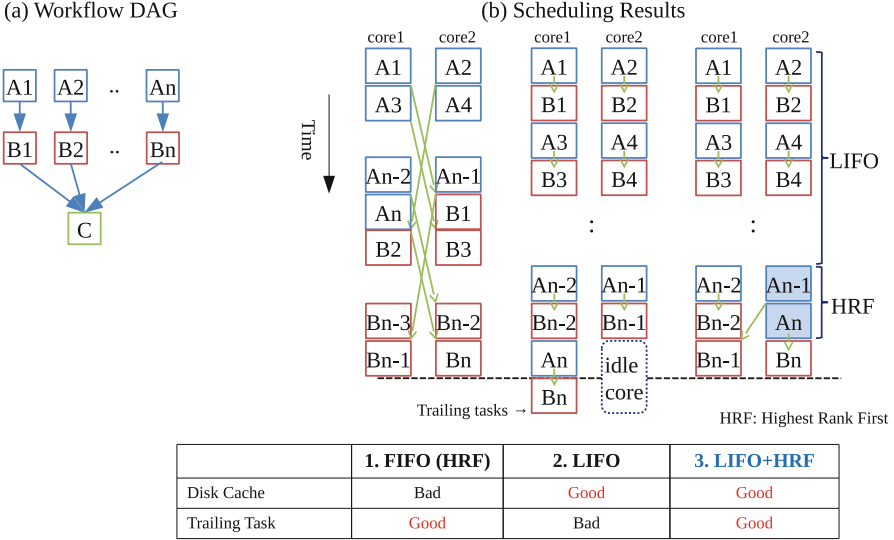


Fig. 6.7 Three scheduling schemes: FIFO, LIFO, and LIFO+HRF

6.3.2 Disk Cache-Aware Scheduling

The paper [26] proposed another I/O-aware scheduling method which improves both disk cache hit rate and core utilization.

Let us consider the scheduling of the workflow DAG shown in Fig. 6.7a. In this workflow, the output file of the task A_1 is the input file of the task B_1 . Figure 6.7b illustrates three scheduling methods for executing the left workflow on a machine with two cores. If the tasks are extracted from a task queue in the FIFO (first in, first out) order, the tasks $A_1 \dots A_n$ are executed before the tasks $B_1 \dots B_n$. In this case, the output file of A_1 may expire from the disk cache before B_1 reads it since $A_3 \dots A_{n-1}$ are executed after A_1 ends and before B_1 starts. In the case of LIFO (last in, first out), the output file of A_1 may remain in the disk cache since the task B_1 is executed immediately after the task A_1 .

However, LIFO has a problem called *trailing task problem* [1]. In the case of FIFO, only B_n is a trailing task. In the case of LIFO, A_n and B_n are trailing tasks since they are executed consecutively.

In order to solve disk cache hit rate and trailing task problem at the same time, the paper [26] proposed a new scheduling method. First, *rank* is defined as the vertical position of a task in the DAG based on the distance from the target task. The scheduling method to execute tasks in the order of higher rank is called *highest rank first* (HRF). The new scheduling method is a hybrid of LIFO and HRF. N_{HR} is defined as the number of highest ranked tasks in the queue and N_{core} as the number of cores of a node. The LIFO+HRF scheduling algorithm is:

- If $N_{HR} > N_{core}$, select a task in order of LIFO.
- If $N_{HR} \leq N_{core}$, select a task in order of HRF.

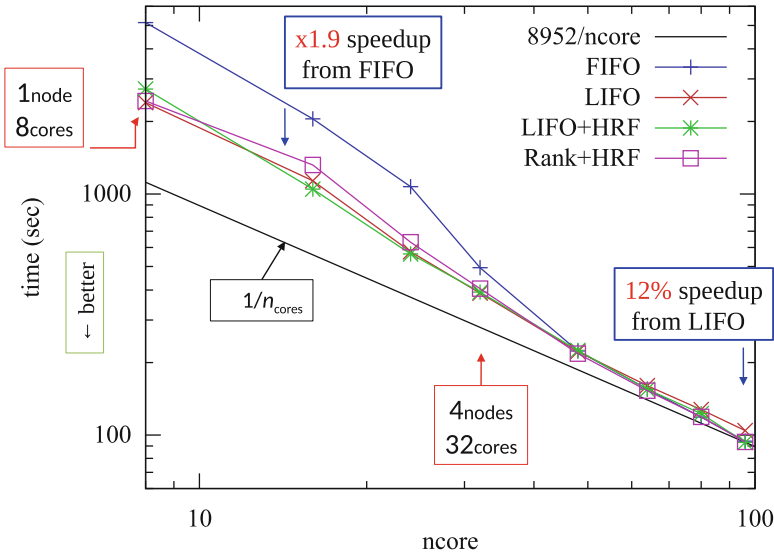


Fig. 6.8 Strong scaling of Montage workflow

The diagram of LIFO+HRF in Fig. 6.7 shows that tasks $A_1 \dots A_{n-2}$ and $B_1 \dots B_{n-2}$ are retrieved with the LIFO method and that A_{n-1} and A_n are retrieved with the HRF method. The paper [26] also discusses another scheduling method called *rank equalization*, but we omit it here.

Figure 6.8 shows the elapsed time (only for parallel tasks) of Montage workflow executed using 1–12 nodes with eight cores per node. In the range of 1–4 nodes, the FIFO scheduling is about 1.9 times slower than the other methods. This result shows that the disk cache hit rate is a significant factor for performance when data size per node is large. In the range of 6–12 nodes, the LIFO scheduling is slower than other scheduling methods. In the experiment using 12 nodes with 96 cores, the LIFO+HRF scheduling improved the performance by 12% from LIFO scheduling. This improvement is due to the reduction of the trailing task problem by the HRF method. In conclusion, the LIFO+HRF scheduling method demonstrates excellent performance for any number of cores in this experiment by solving issues of disk cache and trailing tasks.

6.4 Task Dispatching for Batch Queuing Systems

A traditional scheduler for batch queuing systems takes the CPU load average as the main standard for load balancing. However, the cost of a file access can be a significant parameter for nonuniform storage access (NUSA) file systems like Gfarm, where the difference in throughput between local and remote access may

be significant. We describe data-aware task dispatching for batch queuing systems in the following subsection:

- Data-aware task dispatch (DAD) [12] in Sect. 6.4.1.
- Improved data-aware task dispatch (IDAD) [11] in Sect. 6.4.2.

6.4.1 Data-Aware Task Dispatch

The paper [12] proposed data-aware task dispatch (DAD). It introduces *fileLocality*, a parameter that indicates the difficulty of accessing the dataset, and combines it with load average as a comprehensive *Score* to determine the most suitable node.

The *fileLocality*(t, h), which indicates the difficulty of accessing the dataset referenced by task t when it is dispatched to a specific compute node h , is defined as follows:

$$fileLocality(t, h) = \left[\sum_{y=1}^n locality(f_y, h) \right. \\ \left. / \sum_{y=1}^n (sizeof(f_y) + 1) \right] / 2 \quad (6.1)$$

$$locality(f_i, h) = \begin{cases} -sizeof(f_i) & \text{if } on(f_i, h) \\ sizeof(f_i) & \text{other} \end{cases}$$

where the *locality*(f_i, h) is a value determined by the size of file f_i and whether compute node h has a replica of f_i . If one of the replicas of f_i is on h , the cost of accessing it will be smaller, and therefore the file size of f_i will be subtracted to make the “cost” smaller, and vice versa. The *fileLocality*(t, h) is the normalized sum of the *locality*(f_y, h) ranges [0, 1].

The comprehensive *Score* can be calculated in advance using the *fileLocality* as follows:

$$Score(t, h) = fileLocality(t, h) \times \beta \\ + load(h) \times (1 - \beta) \quad (0 \leq \beta \leq 1) \quad (6.2)$$

The load average *load* and *fileLocality* are unified into *Score* using parameter β . Here, β is a modifier used to adjust the strength of DAD. When $\beta = 1$, the scheduler will ignore the CPU load at dispatch. Although there should be a method for acquiring the optimal value of β , we leave this for future work and only show the effectiveness of this particular parameter.

Score can now be used to judge whether a host is desirable for a job execution in the exact way in which the load average is used in a CPU-focused scheduler, with consideration of both the CPU load and the file locality.

In DAD, β is a key parameter for striking a balance between *fileLocality* and CPU load. However, in real-world situations, tasks dispatched by a scheduler may have quite different characteristics. Some tasks may be I/O-intensive which requires β to be set to a value close to 1, while others could be CPU-intensive and prefer smaller β values. Since β in Equation 6.2 is a global parameter that affects all tasks dispatched by the scheduler, it is not easy to determine a suitable β that works for all kinds of tasks.

6.4.2 Improved Data-Aware Task Dispatch

As discussed in the previous subsection, DAD has parameter β to strike a balance between *fileLocality* and CPU load, yet β is quite difficult to calculate. Considering the fact that CPU load does not have a major impact on execution time, the paper [11] proposed a more data-centric approach called improved data-aware task dispatch (IDAD).

A traditional scheduler takes the CPU load average as the main standard for load balancing. Just like DAD, IDAD also defines a *Score* for selecting the best node at dispatch phase. In DAD, the only user-defined parameter is β , and it is a global parameter that affects all tasks scheduled. For precise control of each task, we introduce a per-task parameter called Remote Degradation Rate (RDR) to indicate the extent to which a task is data-intensive.

RDR is defined as follows:

$$RemoteDegradRate = \frac{RemoteTime(t) - LocalTime(t)}{LocalTime(t)} \quad (6.3)$$

where *RemoteTime(t)* is the execution time when a task t runs on a remote node. Likewise, *LocalTime(t)* is the execution time when a task t runs on a local node.

The range of RDR is $[0, \infty)$. When *RemoteTime(t)* equals *LocalTime(t)*, RDR is zero, which means for this specific task t , executing remotely or locally does not affect execution time, and therefore it is not a data-intensive job.

The *Score(t, h)* then can be defined as follows:

$$Score(t, h) = RDR(t) * \frac{\sum_{y=1}^n RemoteSizeof(f_y, h)}{\sum_{y=1}^n Sizeof(f_y)} \quad (6.4)$$

$$RemoteSizeof(f_i, h) = \begin{cases} 0 & \text{if } on(f_i, h) \\ sizeof(f_i) & \text{other} \end{cases}$$

where the $\sum_{y=1}^n RemoteSizeof(f_y, h)$ is the total size of files accessed remotely and $\sum_{y=1}^n Sizeof(f_y)$ is the total size of files accessed by the task.

At the dispatch phase, when the scheduler selects the best node h for task t , IDAD calculates the $Score(t, h)$ for each free node h ; and the node with the lowest $Score$ is chosen as the execution node.

The paper [12] discussed that some specific task orders may harm locality and cause a drastic degradation in performance. This is also true in IDAD. To alleviate this issue, delay scheduling (DS) [28] is applied in IDAD. In DS with IDAD, tasks are classified by $Score(t, h)$ in Equation 6.4 and a local threshold value. A node h is local to task t if $Score(t, h)$ is smaller than the local threshold $lThreshold$. Otherwise, h is considered remote.

6.4.3 Evaluation

The paper [11] implemented IDAD which interacts with Gfarm file system and evaluated IDAD with BLAST benchmarks, comparing it with DAD and the stock FIFO Torque scheduler. Tasks of Blastn and Blastx are submitted to two different queues to avoid excess judgment regarding the DS. In our experiment, four nodes are used as storage and execution nodes. Twenty-two different database files were replicated once (two replicas) and distributed evenly to four compute nodes. In this evaluation, Blastn and Blastx are submitted to IDAD, DAD, and the stock FIFO Torque scheduler. As the first step of IDAD, a sampling evaluation to acquire the RDR of Blastn and Blastx is performed by executing tasks on both remote and local machines. RDR is calculated using Eq. 6.3.

The RDRs of Blastn and Blastx are 6.176 and 0.262, respectively. In this evaluation, $lThreshold$ is set to 0.3 with intent to classify Blastx as a CPU-intensive job and Blastn as an I/O-intensive job. Here, the $lThreshold$ is altered to a large and then a small value to see how it affects the result. For DAD, since there is no viable way of calculating the optimal β defined, DAD with different values of β and $Delay$ is evaluated to acquire the best result. Figure 6.9 shows the makespan of 100 Blastn tasks and 10 Blastx tasks, which is the time difference between starting the first task and finishing the last task. As you can see in the figure, the shortest makespan using DAD is 179.139 s when $\beta = 0.8$ and $Delay = 2$.

The best-case makespans of DAD, IDAD, and the stock FIFO Torque scheduler are shown in Fig. 6.10. BEST in the figure is obtained under the condition that all nodes hold all datasets required by the task, which means whichever node the task is dispatched to, it will access files locally. Therefore, it can be considered the lower bound for the evaluation. In this figure, IDAD-0 denotes the result of IDAD with $Delay = 0$.

As you can see in Fig. 6.10, the best makespan using IDAD is 162.265 s ($Delay = 2$), whereas the best case with DAD is 179.139 s ($\beta = 0.8$ and $Delay = 2$), which is a 10.40% time reduction. There is a much more obvious improvement, 35.05%, when compared with the stock FIFO Torque scheduler.

Fig. 6.9 Makespan of Blastn and Blastx using data-aware dispatch

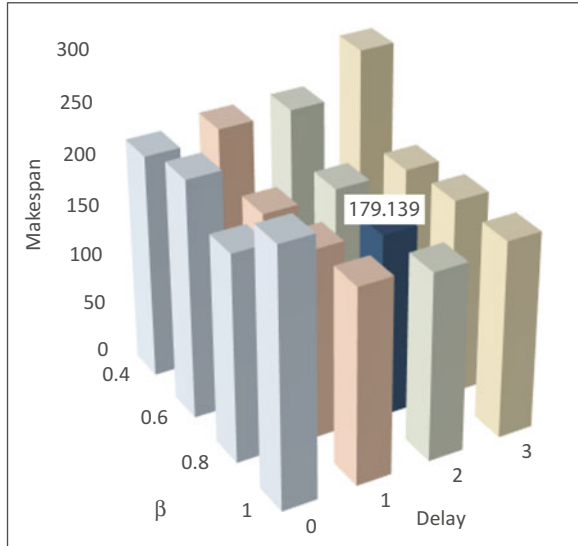
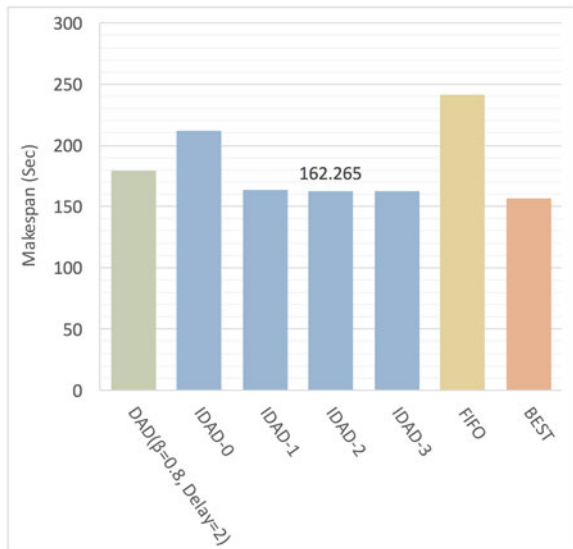


Fig. 6.10 Makespan comparison



References

1. Armstrong, T.G., Zhang, Z., Katz, D.S., Wilde, M., Foster, I.T.: Scheduling many-task workloads on supercomputers: dealing with trailing tasks. In: 2010 3rd Workshop on Many-Task Computing on Grids and Supercomputers, pp. 1–10. IEEE (2010). <https://doi.org/10.1109/MTAGS.2010.5699433>. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5699433>

2. Dahlin, M.D., Wang, R.Y., Anderson, T.E., Patterson, D.A.: Cooperative caching: using remote client memory to improve file system performance. In: Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation (1994)
3. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. *Commun. ACM* **51**(1), 107–113 (2008). <https://doi.org/10.1145/1327452.1327492>
4. Fusion-Io: NVM Primitives Library (2014). <http://opennvm.github.io/nvm-primitives-documents/>
5. Ghemawat, S., Gobiolf, H., Leung, S.T.: The Google file system. In: Proceedings of the 19th ACM Symposium on Operating Systems Principles, pp. 20–43 (2003)
6. Hadoop Distributed File System. <http://hadoop.apache.org/>
7. Herlihy, M., Luchangco, V., Moir, M., Scherer III, W.N.: Software transactional memory for dynamic-sized data structures. In: Proceedings of the Twenty-Second Annual Symposium on Principles of Distributed Computing (PODC '03), pp. 92–101. ACM, New York (2003). <https://doi.org/10.1145/872035.872048> <http://doi.acm.org/10.1145/872035.872048>
8. Jacob, J.C., Katz, D.S., Berriman, G.B., Good, J.C., Laity, A.C., Deelman, E., Kesselman, C., Singh, G., Su, M.H., Prince, T.A., Williams, R.: Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking. *Int. J. Comput. Sci. Eng.* **4**(2), 73–87 (2009). <https://doi.org/10.1504/IJCSE.2009.026999> <http://dl.acm.org/citation.cfm?id=1568665.1568666>
9. Josephson, W.K., Bongo, L.A., Li, K., Flynn, D.: DFS: a file system for virtualized flash storage. *ACM Trans. Storage* **6**(3), 14:1–14:25 (2010)
10. Karypis, G., Kumar, V.: Multilevel algorithms for multi-constraint graph partitioning. In: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing, Supercomputing '98, pp. 296–310. Springer, IEEE Computer Society, Washington, DC (1998). <https://doi.org/10.1109/SC.1998.10018> <http://dl.acm.org/citation.cfm?id=509086>
11. Li, X., Tatebe, O.: Improved Data-Aware Task Dispatching for Batch Queuing Systems. In: 2016 Seventh International Workshop on Data-Intensive Computing in the Clouds (DataCloud), pp. 37–44. IEEE (2016). <https://doi.org/10.1109/DataCloud.2016.009> <http://ieeexplore.ieee.org/document/7845280/>
12. Li, X., Tatebe, O.: Data-aware task dispatching for batch queuing system. *IEEE Syst. J.* **11**(2), 889–897 (2017). <https://doi.org/10.1109/JSYST.2015.2471850> <http://ieeexplore.ieee.org/document/7273750/>
13. Ohtsuji, H., Tatebe, O.: Active-storage mechanism for cluster-wide RAID system. In: Proceedings of IEEE International Conference on Data Science and Data Intensive Systems (DSDIS), pp. 25–32 (2015)
14. Oyama, Y., Ishiguro, S., Murakami, J., Sasaki, S., Matsumiya, R., Tatebe, O.: Reduction of operating system jitter caused by page reclaim. In: Proceedings of the 4th International Workshop on Runtime and Operating Systems for Supercomputers (2014)
15. Oyama, Y., Ishiguro, S., Murakami, J., Sasaki, S., Matsumiya, R., Tatebe, O.: Experimental analysis of operating system jitter caused by page reclaim. *J. Supercomput.* **72**(5), 1946–1972 (2016)
16. Oyama, Y., Murakami, J., Ishiguro, S., Tatebe, O.: Implementation of a deduplication cache mechanism using content-defined chunking. *Int. J. High Perform. Comput. Netw.* **9**(3), 190–205 (2016)
17. Ren, K., Gibson, G.: Tablefs: Enhancing metadata efficiency in the local file system. In: Proceedings of the 2013 USENIX Conference on Annual Technical Conference, USENIX ATC'13, pp. 145–156. USENIX Association, Berkeley (2013). <http://dl.acm.org/citation.cfm?id=2535461.2535480>
18. Ren, K., Zheng, Q., Patil, S., Gibson, G.: Indexfs: Scaling file system metadata performance with stateless caching and bulk insertion. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC'14, pp. 237–248. IEEE Press, Piscataway (2014). <https://doi.org/10.1109/SC.2014.25>

19. Sasaki, S., Matsumiya, R., Takahashi, K., Oyama, Y., Tatebe, O.: RDMA-based cooperative caching for a distributed file system. In: Proceedings of the 21st IEEE International Conference on Parallel and Distributed Systems, pp. 344–353 (2015)
20. Sasaki, S., Takahashi, K., Oyama, Y., Tatebe, O.: RDMA-based direct transfer of file data to remote page cache. In: Proceedings of 2015 IEEE International Conference on Cluster Computing, pp. 214–225 (2015)
21. Schloegel, K., Karypis, G., Kumar, V.: Parallel static and dynamic multi-constraint graph partitioning. *Concur. Comput. Pract. Exp.* **14**(3), 219–240 (2002). <https://doi.org/10.1002/cpe.605>
22. Takatsu, F., Hiraga, K., Tatebe, O.: Design of object storage using open VM for high-performance distributed file system. *J. Inf. Process.* **24**(5), 824–833 (2016)
23. Takatsu, F., Hiraga, K., Tatebe, O.: PPFs: a scale-out distributed file system for post-petascale systems. *J. Inf. Process.* **25**, 538–447 (2017)
24. Tanaka, M., Tatebe, O.: Pwrake: A parallel and distributed flexible workflow management tool for wide-area data intensive computing. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC '10), pp. 356–359. ACM Press, New York (2010). <https://doi.org/10.1145/1851476.1851529>. <http://dl.acm.org/citation.cfm?id=1851476.1851529> <http://portal.acm.org/citation.cfm?id=1851476.1851529>
25. Tanaka, M., Tatebe, O.: Workflow scheduling to minimize data movement using multi-constraint graph partitioning. In: 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012), pp. 65–72. IEEE (2012). <https://doi.org/10.1109/CCGrid.2012.134>. <http://dl.acm.org/citation.cfm?id=2310096.2310129>
26. Tanaka, M., Tatebe, O.: Disk Cache-Aware Task Scheduling For Data-Intensive and Many-Task Workflow. In: IEEE Cluster 2014, pp. 167–175. IEEE, Madrid (2014). <https://doi.org/10.1109/CLUSTER.2014.6968774>. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6968774>
27. Tatebe, O., Hiraga, K., Soda, N.: Gfarm grid file system. *N. Gener. Comput.* **28**, 257–275 (2010)
28. Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S., Stoica, I.: Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling. In: Proceedings of the 5th European conference on Computer systems – EuroSys '10, p. 265. ACM Press, New York (2010). <https://doi.org/10.1145/1755913.1755940>. <http://portal.acm.org/citation.cfm?doid=1755913.1755940>

Chapter 7

Approaches for Memory-Efficient Communication Library and Runtime Communication Optimization



Takeshi Nanri

Abstract This article summarizes the works established in Advanced Communication for Exa (ACE) project. The most important motivation of this project was the severe demands for scalable communication toward Exa-scale computations. Therefore, in the project, we have built a PGAS-based communication library, Advanced Communication Primitives (ACP). Its fundamental communication model is one-sided, based on PGAS model, so that it can consume internal memory footprint as small as possible. Based on this model, several applications including simulations of magnetohydrodynamic, molecular orbitals, and particles were tuned to achieve higher scalability. In addition to that, some communication optimization techniques have been investigated. Especially, tuning methods of collective communications, such as message ordering, algorithm selection, and overlapping, are studied. Also, in this project, a network simulator NSIM-ACE is developed. It simulates behavior of packets for one-sided communications to study the effects of congestions on interconnects.

7.1 Motivation

Currently, there have been various discussions about possible designs of Exa-scale computers. Most of those designs predict that the number of nodes and the number of cores will be significantly increased, and the interconnect topology will be more complicated. On such systems, communication libraries should be re-designed to fulfil the requirements of scalability. Especially, memory usage and performance tuning will become the key issues.

As for memory usage, in the existing communication libraries, each process prepares some amount of receive buffer for each of other processors. This works efficiently up to hundreds of thousands of processes. However, when the number of

T. Nanri (✉)
Kyushu University, Fukuoka, Japan
e-mail: nanri.takeshi.995@m.kyushu-u.ac.jp

processes becomes over 100 million, the total amount of memory for the buffer will be more than 100 GB/process, even when the amount of each receive buffer is 1 KB. At the same time, the available amount of memory per process is predicted to be the same level or reduced on Exa-scale computers. Therefore, communication libraries on such systems must be based on memory saved protocols.

On the other hand, as for performance tuning, static and manual methods are applied on the existing communication libraries. For example, the thresholds for changing algorithms of collective communications are decided by using some benchmark programs at the installation of the library. As the number of processes increases, and the interconnect topology becomes complicated, the search space of the optimization will be increased explosively. In addition to that, because of the complexed topology, it becomes quite difficult to predict performance statically. Therefore, some automatic and dynamic method will be needed for tuning communication libraries.

Another important point for performance tuning is the information of the programs. Existing communication libraries can only achieve information about parameters that have been specified at the invocation of communication functions. Therefore, libraries cannot analyze how those functions are used in the programs. For example, if the library can detect that the invoked function will be repeatedly invoked for many times, it can pay some overhead to apply more aggressive optimization at runtime. Or, there can be special approaches for implementing some popular patterns of computation and communication.

7.2 PGAS-Based Communication Library ACP

7.2.1 Overview

ACP library is designed to support low-overhead data transfer with just-enough amount of memory in communications (Fig. 7.1). It consists of the basic layer and the middle layer [18, 19].

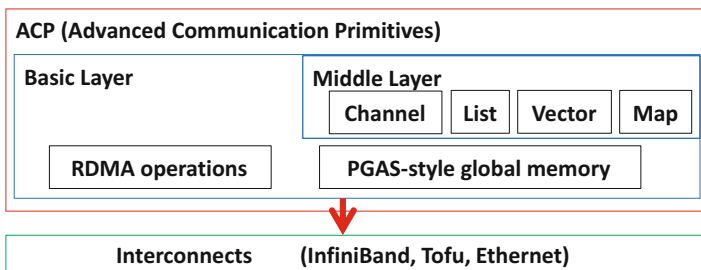


Fig. 7.1 Structure of ACP library

7.2.2 Basic Layer

The basic layer of ACP is a thin abstraction of underlying communication devices [10]. Currently, this layer is implemented on UDP, IBverbs, and Tofu. This layer provides an RDMA-based communication model. It supports a global address space shared among all of the processes. Any local memory space of any process can be mapped to this space via the registration function.

The basic layer provides global address space shared among all of the processes. This space is addressed by 64bit unsigned integer, so that it can be directly handled by the atomic operations of CPUs and devices.

Any local memory space of any process can be mapped to this space via the registration function, `acp_register_memory`, of this layer. It returns a key with the data type of `acp_atkey_t`. There is also a de-registration function, `acp_unregister_memory`, to unregister the region of the specified key. Global address in a registered region can be retrieved by a query function, `acp_query_ga`. It returns the address with the data type of `acp_ga_t`.

At the registration, in addition to the start address and the size of the local region to be mapped, the color of the global address can be specified. Color is used for specifying device in the node to be used for making remote access to the region. There is a function, `acp_colors`, to query the maximum number of colors available on the current system.

At the initialization, a special region, called starter memory, is allocated on each process. Each of the starter memories of the processes is registered to the global address space, and there is a function, `acp_query_starter_ga`, to query for the global address of the starter memory of the specified rank. This region is mainly used for exchanging global addresses before performing RDMA operations on the global address space.

The RDMA operations on the global memory space of the basic layer is called global memory access (GMA). ACP prepares GMA functions, such as `acp_copy`, `acp_add4`, `acp_add8`, `acp_cas4`, and so on, to perform copy and atomic operations on the global memory space. Unlike other communication libraries, both of the source and the destination of copy function can be remote.

All of the GMA functions are non-blocking. Therefore, each of them returns a GMA handle with the data type of `acp_handle_t` to wait for the completion. Each GMA function has an argument “order” to specify the condition for starting the operation. If a GMA handle is specified for this argument, it will wait for the completion of the GMA of the handle before starting its access. If `ACP_HANDLE_NULL` is specified as the order, it starts the access immediately. If `ACP_HANDLE_ALL` is specified, it will wait for the completions of all of the previously invoked GMAs. If `ACP_HANDLE_CONT` is specified as the order, and if the GMA that is invoked immediately before this one has the same source rank, target rank, source color, and target color, it can start its access with the same condition of the previous one, as far as it will not overtake the order. If there is any difference in those parameters, the behavior is same as the case with `ACP_HANDLE_ALL`. There

can be an implementation that always performs the same way as the case with `ACP_HANDLE_ALL`, even if `ACP_HANDLE_CONT` is specified.

The completion function, `acp_complete`, waits for the completion of the GMA specified by the handle. The completions of GMA functions are ensured in order. This means that the completion of one GMA function ensures the completion of all of the other GMA functions that have been invoked earlier than it on the process. There is also a function, `acp_inquire`, which check the completions of the GMA specified by the handle and the GMAs that have been invoked before it.

7.2.3 Middle Layer

To hide the complexities of programming with RDMA, such as registration of memory regions and exchanging global addresses, the middle layer of ACP is prepared as a set of programmer-friendly interfaces. The interfaces of this layer support common patterns of communications, such as channels, neighbors, and collectives. It also prepares interfaces for managing common data structures, such as queues, lists, and vectors, on the global address space of ACP.

As a common policy, each interface consists of the functions for allocation and de-allocation of the memory regions such as control structures and buffers that are used in it. With these functions, the programmers can specify the exact duration of the usage for each of the memory regions. Then, the library may use this information to reduce its memory consumption without degrading the performance.

7.2.3.1 Data Interfaces

Data interfaces of ACP consists of the following data types [1, 2]:

Vector The vector interface provides the data structure and algorithms of dynamic array. The type of for referencing vectors is `acp_vector_t`, and the type of the iterator of vectors is `acp_vector_it_t`. Vectors can be created via the constructor function, `acp_create_vector`. As an argument, this function accepts the rank to place the instance of the vector.

Deque The deque interface provides the data structure and algorithms of double-ended queue. The type of for referencing deques is `acp_deque_t`, and the type of the iterator of deques is `acp_deque_it_t`. Deques can be created via the constructor function, `acp_create_deque`. As an argument, this function accepts the rank to place the instance of the deque.

List The list interface provides the data structure and algorithms of bidirectional linked list. The type for referencing lists is `acp_list_t`, and the type for the iterator of lists is `acp_list_it_t`. Lists are created via the constructor function, `acp_create_list`. This function prepares an empty list. As an argument, it accepts the rank to place the control structure of the list. Elements

can be added to a list via functions, such as `acp_push_back_list` and `acp_insert_list`. Each of these functions also accepts the rank to allocate the instance of the element, as an argument. Therefore, the control structure and the elements of a list can be placed on different processes.

Map The map interface provides the data structure and algorithms of associative arrays. The type for referencing maps is `acp_map_t`, the type for the iterator of maps is `acp_map_it_t`, and the type for returning result of finding element in a map is `acp_map_ib_t`. Maps are created via the constructor function, `acp_create_map`. This function prepares an empty map distributed among a group of processes. As an argument, it accepts the number of processes, the array of ranks of the processes in the group, the number of slots, and the rank to place the control structure of the map. Elements can be added to a map via the function `acp_insert_map`. In addition to the map to insert, this function accepts the key and the value of the element, as an argument. Each of the key and the value is specified by a pair of the address and the size in byte. The rank to place the element to be inserted is decided according to the hash value calculated from the key. `acp_find_map` searches the key in a map. It returns the value true or false according to the result and the iterator of the element with the key if it has been found.

Workspace Workspace is a temporal memory space that is shared among all of processes. Creation and destruction of a workspace is done collectively via functions `acp_create_ws` and `acp_destroy_ws`. The type of the handle for referencing workspaces is `acp_ws_t`. Once created, each process can perform read/write accesses to the workspace via functions `acp_read_ws` and `acp_write_ws` that are similar to POSIX `pread/pwrite`. Size of the workspace is specified at its creation. Initial value of the workspace is undefined. Distribution of the temporal memory workspace among processes can be controlled by parameters specified via the function `acp_setparams_ws`.

Malloc The malloc interface provides the functions for asynchronous allocation and de-allocation of global memory. It is the same as that of the asynchronous global heap that designs a memory pool to be accessible via RDMA as well as via processor instructions. This interface consists of the `acp_malloc` function that allocates a segment of global memory from specified process by a rank number. A global memory segment allocated by this function can be freed by the `acp_free` function.

7.2.3.2 Communication Interfaces

Communication interface provides a virtual pass for message passing from one process to another (Fig. 7.2). The data transfer supported by it is single direction and in order. Therefore, the implementation of it can be simple and efficient. Before establishing data transfer with the interface, both the sender process and the receiver process must call the allocation function to establish a channel between them. If a



Fig. 7.2 Flow of messages on a channel

channel has become useless, the program can free the channel by calling the de-allocation function at both sides [13, 14].

The allocation function, `acp_create_ch`, of this interface allocates a region of memory according to the role of the process that invoked the function. After that, it starts exchanging the information about the region with another peer of the channel to establish the connection of the channel. Then, it returns a handle of the channel, with the data type of `acp_ch_t`, without waiting for the completion of the connection. The library implicitly progresses the establishment of the connection.

The non-blocking functions for sending and receiving messages, `acp_nbSEND_ch` and `acp_nbRECV_ch`, on a channel can be invoked before the completion of its connection. Each of these functions stores the information about the invocation in the request queue of the channel and returns a request handle with the data type of `acp_request_t`. It starts transferring messages after it has completed the connection. The wait function, `acp_wait_ch`, waits for the completion of the request.

The non-blocking de-allocation function, `acp_free_ch`, starts freeing memory regions of the channel specified by the handle. The behavior of the functions of send and receive on the channel that has been specified for this function is not defined.

7.2.4 *Enabling ACP as a Communication Layer Among MPI Task*

MPI (message passing interface) is a programming interface for process-parallel programming used as a de facto standard. There exists various scientific programs that are tuned by using communication functions of this interface. In most of the cases, loops are divided among processes to parallelize the program mainly because of the Amdahl's law and the easiness of programming. However, in recent years, parallelisms of loops are consumed mainly by the parallel architectures within one computational node, such as SIMD ways, multiple cores, and accelerators. Therefore, another level of parallelism, tasks, is becoming important. With task-level parallelism, a program is divided into smaller tasks that solve subproblems. Then, these tasks are executed in parallel, according to the dependencies among them.

Therefore, ACP has been designed to be used as a communication layer for connecting multiple MPI tasks to enable task-parallel programs with existing tuned

MPI codes [8]. Toward Exa-ela, this layer will need to handle millions of processes. Of course, MPI itself can be the layer to connect those tasks. However, MPI has a problem of scalability, because it consumes amount of memory linear to the number of processes. For example, if the size of one receive buffer is 10KB, the amount of memory consumed for the buffers will be 10GB on each process, when the number of processes becomes one million.

On the other hand, by using ACP, the memory consumption can be minimal. Also, one-sided communications can be implemented by using RDMA (remote direct memory access) facility that is available on most of the recent high-performance interconnects. This memory efficiency makes ACP library as the better candidate for the communication layer that covers all of the processes on large-scale parallel computers.

Both MPI library and ACP library have their own launcher to start processes on computational nodes. Those launchers pass various parameters to the processes that are required in the initialization. Therefore, we have added another launcher, `macprun`, to pass parameters of ACP to the processes of MPI tasks. `macprun` uses the launcher of MPI library as it is, internally. Therefore, no modification is needed to the existing MPI libraries.

The most important part for enabling ACP in MPI processes is the mechanism for passing parameters of ACP to them. The parameters of ACP for initialization are a pair of port numbers of TCP and the rank of the process in ACP. The rank of the process in ACP is different from process to process, but they can be calculated with simple manner. On the other hand, the port numbers are also different from process to process. In addition to that, those numbers depend on the availability of the ports of the platform. Therefore, it is difficult to calculate those numbers at the initialization.

Figure 7.3 shows a sample of the commands invoked within `macprun`. In this sample, three MPI tasks are invoked. The numbers of the processes of those tasks are 6, 4, and 2, respectively. To calculate the rank in ACP of the process, `macprun` passes the offset of the rank of ACP at the tail of the command-line option. After the initialization of MPI, each process is attached with the rank of MPI. Those ranks are contiguous among processes in one MPI task and starts from 0. Therefore, each process calculates its rank of ACP by adding the offset and its rank of MPI.

As for port numbers, on the other hand, `macprun` generates a temporal file that consists a table of ACP ranks and their pair of port numbers. Then the name of the file is passed as the parameter of the command-line option, `--mult-runtime`. At the initialization of ACP in the program, each process reads the line of its rank

```
mpirun machines Nodes.1 -np 6 ./test --mult-runtime Portfile 0 &
mpirun machines Nodes.2 -np 4 ./test --mult-runtime Portfile 6 &
mpirun machines Nodes.3 -np 2 ./test --mult-runtime Portfile 10
```

Fig. 7.3 Sample of `mpirun` commands invoked in `m_acprun` for launching three MPI tasks with 6, 4, and 2 processes

of ACP and retrieves its pair of port numbers. Then, it uses those ports to exchange information among processes of all MPI tasks, to enable communication each other by ACP functions.

At this point, this mechanism relies on the shared file system among all of the computational nodes. In addition to that, the size of the temporal file is linear to the number of processes. These requirements degrade the availability and the scalability of the mechanism. Designing more flexible and scalable mechanism is remaining as a future work.

7.3 Network Simulator NSIM-ACE

To study detailed traffics on interconnects, there have been various network simulators. However, most of them deal with communications with message passing model in which two processes explicitly participate in the data transfer. On the other hand, as RDMA facility become popular on interconnects, communications with one-sided model, such as `acp_copy`, `MPI_Put`, and `MPI_Get`, are becoming another important communication patterns in parallel applications. Therefore, to enable studies on such communication patterns, ACE project has developed a simulator, NSIM-ACE [20, 21], which is an enhancement of existing simulator for two-sided communications, NSIM.

NSIM is an interconnect simulator that simulates behavior of packets. The patterns of packet transfers to be simulated are automatically generated from MGEN programs that are C codes with MPI-style functions, such as `MGEN_Isend`, `MGEN_Irecv`, and `MGEN_Wait`. Invocation times of each function can be adjusted by `MGEN_Comp` function which delays following operations for specified duration of seconds.

With the generated patterns of packets, NSIM performs distributed event simulation on each of them. It uses information of the topology and the routing policy of the interconnect given as input files to simulate arbitrations on switches and congestions on links. Therefore, it can perform detailed simulation of large-scale interconnects.

NSIM-ACE added four functions, `MGEN_rdma_put`, `MGEN_rdma_get`, `MGEN_rdma_poll`, and `MGEN_acp_complete` to simulate one-sided communication model. `MGEN_rdma_put` and `MGEN_rdma_get` are used to specify when the one-sided communication starts. `MGEN_rdma_poll` represents a behavior of the target process of `MGEN_rdma_put` which waits for the arrival of the data. `MGEN_acp_complete` waits for the completion of `MGEN_rdma_put` or `MGEN_rdma_get`.

Simulation of one-sided communications with these functions are done by automatically generating reply packets on target side, at each arrival of requesting packets for `MGEN_rdma_put` and `MGEN_rdma_get`. Therefore, traffics for one-sided communications can be investigated precisely.

7.4 Runtime Optimizations of Collective/Neighboring Communications

Overheads of collective communications and neighboring communications are another important issue for achieving sufficient scalability of applications. This section describes several optimization techniques of these communications introduced in ACE project.

7.4.1 *Runtime Algorithm Selection of Collective Communication*

Usually, for each collective communication, more than algorithms are available. Previously, choices of the appropriate algorithm have been done statically, according to the results of benchmarks on some possible combinations of parameters, such as the number of ranks and the size of messages.

However, as the sizes and the complexities of computer systems for high-performance computing are increased significantly, such static strategy has become insufficient to enable efficient usage of the systems. One of the reasons is simply because the parameters to be considered has become large in number and wide in range. In addition to that, especially for communication libraries, usage of cost-efficient topologies of interconnect networks, such as Fat-Tree, Mesh, or Torus, has increased the difficulty on appropriate choice. On these topologies, there are some additional issues that affect the performance significantly, such as the collisions among independent messages and the distance between the sender and the receiver. This means, even when the number of ranks and the size of the message are the same, best implementation technology can be different according to the situations only known at runtime, such as relative locations of ranks. Therefore, demands for the techniques to choose suitable implementations at runtime are increasing.

In ACE project, a method is proposed for choosing the appropriate algorithm of collective communications at runtime [16]. Collective communications, such as broadcasts, reductions, and all-to-all exchanges, are popularly used in parallel programs of computational sciences to achieve productivity and performance portability. There have been many algorithms introduced for each of these collective communications. The method proposed in this paper is a combination of two approaches, performance prediction and performance measurement.

At first, it gathers the information about the allocations of ranks, and applies them to the performance models of the available algorithms along with the information about the topology and the routing policy of the system to predict the time for completing the communication with them. By comparing the results, algorithms that are predicted to be significantly slower than others are discarded from the candidates. Then, each of the remained algorithms is examined one at each call

of the collective communication to gather the empirical performance data. After all algorithms are examined, the fastest algorithm is chosen to be used for the rest of the calls.

From the results of experiments on a cluster connected via Fat-Tree topology interconnect, the proposed method has chosen the algorithm that were sufficiently fast. It guarantees the accuracy of the performance prediction models established in this work.

7.4.2 Neighboring Communication Algorithm for Multiple NICs

For extra large-scale computers, cost-effective topologies such as Mesh and Torus are preferred. With those topologies, each node usually consists of multiple NICs. Therefore, effective use of them is a key to achieve higher scalability.

As one of such attempts, in ACE project, an algorithm of neighboring communications is proposed to use multiple NICs efficiently [11, 12]. The proposed algorithm divides the message into multiple segments. Then they are distributed among NICs according to the number of NICs and the number of neighboring nodes.

An implementation of this algorithm is examined by using RDMA interface of Fujitsu FX10. From the results, it is shown that the proposed algorithm achieves up to 25% improvement of the throughput.

7.4.3 Active Packet Pacing for Congestion Avoidance of Collective Communications

Since congestions on interconnects can reduce the throughput of data transfer significantly, there have been studies on communication algorithms and routing policies to avoid them. However, as the topology becomes larger and complicated, static control of congestions is becoming difficult.

Therefore, ACE project has developed a technique called active packet pacing that reduces congestions according to the runtime status of the interconnect [17]. This technique monitors traffic of each link. If it detects congestion at a link, it increases the gap between packets of the traffic detected as the cause of the congestion. The amount of gap to be increased is adjusted with the number of hops of the traffic, so that number of packets per link becomes optimal.

The logical effect of the packet pacing is examined by the simulator, NSIM. It has predicted that, with all-to-all communication with pairwise exchange algorithm, the throughput can be gained up to two times higher. Also, the practical effect of it is studied by empirical experiments on Fujitsu FX10. From the results, it is shown that packet pacing can actually increase the throughput of the entire interconnects.

7.5 Optimization of Applications

In addition to the fundamental technologies described so far, ACE project also tackled with techniques to achieve higher scalability of some applications. They include analysis of communications in applications [3–6, 9, 23, 24] and optimizations by using fundamental technologies proposed in the project.

7.5.1 *Overlapping Halo Exchange with One-Sided Communication for Stencil Computation*

Stencil computation is one of the most popular pattern of scientific applications in which the target field is represented by multidimensional array, and each element is updated by the values of its neighboring elements. Usually, process-based parallelization on this program is done by dividing the array into blocks and distributing them to the processes so that calculations of each block can be done in parallel.

In the parallelized stencil program, at each step of updating array, each process needs to exchange data on the boundary of the block with the neighboring processes. This type of communication is called as halo exchange. The target processes and the area of this halo exchange depend on how the array is divided and relative positions of elements that are used to update one element.

If this area to be transferred in a halo-exchange communication is not contiguous, multiple operations for sending data are required. However, these operations can cause significant loss of performance because of their overhead. To avoid such overhead, in most of the cases, the elements in that area are copied into a contiguous region of a memory, called a send buffer, so that they can be transferred by one operation. This technique is called packing. After the packed elements are transferred to the receive buffer of the target process, they need to be unpacked to the appropriate positions of the destination block.

Therefore, ACE project examined the effect of overlapping with four different communication interfaces, ISND, ACTV, PASV, and ACP [15]. ISND uses non-blocking two-sided communication interface, `MPI_Isend` and `MPI_Irecv`, for halo exchange. ACTV and PASV use one-sided communication interface with active target synchronization and passive target synchronization, respectively. ACP uses ACP version. Each program performs two-dimensional stencil computation. The array is allocated so that X dimension is contiguous. Therefore, packing and unpacking are performed before and after each halo-exchange communication, respectively.

Figure 7.4 shows the parallel stencil program with two-sided communication interface of MPI. Before entering the main loop, each process calls `MPI_Irecv` to notify the system that its receive buffers are ready to receive messages. Then, in the main loop, it performs packing and starts sending the packed data to the

Fig. 7.4 Stencil computation with send/rcv of MPI

```

setup arrays

MPI_Irecv from left
MPI_Irecv from right
for (steps)
    pack
    MPI_Isend to left
    MPI_Isend to right

    calculate inner elements

    MPI_Waitall

    unpack

    MPI_Irecv from left
    MPI_Irecv from right

    calculate elements of edges

```

Fig. 7.5 Stencil computation with active target of MPI

```

setup arrays
MPI_Win_allocate()

for (steps)
    pack
    MPI_Win_fence
    MPI_Put to left
    MPI_Put to right

    calculate inner elements

    MPI_Win_fence

    unpack

    calculate elements of edges

```

processes on the left and right by using `MPI_Isend`. While the data is transferred, calculations on inner elements that do not depend on the result of halo exchange are performed to overlap the communication. After that, `MPI_Waitall` is called to wait for the completion of the communication. Since the receive buffers become ready again after unpacking, `MPI_Irecv` functions are called to start receiving as soon as possible. Then calculations on the elements that depend on the received data are performed.

Figure 7.5 shows the program with active target synchronization mode. The flow is almost similar with the program with two-sided communication except for the absence of receiving functions. `MPI_Put` is the only functions for data transfer. For synchronization, `MPI_Win_fence` is used at two points in the main loop. The

Fig. 7.6 Stencil computation with passive target of MPI

```

setup arrays
MPI_Win_allocate()
*lrdy = 1; *rrdy = 1;
for (steps)
  ctr++
  pack
  MPI_Win_flush_all
  lflg = 0; rflg = 0;
  while ((lflg == 0) || (rflg == 0))
    if ((lflg == 0) && (*lrdy >= ctr))
      lflg = 1
      MPI_Put data to left
      MPI_Put ctr to rack on left
    if ((rflg == 0) && (*rrdy >= ctr))
      rflg = 1
      MPI_Put data to right
      MPI_Put ctr to lack on right

  calculate inner elements

  lflg = 0; rflg = 0;
  while ((lflg == 0) || (rflg == 0))
    if ((lflg == 0) && (*lack >= ctr))
      lflg = 1
    if ((rflg == 0) && (*rack >= ctr))
      rflg = 1

  unpack

  MPI_Put ctr to rrdy on left
  MPI_Put ctr to lldy on right

  calculate elements of edges

```

first one makes sure that every process is ready to receive data, while the second one waits for the completion of all data transfer.

Figure 7.6 shows the program with passive target synchronization mode. In this mode, to avoid incorrect order of data transfers, synchronization between the source and the target processes of each communication must be explicitly described in the program. With this program, two types of synchronizations are required. To make sure that the buffers on the target process are ready to receive buffers, a short message, called RDY, is sent from the target to the source. On the other hand, to notify the target that the source has sent a data to the buffer, another short message, called ACK, is sent from the source to the target. These messages are sent by using extra MPI_Put functions.

For both of these synchronizations, counters are used. Each process keeps three types of counters, local, ready, and ack. Each process has one local counter that represents its current step of the calculation. On the other hand, ready counters

and ack counters are prepared for each side of the halo exchange. A ready counter represents the availability of the buffer of the target process of the halo exchange on that side. It is updated by `MPI_Put` operation on the target. Similarly, an ack counter represents availability of the data sent from the source of the halo exchange on that side.

In Fig. 7.6, “ctr” is the local counter, “lrdy” and “rrdy” are the ready counters, and “lack” and “rack” are the ack counters. At each step, local counter is incremented. Then, the values of the ready counters are repeatedly checked to wait for the availability of the buffer. When the buffer of the target of one side has become available, it invokes an `MPI_Put` for sending data, followed by another `MPI_Put` for notification to the target. After `MPI_Put` functions to the targets of both sides, it computes inner elements of the matrix. This is where the communication and computation are overlapped.

Then, the process checks the ack counters to wait for the availability of the data from the sources on both sides. After completion of the waits, it unpacks received data. After that, before calculating elements on the edges with that unpacked data, ready notifications are sent to the sources of the both sides, because after unpacking, data in the buffer has been copied to the arrays.

As the flow shows, passive-mode requires additional `MPI_Put` functions for the synchronizations between the source and the target. However, there is no collective operation among all of the processes.

Figure 7.7 shows the ACP version of this program. The flow is quite similar to the program with passive target synchronization. Transferring data and signal messages are all done with a copy function on the global address space. Therefore, all buffers and counters are registered to the global address space in the beginning of the program.

From the results, it is shown that ACP could hide communication time efficiently except for the case when the number of processes is 64. Possible reason for the low efficiency as the number of processes increases, the time for computation becomes shorter so that it cannot hide the entire communication of halo exchange. With `MVAPICH2`, passive target synchronization worked better than other methods in most of the cases. However, the difference between the non-blocking two-sided communication and passive target synchronization is not significant. In addition to that, performance with active target synchronization is almost ten times slower than the time with passive target synchronization. With `Open MPI`, passive target synchronization was the best choice. Other communication methods show almost similar time. Though the detailed analysis on the results is remained as a future work, from these results, using ACP or the passive target synchronization of `MPI` has advantage over other communication methods in the effect of overlapping communication with computation.

However, as the size of the array is increased, this advantage of ACP and passive target synchronization disappears. The reason for this characteristic is being studied. At this point, load imbalances among processes and the overheads for handling messages are the factors to be investigated.

Fig. 7.7 Stencil computation with ACP

```

setup arrays
register ctr, lbuf and rbuf
register lflg and rflg
register lrdy and rrdy
*lrdy = 1; *rrdy = 1;
for (steps)
  ctr++
  pack
  lflg = 0; rflg = 0;
  while ((lflg == 0)|| (rflg == 0))
    if ((lflg == 0)&&>(*lrdy >= ctr))
      lflg = 1
      acp_copy data to left
      acp_copy ctr to rack on left
    if ((rflg == 0)&&>(*rrdy >= ctr))
      rflg = 1
      acp_copy data to right
      acp_copy ctr to lack on right

  calculate inner elements

  lflg = 0; rflg = 0;
  while ((lflg == 0)|| (rflg == 0))
    if ((lflg == 0)&&>(*lack >= ctr))
      lflg = 1
    if ((rflg == 0)&&>(*rack >= ctr))
      rflg = 1

  unpack

  acp_copy ctr to rrdy on left
  acp_copy ctr to lrdy on right

  calculate elements of edges

```

7.5.2 *Studies on Communication Methods in N-Body Simulation*

N-body simulation is another important method used in many scientific problems, such as astronomy and molecular dynamics. Usually, this type of simulation is parallelized by domain decomposition among processes. In this method, each process calculates behavior of particles in the sub-domain attached to it. Then, according to the results of the calculation, data of the particles moved to the different domain needs to be transferred to the process of the new domain. Therefore, pattern of communication in this method is unpredictable. ACE project examined several methods, including the one that uses ACP, to perform this kind of dynamic communication pattern [22].

One way of performing this kind of dynamic communication pattern is to exchange information about the particle movement at each step. This is done by

calling collective communications, such as `MPI_Reduce_scatter`, after the calculation of the particle movement of the step.

Another way is to use one-sided communication. In this case, each process needs to invoke Remote Atomic Fetch-and-Add operation for each target of data transfer to decide the target location so that the transferred data will not be overwritten by other processes. This can be implemented by RMA interface of MPI and ACP.

Results of the experiments showed that implementation with ACP performed well in most of the cases. It is because ACP can utilize RDMA facility of the interconnect efficiently to overlap the communication with computation.

7.5.3 *Memory-Efficient Master-Worker Model for Molecular Orbital Calculation*

OpenFMO is one of the approaches for simulating molecular orbitals. It divides the molecular into segments and applies master-worker model to hierarchically parallelize the calculation. Originally, it is coded with MPI. However, as the number of processes increase, memory consumption of `MPI_COMM_WORLD` is predicted to be a severe issue of scalability.

Therefore, ACE project distributed the workers of OpenFMO as small MPI tasks and connected them with ACP [7]. Since communications required between the master and each worker is one-sided, the connection can be done by basic layer of ACP. From estimation on the memory consumption with MPI-only versus MPI+ACP, it is predicted that MPI+ACP can reduce the memory footprint significantly.

References

1. Ajima, Y.: Reducing manipulation overhead of remote data structure by controlling remote memory access order. In: ExaComm 2016 Workshop, Frankfurt, Germany, 23 June 2016. https://doi.org/10.1007/978-3-319-46079-6_7
2. Ajima, Y., Nose, T., Saga, K., Shida, N., Sumimoto, S.: ACPdl: data-structure and global memory allocator library over a thin PGAS-layer. In: Proceedings of the First International Workshop on Extreme Scale Programming Models and Middleware, pp. 11–18 (2015)
3. Fukazawa, K., Nanri, T.: Performance of large scale MHD simulation of global planetary magnetosphere with massively parallel scalar type supercomputer including post processing. In: Proceedings of 14th IEEE International Conference on High Performance Computing and Communication, pp. 976–982, Liverpool, United Kingdom, Jun 2012. <https://doi.org/10.1109/HPCC.2012.142>
4. Fukazawa, K., Nanri, T., Umeda, T.: Performance evaluation of magnetohydrodynamics simulation for magnetosphere on K computer. In: Tan, G., Yeo, G.K., Turner, S.J., Teo, Y.M. (eds.) AsiaSim 2013, Communications in Computer and Information Science, vol. 402, pp. 570–576. Springer, Berlin/Heidelberg (2013). ISBN: 978-3-642-45036-5. https://doi.org/10.1007/978-3-642-45037-2_61

5. Fukazawa, K., Nanri, T., Umeda, T.: Performance measurements of MHD simulation for planetary magnetosphere on peta-scale computer FX10. In: *Parallel Computing: Accelerating Computational Science and Engineering (CSE)*. *Advances in Parallel Computing*, vol. 25, pp. 387–394. IOS Press (2014). <https://doi.org/10.3233/978-1-61499-381-0-387>
6. Honda, H.: Performance evaluation of Hartree-Fock program developed by ruby scripting language. In: *1st Pan-American Congress on Computational Mechanics (PANACM 2015)*, Apr 2015
7. Honda, H.: Development of ACP middle layer communication library for molecular orbital calculation. In: *International Congress of Quantum Chemistry 2015 Satellite Symposium*, June 2015
8. Honda, H., Morie, Y., Nanri, T.: Development of a memory efficient communication method for connecting MPI programs by using ACP library. In: *The 35th JSST Annual Conference International Conference on Simulation Technology*, Kyoto, Japan, 27–29 Oct 2016
9. Kobayashi, T.: A new bottleneck in large-scale numerical simulations of transient phenomena, and cooperation between simulations and the post-processes. In: *1st Pan-American Congress on Computational Mechanics (PANACM 2015)*, Apr 2015
10. Morie, Y.: Implement and evaluation of ACP basic layer of InfiniBand. In: *International Workshop on Information Technology. Applied Mathematics and Science (IMS 2015)*, Kyoto, Japan, Mar 2015
11. Morie, Y., Nanri, T.: Task allocation optimization for neighboring communication on fat tree. In: *14th IEEE International Conference on High Performance Computing and Communication 9th IEEE International Conference on Embedded Software and Systems, HPCC-ICSS 2012*, pp. 1219–1225, Liverpool, UK, 25–27 June 2012
12. Morie, Y., Nanri, T.: Neighbor communication algorithm with making an effective use of NICs on multidimensional-mesh/torus. In: *International Conference on Simulation Technology (JSST2013)*, Tokyo, Sep 2013
13. Nanri, T.: Channel interface: a primitive model for memory efficient communication. In: *23rd Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, Turku, Finland, Feb 2015
14. Nanri, T.: Performance and memory usage evaluations for channel interface of advanced communication primitives library. In: *1st Pan-American Congress on Computational Mechanics (PANACM 2015)*, Apr 2015
15. Nanri, T., Fukazawa, K.: Effect of overlapping halo exchange with one-sided communication. In: *the 35th JSST Annual Conference International Conference on Simulation Technology*, Oct 2016
16. Nanri, T., Kurokawa, M.: Efficient runtime algorithm selection of collective communication with topology-based performance models. In: *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)' 12*, Las Vegas, 16–19 July 2012
17. Shibamura, H.: Active packet pacing as a congestion avoidance technique in interconnection network. In: *International Conference on Parallel Computing 2015 (ParCo 2015)*, Sept 2015
18. Sumimoto, S., Ajima, Y., Saga, K., Nose, T., Shida, N., Nanri, T.: The design of advanced communication to reduce memory usage for exa-scale systems. In: *Proceedings of 12th International Meeting on High Performance Computing for Computational Science*, Porto, Portugal, 28–30 June 2016, to be published as Springer's *Lecture Notes in Computer Science (LNCS)*
19. Sumimoto, S., Ajima, Y., Nose, T., Saga, K., Shida, N., Yoshiyuki, M., Nanri, T.: Parallel application experiences using advanced communication primitives. In: *25th Euromicro International Conference on Parallel, Distributed and network-based Processing (PDP 2017)*, 6–8 Mar 2017
20. Susukita, R., Morie, Y., Nanri, T., Shibamura, H.: Performance Evaluation of RDMA Communication Patterns by Means of Simulations. In: *2015 Joint International Mechanical, Electronic and Information Technology Conference*, Dec 2015

21. Susukita, R., Morie, Y., Nanri, T., Shibamura, H.: NSIM-ACE: an interconnection network simulator for evaluating remote direct memory access. In: Proceedings of 6th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2016) (2016)
22. Susukita, R., Morie, Y., Nanri, T.: Efficient communications of particle data in particle-based simulations. In: Proceedings of 35th JSST Annual Conference International Conference on Simulation Technology (JSST 2016) (2016)
23. Takami, T., Fukudome, D.: An efficient pipelined implementation of space-time parallel applications. In: Parallel Computing: Accelerating Computational Science and Engineering (CSE). Advances in Parallel Computing, vol. 25, pp. 273–281. IOS Press (2014). <https://doi.org/10.3233/978-1-61499-381-0-273>
24. Takami, T., Fukudome, D.: An identity parareal method for temporal parallel computations. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) Lecture Notes in Computer Science, vol. 8384, pp. 67–75 (2014). https://doi.org/10.1007/978-3-642-55224-3_7

Chapter 8

A Development Platform for Embedded Domain-Specific Languages



Shigeru Chiba, YungYu Zhuang, and Thanh-Chung Dao

Abstract The use of domain-specific languages (DSLs) is a promising approach to helping programmers write an efficient program for high-performance computing. The programmers would feel difficulties in writing such a program by hand with only low-level abstractions, such as arrays and loops, provided by a general-purpose language. This chapter presents our new implementation technique for domain-specific languages. Since existing techniques are not satisfactory, we developed our technique called *deep reification*. This chapter also presents *Bytespresso*, which is our prototype system to use deep reification. Several Java-embedded DSLs implemented with Bytespresso are presented to assess the effectiveness of deep reification and Bytespresso. Program fragments written in these DSLs are embedded in Java, but they are dynamically off-loaded to native hardware to obtain good execution performance. Since they are embedded in Java, the syntax of Java is reused by those DSLs, and hence the development costs of these DSLs are reduced.

8.1 Embedded Domain-Specific Languages

The usefulness and necessity of domain-specific languages (DSLs) are getting widely recognized in the high-performance computing (HPC) area. DSLs provide higher-level abstractions for a specific domain than general-purpose languages. These abstractions consist of domain-specific data types and operators, and they

S. Chiba (✉)
The University of Tokyo, Bunkyo-ku, Tokyo, Japan
e-mail: chiba@acm.org

Y. Y. Zhuang
National Central University, Taoyuan City, Taiwan (R.O.C.)
e-mail: yungyu@acm.org

T.-C. Dao
School of Information and Communication Technology, Hanoi University of Science and Technology, Hanoi, Vietnam
e-mail: chungdt@soict.hust.edu.vn

hide various techniques for their efficient implementation from the programmers' views. We first briefly overview the motivation of this approach by presenting several well-known implementation techniques for it.

8.1.1 *Domain-Specific Data Types and Operators*

Providing useful data types and operators for a particular application domain is a promising approach to reduce programming costs in high-performance computing. Since the underlying hardware and software stack is getting complicated, writing a raw Fortran or C/C++ program with built-in data types for simple arrays is also getting harder. During writing such a program, programmers have to consider various nonfunctional concerns such as how to exploit parallel computation provided by hardware, how to improve a cache hit ratio, and how to replicate data among distributed nodes. If domain-specific data types and operators are provided to abstract these nonfunctional concerns away, programmers can more focus on application-specific concerns. They would not be bothered about parallelism or distribution and hence reduce programming costs.

Such data types and operators can be implemented as a library in object-oriented languages. Since objects encapsulate implementation details of data manipulation and they provide methods as operators for processing the data, objects and methods are appropriate vehicles for implementing data types and operators for high-performance computing. We do not have to modify a programming language to provide data types and operators. Suppose that we have a library providing `Matrix` and `Vector` objects. Then we would be able to write the following code:

```
d = (a * p) * q
```

Here, `a` is a `Matrix` object, `p` and `q` are `Vector` objects, and `d` is a variable of `double`. The first `*` is a `multiply` method in the `Matrix` class, and the second is a method in `Vector`. If the language does not support operator overloading, the code above would be like this:

```
d = (a.multiply(p)).multiply(q)
```

In either case, implementation details will be hidden from the programmer. `a` might be a sparse matrix and uses the compressed sparse row format. It might be a large matrix, and the data are allocated on multiple distributed nodes where data are exchanged through a MPI library. Since the implementation of the `Matrix` and `Vector` is provided by our library, the programmer only has to select an appropriate implementation, for example, by selecting a subclass `SparseMatrixByMPI` of `Matrix` when creating the object that the variable `a` refers to.

8.1.2 C++ *Template Libraries*

A challenge of the approach to providing domain-specific data types and operators by object orientation is an efficient implementation. Despite vigorous research activities for a long time, object-oriented mechanisms such as method calls tend to involve runtime penalties.

Listing 8.1 N-body simulation in Java

```

1 Func f = (Vec4Array pos, int i, Vec3 pi, float wi) -> {
2   Vec3 a = pos.sum((Vec4Array p, int j, Vec3 pj, float wj)
3     ->{
4       Vec3 r = pi.sub(pj);
5       float ra = reciprocalSqrt(r.mult(r) + soft);
6       return r.scale(wj * (ra * ra * ra));
7     });
8   Vec3 v2 = vel.get(i).add(a.scale(delta)).scale(damping);
9   vel.set(i, v2);
10  return pi.add(v2.scale(delta));

```

Let us see the example taken from our paper [2]. Listing 8.1 written in Java defines the kernel computation of the all-pairs approach to N-body simulation [9]. `Func`, `Vec4Array`, and `Vec3` are data types provided by a library. `Func` is a type of lambda expression. `Vec4Array` is an array of four-dimensional vectors. `Vec3` is a three-dimensional vector. `reciprocalSqrt` is an operator provided by the library as well as methods in the library classes, such as `add` and `sub` (subtraction). Since the programmer does not have to implement these classes, Listing 8.1 is a natural program derived from the following formula:

$$a_i = \sum_j w_j (r \cdot r + \varepsilon^2)^{-\frac{3}{2}} r \quad \text{where } r = p_j - p_i \quad (8.1)$$

$$v'_i = \delta(v_i + \Delta t \cdot a_i) \quad (8.2)$$

$$p'_i = p_i + \Delta t \cdot v'_i \quad (8.3)$$

Here, a_i corresponds to the variable `a`. v_i corresponds to `vel.get(i)` and v'_i corresponds to the variable `v2`. p_i is the value returned by the lambda expression `f`. Note that \sum is represented by the `sum` method on `pos`, which is an array of positions p . The `sum` method takes a lambda expression to compute $w_j (r \cdot r + \varepsilon^2)^{-\frac{3}{2}} r$ and hides how to iterate over array elements.

Although the program is a straightforward translation into Java except that domain-specific operators are not simple symbols but methods, the runtime performance is not satisfactory when compared with the equivalent C code optimized by hand without using higher-level abstraction such as objects and a lambda expression. Figure 8.1 shows the execution performance of the N-body program written in

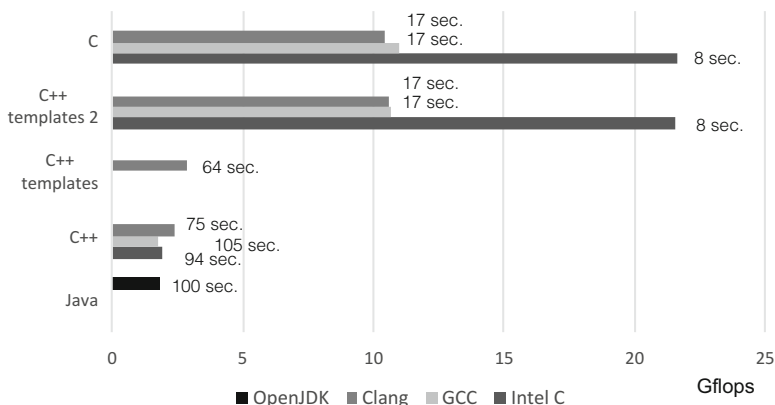


Fig. 8.1 The execution performance of N-body programs

several languages. We used OpenJDK (version 1.8.0_151), the Intel C compiler (version 17.0.1) with `-fast`, GCC 5.4.0 with `-Ofast`, and Clang 5.0.0 with `-Ofast`. All the programs were run on a machine with dual Intel Xeon E5-2637v3. The hardware and compilers that we used for this experiment were slightly upgraded from those in our paper [2]. Although the execution performance depends on the optimization by a compiler and even minor differences of library design, the performance of Listing 8.1 in Java is not comparable with the equivalent C code. Furthermore, a C++ program naively translated from Listing 8.1 was also slow. As presented in Listing 8.2, this C++ program also uses lambda expressions and several objects such as `Vec3` and `Vec4Array`.

Listing 8.2 N-body simulation in C++

```

1 static const auto func
2 = [vel](Vec4Array* pos, int i, Vec3 pi, float wi) -> Vec3
3 {
4     Vec3 a = pos->sum([&pi](Vec4Array* pos, int j, Vec3 pj
5     ,
6     float wj) -> Vec3 {
7         Vec3 r = pi.sub(pj);
8         float ra = 1.0f / sqrtf(r.mult(r) + 0.01f);
9         return r.scale(wj * (ra * ra * ra));
10    });
11    Vec3 v2 = (vel->get(i)).add(a.scale(0.016f)).scale(1.0
12    f);
13    vel->set(i, v2);
14    return pi.add(v2.scale(0.016f));
15 };

```

To avoid runtime penalties due to object orientation, domain-specific data types and operators in C++ tend to be implemented by a template library. In the case of our N-body program, the implementation using templates did not significantly improve the performance. “C++ templates” in Fig. 8.1 indicates the performance of our implementation effort using templates. Since in C++ 17 a lambda expression can be passed as a template argument in only a restricted manner, our template library for the N-body program could be compiled only by Clang, and the performance improvement was not satisfactory. “C++ templates 2” in Fig. 8.1 indicates the performance of another implementation, in which the users do not describe kernel computation in the form of lambda expression. They instead write it in the form of a class. Although its performance is comparable to hand-optimized C code as shown in Fig. 8.1, its programming interface is not satisfactory since the users have to write a new template class.

A main trick of performance optimization by templates is to pass compile-time constant values to a template as template arguments so that the resulting code of instantiating the template will be specialized for those template arguments. Hence when a library designer designs her template library, she has to carefully separate constant part of data types and operators from the rest that are dynamically given at runtime. For our example in Listing 8.1, the `sum` method on `pos` should be specialized for the given lambda expression; the value of the variable `vel` might be given at runtime. Furthermore, a library designer has to consider the constraints on template arguments. For example, a string literal cannot be a template argument. We cannot specialize two objects that refer to each other:

```
template <typename T> class A {
    T b;
};
template <typename T> class B {
    T a;
};
```

If the member variable `b` refers to an instance of `B` and `a` refers to an instance of `A`, then we would want to specialize the type of `b` into `B` and the type of `a` into `A`. However, such specialization would be obtained only by `A<B<A<B< . . . >>>` and `B<A<B< . . . >>>`, which are not feasible.

Furthermore, we cannot predict with confidence that such specialization is properly performed by a C++ compiler and it actually improves execution performance. This depends on the compiler implementation, and a library designer has to struggle with it in a trial-and-error fashion. Limitations of C++ template libraries would be that this trade-off between a user-friendly programming interface and execution performance is not predictable.

Finally, this approach using a C++ template is only applicable when a program is written in C++. Although most programs for high-performance computing could be written in C++, some programs might be written in other languages such as Go. When we want to use OpenCL, this approach is not effective. Since a kernel

function in OpenCL is described in the form of string literals (or character strings), the specialization by C++ templates is not naively applicable to an OpenCL kernel function.

8.1.3 *Pragmas*

Since a C++ library with templates has some limitations, there have been other approaches proposed. The most extreme approach is to develop an external domain-specific language (DSL). It is a new language designed for data types and operators for a particular application-domain. Although it can provide the most natural syntax and potentially the best optimizing compiler, a drawback of this approach is its development cost. Not only a compiler but also a development environment such as an editor have to be developed.

Another approach with lower development costs is to add pragmas to an existing language. Pragmas customize a compilation process, in particular, how target native code is generated, although it does not change syntax. OpenMP is one of the most successful pragma-based systems. A `for` statement with the `parallel` pragma of OpenMP can be regarded as a domain-specific operator for parallel looping although the syntactic expression of the `for` statement is still a normal one. In other words, pragmas implement domain-specific data types and operators by changing the semantics of existing language constructs while keeping the original presentation. In practice, the semantic changes are restricted to be compatible to the original semantics. The programs including pragmas should run even when all the pragmas are ignored. For example, a `for` statement with the OpenMP pragma should be a valid `for` statement even when the pragma is removed.

For the development of pragmas, we can use various compiler frameworks such as Eclipse [14], ROSE [10], and Roslyn [8]. GCC and LLVM compilers can be also used as a compiler framework. They mitigate relatively high development costs of pragmas. Most pragmas can be implemented by a translator of abstract syntax trees (ASTs) from the original trees into transformed trees according to pragmas.

A drawback of this pragma approach is that syntactic presentation is restricted. A pragma can only add extra semantics to an existing language construct such as a `for` statement. The syntax or presentation of that language construct cannot be changed. This restriction on syntax may also complicate the behavior of domain-specific data types and operators by pragmas. For example, since OpenMP pragmas cannot control what programmers write in the body of a `for` statement, the programmers may write an arbitrary body. Hence they may be surprised at unexpected behavior of the statement when the computation in the body has forward or backward dependency. For example,

```
#pragma omp parallel for
for (int i = 0; i < N; i++)
    array[i] += array[i - 1];
```

The result of this `for` statement might be different from the result of the `for` statement without the `pragma` since OpenMP does not guarantee that `array[i]` is updated after updating `array[i-1]`. If custom syntax can be provided for `parallel for` statements, providing such syntax would be preferable so that programmers could not write a body including forward or backward dependence:

```
parallel_for (array, N) {
    return self + 1.0;
}
```

Here, `parallel_for` is a custom `for` statement with new syntax. It updates every element of `array` with size `N`. The new value is obtained by executing the body `self + 1.0`. The built-in variable `self` refers to the old value of each element. Although it enables only a limited kind of computation, the programmers cannot write an expression with forward or backward dependence. Restricting the expressiveness of data types and operators is sometime useful to avoid unexpected behavior.

8.1.4 Deep Embedding

Deep embedding is a technique for implementing an embedded domain-specific language (DSL). An embedded DSL is a library providing domain-specific data types and operators through an application programming interface (API) regarded as a DSL. Deep embedding is a variant of the technique called fluent API [3]. The functionality of a library with a fluent API is invoked by a chain of methods.

```
Matrix m;
Vector v, p, q;
...
q = m.mult(v).add(p);
```

The variables `m` refers to a `Matrix` object, and `v`, `p`, and `q` refer to `Vector` objects. The last line above expresses $q = m * p + v$, matrix-vector multiplication and vector addition. The `mult` method executes multiplication, and the `add` method executes addition. These methods return the resulting vector.

A deep-embedding library, however, provides methods returning an abstract syntax tree (AST). For the example above, both `mult` and `add` return an AST corresponding to an expression for multiplication or addition. Hence, the type of `q` has to be changed from `Vector` to `VectorExpr` (vector expression). To obtain the value of vector type, a method for materialization, for example, `eval`, has to be called on the AST.

```
Matrix m;
Vector v, p, r;
VectorExpr q;
...
q = m.mult(v).add(p);
r = q.eval();
```

The result of $m * v + p$ is stored in `r`. A unique feature of deep embedding is that programmers explicitly specify when an AST is materialized (by calling `eval`) into a value. Hence, a deep-embedding library can generate an optimized program for executing an AST to be materialized when the materialization method such as `eval` is called. Then, it can execute the program generated on demand so that it can get a better execution performance. The generated program does not have to be written in the host language. When a deep-embedding library is for Java, it may generate a C++ or assembly program. It may directly generate native machine code [13].

Since a deep-embedding library generates a program, it is similar to an external DSL compiler, but it is a library embedded in its host programming language. The program generation is performed at runtime as a just-in-time (JIT) compiler does. A deep-embedding library is more portable than external DSLs or the pragma approach; it does not require a custom compiler or language processor.

Although the syntactic presentation of the code above is somewhat ugly, it could be improved, for example, if the host language supports operator overloading:

```
q = (m * v) + p;
r = q.eval();
```

In Scala, dots and parentheses can be omitted when a method takes a single parameter. Hence programmers can write the following code:

```
q = m mult v add p
```

It can be also possible to implement a precedence rule among `mult` and `add` by exploiting types [7]. However, in either cases, the `eval` method is still necessary for explicit materialization.

Explicit materialization is a drawback of deep embedding since programmers are aware of AST construction. They also have to distinguish AST types and value types such as `VectorExpr` and `Vector`. This drawback is mitigated in Lightweight Modular Staging (LMS) [12] by relying on Scala's powerful type system and advanced features such as user-defined implicit conversion (and often Scala-Virtualized compiler extension [11]). Yin-Yang [5] uses Scala's macro system to further mitigate this drawback. These techniques for mitigation, however, are not available in other mainstream programming languages, which do not provide as powerful programming capabilities as Scala's.

8.2 Deep Reification

In our paper [2], we proposed an alternate approach called deep reification. Like deep embedding, deep reification enables a library providing domain-specific data types and operators through a language-like API.

The idea of deep reification is simple. Deep reification is a language mechanism for obtaining the AST of the source code of a given lambda expression (or a function closure) at *runtime*. Deep reification obtains not only the AST of the given lambda expression but also the ASTs of all the methods directly or indirectly invoked from that lambda expression. It can be regarded as a mechanism for obtaining an abstract syntax *forest*. The obtained ASTs also come with runtime values captured by the given lambda expression and accessed from the directly or indirectly invoked methods. All the types appearing in the obtained ASTs are also collected.

Listing 8.3 Deep reification for N-body simulation

```

1 dsl.run(() -> {
2   pos1.tabulate(i -> new Vec4(i, i, i, 2));
3   vel.tabulate(i -> new Vec4(i, i, i, 2));
4   dsl.repeat(R, () -> {
5     pos2.map(f, pos1);
6     pos1.map(f, pos2);
7   });
8   Vec3 g = pos1.sum((Vec4Array pos, int i, Vec3 v, float w)
9     -> new Vec3(v.x / w, v.y / w, v.z / w));
10  Util.print(g.x / N).println();
11 });

```

Listing 8.3 shows an example. It performs the N-body simulation using the lambda expression `f` shown in Listing 8.1. The `run` method on `dsl` performs deep reification. It takes a lambda expression and obtains the AST of its source code. Since this lambda expression refers to `f`, which is another lambda expression in Listing 8.1, the AST of `f` is also obtained. Likewise, other methods such as `repeat` and `map` invoked in Listings 8.1 and 8.3 are also reified and their ASTs are obtained.

Domain-specific data types and operators are implemented by deep reification as follows. Its approach is similar to deep embedding. A code snippet including domain-specific data types and operators is an embedded DSL program. It is written in the form of lambda expression such as one passed to `run` in Listing 8.3. This lambda expression is passed to some method provided by the DSL library and the method extracts the ASTs of that lambda expression and the methods invoked from that lambda expression. The `run` method in Listing 8.3 is such a method provided by a DSL library. Then, the extracted ASTs are translated into an optimized program, which will be executed to get the result of the DSL program written by the user. If a host language is Java, the optimized program translated from the ASTs may be a C++ program. It will be compiled dynamically by an external C++ compiler and run, while the host program in Java is running. The compiled binary would communicate to the Java virtual machine through Java Native Interface (JNI) or inter-process communication such as a socket and a pipe. The latter means allows us to run the compiled binary on a remote machine with rich computational resources. Otherwise, the compiled binary can be run as a stand-alone program without any communication to the host program, while it is running. In this case, the host

program can be regarded as a program generator where the specification of the generated program is embedded. The generated and compiled binary will run after the host program completely finishes.

Any practical implementation of deep reification needs a means for delimiting acquisition of ASTs for methods directly/indirectly invoked by the root lambda expression. One simple means is to use naming conventions. In Java, the methods contained in system packages such as `java.lang.*` can be eliminated from the acquisition. Another approach is to let the application programmers annotate methods to delimit further acquisition of ASTs. When extracted ASTs as a DSL program are translated into an efficient C++ program, some methods will be *native* methods, and their ASTs will not be translated into C++ code *as is*. They will be translated, independently of their ASTs, into predefined code supplied by DSL implementation; further traversal to obtain ASTs will not be necessary. Hence, the annotation to specify such a *native* method can be used to delimit AST acquisition.

A difference from deep embedding is that a DSL program is written by borrowing the syntax of the host language, while it is written in the form of method chaining in deep embedding. Deep reification also borrows parts of the semantics of a host language. In the deep-embedding approach, borrowing host language features such as function calls is difficult in a DSL program. The syntax for a function call has to be explicitly implemented by the DSL. The DSL cannot reuse a host language mechanism similar to a function call. Since deep reification traces a method-call chain and variable accesses referring to the outside of a given closure, DSL implementation can exploit that tracing functionality when implementing its equivalent language constructs such as a function call. Deep reification provides for DSL implementation a smoother connection to its host language.

The approach based on deep reification is similar to a JIT compiler, but the dynamic compilation in this approach would be heavier than typical JIT compilers. Hence, a DSL program in this approach is appropriate for off-loading high-performance computation when executing that computation takes a long time relatively to the compilation time. On the other hand, like the deep-embedding approach, the deep-reification approach is portable since it does not need a custom virtual machine or compiler.

Deep reification is also similar to syntactic macros such as ones found in Lisp. Syntactic macros allow programmers to extract an AST of the expression given as a macro argument. The resulting AST returned by a macro function lexically substitutes the original macro-call expression. Although a syntactic macro can extract only the AST of the expression written as the argument to the macro, deep reification can extract the AST of an expression written in a different place from the place where the execution of deep reification. In Listing 8.3, the `map` method in line 5 and 6 gets a lambda expression referred to by the variable `f` and extracts its AST. The lambda expression does not have to necessarily be directly written in line 5 and 6. If `map` is a macro function, the lambda expression has to be written in line 5 and 6. Furthermore, deep reification allows programmers to write the following method and use it:

```
void mapmap(Func f, Vec4Array pos1, Vec4Array pos2) {
    pos2.map(f, pos1);
    pos1.map(f, pos2);
}
```

Then line 5 and 6 in Listing 8.3 can be replaced with the following single call:

```
mapmap(f, pos1, pos2);
```

Defining a convenience method such as `mapmap` is not possible if `map` is a macro function.

8.3 Bytespresso

To show our idea of deep reification, we implemented a Java library that provides deep reification in Java [2]. Our Java library named *Bytespresso* extracts an AST by bytecode decompilation. It needs to launch the Java virtual machine with the option `jdk.internal.lambda.dumpProxyClasses`. This option generates the bytecode of a dynamically generated lambda expression. For deep reification, Bytespresso reads the bytecode of a given lambda expression and decompiles it to construct an AST. It does not need source code; it only needs Java bytecode.

To support the implementation of an embedded DSL by deep reification, Bytespresso also provides a translator from ASTs to C or CUDA code. The code generated by the translation is normally compiled by an external C (or CUDA) compiler and executed in a separate process from the Java virtual machine. The generated code and the host Java code communicate with each other through a socket for portability.

Delimiting AST acquisition by deep reification, Bytespresso provides the `@Native` annotation. When a method has this annotation, the AST of the method body is not extracted by deep reification. The ASTs for the methods invoked by that method are not extracted either. Although a `@Native` method is translated into a C function by Bytespresso, the body of that C function is the argument to `@Native`. The following method is an example of `@Native` method:

```
@Native("struct timeval time; gettimeofday(&time, NULL); "
        + "return time.tv_sec * 1000000 + time.tv_usec;")
public static long time() {
    return System.nanoTime() / 1000;
}
```

The body of the C function is given as a string literal. Furthermore, Bytespresso also provides the `@Foreign` annotation for delimiting. Unlike `@Native`, the translator provided by Bytespresso does not generate a C function for a `@Foreign` method. A call to this method is translated into a call to the existing C function with the same name. For example,

```
@Foreign public static float sqrtf(float f) {
    return (float)Math.sqrt(f);
}
```

A call to this method is translated into a call to the C function `sqrt` in the standard library. The body of the method is ignored.

The translator provided by Bytespresso does not preserve the original semantics of Java. For example, the generated code does not perform an array boundary checking although DSL designers, who implement their DSL compilers, can modify the translator from ASTs to C so that array boundary checking will be done. Garbage collection is optional for the generated code. If it is required, a conservative garbage collector [1] is used. Note that a program processed by Bytespresso is a DSL program. It should be a different language from Java and can provide different semantics although it has to use the same syntax as Java. The choice of which part of Java's semantics is reused by the DSL is the responsibility of the DSL designer. It should be decided to fit the aim of the DSL.

8.3.1 *N-Body Simulation*

We first show our small array library built with Bytespresso. It provides several classes used in Sect. 8.1.2 such as `Vec4Array` (an array of vectors in four dimensions) and `Vec3` (a vector in three dimensions). `Vec4Array` provides a method for computing sums:

```
@Inline public Vec3 sum(Func f) {
    Vec3 v = new Vec3(0, 0, 0);
    for (int i = 0; i < size; i++) {
        Vec3 vi = get(i);
        Vec3 v2 = f.apply(this, i, vi, getW(i));
        v = v.add(v2);
    }
    return v;
}
```

This is a natural Java program that can be also executed by the Java virtual machine. `get(i)` obtains the *i*-th element (with only three components) of the vector. `getW(i)` obtains the fourth dimension of the *i*-th element. `f.apply` runs the lambda expression *f* with the arguments.

However, this array library is an embedded DSL. Although a program written in this DSL lexically looks like a lambda expression in Java, it is extracted from a host Java program, translated into efficient C++ code, compiled, and executed out of the Java virtual machine. The execution semantics of the DSL code is slightly different from Java's, for example, with respect to array boundary checking.

The vector elements of a `Vec4Array` object are stored in a `FloatArray2D` object. `FloatArray2D` is a class provided by Bytespresso for a two-dimensional array of `float`. The AST-to-C translation of the code related to this object is specially treated, and hence every instance of `FloatArray2D` is translated into a static global variable in C. An instance of `FloatArray2D` has to be created

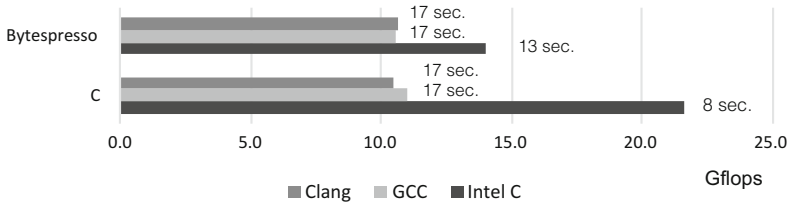


Fig. 8.2 The execution performance of N-body programs by Bytespresso

with its fixed size before deep reification is executed. For example, if the size is 128 by 128, then the instance is translated into:

```
static double gvar[128][128];
```

This is a statically allocated array with a fixed size. It will help the back-end C compiler generate efficient binary.

Note that a DSL program extracted from its host Java program by Bytespresso is compiled into C and executed by a separate process. Therefore, the host Java program and the DSL program run in separate execution environments, and they do not share objects or any type of variables. All the Java objects referred to by a DSL program are also extracted by Bytespresso. Then their copies are made and embedded into a generated C program as a statically allocated variable. When data have to be exchanged between a host Java program and a DSL program, they have to be explicitly passed by remote method invocation. The AST-to-C translator of Bytespresso supports remote method invocation between host and DSL programs. A method annotated with `@Remote` is treated as a method that can be remotely called. Bytespresso currently supports the only primitive data types and arrays as a parameter of remote method invocation.

Figure 8.2 shows the execution performance of our array library with Bytespresso. We ran the simple N-body simulation shown in Listing 8.3 on the same hardware and software as in Fig. 8.1 in Sect. 8.1.2. The performance of Bytespresso is comparable to the C program written by hand except when the back-end C compiler is the Intel C compiler. It seems that the Intel compiler was able to effectively apply SIMD vectorization when the source code was written by hand and had a simpler structure.

The performance of Bytespresso shown in Fig. 8.2 was achieved by aggressive inlining specified by the annotation `@Inline` for the `sum` method. Our array library supports not only single-thread execution but also OpenMP and CUDA. It provides several subclasses of `Vec4Array`, and, by switching them, programmers can change which kind of program is generated from a DSL program written as a lambda expression in Java. When a subclass of `Vec4Array` for CUDA is selected, the AST-to-C translator of Bytespresso generates a CUDA program that computes the `sum` method by using a GPGPU. Therefore, a call to the `sum` method may cause dynamic method dispatch. Bytespresso aggressively inlines a method and attempts to statically resolve such dynamic dispatch. The call to the `sum` method

in Listing 8.1 and all other calls that might have been dynamic calls were statically resolved by Bytespresso and translated into normal calls to C functions specialized for the call sites. Some calls to the specialized functions are further inlined. To help this optimization, the `final` modifier should be added to object fields if possible. Furthermore, Bytespresso provides `@Final` annotation to specify that the value of a field never changes, while a DSL program is running. Since it can be updated while a host Java program is running before deep reification is performed, `@Final` is different from the `final` modifier.

Listing 8.4 2-dimensional 5-point stencil

```
Boundary b = new FixedEndBoundary();
b.initializer(new Initializer() { ... });
GridFloat2D grid = new CpuGridFloat2D(xsize, ysize, b);
Initializer init = new Initializer() {
    public float value(int i, int j) { return 273.15f; }
};
Kernel k = new Kernel() {
    public float newValue(Float2Array oldValue, Cursor cur,
        int t, Reduction r) {
        float v = c0 * (cur.north(oldValue) + cur.south(oldValue))
            + c1 * (cur.east(oldValue) + cur.west(oldValue))
            + c2 * cur.self(oldValue);
        return v;
    }
};
grid.initialize(init)
    .each(Reduction.NO, 1, Predicate.FOREVER, k)
    .repeat(N, new MPIDriver(nodes));
```

8.3.2 Auto-Parallelization

Another example is a simple framework for stencil computation. It is an embedded DSL, and a program written in this DSL is translated into an efficient C program to be run. The abstraction provided by the framework hides all the details of efficient implementation. The framework users do not have to care about the implementation details.

Stencil computation is a well-known programming model useful for, for example, solving a partial differential equation. Listing 8.4 is a program using our framework for stencil computing. It performs five-point stencil computation in single precision. It first constructs a two-dimensional grid with a concrete boundary condition. In Listing 8.4, we constructs two-dimensional grid, which is a `CpuGridFloat2D` object with a `FixedEndBoundary` object. Then the `initialize` method on

a grid object registers an initializer that sets each grid point to an initial value, and the `each` method registers a kernel function for stencil computation. Finally, the `repeat` method performs deep reification and generates a C program. Since a `MPIDriver` object is given to the `repeat` method in Listing 8.4, the generated program is an MPI program in C. It is supposed to be compiled and submitted to a job queue of supercomputer after the Java program in Listing 8.4 finishes.

The kernel function receives the old values of a grid (`oldValue`) and the current position (`cur`). It has to return a new value at the current position of the grid. `cur.north` obtains the old value at the upper position, and `cur.self` obtains the old value at the current position. The other parameters `t` and `r` are the current time and an object for computing reduction, which is not specified in Listing 8.4.

This framework for stencil computation provides several components, and the users can write their application program by selecting appropriate components. They can choose boundary conditions and how their program is executed, by CPU, GPGPU, or MPI. Then the framework generates a program for the given configurations.

Listing 8.5 The Himeno benchmark using our framework

```
grid.initialize(cInitPressure)
  .each(Reduction.SUM, 1, Predicate.FOREVER, new Kernel()
    {
      public float newValue(FloatArray3D p, Cursor cur, int
t,
                          Reduction r) {
        float s0 = cur.self(a0) * cur.east(p)
          + cur.self(a1) * cur.south(p)
          + cur.self(a2) * cur.down(p)
          + cur.self(b0)
            * (cur.southeast(p) - cur.northeast(p)
              - cur.southwest(p) + cur.northwest(p)
            )
          + cur.self(b1)
            * (cur.downsouth(p) - cur.downnorth(p)
              - cur.upsouth(p) + cur.upnorth(p)
            )
          + cur.self(b2)
            * (cur.downeast(p) - cur.downwest(p)
              - cur.upeast(p) + cur.upwest(p)
            )
          + cur.self(c0) * cur.west(p)
          + cur.self(c1) * cur.north(p)
          + cur.self(c2) * cur.up(p)
          + cur.self(wrk1);
        float ss = (s0 * cur.self(a3) - cur.self(p))
          * cur.self(bnd);
        r.apply(ss * ss);
        return cur.self(p) + omega * ss;
      }
    })
  .repeat(N, drv);
```

An interesting research question is whether this framework using Bytespresso can generate a program appropriate for the given configuration, in particular, a back-end compiler. To reduce development costs, existing software tools should be reused if they are appropriate to use, and a generated program should be optimizable easily by such a back-end compiler. For example, there are automatic parallelizing compilers available.

To examine whether our framework can generate a program that such a compiler can parallelize, we rewrote the Himeno benchmark [4] to use our framework and ran it on the Fujitsu FX10 supercomputer. The back-end compiler was the Fujitsu C compiler. To exploit its automatic parallelization, a compiled program has to be a *good* one that the compiler can easily analyze and parallelize. To generate a *good* program, our framework applies function inlining to the whole kernel loop in the framework implementation. It also transforms an object used for reduction into a set of local variables. This technique is often known as object inlining. Without this transformation, the back-end compiler could not parallelize the code. The framework also allocates global variables in a generated C program, which holds grid data, in a specified order on memory. Since all these optimization techniques are specific to the Fujitsu C compiler, we developed framework components for that compiler so that the framework users can choose when their target machine is the Fujitsu FX10 supercomputer.

The Himeno benchmark runs a single three-dimensional Jacobi kernel. The problem size was XL ($512 \times 512 \times 1024$). The original benchmark is written in C with MPI, and each MPI process is single-threaded in single precision. The benchmark score mainly reflects the memory bandwidth. Listing 8.5 shows the kernel of the benchmark program. The variables `a0`, `a1`, and so forth refer to coefficient matrices, which are `FloatArray3D` objects created in the benchmark program.

Figure 8.3 shows the result. Bytespresso could successfully achieve comparable performance with the hand-optimized C code. Every node of the machine had 32 GByte memory and one SPARC64 IXfx (1.848 GHz) processor with 16 cores. The back-end compiler was Fujitsu C compiler 1.2.1. The compiler option `-Kfast,parallel,noprefetch,ocl` was given. The page size was set to 256 MB.

8.3.3 NAS Parallel Benchmarks

We also wrote a vector matrix library built with Bytespresso. It runs on the MPI environment, and hence it can process large vector and matrices. Like our small array library used for the N-body simulation in Sect. 8.3.1, it is an embedded DSL. A program written in this DSL looks like a normal Java program, but it is executed after being translated into a C program using MPI. The library users do not have to care about data distribution or exchanges through MPI. Such details are hidden by the library.

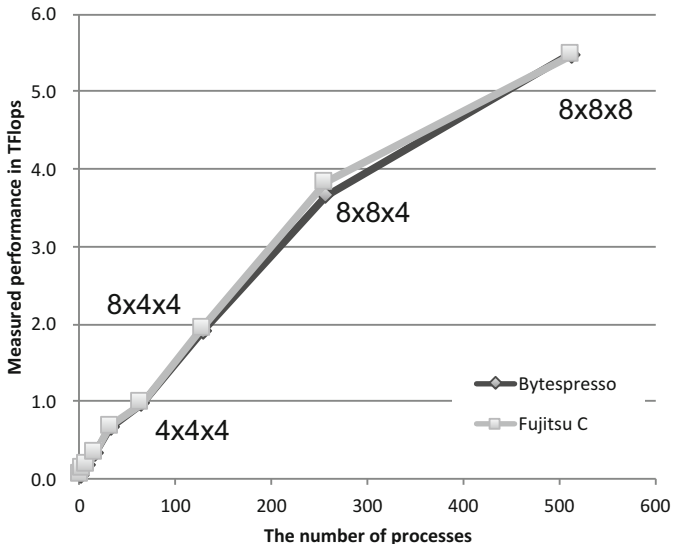


Fig. 8.3 Strong scale performance of the Jacobi kernel of Himeno on FX10

Listing 8.6 The CG benchmark using our framework

```

public double conj_grad(Vector x, Vector z, Matrix.Sparse a,
                        Vector p, Vector q, Vector r, double rnorm)
{
    q.set(0.0);
    z.set(0.0);
    r.set(x);
    p.set(r);
    double rho = r.norm();
    for (int cgit = 1; cgit <= cgitmax; cgit++) {
        q.setToMult(a, p); // q = A * p
        double d = inner(p, q); // d = p * q
        double alpha = rho / d;
        z.setToAdd(z, alpha, p); // z = z + alpha * p
        r.setToAdd(r, -alpha, q); // r = r - alpha * q
        double rho0 = rho;
        rho = r.norm(); // rho = r * r
        double beta = rho / rho0;
        p.setToAdd(r, beta, p); // p = r + beta * p
    }
    r.setToMult(a, z); // r = A * z
    r.setToSub(x, r); // r = x - r
    double sum = r.norm(); // sum = r * r
    return Util.sqrt(sum);
}
    
```

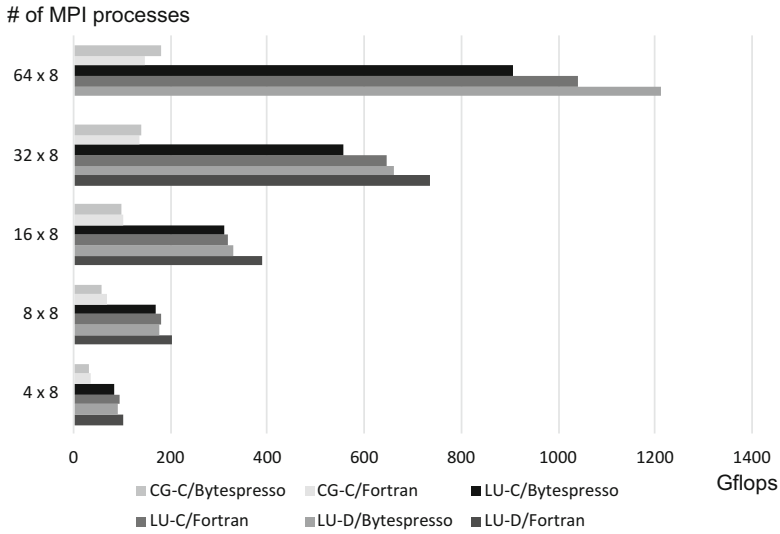



Fig. 8.4 NAS Parallel Benchmarks on TSUBAME 3.0

To evaluate the performance of this vector matrix library, we rewrote the CG benchmark from the NAS parallel benchmarks 3.0 so that it will use our library. Since the library provides high-level abstraction, vectors and matrices, the CG benchmark became a largely simplified program than the original one containing a large number of `do` loops. The original program `cg-omp.f` contains 1156 lines, while ours corresponding one contains only 751 lines including comments but excluding the library code. Listing 8.6 shows a main part of the program, the conjugate gradient routine. Since the generated program by our library runs using MPI, for example, the `setToMult` method for computing matrix-vector multiplication synchronously runs with other MPI processes. It first computes the multiplication of a sub-matrix and a sub-vector stored on local memory, and then it exchanges the results among the other nodes through `MPI_Irecv`, `MPI_Send`, and `MPI_Wait`. These MPI primitives are implemented by `@Native` methods of Bytespresso.

Besides the CG benchmark, we also wrote a program equivalent to the LU benchmark of NAS parallel benchmark suites. This program does not use high-level abstraction such as a matrix; it directly uses four-dimensional double-precision arrays as the original benchmark program does. We wrote this benchmark program to examine the basic execution performance of programs generated by the AST-to-C translator that Bytespresso provides.

The execution performance of the CG and LU benchmark programs in Fig. 8.4. We ran them on the TSUBAME 3.0 supercomputer at Tokyo Tech. Each node of the machine has dual Intel Xeon E5-2680v4 processors with 256 GB memory. For compilation, we used GCC 4.8.5 with the OpenMPI library. The optimization option

-Ofast was given to the compiler. We created eight MPI processes per node, and the maximum number of nodes we used was 64. The CG benchmark using Bytespresso achieved comparable performance to the original Fortran program although the slow down due to Bytespresso was not negligible for the LU benchmark.

8.3.4 ExaStencil

Our last example is an embedded version of the ExaSlang 4 DSL [6]. ExaSlang is an external DSL for stencil computation. It is being developed by the SPPEXA ExaStencil project. It provides stencil operators and data grids. It also provides a loop-over statement, which abstracts a (sequential or parallel) iteration over a grid.

Since ExaSlang supports multigrid methods, a set of multi-level data grids is called a field. A data grid at a particular level is specified by @. For example, GradientX@finest represents a data grid at the finest level of the GradientX field. A stencil operator consists of stencil coefficients.

```
Stencil SmootherStencil_u@all {
  [ 1, 0] => -1.0
  [-1, 0] => -1.0
  [ 0, 1] => -1.0
  [ 0,-1] => -1.0
  [ 0, 0] => 4.0*alpha + GradientX@current * GradientX@current
}
```

This declares a stencil operator named SmootherStencil_u available at all levels. The right operand of => is a coefficient at the position specified by the left operand of =>. The declaration above defines the following stencil:

$$\begin{pmatrix} & -1 & \\ -1 & 4\alpha + I_x^2 & -1 \\ & -1 & \end{pmatrix}$$

Here, I_x is the element of GradientX at the current position. The stencil can be used in a function:

```
Function Smoother@all ( ) : Unit {
  // omitted
  loop over Flow_u@current {
    Flow_u[next]@current = Flow_u[active]@current
      + ((1.0 / diag(SmootherStencil_u@current))
        * (RHS_u@current
          - SmootherStencil_u@current * Flow_u[active]@current
          - GradientX@current * GradientY@current * Flow_v[
            active]@current))
  }
  // omitted
}
```

This `Smoother` function is available at all levels, and it iterates the loop body over the current level of the `Flow_u` field. In the body, the `SmootherStencil_u` for the current level is applied to the current level of (the active slot of) the `Flow_u` field.

We implemented an embedded version of this DSL by using Bytespresso. In the embedded version, stencil operators and fields are defined as Java objects. For example, the `SmootherStencil_u` shown above is written as follows:

```
final Stencil smootherStencil_u
= new StencilBuilder()
  .add(1, 0, -1.0)
  .add(-1, 0, -1.0)
  .add(0, 1, -1.0)
  .add(0, -1, -1.0)
  .add(0, 0, (current, x, y) ->
    4.0 * alpha + gradientX.at(current).get(x, y)
    * gradientX.at(current).get(x, y))
  .build();
```

To obtain better performance, an instance of `CustomStencil` should be used:

```
final CustomStencil smootherStencil_u = new CustomStencil() {
  public double calc(final LayeredNodeField layeredField,
    final int current, final NormalIndex x,
    final NormalIndex y) {
    return -1.0 * (layeredField.get(current, x, 1, y, 0)
      + layeredField.get(current, x, -1, y, 0)
      + layeredField.get(current, x, 0, y, 1)
      + layeredField.get(current, x, 0, y, -1))
      + (4.0 * alpha + gradientX.get(current, x, y)
        * gradientX.get(current, x, y))
      * layeredField.get(current, x, 0, y, 0);
  }
};
```

This object directly represents the expression computed by a stencil operator. Likewise, the `Smoother` function is written in our DSL as follows:

```
@Inline public void smoother(int current) {
  // omitted
  flow_u.next(current).loopOver(current, (c, x, y) ->
    flow_u.active(c).get(c, x, y)
    + smootherStencilDiagInv_u.calc(
      (rhs_u.get(c, x, y)
        - smootherStencil_u.calc(flow_u.active(c), c, x, y))
        - gradientX.get(c, x, y) * gradientY.get(c, x, y)
        * flow_v.active(c).get(c, x, y)),
      c, x, y));
  // omitted
}
```

This function takes a parameter specifying a level. Its expression is more verbose than ExaSlang but keeps the same abstractions.

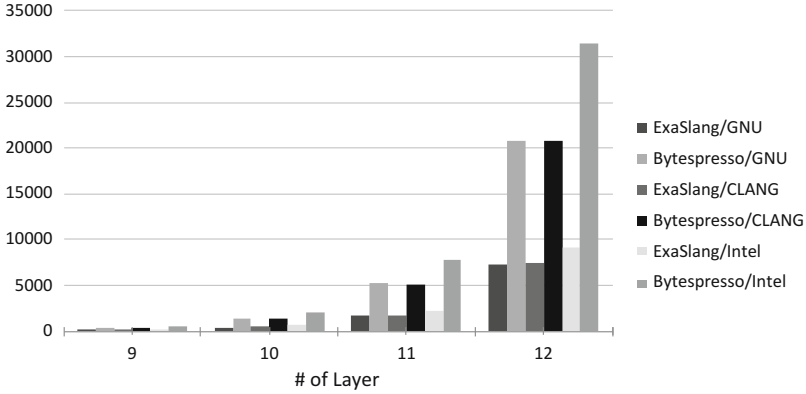


Fig. 8.5 Execution time of ExaSlang programs (msec)

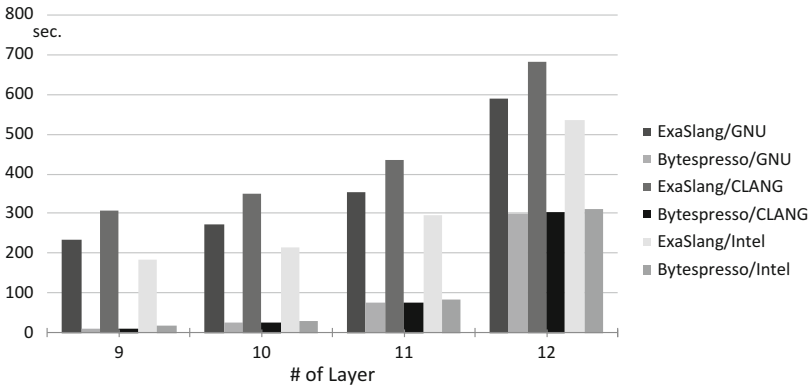


Fig. 8.6 Compile and execution time of ExaSlang programs (sec)

We compared the execution performance of programs that compute two-dimensional optical flow by the multigrid method. One program was written in the original ExaSlang language, while the other was written in our embedded DSL. Both are single-threaded. We compiled the two programs by GCC 5.4.0, Intel C compiler 17.0.1, and Clang 5.0.0 with `-O3`, and ran them on the machine with Intel Xeon E5-2637v3. We examined with different numbers of layers for the multigrid method. Figure 8.5 shows the execution time of the kernel part of the two programs. The program written in our embedded DSL was only three times slower than the program in the original ExaSlang language. Figure 8.6 shows the sum of the compilation time and the execution time of the two programs. Since the compilation by the current ExaSlang compiler is slow, our embedded DSL was rather faster.

8.4 Summary

This chapter presented *Bytespresso*, a prototype system to use our deep-reification technique, in Java. Bytespresso allows DSL developers to easily implement efficient DSLs embedded in Java. The embedded DSL code is dynamically extracted and off-loaded from the Java virtual machine onto native hardware. The DSL developers can exploit an existing tool chain, including an external optimizing compiler, when off-loading DSL code. Since the syntax of the DSLs are borrowed from Java's and a few language mechanisms in Java, such as a method call, are reused, the development costs of the DSLs can be reduced.

Acknowledgements I would like to thank Maximilian Scherr and Toshiyuki Takahashi for their various contributions to this work. This is partly supported by JST/DFG SPPEXA ExaStencil project. We deeply thank Christian Lengauer, Sebastian Kuckuk, Christian Schmitt, Matthias Bolten, Frank Hannig, and Harald Köstler. We also thank Shuichi Chiba at Fujitsu Ltd. for the experiment on FX10.

References

1. Boehm, H.J., Weiser, M.: Garbage collection in an uncooperative environment. *Softw. Pract. Exp.* **18**(9), 807–820 (1988)
2. Chiba, S., Zhuang, Y., Scherr, M.: Deeply reifying running code for constructing a domain-specific language. In: *Proceedings of the 13th International Conference on Principles and Practices of Programming on the Java Platform: Virtual Machines, Languages, and Tools, PPPJ '16*, pp. 1:1–1:12. ACM, New York (2016)
3. Fowler, M.: *Fluentinterface* (2005). <https://www.martinfowler.com/bliki/FluentInterface.html>
4. Himeno, R.: Himeno benchmark (2001). <http://accr.riken.jp/2444.htm>
5. Jovanovic, V., Shaikhha, A., Stucki, S., Nikolaev, V., Koch, C., Odersky, M.: Yin-yang: concealing the deep embedding of DSLs. In: *Proceedings of the 2014 International Conference on Generative Programming: Concepts and Experiences, GPCE 2014*, pp. 73–82. ACM, New York (2014)
6. Kuckuk, S., Haase, G., Vasco, D.A., Köstler, H.: Towards generating efficient flow solvers with the ExaStencils approach. *Concurr. Comput. Pract. Exp.* **29**(17), 4062:1–4062:17 (2017)
7. Nakamaru, T., Ichikawa, K., Yamazaki, T., Chiba, S.: Silverchain: A fluent API generator. In: *Proceedings of the 16th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences, GPCE 2017*, pp. 199–211. ACM, New York (2017)
8. .NET Foundation: The .net compiler platform “roslyn” (2014). <https://github.com/dotnet/roslyn>
9. Nyland, L., Harris, M., Prins, J.: Fast n-body simulation with CUDA. In: Nguyen, H. (ed.) *GPU Gems 3*, chap. 31, pp. 677–695. Addison-Wesley, Boston (2007)
10. Quinlan, D., Schordan, M., Miller, B., Kowarschik, M.: Parallel object-oriented framework optimization. *Concurr. Comput. Pract. Exp.* **16**(2–3), 293–302 (2004)
11. Rompf, T., Amin, N., Moors, A., Haller, P., Odersky, M.: Scala-virtualized: linguistic reuse for deep embeddings. *Higher-Order Symb. Comput.* **25**(1), 165–207 (2012)

12. Rompf, T., Odersky, M.: Lightweight modular staging: a pragmatic approach to runtime code generation and compiled DSLs. In: Proceedings of the Ninth International Conference on Generative Programming and Component Engineering, GPCE '10, pp. 127–136. ACM, New York (2010)
13. Rompf, T., Sujeeth, A.K., Brown, K.J., Lee, H., Chafi, H., Olukotun, K.: Surgical precision JIT compilers. In: Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, pp. 41–52. ACM, New York (2014)
14. The Eclipse Foundation: Eclipse IDE (2001). <http://www.eclipse.org>

Chapter 9

Xevolver: A User-Defined Code Transformation Approach to Streamlining Legacy Code Migration



Hiroyuki Takizawa, Reiji Suda, Daisuke Takahashi, and Ryusuke Egawa

Abstract Since different systems usually require different performance optimizations, an application code is likely “specialized” for a particular system configuration to fully extract the system performance. This is one major reason why migration of an existing application code, so-called legacy code migration, is so labor-intensive and error-prone especially in the high-performance computing (HPC) area. To make matters worse, the diversity of system architectures would increase the number of system configurations that have to be considered during the life of an application. As a result, the increasing system complexity and diversity will force programmers to further invest enormous time and effort on HPC application development and maintenance. For long-term software development and maintenance, an HPC application code should not be optimized for a particular system configuration. However, simply excluding system-awareness from the code just results in reducing the performance, because system-awareness can be regarded as the necessary information to exploit the performance of the target system. The goal of this project is to express system-awareness separately from the HPC application code to achieve high performance while keeping the code maintainability. To this end, we have been developing a code transformation framework, Xevolver, so that users can express system-awareness as user-defined code transformation rules separately from HPC application codes. By defining custom code transformations for individual cases, an HPC application code itself does not need to be modified for each system configuration and is transformed for each system by using the code transformations defined separately from the code.

H. Takizawa (✉) · R. Egawa
Cyberscience Center, Tohoku University, Sendai, Japan
e-mail: takizawa@tohoku.ac.jp; egawa@tohoku.ac.jp

R. Suda
Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, Japan
e-mail: reiji@is.s.u-tokyo.ac.jp

D. Takahashi
Center for Computational Sciences, University of Tsukuba, Tsukuba, Japan
e-mail: daisuke@cs.tsukuba.ac.jp

9.1 Introduction

As the end of Moore's law is approaching, high-performance computing (HPC) system architectures will become more complicated and heterogeneous so as to satisfy never-ending demands for achieving higher performance than ever before without relying on advances in device technology. Moreover, due to some severe limitations such as power consumption, HPC system architecture design cannot take the so-called "one-size-fits-all" approach whose ultimate goal is to enable a single architecture to achieve high performance on any applications. As a result, future HPC system architectures will diverge to adapt to their important applications. In this way, the complexity and diversity of HPC systems are increasing more and more in an upcoming extreme-scale computing era. This results in also complicating and diverging system software stacks such as runtimes, compilers, libraries, languages, and so on. While HPC systems could sometimes change revolutionarily in terms of both hardware and software, it is not affordable to rewrite each application for every new system. Therefore, it is often required to incrementally or evolutionarily migrate an existing application to newly available systems while carefully checking if its behavior is unchanged.

An HPC system is a huge collection of various hardware and software components that can affect the performance of an HPC application code. The performance characteristics of an application code could change drastically depending on various factors. The increasing system complexity makes it harder to exploit the full potential of an HPC system. Since different systems usually require different performance optimizations, an application code is likely "specialized" for a particular system configuration of specific system architecture, system scale, software stack, and so on, to fully extract the system performance. This is one major reason why migration of an existing application code, so-called *legacy code migration*, is so labor-intensive and error-prone especially in the HPC area. To make matters worse, the diversity of system architectures would increase the number of system configurations that have to be considered during the life of an application. Accordingly, the increasing system complexity and diversity will force programmers to further invest enormous time and effort for HPC application development and maintenance.

As with the survival in the nature, HPC applications also have to be adaptable to various system changes to survive in the extreme-scale computing era. Today, an HPC application is too specialized for a particular system, and system-specific performance optimizations are directly applied to the application code. As a result, system-specific performance optimizations are tightly interwoven with the algorithm description, resulting in severely degrading the adaptability to system changes. For streamlining the process of legacy code migration, therefore, one important research issue is how to prevent developing an HPC application with being aware of only a particular system configuration. Such *system-awareness* should not be expressed in an HPC application code.

Because of the system diversity mentioned above, an HPC application code should not be optimized for a particular system configuration. However, simply

excluding system-awareness from the code just results in reducing the performance, because system-awareness can be regarded as the necessary information to exploit the performance of the target system. Therefore, the goal of this project is to express system-awareness separately from the HPC application code to achieve high performance while keeping the code maintainability. To this end, we have been developing a code transformation framework, *Xevolver* [18, 21], so that users can express system-awareness as user-defined code transformation rules separately from HPC application codes. By defining custom code transformations for individual cases, an HPC application code itself does not need to be modified for each system configuration and is transformed for each system by using the code transformations defined separately from the code.

9.2 The Xevolver Code Transformation Framework for Separation of System-Awareness from Application Codes

In practice, there are repetitive patterns in the code modifications for system-specific performance optimization. We can hence assume that those code modifications could be replaced with a smaller number of code transformations. Under this assumption, we have been developing Xevolver to enable users to express their own code optimizations for special demands of individual systems and individual applications [18, 21]. Instead of simply modifying a code by hand, users can easily define custom code transformations to optimize and specialize an application code for a particular system. In Xevolver, such code transformation rules can be defined separately from an application code. Accordingly, Xevolver enables to express system-specific and/or application-specific code optimizations separately from application codes.

To define a user-defined code transformation rule, what users have to do is to simply write two versions of a code, or a code pattern; the original version and its transformed version. Then, one of our tools named *Xevtgen* [16] generates a machine-usable code transformation rule from such a simple description about the code transformation. Therefore, users do not need to care about any special knowledge internally required for implementing their code transformations.

The rest of this section gives a small tutorial on how to use Xevolver.

9.2.1 Commands

A high-level interface of Xevolver, named *Xevtgen* [16], provides three important commands, *xevparse*, *xevunparse*, and *xevtgen*. The first command, *xevparse*, is used to translate a Fortran code to its abstract syntax tree in an XML format, called an *XML AST*. Inversely, the second command, *xevunparse*,

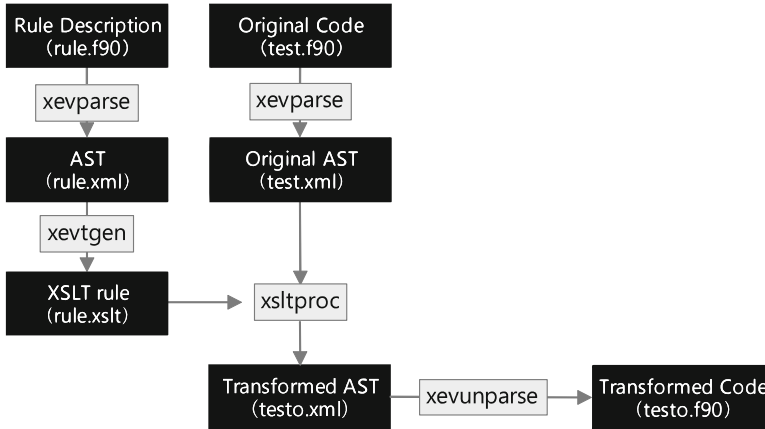


Fig. 9.1 Code transformation by using the three commands provided by Xevolver

is to translate an XML AST to its corresponding Fortran code. In Xevolver, a code transformation rule is also written in Fortran. If an XML AST represents a code transformation rule, the third command, `xevtgen`, translates the AST to an XSLT rule [6], which is a standard XML format to describe XML data conversion. Accordingly, in Xevolver, a target code and its transformation rules are both written in Fortran and converted into XML. As a code transformation is executed as an XML data conversion, various XSLT processors such as `xsltproc` [22] are available for the transformation. Figure 9.1 illustrates how to use these three commands. In the figure, a code, `test.f90`, is transformed to its transformed version, `testo.f90`, and the code transformation rule is defined in `rule.f90`.

9.2.2 Rule Description

Figure 9.2 shows an example of code transformation rules written in Fortran with a special directive starting with `!$xev tgen`. In the example, `!$xev tgen` is followed by `trans` and `exp`. Here, `trans` indicates that the directive defines a transformation rule, and `exp` means that the transformation is applied to an expression. Then, a pair of `src('i**2')` and `dst('i*i')` represents a rule that replaces expression `i**2` with another expression of `i*i`, because `src` and `dst` are clauses indicating the source and destination code patterns of a code transformation, respectively. A string surrounded by back quotations is an expression written in the Fortran syntax.

As illustrated in Fig. 9.1, an XSLT rule for transforming XML ASTs is generated by running the `xevparse` command to convert the code in Fig. 9.2 to its XML AST

```

1 program rule1
2
3 !$xev tgen trans exp src('i**2') dst ('i*i')
4
5 end program

```

Fig. 9.2 A simple rule of replacing a particular expression

```

1 program rule2
2 integer :: i,j,k,inp
3 integer :: ary(1000)
4
5 !$xev tgen trans stmt src begin
6 do k=1,10
7 do j=1,10
8 do i=1,10
9 inp=(k-1)*100+(j-1)*10+i
10 ary(inp)=inp
11 end do
12 end do
13 end do
14 !$xev end tgen src
15
16 !$xev tgen trans stmt dst begin
17 do inp=1,1000
18 ary(inp)=inp
19 end do
20 !$xev end tgen dst
21
22 end program

```

Fig. 9.3 A simple rule of changing loop structures

and then executing the `xevtgen` command on the XML AST. Therefore, by just writing two versions of an expression, users can transform the original expression to another one.

Figure 9.3 shows another example of code transformation rules. In the rule, `!$xev tgen trans` in Line 5 is followed by `stmt`, and thus the rule is applied to statements. Specifically, a loop nest indicated with the `src` clause is transformed to another one specified by the `dst` clause. By using the rule, the triple-nested loops described in Lines 6–13 are transformed into a single-nested loop in Lines 17–19.

Although both of the two rules in Figs. 9.2 and 9.3 can actually work, each of the rules is applied only to a very specific code fragment and hence not useful. For example, if the rule in Fig. 9.2 is applied to the code in Fig. 9.4, only `i**2` is transformed, and other similar expressions such as `j**2`, `(k+1)**2`, and `inp**2` are unchanged, even though users may intend to replace those expressions in the same way. Similarly, the rule in Fig. 9.3 does not transform the loop nest in Fig. 9.4 even though users may intend to transform any triple-nested loops in the same way; the rule can transform a loop nest only if the AST of the loop nest is exactly the same as that of the loop nest written in Lines 5–14. Consequently, the rules in Figs. 9.2 and 9.3 can transform only particular code fragments and can be seen as simple text replacement.

```

1  program test
2  integer :: data(1000)
3  integer :: i = 1, j = 1, k = 1, inp
4
5  data(1) = i**2 + j**2 + (k + 1)**2
6  do k = 1, 10
7    do j = 1, 10
8      do i = 1, 10
9        inp = (k-1)*100+(j-1)*10+i
10       data(inp) = inp**2
11     end do
12   end do
13 end do
14 end program

```

Fig. 9.4 A Fortran code to be transformed

```

1  program rule3
2
3  !$xev tgen var(il) exp
4  !$xev tgen trans exp src('il**2') dst('il*il')
5
6  end program

```

Fig. 9.5 A rule of replacing a code pattern

To generalize a rule so as to transform a certain *code pattern* to another, Xevolver provides special variables called *Tgen variables* that match any expressions or statements. In Line 3 of Fig. 9.5, a Tgen variable named *il* is defined to represent any kinds of expressions including variable references. Then, in Line 4, a code transformation rule is expressed by using the Tgen variable. Since *il* matches any expressions, the rule defined in Fig. 9.5 can transform all of *i**2*, *j**2*, $(k+1)**2$, and *inp**2* in Fig. 9.4. Similarly, a group of statements such as a loop body can be represented as a *Tgen list* of statements. A Tgen list of statements is a kind of Tgen variables and matches zero or more statements. For example, although the rule in Fig. 9.6 is similar to that in Fig. 9.3, one Tgen list defined in Line 5 is used to represent the loop body. As a result, the rule is applied to a loop nest as long as the loop structure and the first statement in the loop body are the same as the source code pattern in Lines 7–16 of Fig. 9.6. In this way, users can generalize a code transformation rule so that the rule can be reusable for other similar cases.

If either *i*, *j*, or *k* is used in the second statement or later within the loop body in Fig. 9.4, the code transformation rule in Fig. 9.6 changes the behavior of the code. Thus, the rule should be applied only if those variables do not appear within the loop body except for the first statement. A *Tgen condition* is provided to represent such a condition. Figure 9.7 shows an example of code transformation rules using Tgen conditions. In Lines 4–6, Tgen conditions, *has_i*, *has_j*, and *has_k*, are defined to indicate expressions containing variables, *i*, *j*, and *k*, respectively. Then, in Line 8, *stlc* is defined as a Tgen list under condition `cond(not(or(has_i,has_j,has_k)))`, which means a negation of logical sum of the above three Tgen conditions. Hence, *stlc* means a list of statements

```

1  program rule4
2  integer :: i,j,k,inp
3  integer :: ary(1000)
4
5  !$xev tgen list(stl) stmt
6
7  !$xev tgen trans stmt src begin
8  do k=1,10
9  do j=1,10
10 do i=1,10
11 inp=(k-1)*100+(j-1)*10+i
12 !$xev tgen stmt(stl)
13 end do
14 end do
15 end do
16 !$xev end tgen src
17
18 !$xev tgen trans stmt dst begin
19 do inp=1,1000
20 !$xev tgen stmt(stl)
21 end do
22 !$xev end tgen dst
23
24 end program

```

Fig. 9.6 A rule of changing loop structures with any loop bodies

```

1  program rule5
2  integer :: i, j, k, inp
3
4  !$xev tgen condef(has_i) contains exp('i')
5  !$xev tgen condef(has_j) contains exp('j')
6  !$xev tgen condef(has_k) contains exp('k')
7
8  !$xev tgen list(stlc) stmt cond(not(or(has_i, has_j, has_k)))
9
10 !$xev tgen trans stmt src begin
11 do k=1,10
12 do j=1,10
13 do i=1,10
14 inp=(k-1)*100+(j-1)*10+i
15 !$xev tgen stmt(stlc)
16 end do
17 end do
18 end do
19 !$xev end tgen src
20
21 !$xev tgen trans stmt dst begin
22 do inp=1,1000
23 !$xev tgen stmt(stlc)
24 end do
25 !$xev end tgen dst
26
27 end program

```

Fig. 9.7 A Tgen list of statements satisfying some conditions

not containing neither *i*, *j*, nor *k*. By using `stlc`, the rule defined in Lines 10–25 transforms a loop nest only if its loop body satisfies the condition given in Line 8.

If a rule should be applied to a specific code region, the region is expressed as a *Tgen context*, and the rule is defined for the Tgen context. Figure 9.8 shows an

```

1  program rule6
2
3  !$xev tgen var(e1,e2,e3,e4) exp
4  !$xev tgen list(stl1) stmt
5  !$xev tgen ctxdef(c1) stmt begin
6  do e1 = e2, e3, e4
7  !$xev tgen stmt(stl1)
8  end do
9  !$xev end
10
11 !$xev tgen var(i1) exp
12 !$xev tgen trans exp src('i1**2') dst('i1*i1') context(c1)
13
14 end program

```

Fig. 9.8 A rule using a Tgen context

example of using a Tgen context. In the rule, four Tgen variables are defined in Line 3, and a Tgen list is defined in Line 4. Then, a Tgen context, `c1`, is defined in Lines 5–9. The Tgen context represents a `do` loop of any index range whose body can be a list of any statements. In Line 12, a code transformation rule is defined referring to the Tgen context, `c1`, and hence is applied to an expression only if the expression is in a code region expressed by the context. Therefore, if the rule in Fig. 9.8 is applied to the code in Fig. 9.4, the expressions in Line 5 are unchanged because they are not in a code region matching the Tgen context; only `inp**2` in the loop body is replaced with `inp*inp`.

If a code matches the source patterns of multiple code transformation rules, the order of their definitions coincides with the order of applying them to the code. Therefore, the order of rule definitions written in a rule description file can represent the priorities of the rules.

9.3 Case Studies of Using the Xevolver Framework

Using the rule description introduced in Sect. 9.2.2, we have demonstrated that Xevolver can express various code transformation rules specific to individual systems and/or applications.

In [16], the Xevolver is used to adapt one of our target applications, Numerical Turbine [10], to an OpenACC [3] platform. As seen in other real-world simulation codes, Numerical Turbine has a lot of similar loop nests that have almost the same loop structure as shown in Fig. 9.9. One problem in running the code on an OpenACC platform is that the loop structure is not OpenACC-friendly. Numerical Turbine has been optimized for NEC SX vector computing systems, and thus the loop structure is optimized so as to maximize the innermost loop parallelism

```

1 DO 200 M=1,MF
2   DO 200 K=1,KF
3     DO 200 J=1,JF
4       DO 200 L=lstart,lend
5         II1 = IS(L)
6         II2 = II1+1
7         II3 = II2+1
8         IIF = IT(L)
9         IIE = IIF-1
10        IID = IIE-1
11        DO 200 I=II2,IIF
12          IF (I.LE.(IS(L)+2).OR.I.GE.(IT(L)-1)) THEN
13            STBC=0.0D0
14          ELSE
15            STBC=1.0D0
16          END IF
17        ... (skip) ...

```

Fig. 9.9 The original structure of kernel loops in Numerical Turbine

```

1 DO 200 M=1,MF
2   DO 200 K=1,KF
3     DO 200 J=1,JF
4       DO 200 I=1,inum
5         DO 200 L=lstart,lend
6           IF (I.GE.IS(L).AND.I.LE.IT(L)) THEN
7             EXIT
8           END IF
9
10          IF (I.LE.(IS(L)+2).OR.I.GE.(IT(L)-1)) THEN
11            STBC=0.0D0
12          ELSE
13            STBC=1.0D0
14          END IF
15        ... (skip) ...

```

Fig. 9.10 The transformed structure of kernel loops in Numerical Turbine

exploited by the vector processor. For an OpenACC platform, therefore, the loop structure should be changed to another one as shown in Fig. 9.10.¹

One conventional way of optimizing those loops for other systems is to simply change the loop structure by directly modifying the code. All the similar loop nests would be modified in almost the same way. Such an optimization task of manually modifying the code would be labor-intensive and error-prone. Moreover, the code modifications would result in specializing the code for the new systems. Due to the system complexity and diversity, the modified version could be more complex and make it more difficult to keep the maintainability and performance portability.

On the other hand, in the case of using Xevolver, the original code is almost unchanged, and system-dependent information is expressed as user-defined code transformation rules. As a result, application developers can maintain their codes in their manageable ways and do not usually need to care about the optimized

¹In the case study, some OpenACC directives are also needed, but not shown in the example for simplicity.

```

1  program nt_opt
2
3  !$xev tgen list(body) stmt
4  !$xev tgen var(lstart,lend,II2,IIF) exp
5  !$xev tgen condef(has_doi) contains stmt begin
6  DO I=II2,IIF
7  !$xev tgen stmt(body)
8  END DO
9  !$xev tgen end
10 !$xev tgen list(stmt_with_doi) stmt cond(has_doi)
11 !$xev tgen src begin
12 DO L=lstart,lend
13 !$xev tgen stmt(stmt_with_doi)
14 END DO
15 !$xev end tgen src
16 !$xev tgen dst begin
17 DO I=1,inum
18 DO L = lstart, lend
19 IF (I .GE. IS(L) .AND. I .LE. IT(L)) THEN
20 EXIT
21 END IF
22 !$xev tgen stmt(body)
23 END DO
24 END DO
25 !$xev end tgen dst
26 end program nt_opt

```

Fig. 9.11 The code transformation rule of adapting Numerical Turbine to an OpenACC platform

codes because the codes are transformed just before the compilation. A simplified version of the code transformation rule used in our case study is shown in Fig. 9.11. In the rule, a Tgen variable named *body* is used to define a Tgen condition, *has_doi*, which means a *do* loop whose index variable name is *I*. Then, a Tgen list, *stmt_with_doi*, is defined in Line 10 for expressing a *do* loop nest that satisfies the Tgen condition. Consequently, the structure of a *do* loop nest in the body of another *do* statement in Lines 11–15 is replaced to another loop structure shown in Lines 16–25. One important point in this rule description is that the body of the inner loop whose index variable name is *I* is expressed by *body* in Line 7, and *body* is then used in Line 22. In this way, a Tgen condition can be used to extract a part of a code pattern and to use it in the destination pattern.

The rule in Fig. 9.11 performs a kind of loop interchange with a special conditional branch, which is needed so that the loop interchange does not change the behavior of the code. The code transformation rule is specific to this particular application, Numerical Turbine, and hence we cannot expect a compiler to do the same loop transformation. Conventionally, in such a case, the code needs to be modified to achieve high performance. However, in Xevolver, the code modifications for such system-specific and/or application-specific reasons can be expressed as a user-defined code transformation rule. Accordingly, Xevolver can separate system-awareness from the code by defining an application-specific code transformation rule. In our case study, 44 loop nests are transformed by the rule in Fig. 9.11.

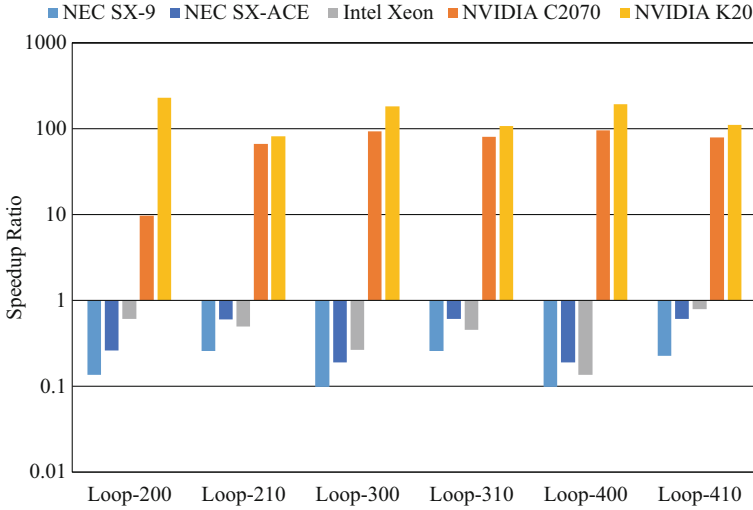


Fig. 9.12 The impact on the performance of Numerical Turbine

Figure 9.12 shows the performance impact of the code transformation. The horizontal axis indicates some main kernels, and the vertical axis shows the speedup ratio of the transformed code to the original code. If the speedup ratio is less than one, the performance is degraded by the code transformation. The original code is optimized for NEC SX vector computers, and the code transformation rule is defined to adapt the original code to an OpenACC platform. As a result, the performance of graphics processing units (GPUs), i.e., NVIDIA C2070 and NVIDIA K20, significantly improves, while the SX performance degrades. It is obvious that different systems require different optimizations. In the case of using Xevolver, the performance degradation is not a problem, because the original code is used for the SX systems and transformed only for GPU systems. Therefore, we can achieve high performance-portability across those totally different systems.

If the destination code pattern defined in Lines 16–25 of Fig. 9.11 is replaced with another one, the loop structure of the 44 loop nests in Numerical Turbine would be transformed in a different way. Therefore, use of Xevolver can make it easy to optimize the code for another platform without complicating the original code.

Another case study is to use Xevolver for data layout optimization [19]. Data layout optimization is important to achieve high performance on modern HPC systems, because it can change the memory access patterns critically affecting the performance. In many cases, data layout optimization is done by changing the data structures accessed in kernel loops. Each of data structures and kernel loops could require its own code transformation, and hence custom code transformation rules are really needed for data layout optimization.

```

1  program dl
2
3  !! Rule 1
4  !! Replace the type definition of AoS data structure
5  !$xev tgen src begin
6      type aos_t
7          real:: x
8          real:: y
9      end type aos_t
10
11     type(aos_t) aosdata(100)
12 !$xev end tgen src
13 !$xev tgen dst begin
14     type soa_t
15         real:: x(100)
16         real:: y(100)
17     end type soa_t
18
19     type(soa_t) soadata
20 !$xev end tgen dst
21
22 !! Rule 2
23 !! Change the way of accessing the data structure
24 !$xev tgen var(idx) exp
25 !$xev tgen var(x) name
26 !$xev tgen trans exp src('aosdata(idx)%x') dst('soadata%x(idx)')
27
28 end program dl

```

Fig. 9.13 A transformation rule for AoS-to-SoA transformation

Figure 9.13 shows an example of typical data layout optimization, so-called AoS-to-SoA conversion, in which an array of structures (AoS) is converted to a structure of arrays (SoA). For the conversion, two kinds of rules are defined. One is to replace the definition of an AoS data structure to that of an SoA data structure. The other is to change the way of accessing the data structure. The former rule is defined in Lines 5–20, while the latter one is defined in Line 26. In this example, the former one is defined as simple text replacement and used only once in the code. On the other hand, the latter one is applied to every access to the data structure and hence potentially changes a large number of code lines scattered over the code. It is cumbersome and error-prone to do the same code modifications by hand. Xevolver can express such an application-specific rule for mechanically changing the way of accessing a data structure. As a result, Xevolver can make it easy to change a data structure for individual cases, i.e., different systems, compilers, and/or libraries. Thus, Xevolver can prevent specializing a data structure for a particular system configuration. Accordingly, Xevolver is helpful for long-term maintenance and evolution of an HPC application code.

Furthermore, in the Xevolver project [21], we have conducted various case studies such as [8, 9, 20] to demonstrate the usefulness and practicality of the user-defined code transformation approach in migrating real-world applications to new systems.

9.4 Hierarchical Abstraction of HPC Systems

We do not claim that everything for performance optimization should be expressed as user-defined code transformations. Rather, our claim is that appropriate abstractions should be used for appropriate purposes. User-defined code transformations should be used to express code modifications that are unavoidable even if all the abstractions are properly used. Abstraction technologies such as numerical libraries are strongly required to hide complicated system configurations from application developers.

We are developing numerical libraries to hierarchically abstract HPC system configurations. Our numerical libraries are optimized for multiple platforms, such as GPU, the Intel many integrated core (MIC) architecture, and standard CPU cluster systems, so that application developers can use different implementations with common interfaces, resulting in high performance-portability. The numerical libraries are designed to support as many data structures as possible to cover various use cases while achieving high performance. We also investigate auto-tuning technologies to adapt the optimized implementations of numerical libraries to similar platforms in order to achieve high performance-portability.

One of the libraries developed in our project is FFTE for GPU/MIC.

FFTE [17] is a Fortran subroutine library for computing the fast Fourier transform (FFT) in one or more dimensions. It includes real, complex, mixed-radix, and parallel transforms. Portability is important for any numerical libraries. Thus, FFTE supports several programming models such as OpenMP, MPI, OpenMP + MPI, and CUDA Fortran + MPI. FFTE may be faster than other publicly available FFT implementations and vendor-tuned libraries. In a GPU cluster, GPUs are connected to individual nodes as PCI Express devices. Many FFT implementations work well within GPU cards [12, 13]. However, PCI Express transfer is often a performance bottleneck in FFT because FFT requires a large number of memory access operations per arithmetic operation. One goal for parallel FFTs on GPU clusters is to minimize the PCI Express transfer time and the MPI communication time. The advanced features of MVAPICH2-GDR [11] made it easy to overlap PCI Express transfers and MPI communication.

Figure 9.14 shows the parallel one-dimensional FFT performance comparison among FFTE 6.0 (GPU), FFTE 6.0 (CPU), and FFTW 3.3.3 (CPU) [4] on the HAPACS base cluster. The results indicate that FFTE 6.0 is well optimized for a GPU cluster system, and thus FFTE 6.0 (GPU) outperforms the others especially for large problems. Application developers can benefit from using the FFTE library without knowing its implementation details. Therefore, by using the library, we can achieve high performance on FFT without specializing an application code.

The Intel Xeon Phi processor has emerged as an important computational accelerator in high-performance computing systems. Knights Landing processor [15] is the second-generation Intel Xeon Phi product. The Intel Advanced Vector

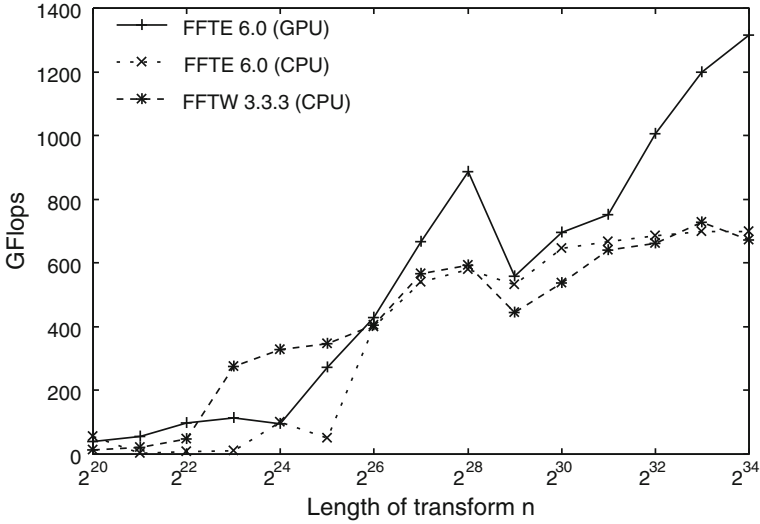


Fig. 9.14 Performance of parallel one-dimensional FFTs on the HA-PACS base cluster (128 nodes, 512 MPI processes)

Extensions 512 (AVX-512) [5] is a 512-bit single-instruction multiple data (SIMD) instruction sets on the Intel Xeon Phi processor. We have vectorized our FFT kernels using the Intel AVX-512 instructions. Both vectorization and parallelization are of particular importance with respect to Intel Xeon Phi processors. The implementation of parallel one-dimensional FFTs in FFTE is based on the six-step FFT algorithm [1]. When we parallelize the six-step FFT by using OpenMP, the outermost loop of each FFT step is distributed across the cores. In this case, the outermost loop length may not have sufficient parallelism for many-core processors. Loop collapsing increases the loop length by collapsing nested loops into a single-nested loop. For using the OpenMP collapse clause, which is supported from OpenMP 3.0 [14], the loops must be perfectly nested. Since the outer loops in the six-step FFT are perfectly nested, they can be collapsed into a single-nested loop.

Figure 9.15 shows the one-dimensional real FFT performance comparison among FFTE 6.2alpha (with collapse), FFTE 6.2alpha (without collapse), FFTW 3.3.6-pl1, and the MKL 2017 Update 1 on Intel Xeon Phi 7250. As shown in figure, FFTE (with collapse) is faster than FFTE (without collapse) and FFTW. Besides, FFTE (with collapse) is faster than MKL for the cases of $n = 2^{22}$, $2^{24} \leq n \leq 2^{25}$ and $n = 2^{29}$ on 272 threads. Accordingly, an loop optimization technique, loop collapse is effective to improve the performance of an Xeon Phi processor by increasing the outermost loop parallelism exploited by OpenMP parallel processing.

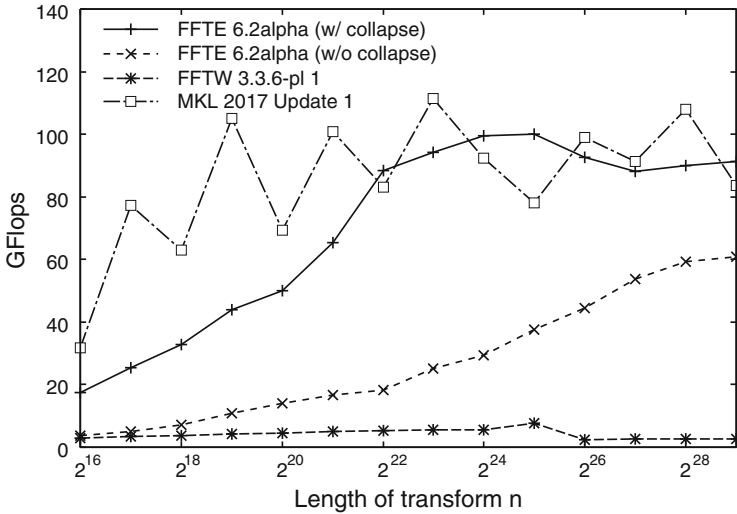


Fig. 9.15 Performance of one-dimensional real FFTs (Intel Xeon Phi 7250, 272 threads)

9.5 HPC Refactoring Catalog

In this project, we are cataloging expert knowledge and experiences about code optimizations as the *HPC refactoring catalog*. The HPC refactoring catalog is open to the public (<https://one.sc.cc.tohoku.ac.jp/hpcoref/>) so that programmers can share and reuse the knowledge and experiences. Description about the code optimization, code examples, and performance evaluation results are provided in the catalog, and code transformation rules are also provided for some important code optimization techniques.

One example of using the catalog for application migration is porting an application from Xeon-based scalar systems to a vector-parallel computer, NEC SX-ACE [2]. The application code has been developed for estimating the risk of a heat stroke by simulating the changes in the body temperature under various environments [7]. Although this code is memory-intensive and thus suitable for vector computing systems with high memory bandwidth, the code has been developed for Xeon-based scalar systems. In this project, hence, the code is migrated to the SX-ACE system, which has a higher memory bandwidth than other existing scalar systems. Figure 9.16 shows a part of the main routine of the code. In this code, since the value of BCOEF changes every iteration, the triple-nested loops cannot be vectorized by the NEC SX compiler. As a result, the vectorization ratio of this routine is only 15.4%. Since a higher vectorization ratio is mandatory for exploiting the SX-ACE system, the routine should be optimized so that the compiler can vectorize the kernel loop.

```

1 DO K=1,MODELZ
2   DO J=1,MODELX
3     DO I=1,MODELX
4       ... (skip)...
5         IF (UOLD(I,J,K) .GE. US(I,J,K)) THEN
6           BCOEF=2**((UOLD(I,J,K)-US(I,J,K))/6)
7           TEMP_DIFF=UOLD(I,J,K)-US(I,J,K)
8         ELSE
9           TEMP_DIFF=0E0
10        ENDIF
11      ... (skip)...
12        IF (UOLD(I,J,K) .LT. 39E0) THEN
13          BCOEF=1E0
14        ELSE IF (UOLD(I,J,K) .LT. 44E0) THEN
15          BCOEF=1E0+0.8E0*(UOLD(I,J,K)-39E0)
16        ELSE
17          BCOEF=1E0+(5E0*0.8E0)
18        ENDIF
19      ENDIF
20    ELSE
21      BCOEF=1E0
22      TEMP_DIFF=0E0
23    ENDIF
24  ... (skip)...
25  U(I,J,K)=UOLD(I,J,K)+(DT*SAR(I,J,K)/CP(LK(I,J,K))&
26    &+DT*A(LK(I,J,K))*storetemp2(i,j,k)/(ROU(LK(I,J,K))*CP(LK(I,J,K)))&
27    &-(DT*BCOEF*B(LK(I,J,K))*(UOLD(I,J,K)-henkaTB))/(ROU(LK(I,J,K))
28  ... (skip)...

```

Fig. 9.16 Original code of the heatstroke risk simulation

```

1 [Before]
2 integer :: x
3   x = 0
4   do i=1,MAX
5     if(a(i) < 5000) then
6       x = 100
7     end if
8     b(i) = a(i) + c(i)*x
9   end do
10
11 [After]
12 integer :: x(MAX)
13   x = 0
14   do i=1,MAX
15     if(a(i) < 5000) then
16       x(i) = 100
17     end if
18     b(i) = a(i) + c(i)*x(i)
19   end do

```

Fig. 9.17 Code optimization pattern: “Using arrays instead of scalar variables for promoting vectorizations”

The catalog provides more than 50 system-aware code optimization patterns, and one of the patterns for the SX-ACE system is shown in Fig. 9.17. In this optimization pattern, since the value of x may change every iteration, the SX compiler cannot vectorize the loop. By changing scalar variable x to a one-dimensional array so that each element stores the value at an iteration, we can promote the compiler’s vectorization of the loop. Although the number of nested loops and variables

```

1  DO K=1,MODELZ
2  DO J=1,MODELX
3  DO I=1,MODELZ
4  ... (skip) ...
5  IF (UOLD(I,J,K) .GE. US(I,J,K)) THEN
6  BCOEF_(I,J,K)=2*((UOLD(I,J,K)-US(I,J,K))/6)
7  TEMP_DIFF=UOLD(I,J,K)-US(I,J,K)
8  ELSE
9  TEMP_DIFF=0E0
10 ENDIF
11 ... (skip) ...
12 IF (UOLD(I,J,K) .LT. 39E0) THEN
13 BCOEF_(I,J,K)=1E0
14 ELSE IF (UOLD(I,J,K) .LT. 44E0) THEN
15 BCOEF_(I,J,K)=1E0+0.8E0*(UOLD(I,J,K)-39E0)
16 ELSE
17 BCOEF_(I,J,K)=1E0+(5E0*0.8E0)
18 ENDIF
19 ELSE
20 BCOEF_(I,J,K)=1E0
21 TEMP_DIFF=0E0
22 ENDIF
23 ... (skip) ...
24 U(I,J,K)=UOLD(I,J,K)+(DT*SAR(I,J,K)/CP(LK(I,J,K))&
25 &+DT*(LK(I,J,K))*storetemp2(i,j,k)/(ROU(LK(I,J,K))*CP(LK(I,J,K)))&
26 &-(DT*BCOEF_(I,J,K)*B(LK(I,J,K))*...

```

Fig. 9.18 Optimized code of the heatstroke risk simulation

in the optimization pattern are different from those in the target application, the vectorization-obstructing factor in the pattern is the same as that in the target application.

Using this optimization pattern as a reference, the target application can be optimized in a similar way. Since the loops are triple-nested and the value of BCOEF changes every iteration, a three-dimensional array, BCOEF_(I, J, K), is used instead of a scalar variable, as shown in Fig. 9.18.

By replacing every reference to BCOEF with a reference to an element of BCOEF_(I, J, K), the triple-nested loops are successfully vectorized. Tabel 9.1 shows the effects of the code optimization on the single-core execution performance of the SX-ACE system. By applying this optimization, the vectorization ratio increased to 99.16% while keeping a certain level of the vector length. As a result, a 20x performance improvement is achieved. In this way, programmers can perform code migration and optimization based on system-aware optimization patterns in the catalog. Furthermore, since the performance impacts of the code optimizations are also quantitatively provided in the catalog, the programmers can estimate how the code optimizations will affect the performance of their codes. Therefore, we can expect that programmers' burden of finding effective code optimizations in a try-and-error manner is reduced by using the catalog (Table 9.1).

Table 9.1 Effects of the code optimization

	Vectorization ratio [%]	Ave. vector length	Exec. time [sec]
Original	15.4	216.5	3,209.7
Optimized	99.1	196.4	149.1

9.6 Conclusions

In the Xevolver project, we have explored an effective way of streamlining legacy code migration. One major reason why legacy code migration is so challenging is that an HPC application code is too specialized for a particular system configuration. Therefore, we have considered how to separate such system-specific information from HPC application codes.

Various approaches have been proposed so far to hide the implementation details that are specific to a particular system. Numerical libraries are one of the most powerful abstractions for hiding the system-specific implementations from application developers. Therefore, we have discussed how to optimize some important numerical libraries such as FFT for modern HPC systems.

Even if all the abstractions are properly used in an HPC application code, the code still needs to be optimized to fully exploit the performance of a target system. However, such optimization is likely to specialize the code only for the target system, leading to lower maintainability and performance portability of the code. Motivated by this issue, we have been developing a code transformation framework, Xevolver, so that users can define their own code transformation rules for system-specific performance optimizations. Various case studies have been made to demonstrate the effectiveness and practicality of Xevolver. Therefore, we conclude that the under-defined code transformation approach is useful and helpful for streamlining legacy code migration by separating system-awareness from HPC application codes.

Expressing system-aware code modifications as code transformations is also useful to share expert knowledge and experiences in a machine-usable way. Our HPC refactoring catalog is maintained by several practitioners to record and share their experiences and hence is expected to further grow in the future.

Since the main goal of the Xevolver code transformation framework is separation of system-awareness, it does not automate system-aware performance optimization at all. It just replaces manual code modifications with code transformations. As a result, users have to be responsible for ensuring that the transformations do not change the behaviors of their codes. Therefore, as one future research direction, it would be interesting to explore an effective way to check the equivalence between the original and transformed versions of a code.

Acknowledgements The authors would like to sincerely thank all the members of the Xevolver project for their significant contributions.

References

1. Bailey, D.H.: FFTs in external or hierarchical memory. *J. Supercomput.* **4**, 23–35 (1990)
2. Egawa, R., Komatsu, K., Momose, S., Isobe, Y., Musa, A., Takizawa, H., Kobayashi, H.: Potential of a modern vector supercomputer for practical applications: performance evaluation of sx-ace. *J. Supercomput.* **73**(9), 3948–3976 (2017). <https://doi.org/10.1007/s11227-017-1993-y>
3. Farber, R. (ed.): *Parallel Programming with OpenACC*. Morgan Kaufmann, Cambridge (2016)
4. FFTW home page. <http://www.fftw.org/>
5. Intel Corporation: Intel architecture instruction set extensions programming reference (2016). <https://software.intel.com/sites/default/files/managed/26/40/319433-026.pdf>
6. Kay, M.: *XSLT 2.0 and XPath 2.0 Programmer’s Reference (Programmer to Programmer)*, 4th edn., Wiley, Indianapolis (2008)
7. Kojima, K., Hirata, A., Hasegawa, K., Kodera, S., Laakso, I., Sasaki, D., Yamashita, T., Egawa, R., Horie, Y., Yazaki, N., Kowata, S., Taguchi, K., Kashiwa, T.: Risk management of heatstroke based on fast computation of temperature and water loss using weather data for exposure to ambient heat and solar radiation. *IEEE Access* **6**, 3774–3785 (2018). <https://doi.org/10.1109/ACCESS.2018.2791962>
8. Komatsu, K., Egawa, R., Hirasawa, S., Takizawa, H., Itakura, K., Kobayashi, H.: Migration of an atmospheric simulation code to an OpenACC platform using the Xevolver framework. In: *The Third International Symposium on Computing and Networking, International Workshop on Legacy HPC Application Migration (LHAM2015)*, pp. 515–520 (2015)
9. Komatsu, K., Egawa, R., Hirasawa, S., Takizawa, H., Itakura, K., Kobayashi, H.: Translation of large-scale simulation codes for an OpenACC platform using the Xevolver framework. *Int. J. Netw. Comput.* **6**(2), 167–180 (2016)
10. Miyake, S., Yamamoto, S., Sasao, Y., Momma, K., Miyawaki, T., Ooyama, H.: Unsteady flow effect on nonequilibrium condensation in 3-D low pressure steam turbine stages. In: *ASME Turbo Expo 2013, San Antonio* (2013)
11. MVAPICH: MPI over InfiniBand, omni-path, ethernet/iWARP, and RoCE. <http://mvapich.cse.ohio-state.edu/>
12. Nukada, A., Matsuoka, S.: Auto-tuning 3-D FFT library for CUDA GPUs. In: *Proceedings of Conference on High Performance Computing Networking, Storage and Analysis (SC’09)*, Portland (2009)
13. NVIDIA Corporation: CUFFT library user’s guide (2017). http://docs.nvidia.com/cuda/pdf/CUFFT_Library.pdf
14. OpenMP Architecture Review Board: OpenMP application program interface. <http://www.openmp.org/mp-documents/spec30.pdf>
15. Sodani, A., et al.: Knights landing: second-generation Intel Xeon Phi product. *IEEE Micro* **36**, 34–46 (2016)
16. Suda, R., Takizawa, H., Hirasawa, S.: Xevtgen: fortran code transformer generator for high performance scientific codes. *Int. J. Netw. Comput.* **6**(2), 263–289 (2016)
17. Takahashi, D.: FFTE: a fast Fourier transform package. <http://www.ffte.jp/>
18. Takizawa, H., Hirasawa, S., Hayashi, Y., Egawa, R., Kobayashi, H.: Xevolver: an XML-based code translation framework for supporting HPC application migration. In: *IEEE International Conference on High Performance Computing (HiPC)*, Goa (2014)
19. Takizawa, H., Yamada, T., Hirasawa, S., Suda, R.: A use case of a code transformation rule generator for data layout optimization. In: *Sustained Simulation Performance 2016*, pp. 21–30. Springer, Berlin/Heidelberg (2016)
20. Takizawa, H., Reimann, T., Komatsu, K., Soga, T., Egawa, R., Musa, A., Kobayashi, H.: Vectorization-aware loop optimization with user-defined code transformations. In: *Workshop on Re-emergence of Vector Architectures (REV-A)*, Honolulu, pp. 685–692 (2017)
21. The Xevolver Project: JST CREST “an evolutionary approach to construction of a software development environment for massively-parallel heterogeneous systems” (2011). <http://xev.arch.is.tohoku.ac.jp/>
22. The XSLT C library for GNOME. <http://xmlsoft.org/libxslt/>

Chapter 10

Numerical Library Based on Hierarchical Domain Decomposition



**Ryuji Shioya, Masao Ogino, Yoshitaka Wada, Kohei Murotani,
Seiichi Koshizuka, Hiroshi Kawai, Shin-ichiro Sugimoto, and Amane Takei**

Abstract We have been developing an open-source computer-aided engineering (CAE) software, ADVENTURE, which is a general-purpose parallel finite element analysis system and can simulate a large-scale analysis model with supercomputer. For supercomputer architecture such as an exa-scale system, to obtain high computational efficiency for software that requires large-scale numerical calculation data processing, a programming model that considers the hierarchical structure of hardware, such as a microprocessor and memory, is necessary. From this point of view, ADVENTURE system was developed using the hierarchical domain decomposition method (HDDM) as the basic technology for a large-scale data system. HDDM is technology developed by ourselves mainly for numerical analysis method. In particular, we have developed application-specific system software that

R. Shioya (✉) · H. Kawai
Toyo University, Tokyo, Japan
e-mail: shioya@toyo.jp; kawai063@toyo.jp

M. Ogino
Nagoya University, Nagoya, Japan
e-mail: masao.ogino@cc.nagoya-u.ac.jp

Y. Wada
Kindai University, Higashi-osaka, Japan
e-mail: wada@mech.kindai.ac.jp

K. Murotani
Railway Technical Research Institute, Tokyo, Japan
e-mail: murotani.kohei.03@rtri.or.jp

S. Koshizuka
University of Tokyo, Tokyo, Japan
e-mail: koshizuka@sys.t.u-tokyo.ac.jp

S. Sugimoto
Hachinohe Institute of Technology, Hachinohe, Japan
e-mail: sugimoto@hi-tech.ac.jp

A. Takei
Miyazaki University, Miyazaki, Japan
e-mail: takei@cc.miyazaki-u.ac.jp

can obtain high performance by focusing on simulation of continuum mechanics by finite element method (FEM) and particle method which are highly demanded by academic research and industry. We have developed four research items “DDM I/O (input/output) library,” “DDM solver library,” “DSL for continuum mechanics,” and “continuous mechanics simulator.” The software, which is the result of our research, is released as open-source software on the sub-project page in the ADVENTURE project homepage. In this chapter, some of those libraries are described in detail.

10.1 Numerical Library Based on Hierarchical Domain Decomposition

10.1.1 Introduction

For next-generation parallel computing architecture such as a post peta-scale system, to obtain high computational efficiency for software that requires large-scale numerical calculation data processing, a programming model that considers the hierarchical structure of hardware, such as a microprocessor and memory, is necessary. In particular, it is necessary to assume that all processes from pre/post processing, such as input data generation and visualization, to solver processing, such as the numerical analysis method, are performed on a supercomputer. Therefore, we have developed a large-scale numerical data processing system that uses the hierarchical domain decomposition method (HDDM) [1] as basic technology for a large-scale numerical data processing system on next-generation parallel computers. The HDDM is the technology that we developed for the numerical analysis method. In particular, we developed application-specific system software that can obtain high performance by focusing on simulation of continuum mechanics by finite element method (FEM) and particle method which are highly demanded by academic research and industry.

The target application is the open-source computer-aided engineering (CAE) software ADVENTURE [2], which we have developed and has been proven in large-scale parallel computing using the HDDM. It is also used in the HPCI strategic program and for social and scientific priority issues to be managed by using a post-K computer. However, ADVENTURE does not include software for a particle method; thus, we have developed a new application for a particle solver. We have developed four research items: a “DDM I/O (input/output) library,” “DDM solver library,” “DSL for continuum mechanics,” and “continuous mechanics simulator.” The software, which is the result of our research, is released as open-source software on a sub-project page on the ADVENTURE Project homepage [3].

As a “DDM I/O library,” we have developed the following libraries: “AdvIO2,” a standard I/O library that corresponds to particle method simulation; “AdvMetis2,”

a tool that has multilevel domain decomposition and a parallel mesh subdivision function and enables large-scale data generation using MPI-OpenMP hybrid parallel processing and a restart function; “DDM compression technology,” which compresses the calculation data of an FEM and particle method by deleting internal freedom; “LexADV_VSCG,” which achieves offline visualization of over $100,000 \times 100,000$ pixels for super high-resolution simulation and is a highly portable library that supports mesh and particle drawing; and “LexADV_WOVis,” which is a library that performs MPI-OpenMP hybrid parallel visualization using the data structure of DDM and z-buffer image synthesis processing using the original tournament method.

Additionally, we have developed “LexADV_EMPS,” which is the framework of a distributed memory parallel moving particle simulation (MPS) explicit solver with a bucket-based two-level domain decomposition and halo communication pattern generation function to reduce communication between adjacent processes for the particle method. Using AdvMetis2, we succeeded in generating one trillion degree of freedom (DOF) mesh assumed in a post peta-scale system. Moreover, using “LexADV_EMPS” we performed particle method simulation over hundreds of millions, which was difficult previously.

As a “DDM solver library,” we have developed the following libraries: “LexADV_IsDDM,” which is an iterative method library with high scaling performance of 85% or more with a K computer; “LexADV_TryDDM,” which is an iterative method library that explicitly constructs the Schur complement equation with domain decomposition mesh, global coefficient matrix, and right-hand side vector as input; and a scaled-BDD method with high-speed and high-stability convergence. We succeeded in reducing the number of iterations and calculation time in a multiple material model. Using “LexADV_IsDDM,” we succeeded in the structural analysis of 100 billion DOFs with an unstructured mesh, which was conventionally difficult. Using “LexADV_TryDDM” made it easy to compare the BDD method with IC decomposition and SSOR preprocessing.

As a “DSL for continuum mechanics,” we have developed the following libraries: a translator from a LaTeX-based numerical expression description to program code that targets physical models for elements, cells, and particles and “LexADV_AutoMT,” which is a library optimized for small-scale matrices and tensor operations for multi-core, many-core processors and GPUs.

Because the translator generates a program that calls “LexADV_AutoMT,” the user does not need to consider the computer architecture, and it is possible to obtain high performance.

As a “continuous mechanics simulator,” we have developed a large-scale electromagnetic field simulator using the DDM iteration method. It improved the efficiency of analysis for models that include mobile objects, such as motors, using the development of distributed memory parallel algorithms, and improved accuracy using the development of heterogeneous boundary smoothing technology for three-dimensional (3D) voxel data constructed from medical images.

We have also developed a large-scale fluid simulator based on the MPS explicit method using the “LexADV_EMPS library,” particularly functions such as fluid-rigid body coupled analysis, a higher-order precision differential model, and surface tension model.

In the following sections, some of these libraries are described in detail.

10.1.2 LexADV_TryDDM

An efficient algorithm to solve large linear systems derived from finite element analysis is of great importance in the post peta-scale computing era. The domain decomposition method (DDM) [4, 5] is a well-known technique of parallel finite element analysis. Particularly, the iterative substructuring method that solves a condensed interface system, called the Schur complement system, is a widely used algorithm in the implementation of the DDM. Moreover, the HDDM [1] that adopts the master-slave model and a multiple master system in parallel computation is expected to obtain high parallel efficiency on various distributed memory parallel computers. Most software packages that implement finite element analysis based on the DDM adopt the implicit construction of the Schur complement. As a result, the program structure is too complex. Thus, it is difficult to evaluate and develop an algorithm for the DDM using existing software packages. To solve these problems, we have developed a parallel DDM library called LexADV_TryDDM for finite element analysis with the explicit construction of the Schur complement equation. Particularly, two methods for the parallel construction of the Schur complement equation have been implemented. These methods are based on a global Schur complement and local Schur complement.

LexADV_TryDDM is a library for solving linear systems of the form

$$Ax = b, \tag{10.1}$$

where A is a symmetric positive definite matrix. The library is programmed in the C language, requires message passing interface (MPI) for message passing, and makes use of ADVENTURE_IO (AdvIO) [6], MUMPS [7], and Lis [8]. LexADV_TryDDM has the following features:

- solves linear equations using the DDM;
- supports the compressed sparse row (CSR) format for the coefficient matrix;
- supports the dense vector for the right-hand side;
- supports domain decomposed mesh data using ADVENTURE_Metis (AdvMetis) [6];
- uses the AdvIO library for reading input data;
- uses the MUMPS library to construct the Schur complement equation;
- uses the Lis library to solve the Schur complement equation;

- is written in the C language;
- uses the MPI library for parallel processing.

10.1.2.1 Iterative Substructuring Method

LexADV_TryDDM solves a linear system based on a nonoverlapping DDM. Using the nonoverlapping domain decomposition, the original linear system can be reordered using

$$\begin{bmatrix} A_{II} & A_{IB} \\ A_{IB}^T & A_{BB} \end{bmatrix} \begin{bmatrix} x_I \\ x_B \end{bmatrix} = \begin{bmatrix} b_I \\ b_B \end{bmatrix}, \quad (10.2)$$

where the subscripts I and B correspond to variables in the *interior* of subdomains and on the *interface* boundary, respectively. We consider a partitioned system where the variables in the interior and on the interface are

$$A_{II}x_I = b_I - A_{IB}x_B, \quad (10.3)$$

$$Sx_B = g, \quad (10.4)$$

respectively, where $S = \sum_{i=1}^N R^{(i)T} S^{(i)} R^{(i)}$ is the Schur complement matrix, where

$$S^{(i)} = A_{BB}^{(i)} - A_{IB}^{(i)T} \left(A_{II}^{(i)} \right)^{-1} A_{IB}^{(i)} \quad (10.5)$$

is a local Schur complement matrix of subdomain (i) , N denotes the number of subdomains, $R^{(i)}$ is a subdomain Boolean matrix that restricts the interface, and g is a condensed right-hand side vector.

LexADV_TryDDM solves the interface system and then interior systems. A flowchart of the library is summarized as follows:

1. Call `read_ab_data()`
 - (a) Read coefficient matrix A in CSR format and right-hand side vector b in dense format.
2. Call `try_ddm()`
 - (a) Read the domain decomposed mesh generated by AdvMetis
 - (b) Reorder A and b in the order of the interior first.
 - (c) Compute S and g using MUMPS.
 - (d) Compute x_B as a solution of Eq. (10.4) using Lis.
 - (e) Compute x_I as a solution of Eq. (10.3) using MUMPS.
 - (f) Compute a solution x from x_I and x_B .

```

void read_ab_data(
    char *mat_file, /* input filename */
    MAT_DATA *A, /* Matrix 'A' */
    VEC_DATA *b /* Vector 'b' */
);

```

```

int try_ddm(
    int argc, char* argv[],
    int *csr_ptr, /* row pointer of 'A' */
    int *csr_idx, /* column index of 'A' */
    double *csr_val, /* value of 'A' */
    int A_nnz, /* num. of nonzero of 'A' */
    int A_matdim, /* dimension of 'A' */
    int b_dim, /* dimension of 'b' */
    double *b_val, /* value of 'b' */
    char *mesh_file, /* file name of domain decomposed mesh */
    char *mesh_type /* type of finite element */
);

```

Table 10.1 Computational time in a static stress analysis of a holed plate model with a 0.1 million degree of freedom mesh

Software	Total time [s]	Solving Eq. (10.4) [s]	Other [s]
LexADV_TryDDM	71.2	16.5	54.7
ADVENTURE_solid [6]	59.3	8.3	51.0

10.1.2.2 Numerical Examples

Numerical examples of LexADV_TryDDM are presented. Tables 10.1 and 10.2 show performances for solving a linear system of 0.1 million DOFs. As can be observed in Table 10.1, the library focuses on encouraging the research of DDM; however, it has computational time performance that is sufficient for practical use. Moreover, the library can use various conventional iterative methods and then enables a comparison between DDM specified preconditioners and others.

10.1.3 Visualization Library for a Post Peta-Scale Computer

A large bottleneck on the exa- and peta-scale supercomputers is the scientific visualization [11] because of parallel environment with a huge number of processors. We'd like to review a trend of the recent and next-generation supercomputer and point the problem of the visualization on. GPGPU accelerates performance,

Table 10.2 Comparison of iteration counts with different preconditioners

Software	Preconditioner	Iter. ($N = 50$)	Iter. ($N = 250$)
LexADV_TryDDM	Jacobi	2,301	3623
	Gauss-Seidel	1,120	1,780
	ILU(0)	271	433
	BDD-DIAG [9]	175	94
ADVENTURE_solid	BDD-DIAG	176	96
	BDD [10]	61	46

which has reduced the computational time by 1/30 in the past decade, using a moderate comparison. The Kei supercomputer can produce a huge size of total result files, which exceeds several 10^{12} bytes. The network speed required to transfer reduced data is lower than we expect over the Internet, even if a data reduction technique is used. The problem is now evident, and the post processing of numerical analysis cannot be obviously managed in local computers. We have two solutions for the problem. The first is a reversible reduction technique that compresses a dataset constituted of coarse and medium fine facets or volumes using a parallel supercomputer [12]. The compressed dataset turns into a smaller dataset than the original dataset, and interactive post processing can be achieved. The second solution is parallel visualization in each node of a supercomputer. The visualization is conducted sequentially after finite element or numerical simulations [13]. Interactive visualization as post processing in the analysis allows effective work to examine the results. Regarding user experience, a good experience with interactivity could not be achieved in visualization on a supercomputer in the past. Thus, a versatile scientific computer visualization library, LexADV_VSCG [14], which achieves very high portability for any hardware and software environment of a supercomputer, is designed and implemented in the present project. The fundamental ideas, design, and implementation of LexADV_VSCG are explained. Important facts to visualize on a supercomputer are presented in this work.

10.1.3.1 Fundamental Ideas and Design of the LexADV_VSCG Library

LexADV_VSCG is required because of the obvious concept of visualization on a supercomputer. LexADV_VSCG is designed to have the following significant features [14]:

- simple and standard implementation for any operating system (OS): independent from the OS;
- multiple images to be considered as one image using the z-buffer: parallel processing;
- no other special hardware and vis libraries: independent from specific environment issues;

- manages very fine resolution using well-designed software: key idea for ultra-large-scale data;
- impressive image using a well-designed algorithm and implementation: key idea is to use a very fine resolution.

The significant features are surveyed and discussed thoroughly for a next-generation supercomputer. To establish the features, the software rendering approach is used without any other software library. The parallel visualization feature is not provided by LexADV_VSCG; however, the feature is provided by LexADV_WOVis, as mentioned later. LexADV_VSCG is the elemental library that serves the API for scientific and engineering visualization and requires no specific acceleration hardware or other visualization software. The LexADV_VSCG library generally becomes a part of an application program. Each pixel of an image in the library is extended to represent the relationship between the view vector and normal of a facet and z-buffer. The key data of each pixel is an extended z-buffer bit. The extended z-buffer merges many generated images into one image on parallel computers where the images are generated in parallel. Each image at a node is standardly maintained at each storage of the node or stored in the memory of the node. For instance, when a total of 100,000 cores are installed in the supercomputer, an analysis model should also be partitioned into 100,000 subdivided subsets. A camera as a viewpoint is located at 500 positions on a unit sphere, and no fewer than 500,000,000 images are generated. In this case, all the images should be merged into 500 images. The merging task requires an enormous amount of transferring and receiving data between nodes. The merging processes are successively computed as illustrated in Fig. 10.1. When the processes are iteratively conducted 16 times, 216 images are unified to obtain a complete image. Level of detail (LOD), which is a visualization technique, is a useful approach to managing a large-scale finite element model. LOD requires several levels of datasets to represent the model. In the latest parallel computer system, we must completely manage hundreds of thousands of datasets. Additionally, a large-scale finite element model can be produced on the supercomputer. When all the calculation tasks from A to Z of an analysis are processed on the supercomputer, the cost of all of the computing we require would be larger than the initial assumption.

10.1.3.2 How to Achieve Interactive Visualization

The objective of the development of LexADV_VSCG is to provide an API to obtain visualized images for very large-scale finite element and particle models. LexADV_VSCG simply provides a fundamental rendering algorithm, which is ray casting and glow shading; however, LexADV_VSCG can generate very high-resolution images for exa-scale and next-generation supercomputers. The implementation is based on basic visualization algorithms by rendering triangle facets and shading calculated using ray-cast direction and triangle surface direction.

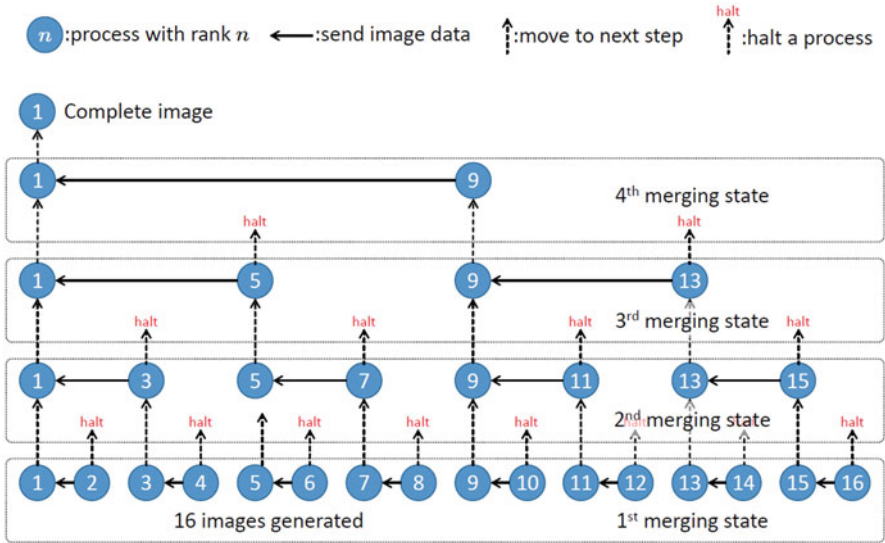


Fig. 10.1 Merging processes of images, where located in each node, with an extended z-buffer in a parallel environment

LexADV_VSCG necessarily requires a huge number of facets with very fine resolution for a high quality image. LexADV_VSCG can provide the interface for particle-based method; particles are represented by an aggregation of triangle facets. A sufficient quality of rendering image can be achieved using LexADV_VSCG for scientific and engineering visualizations with the supercomputer. The interactivity of scientific and engineering visualizations is one of the most practical concerns in scientific and engineering visualization processing. Unfortunately, the latest supercomputer does not prepare a direct interactive connection with a user's local computer. In the supercomputer, all the application programs to be executed are collocated by a job management system. LexADV_VSCG can produce a very fine image with a $10^5 \times 10^5$ -pixel resolution. The image contains sufficient information with regard to a very fine finite element discretized model and a huge number of particles. In a traditional visualization application, a scientist or engineer can easily adjust the appropriate size of the image as required by shrinking or enlarging it. Moving, enlarging, and shrinking are the fundamental operations to view an image with interactivity. The LexADV_VSCG library only generates high-resolution images. It is necessary to propose the practical use of the LexADV_VSCG library for interactive visualization. The library is integrated in a user's application to be run on the supercomputer. The application requires some contrivance to view the results after a numerical simulation. The library has positions on a sphere in advance. A scientist or engineer only generates multiple viewpoint images. A spherical polar coordinate system is used as a name description rule that represents a relationship between a name of an image file and a camera position in LexADV_VSCG. The

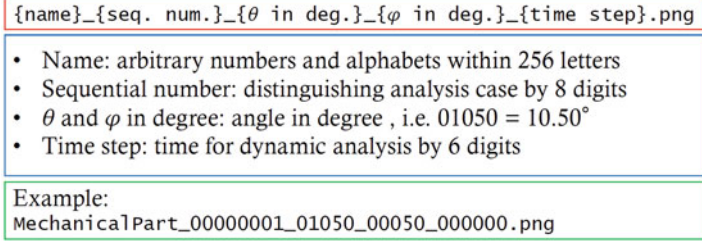


Fig. 10.2 Name description rule representing the viewpoint in a spherical coordinate system

name description rule of an image file, including the coordinate system, is defined as described in Fig. 10.2. The rule is easily extended to other coordinate systems, for example, Euler's angle definition or geographic coordinate system.

In LexADV_VSCG, two description rules for the rotation of an object are available. The first description rule is the direction form of a rotation matrix. The second description rule is Rodrigues vector (k_1, k_2, k_3) . The Rodrigues rotation formula is described as follows:

$$\mathbf{K} = \begin{pmatrix} 0 & -k_3 & k_2 \\ k_3 & 0 & -k_1 \\ -k_2 & k_1 & 0 \end{pmatrix}, \quad (10.6)$$

$$\mathbf{R} = \mathbf{I} + (\sin \theta) \mathbf{K} + (1 - \cos \theta) \mathbf{K}^2, \quad (10.7)$$

where \mathbf{I} is a unit matrix and θ is the length of the Rodrigues vector. The position of a viewpoint is notated in spherical polar coordinates. The distance from a viewpoint to the center of an object is defined as a constant value of one. Amplitude θ is the angle between the z -axis and radius vector. Amplitude φ is the angle between the x -axis and projected vector of the radius vector to the x - y plane. The position of a viewpoint is set at $(0, 0, -1)$ in the implementation. All of positions of a viewpoint are calculated using the iterative subdivision of a dodecahedron in advance. The computational notation is described in the Cartesian coordinate system. To convert from the Cartesian coordinate system to the spherical polar coordinate system, the positions of viewpoints should be calculated by

$$\begin{Bmatrix} x \\ y \\ z \end{Bmatrix} = \mathbf{R}^{-1} \begin{Bmatrix} 0 \\ 0 \\ -1 \end{Bmatrix}, \quad (10.8)$$

$$\begin{cases} \cos \theta = \frac{z}{\sqrt{x^2 + y^2 + z^2}} (0 \leq \theta \leq \pi) \\ \cos \varphi = \frac{x}{\sqrt{x^2 + y^2}}, \sin \varphi = \frac{y}{\sqrt{x^2 + y^2}} (0 \leq \varphi \leq 2\pi), \end{cases} \quad (10.9)$$

where x , y , and z are the camera position in the orthogonal coordinate system, \mathbf{R} is an inverse rotational matrix to be applied to the object, and θ and φ are the position of a viewpoint in the spherical polar coordinate system. A generated multiple image on each node can be processed using OmniEyes [15], which is a specialized image viewing application to be run on a scientist's or engineer's local computer to display a very high-resolution image generated by LexADV_VSCG. The images to be displayed to a scientist or engineer can be easily changed using the mouse operation in a similar manner to a 3D object viewing application. To ensure the magnification, rotation, and movement of a view of an object, the image buffer to be displayed that corresponds to the mouse operation is allocated up to the maximum size of the memory in the computer. The recommended computer specification is over 16 GB of main memory and over 1 GB of video memory. A medium-range notebook PC satisfies the recommended specification.

10.1.3.3 How to Parallelize the Visualization Processes

LexADV_VSCG only provides a single processing visualization approach to obtain visualized results. As mentioned above, the image has z-buffer information to superpose many images to obtain one large image at the pixel level. If over 10 thousand cores are used in an analysis, a scientist or engineer should merge over 10 thousands images into one meaningful image iteratively for several hundred times the number of viewpoints. This task should also be automated to free scientists or engineers from such a boring task. In current and future supercomputer systems, a parallel computer system is naturally considered. According to this consideration, LexADV_WOVis [3] provides parallel features for offline image generation. LexADV_WOVis is a coupler to use the full features of LexADV_VSCG. The release version of LexADV_WOVis requires the following mandatory libraries:

- LexADV_VSCG (Versatile Scientific Computer Graphics) library;
- MPI library for parallel data processing;
- AdvIO library for reading input data;
- the input and output files of ADVENTURE_Solid (AdvSolid) and ADVENTURECluster;
- AutoNOSim library for parallel processing.

LexADV_WOVis fundamentally offers an automatic procedure for superimposing the images of a decomposed finite element discretized model in parallel. Each decomposed finite element model is located at each node in the parallel supercomputer. An application program using LexADV_WOVis automatically gathers all of the distributed images into one high-resolution image. The merged images are repeatedly produced according to the positions of viewpoints on a unit sphere, as mentioned in Sect. 10.1.3.1.

10.1.4 Parallel Explicit MPS Solver Framework

The MPS method [16] is one of the most famous particle methods. A particle method is one of the numerical calculation methods that uses particles to solve the physical laws governed by differential equations, for example, the Navier-Stokes equations. Because the particles, as the calculation indicates, move on the time-marching processes, the particle method is superior to grid methods for the case of solving dynamic problems, such as free surfaces and large deformations. However, the motion of particles makes it difficult to parallelize the particle method in distributed memory parallel computers.

Our target solver is the explicit MPS (EMPS) method, which is one of the most popular particle methods [17]. LexADV_EMPS v0.1b, released as open-source software in October 2014, is the sEMPS framework for solving large-scale problems using particle methods. The target problem size is 10 million to 1 billion particles or more. LexADV_EMPS supports three functions, “domain decomposition,” “halo exchange,” and “dynamic load balance,” which are required in the distributed memory parallel computing of particle methods.

LexADV_EMPS adopts bucket-based domain decomposition [17]. First, after a bounding box for an entire analysis domain is defined, the bounding box is filled with buckets of cubes, as shown in Fig. 10.3a. Because the radius of influence for a particle is defined in the MPS method, the width of the bucket must be the same or larger than the radius of influence. All particles are assigned to buckets. The domain decomposition by the buckets is performed with an equal number of particles in each processing element, as shown in Fig. 10.3b. Each bucket color in Fig. 10.3b represents a subdomain that is assigned to each processing element. The domain decomposition in LexADV_EMPS is performed by ParMETIS.

Each decomposed domain is expanded from the boundaries by one bucket width. The particles in the expanded domain are assigned to one processing element, as shown in Fig. 10.3c. The expanded domains by one bucket width (areas of faint color in Fig. 10.3c) are called “halos.” The consistency of the particle data in a halo is maintained by communication between neighboring domains. The arrows in Fig. 10.3c represent the appearance of a particle datum of the other domains being sent to a halo of a domain. If an imbalance in the number of particles among domains appears as the analysis progresses, this domain decomposition is performed again to recover the balance of the number of particles. Figure 10.4 shows the results of the dam-break analysis, and Fig. 10.5 shows the results of the domain decomposition for the dam-break analysis in Fig. 10.4 by LexADV_EMPS.

10.1.4.1 Distributed Memory Parallel Explicit MPS Implemented by LexADV_EMPS

There are two sample solvers implemented by LexADV_EMPS. The first sample solver is the dam-break analysis with a free surface, as shown in Fig. 10.4. The

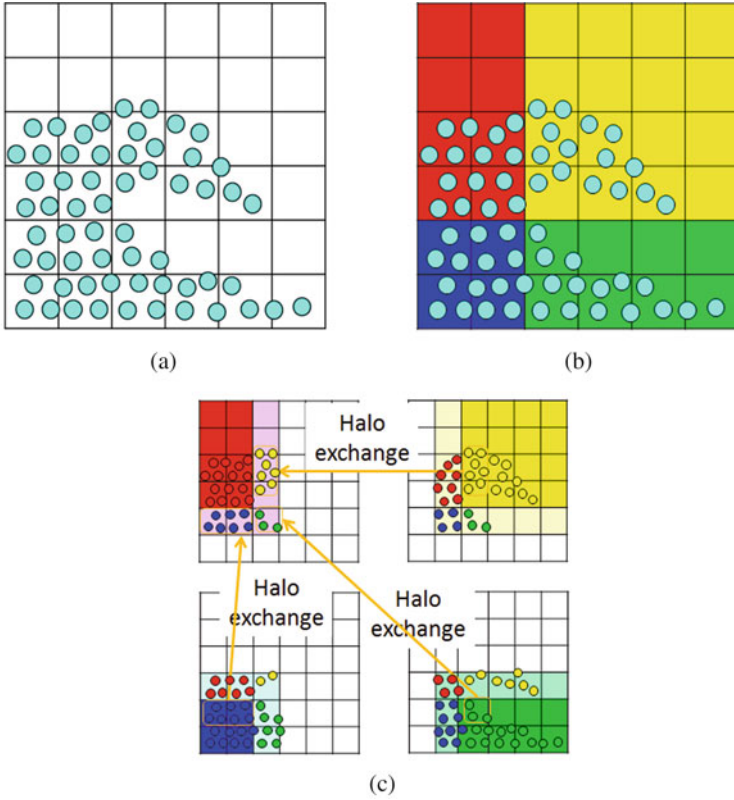


Fig. 10.3 Bucket-based domain decomposition of LexADV_EMPS. (a) Assigning particles into buckets. (b) Domain decomposition for buckets. (c) Halo exchange pattern

second sample solver is the two floating object analysis using the interaction between fluid and rigid bodies, as shown in Fig. 10.6. The yellow object floats and the red object sinks according to each density in Fig. 10.6b. These two sample solvers are as popular as the particle method.

In floating object analysis, we use the coupled method in [18] as the coupled fluid-rigid body interaction algorithm. In the coupled method, rigid bodies are represented by particles with fixed relative configurations, and the rigid body particles are calculated using the same method as that for fluid particles. Translations and rotations of rigid bodies are calculated by the rigid body particles. The positions of the rigid body particles are updated by the motions of the rigid bodies. This algorithm means the volume integral of forces for rigid bodies.

Additionally, the functions for the particle method, for example, “surface tension model by potential model”; “first-, second-, third-, and fourth-order spatial derivative models”; and “collision condition,” are implemented in the sample solver.

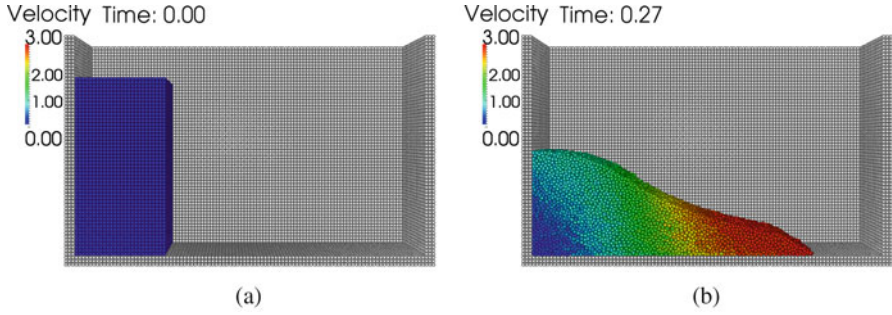


Fig. 10.4 Results of dam-break analysis (colors are the velocity). (a) Initial arrangement. (b) Result at 0.27 s

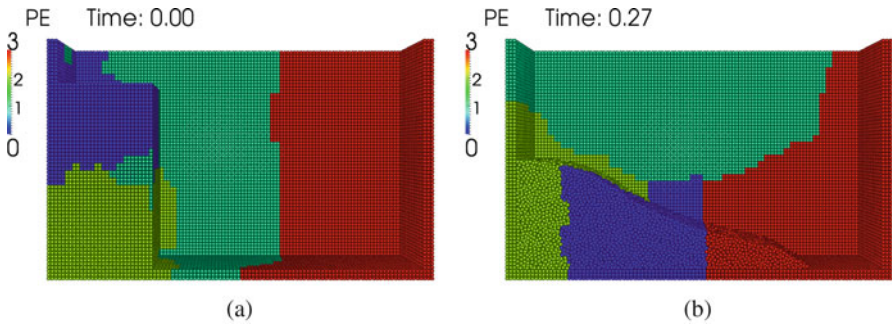


Fig. 10.5 Results of the domain decomposition (colors are processor elements). (a) Initial arrangement. (b) Result at 0.27 s

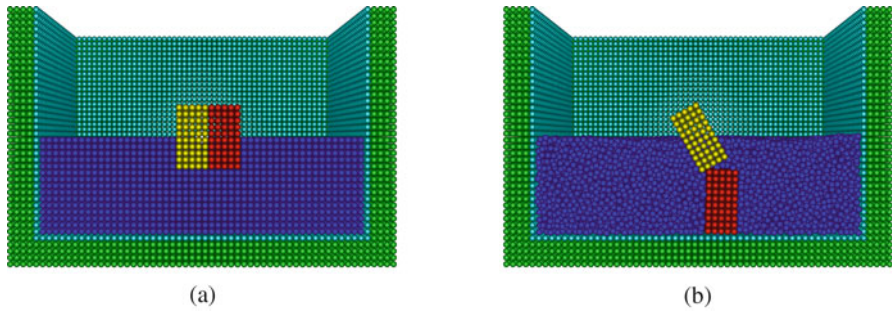


Fig. 10.6 Analysis of two floating objects with different densities. (a) Initial arrangement. (b) Result at 1.0 s

10.1.5 LexADV_AutoMT

Exa-scale supercomputers will soon appear, around 2019–2022. As the exa-age approaches, the gap between the full potential of HPC hardware architectures and the actual performance of code running on these platforms but written by ordinary researchers and developers could increasingly widen. As a result, it would be increasingly more difficult to achieve a high peak performance ratio. Measures must be taken to shrink the so-called “ninja gap.” Domain-specific language (DSL) is one of the promising technologies for achieving both code performance and productivity in software development for numerical simulation. In this section, [19], which is a library for tensor operations in two-dimensional(2D)/3D spaces, is presented. AutoMT stands for “automation”-“matrix”-“tensor,” or “MT” represents “mathematical toolkit.” It also supports a type of DSL, which can perform a translation from a LaTeX formula into C/Fortran source code.

10.1.5.1 AutoMT Library for Tensor and Small-Matrix Operations

Currently, we are developing a DSL dedicated to the continuum mechanics field [20, 21], and its associated library for small-matrix and tensor operations. It is called LexADV_AutoMT (AutoMT).

The numerical library called AutoMT is dedicated to tensor and small-matrix operations used in the continuum mechanics field. It manages tensor quantities in 3D space, such as scalar, vector, second-order tensor, and fourth-order tensor. It can also support the very small corresponding matrix and vectors, such as 3×3 and 6×6 . These tensor and matrix operations appear often in element-wise, cell-wise, and particle-wise calculations in a continuum mechanics-based simulation code.

We consider the following simple example of 3D tensor operations. This is a simple formula that uses scalar, vector, and tensor quantities:

$$s = (aX) \cdot b, \quad (10.10)$$

where s is a scalar, a and b are vectors, and X is a second-order tensor. This example requires a product operation between a vector and tensor, followed by a dot product operation between two vectors. AutoMT supports both C and Fortran languages. Using C, the AutoMT library is called in the following manner:

```
double a[3], X[3][3], b[3], s, tmp[3];
AutoMT_prod_v_t_v (a, X, tmp);
AutoMT_cdot_v_v_s (tmp, b, &s);
```


It can also be written in Fortran as follows:

```
real*8 a(3), X(3, 3), b(3), s, tmp(3)
call automt_prod_v_t_v (a, X, tmp)
call automt_cdot_v_v_s (tmp, b, s)
```

In both cases, a vector is represented as a one-dimensional array variable, whereas a second-order tensor is represented as a 2D array variable.

Implementations of the library for various types of high-performance computing platforms, such as Intel x86, Xeon Phi (Knights Landing), and K computer/Fujitsu PRIMEHPC FX100, are also available. Each implementation is highly tuned for the underlying hardware architecture based on the design concept of a “high-performance design pattern.” In these highly tuned versions, the implementations use preprocessor macros instead of C functions/Fortran subroutines. They represent each data object of 3D vectors, tensors, or small matrices as a set of scalar variables instead of array/structure.

10.1.5.2 AutoMT DSL

Additionally, on top of the AutoMT matrix and tensor library, a LaTeX-like DSL is built. The language supports the concept of tensors and small matrices directly as abstract types, which is often used in the context of solid and fluid mechanics. The syntax of the DSL is based on LaTeX, which is one of the major choices for writing engineering documents and mathematical formulas. The DSL translator converts from DSL source code into the corresponding C/Fortran source code, which is mainly composed of function/subroutine calls to the AutoMT library.

10.1.6 *ADVENTURE_Magnetic*

ADVENTURE_Magnetic (AdvMag) is a finite element analysis solver for the electromagnetic field using the HDDM with parallel data processing techniques. It was designed as part of the *ADVENTURE* Project, and applied numerical analysis techniques were developed in the HDDMPPS. As a result, AdvMag has been able to solve electromagnetic field problems with tens of billions of DOFs and rotating machine models efficiently on massively parallel computers. Its source code is published via the website of the *ADVENTURE* Project.

AdvMag has the following features:

- AdvMag supports nonlinear magnetostatic analysis, time-harmonic eddy current analysis, and nonsteady eddy current analysis. These functions have been published.
- The high-frequency electromagnetic field analysis function is in development.

- A new domain decomposition technique for the HDDM, including moving bodies, is in development. This technique makes it possible to efficiently analyze rotating machines, such as electric generators or motors, on massively parallel computers.
- AdvMag has some parallel processing modes:
 - single mode where all computations are performed as a single process;
 - shared memory parallel mode with OpenMP;
 - distributed memory parallel mode with MPI;
 - hybrid parallel mode with MPI and OpenMP.
- AdvMag can solve electromagnetic field problems with tens of billions of DOFs.

10.1.6.1 Analysis Functions

In this section, the published analysis functions of AdvMag are outlined.

1. Nonlinear magnetostatic analysis function

This function analyzes nonlinear magnetostatic problems. A finite element equation based on the A method [22] is adopted. The nonlinearity of permeability can be considered using Newton's method or the Picard iteration. This function outputs the magnetic flux density [T] and electromagnetic force [N]. Furthermore, it has analyzed two billion DOFs [23].

2. Time-harmonic eddy current analysis function

This function analyzes time-harmonic eddy current problems. Finite element equations based on the A method or A - ϕ method [24] are adopted. In this function, a linear system with complex numbers is solved. Therefore, it outputs the real and imaginary parts of the magnetic flux density [T], eddy current density [A/m²], and electromagnetic force [N]. Furthermore, it has analyzed 3.5 billion DOFs [25].

3. Nonsteady eddy current analysis function

This function analyzes nonsteady eddy current problems. Finite element equations based on the A method or A - ϕ method are adopted. This function outputs the magnetic flux density [T], eddy current density [A/m²], and electromagnetic force [N] in each time step. A new domain decomposition technique for the HDDM, including moving bodies, is in development based on this function.

10.1.6.2 Analysis with Tens of Billions of Degrees of Freedom

In this section, the high-frequency electromagnetic field analysis with tens of billions of DOFs [26] is introduced.

The high-frequency electromagnetic field analysis function is in development based on AdvMag. This function analyzes high-frequency electromagnetic field problems. A finite element equation based on the E method [27] is adopted. For a time-harmonic scheme of electromagnetic fields, a linear system with complex numbers is solved only once. Therefore, our method can solve high-frequency electromagnetic field problems efficiently.

The conjugate orthogonal conjugate gradient (COCG) and the conjugate orthogonal conjugate residual (COCR) methods are adopted to solve the interface problem. In high-frequency electromagnetic field problems, the residual norms of the COCG method oscillate intensely. However, those of the COCR method oscillate slightly and decrease stably.

The simple hyperthermia applicator model with up to 30 billion DOFs is analyzed. This problem is one of the benchmark problems defined as Testing Electromagnetic field Analysis Method Workshop Problem 29 (TEAM29) [28]. The frequency value is 8 MHz. In this analysis, the dielectric phantom of the shape of a cylinder with a specific dielectric constant of 80 and conductivity of 0.52 S/m is placed in. Computations were performed by all nodes of the Oakleaf-FX supercomputer in the Supercomputing Division, Information Technology Center, the University of Tokyo. This computer consists of 4,800 computer nodes of a Fujitsu PRIMEHPC FX10 massively parallel supercomputer. Its peak performance is 1.135 PFLOPS and it has 150 TB of memory. The COCR method was applied to solve the interface problem. The simplified block diagonal scaling was adopted as a preconditioner of the COCR method. As a result, the model with 10, 20, and 30 billion DOFs was solved in 549, 956, and 1,138 s, respectively. Thus, it was shown that it is possible to analyze electromagnetic field problems with complex numbers and tens of billions of DOFs.

10.1.6.3 Hierarchical Domain Decomposition Method for Devices Including Moving Bodies

In this section, the HDDM for devices including moving bodies [29], which is in development based on the nonsteady eddy current analysis function of AdvMag, is introduced. This method is in development mainly for analyzing rotating machines, such as electric generators or motors.

Because the moving body (e.g., a rotor) moves, the connection relation with the stationary body (e.g., a stator) changes. Therefore, in our method, meshes of the stationary and moving bodies are generated independently, with the element surfaces in the connection surface between the stationary and moving bodies coinciding even if the moving body moves. Then, to prevent the DOFs on the connection surface from being located inside the subdomain, meshes of the stationary and moving bodies are decomposed independently. The DOFs on the connection surface appear on the surface of the subdomains; thus, the DOFs on the connection surface are considered as the interface DOFs. As a consequence, changes in the connection relation between the stationary and moving bodies are replaced with changes in the

communication tables. At first, a set of communication tables of the interface DOFs that are created by the domain decomposition is created. This set is not changed by time evolution. Then, sets of communication tables of the DOFs on the connection surface are created based on the connection relation between the stationary and moving bodies. As many sets are created as the number of connection relations. In the interprocess communications at each time step, sets of communication tables of the interface DOFs that are created by the domain decomposition and the DOFs on the connection surface based on the connection relation in each time step are used in combination.

In our method, there is no difference from the HDDM algorithm without moving bodies, except that different sets of communication tables are used for each time step. Tasks such as updating the coordinate values of the moving body and loading and changing the sets of communication tables are added to AdvMag. Therefore, it is expected that it can conduct the analysis of the moving bodies with high parallel efficiency on massively parallel supercomputers. Moreover, this method is highly versatile. Although it is currently applied only to 3D electromagnetic field analysis, it is expected that it can be applied to structural analysis, heat transfer analysis, analysis in two dimensions, and various elements.

As a result, the parallel efficiencies of our method are equivalent to those of the HDDM without moving bodies on massively parallel supercomputers. Furthermore, the rotating machine model with seven million DOFs that is solved in more than 1 month using conventional sequential computation has been successfully solved in only 1.60 h using our method.

10.1.7 Analysis of a High-Frequency Electromagnetic Field Using Anatomical Human Models with Mesh Smoothing

For a high-accuracy analysis of the high-frequency electromagnetic field using AdvMag with anatomical human models [30, 31], different material boundaries should be represented by curved surfaces. However, even if users use voxel data that has been built on the basis of MRI data for anatomical human models, material boundaries are not smooth, and then an unneeded reflection and scattering of the electric fields have been observed in the numerical analysis. To reduce this noise, we have developed a mesh smoothing technique with tetrahedral elements for smoothing stair shapes of different material boundaries. In this proposed method, material boundaries of a voxel mesh are detected automatically, and then triangular prisms are placed on these boundaries. Furthermore, each voxel and prism is divided into tetrahedral elements. The proposed method is relatively simple and robust for large-scale and complicated shape models.

10.1.7.1 Mesh Generation Technique

Anatomical human body models are constructed using a binary data format wherein types of organs (including air area) are encoded using voxels with sides of 4 mm. The size of the adult male model is 160 voxels wide, 80 voxels deep, and 433 voxels high. First, the file containing the anatomical human body model, which is the only input for this computation, is read. Next, ParMETIS [32] is used to partition the input voxel data into a number of parts. After the initial decomposition, all processing can be performed independently for each node without communication between nodes. In each node, the voxel mesh is transformed to a tetrahedral mesh.

10.1.7.2 Confirmation of Mesh Smoothing

Accuracy verifications for the confirmation of mesh smoothing is performed using the voxel mesh model of TEAM29. This is a benchmark problem that is widely used for high-frequency electromagnetic field analysis.

To detect the resonant frequency and compare solutions with actual measurements, the resonance state was investigated. The frequency band of 60–70 [MHz] was calculated for 0.4 [MHz] steps around resonant frequencies, and the response for every frequency step was investigated. Computations were performed using 3,072 cores (192 nodes) of the FX10 supercomputer. A comparison between measured resonant frequencies [28] is shown in Table 10.3. Resonances did not occur in the frequency band of 60–70 [MHz] using the original voxel mesh and one-level smoothing mesh. By contrast, the obtained solution was in very good agreement with that obtained using two-level smoothing. The maximum error rate between the obtained solution and measurement was 6.10% in the mode. Therefore, the solution obtained using the proposed two-level smoothing had sufficiently high accuracy.

Figure 10.7 shows a result of mesh smoothing. The anatomical human body torso model was constructed using voxels with 4 mm pitch. The torso model was made from the entire body model by cutting out width $Z_t = 0.25$ m from the position of height $Z_h = 0.95$ m. Figure 10.7a shows the ribs, backbone, and pelvis before smoothing; Fig. 10.7b shows the same after smoothing. As can be seen, we obtained smooth step shapes on the boundaries. The smoothing algorithm was relatively simple and robust. However, in this algorithm, the insertion of triangular prisms increased the size of the model. With the anatomical human model, when smoothing

Table 10.3 Resonant frequencies in MHz(): error rate between measured data (68.6 [MHz]) and the anatomical solution [%])

Smoothing type	Result
None (Original voxel model)	N/A
One-level smoothing	N/A
Two-level smoothing	64.4 (6.1)

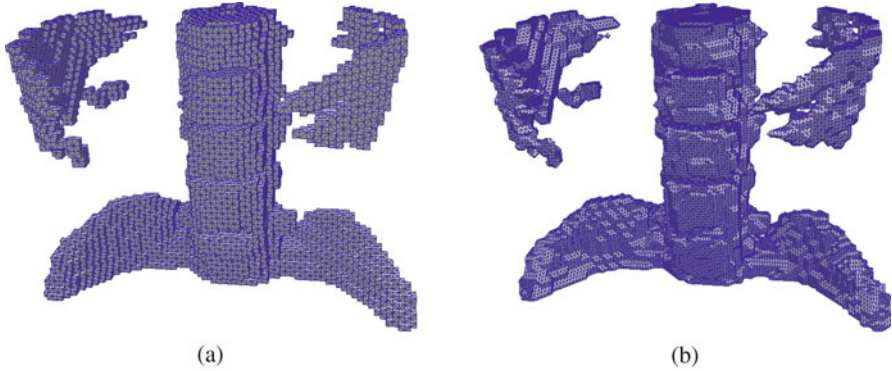


Fig. 10.7 Smoothing example (ribs, backbone, and pelvis). (a) Original. (b) Smoothed

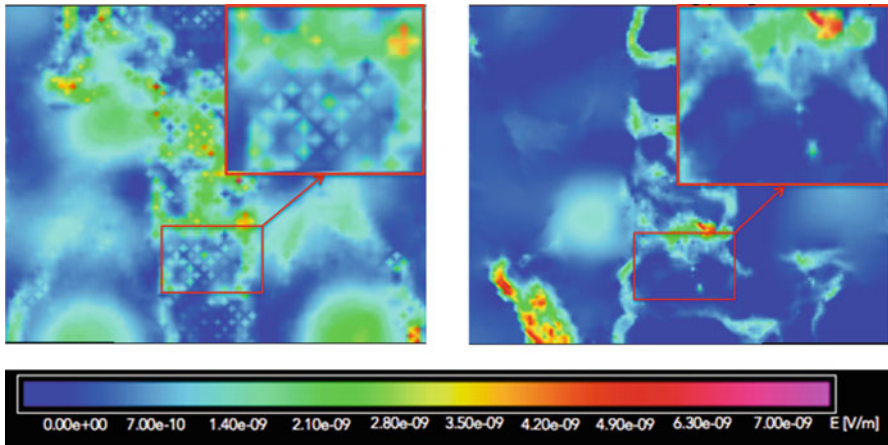


Fig. 10.8 Numerical results of high-frequency electromagnetic field analysis using the torso model (300 MHz, left, without smoothing; right, with smoothing)

was executed, the model size became ten times larger. To solve such large models, we considered using HPC systems. Figure 10.8 shows visualization examples of the electric field. These contour maps are visualized as a cut plane in the human body. Electric fields around the backbone and pelvis can be observed. In the result without smoothing (Fig. 10.8 (left)), reflection and scattering of the electric fields occurred around the bones. By contrast, in the result obtained with smoothing (Fig. 10.8 (right)), the electric field shows a natural distribution in bones and the boundaries of other organs. It is understood that smoothing decreased noise in the electric field along the voxel shape in the vicinity of the spine surface by comparing the enlarged views in Fig. 10.8. From these results, because of mesh smoothing, reflection and scattering of the electric fields were not found.

References

1. Yagawa, G., Shioya, R.: Parallel finite elements on a massively parallel computer with domain decomposition. *Comput. Syst. Eng.* **4**, 495–503 (1994)
2. ADVENTURE Project (2018). <http://adventure.sys.tu.tokyo.ac.jp/>
3. LexADV: Development of a numerical library based on hierarchical domain decomposition for post petascale simulation (2018). <http://adventure.sys.t.u-tokyo.ac.jp/lexadv/index.html>
4. Glowinski, R., Dinh, Q. V., Periaux, J.: Domain decomposition methods for nonlinear problems in fluid dynamics. *Comput. Methods Appl. Mech. Eng.* **40**, 27–109 (1983)
5. Quarteroni, A., Vali, A.: *Domain Decomposition Methods for Partial Differential Equations* (Clarendon Press, Oxford, 1999)
6. Yoshimura, S., Shioya, R., Noguchi, H., Miyamura, T.: Advanced general-purpose computational mechanics system for large scale analysis and design. *J. Comput. Appl. Math.* **149**, 279–296 (2002)
7. Amestoy, P.R., Duf, I.S. and L'Excellent, J.-Y.: Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Methods Appl. Mech. Eng.* **184**, 501–520 (2000)
8. Fujii, A., Nishida, A., Oyanagi Y.: Evaluation of parallel aggregate creation orders: smoothed aggregation algebraic multigrid method. In: *High Performance Computational Science and Engineering*, pp. 99–122. Springer, New York (2005)
9. Ogino, M., Shioya, R., Kanayama, H.: An inexact balancing preconditioner for large-scale structural analysis. *J. Comput. Sci. Tech.* **2**, 150–161 (2008)
10. Mandel, J.: Balancing domain decomposition. *Commun. Numer. Methods Eng.* **9**, 233–241 (1993)
11. Okuda, H., et al.: Parallel Finite Element Analysis Platform for the Earth Simulator: GeoFEM. *Lecture Notes in Computer Science*, vol. 2659, pp. 773–780. Springer, Berlin/Heidelberg (2003)
12. Dongarra, J., et al.: The international exascale software project roadmap. *Int. J. High Perform. Comput. Appl.* **25**, 3–60 (2011)
13. Fabian, N., et al.: The paraview coprocessing library: a scalable, general purpose in situ visualization library. *IEEE Symposium on Large Data Analysis and Visualization*, Providence, pp. 89–96 (2011)
14. Wada, Y., et al.: High-resolution visualization library for ex-ascal supercomputer. In: Dobashi, Y., Ochiai, H. (eds.) *Mathematical Progress in Expressive Image Synthesis III*. Springer Science Mathematics for Industry, pp. 83–94. Springer, Singapore (2016)
15. Wada, Y., et al.: Development of LexADV_VSCG library for a viewer with high resolution image. In: *Proceedings of 21th JSCES Conference*, vol. 21, 2p (2016)
16. Koshizuka, S., Oka, Y.: Moving-particle semi-implicit method for fragmentation of incompressible fluid. *Nucl. Sci. Eng.* **123**, 421–434 (1996)
17. Murotani, K., et al.: Development of hierarchical domain decomposition explicit MPS method and application to large-scale tsunami analysis with floating objects. *J. Adv. Simul. Sci. Eng.* **1**(1), 16–35 (2014)
18. Koshizuka, S., Nobe, A., Oka, Y.: Numerical analysis of breaking waves using the moving particle semi-implicit method. *Int. J. Numer. Methods Fluids* **26**, 751–769 (1998)
19. Kawai, H., et al.: AutoMT, a library for tensor operations and its performance evaluation for solid continuum mechanics applications. *Mech. Eng. Lett.* **1**, Paper No. 15-00349 (2015)
20. Holzapfel, G.A.: *Nonlinear Solid Mechanics – A Continuum Approach for Engineering*. Wiley, New York (2000)
21. Bonet, J., Wood, R.D.: *Nonlinear Continuum Mechanics for Finite Element Analysis*. Cambridge University Press, Cambridge/New York (2008)
22. Kanayama, H., Zheng H., Maeno, N.: A domain decomposition method for large-scale 3-D nonlinear magnetostatic problems. *Theor. Appl. Mech. Jpn.* **52**, 247–254 (2003)

23. Sugimoto, S., Ogino, M., Kanayama, H., Yoshimura, S.: Introduction of a direct method at subdomains in non-linear magnetostatic analysis with HDDM. In: 2010 International Conference on Broadband, Wireless Computing, Communication and Applications, Fukuoka, pp. 304–309 (2010)
24. Kanayama, H., Sugimoto, S.: Effectiveness $A\text{-}\phi$ method in a parallel computing with an iterative domain decomposition method. *IEEE Trans. Magn.* **42**(4), 539–542 (2006)
25. Sugimoto, S., et al.: Improvement of convergence in time-harmonic eddy current analysis with hierarchical domain decomposition method. *Trans. Jpn. Soc. Simul. Tech.* (in Japanese) **7**(1), 110–117 (2015)
26. Sugimoto, S., Takei, A., Ogino, M.: Finite element analysis with tens of billions of degrees of freedom in a high-frequency electromagnetic field. *Mech. Eng. Lett.* **3**, Paper No. 16-0067 (2017)
27. Takei, A., Yoshimura, S., Kanayama, H.: Large scale parallel finite element analyses of high frequency electromagnetic field in commuter trains. *Comput. Model. Eng. Sci.* **31**(1), 13–24 (2008)
28. Kanai, Y.: Description of TEAM workshop problem 29: whole body cavity resonator. Technical report TEAM Workshop in Tucson (1998)
29. Sugimoto, S., Ogino, M., Kanayama H., Takei, A.: Hierarchical domain decomposition method for devices including moving bodies. *J. Adv. Simul. Sci. Eng.* **4**(1), 99–116 (2018)
30. NICT EMC group home page (2018). <http://emc.nict.go.jp/bio/index.html>
31. Conil, E., et al.: Variability analysis of SAR from 20 MHz to 2.4 GHz for different adult and child models using finite-difference time-domain. *Phys. Med. Biol.* **53**, 1511–1525 (2008)
32. Karypis, G., Kumar, V.: A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.* **20**(1), 359–392 (1999)

Chapter 11

Advanced Computing and Optimization Infrastructure for Extremely Large-Scale Graphs on Post-peta-scale Supercomputers



Katsuki Fujisawa, Toyotaro Suzumura, Hitoshi Sato, Koji Ueno, Satoshi Imamura, Ryo Mizote, Akira Tanaka, Nozomi Hata, and Toshio Endo

Abstract In this paper, we present our ongoing research project. The objective of many ongoing research projects in high-performance computing (HPC) areas is to develop an advanced computing and optimization infrastructure for extremely large-scale graphs on the peta-scale supercomputers. The extremely large-scale graphs that have recently emerged in various application fields, such as transportation, social networks, cybersecurity, disaster prevention, and bioinformatics, require fast and scalable analysis. The Graph500 benchmark measures the performance of

K. Fujisawa (✉)

Institute of Mathematics for Industry, Kyushu University, Nishi-ku Fukuoka, Japan
e-mail: fujisawa@imi.kyushu-u.ac.jp

T. Suzumura

Barcelona Supercomputer Center, Barcelona, Spain
e-mail: toyotaro.suzumura@bsc.es

H. Sato · R. Mizote

Artificial Intelligence Research Center, National Institute of Advanced Industrial Science and Technology, Tokyo, Japan
e-mail: hitoshi.sato@aist.go.jp; ryo.mizote@aist.go.jp

K. Ueno

Fixstars Corporation, Gatecity Osaki West Tower, Tokyo, Japan
e-mail: kojiueno5@gmail.com

S. Imamura

Fujitsu Laboratories LTD., Kawasaki, Kanagawa, Japan
e-mail: s-imamura@jp.fujitsu.com

A. Tanaka · N. Hata

Graduate School of Mathematics, Kyushu University, Nishi-ku Fukuoka, Japan
e-mail: tanaka.akira.528@s.kyushu-u.ac.jp; n.hata.122@s.kyushu-u.ac.jp

T. Endo

Global Scientific Information and Computing Center, Tokyo Institute of Technology, Tokyo, Japan
e-mail: endo@is.titech.ac.jp

© Springer Nature Singapore Pte Ltd. 2019

M. Sato (ed.), *Advanced Software Technologies for Post-Peta Scale Computing*,
https://doi.org/10.1007/978-981-13-1924-2_11

207

any supercomputer performing a breadth-first search (BFS) in terms of traversed edges per second (TEPS). In 2014–2017, our project team has achieved about 38.6TeraTEPS on K computer and been a winner at the 8th and 10th to 15th Graph500 benchmark. We commenced our research project for developing the Urban OS (Operating System) for a large-scale city in 2013. The Urban OS, which is regarded as one of the emerging applications of the cyber-physical system (CPS), gathers big data sets of the distribution of people and transportation movements by utilizing sensor technologies and storing them in the cloud storage system. In the next step, we apply optimization, simulation, and deep learning techniques to solve them and check the validity of solutions obtained on the cyberspace. The Urban OS employs the graph analysis system developed by this research project and provides a feedback to a predicting and controlling center to optimize many social systems and services. We briefly explain our ongoing research project for realizing the Urban OS.

11.1 Introduction

The objective of many ongoing research projects in high-performance computing (HPC) areas is to develop an advanced computing and optimization infrastructure for extremely large-scale graphs on the peta-scale supercomputers. The extremely large-scale graphs that have recently emerged in various application fields, such as transportation, social networks, cybersecurity, disaster prevention, and bioinformatics, require fast and scalable analysis (Fig. 11.1) [16, 22, 24]. In recent year, the demands for high-speed graph processing have been remarkably increasing after converting the real-world data into the graph data. The graph processing cycle starts from the target relation and the generation of a graph. Next, we analyze and process it by utilizing graph algorithms. We can finally understand the relationship and characteristic of the target. The graph consists of the node and edge sets. For example, a node corresponds to an intersection in a road network, and an edge corresponds to a road between two intersections. In the analysis of social networks such as Twitter, a node corresponds to a user, and an edge corresponds to the Twitter follower relationship between two users. Besides, we usually handle even larger-scale graph data in the cybersecurity and neural network. The number of vertices in the graph networks has grown from billions to trillions and that of the edges from hundreds of billions to tens of trillions (Fig. 11.2). For example, a graph that represents the interconnections of all the neurons of the human brain has over 89 billion vertices and over 100 trillion edges. To analyze these extremely large-scale graphs, we require a new-generation exascale supercomputer, which will not appear until the 2020s, and therefore, we propose a new framework of software stacks for extremely large-scale graph analysis systems, such as parallel graph analysis and optimization libraries on multiple CPUs and GPUs, hierarchal graph stores using nonvolatile memory (NVM) devices, and graph processing and visualization systems.

We have a research team that joined the JST (Japan Science and Technology Agency) CREST (Core Research for Evolutional Science and Technology) post-

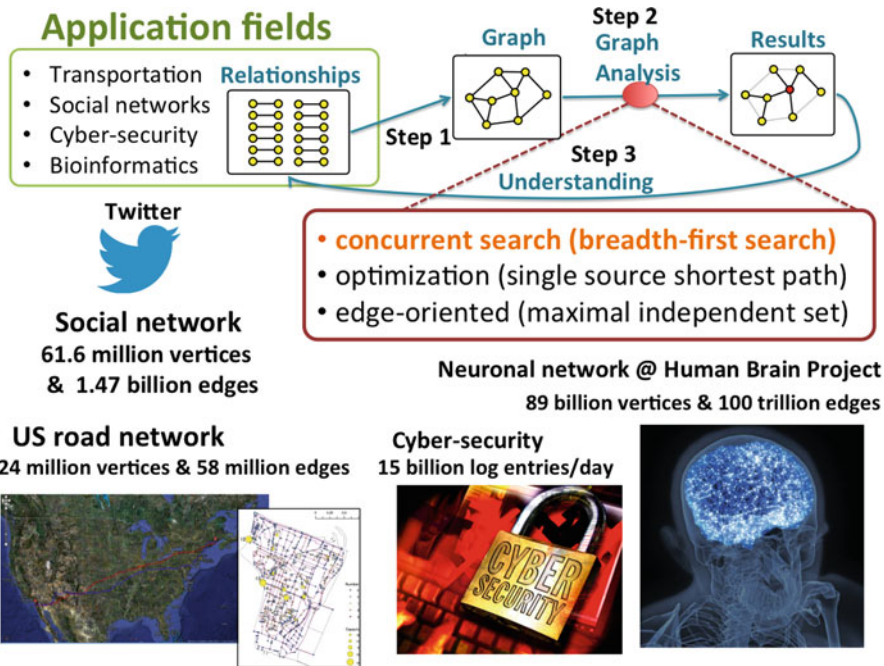


Fig. 11.1 Graph analysis and its application fields. (Image: Illustration by Mirko Ilic)

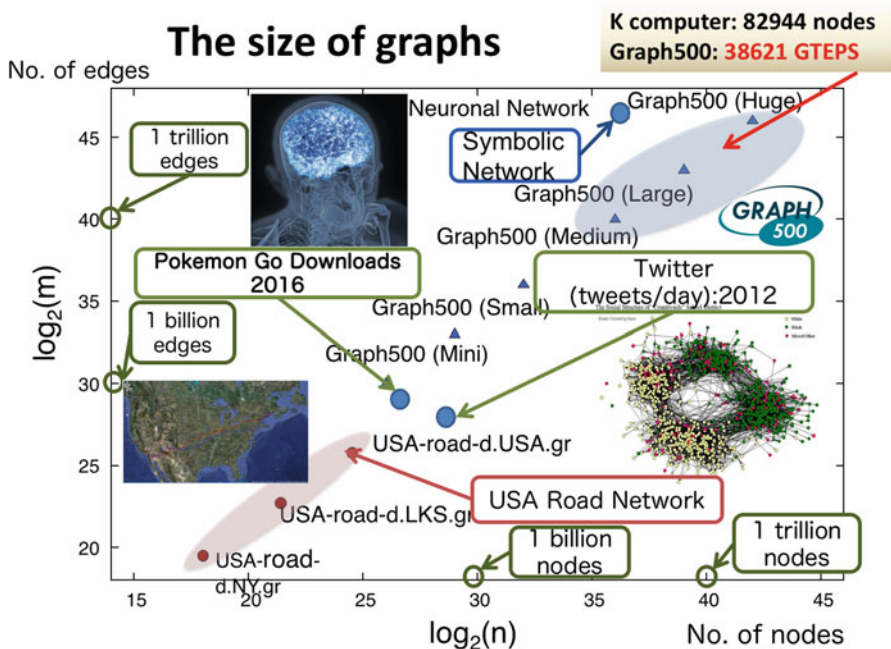


Fig. 11.2 Size of graphs in various application fields and Graph500 benchmark

Peta High Performance Computing project¹ from October 2011 to March 2017. The objective of our researches for the JST CREST project has been developed advanced computing and optimization infrastructures for extremely large-scale graphs on post-peta-scale supercomputers. In this paper, we explain our ongoing research project and show its remarkable results.

11.2 Graph500 and Green Graph500 Benchmarks

The Graph500² and Green Graph500 benchmarks are designed to measure the performance of a computer system for applications that require irregular memory and network access patterns. Following its announcement in June 2010, the Graph500 list was released in November 2010, since when it has been updated semiannually. The Graph500 benchmark measures the performance of any supercomputer performing a breadth-first search (BFS) in terms of traversed edges per second (TEPS). The detailed instructions of the Graph500 benchmark are described as follows:

1. Step 1: Edge List Generation

First, the benchmark generates an edge list of an undirected graph with $n(= 2^{SCALE})$ vertices and $m(= n \cdot edge_factor)$ edges;

2. Step 2: Graph Construction

The benchmark constructs a suitable data structure, such as CSR (compressed sparse row) graph format, for performing BFS from the generated edge list;

3. Step 3: BFS

The benchmark performs BFS to the constructed data structure to create a BFS tree. Graph500 employs TEPS (traversed edges per second) as a performance metric. Thus, the elapsed time of a BFS execution and the total number of processed edges determine the performance of the benchmark;

4. Step 4: Validation

Finally, the benchmark verifies the results of the BFS tree. Note that the benchmark iterates Step 3 and Step 4 64 times from randomly selected start points, and the median value of the results is adopted as the score of the benchmark.

We implemented the world's first GPU-based BFS on the TSUBAME 2.0 supercomputer at the Tokyo Institute of Technology and gained fourth place in the fourth Graph500 list in 2012 [27]. The rapidly increasing number of these large-scale graphs and their applications has attracted significant attention in recent Graph500 lists (Fig. 11.2). In 2013, our project team gained first place in both the big and small data categories in the second Green Graph500 benchmarks. The Green

¹<http://opt.imi.kyushu-u.ac.jp/graphcrest/eng/>

²<https://graph500.org>

Graph500 list collects TEPS-per-watt metrics. Our another implementation, which uses both DRAM and NVM devices and whose objective is to analyze extremely large-scale graphs that exceed the DRAM capacity of the nodes, gained fourth place in the big data category in the second Green Graph500 list.

As we have mentioned in this section, our project team have challenged the Graph500 and Green Graph500 benchmarks, which are designed to measure the performance of a computer system for applications that require irregular memory and network access [10–15, 21, 23, 26–28, 31–33]. We briefly explain four major papers of our research projects for Graph500 and Green Graph500 benchmarks.

1. “Highly Scalable Graph Search for the Graph500 Benchmark” [26]

We found that the provided reference implementations are not scalable in a large distributed environment. We devised an optimized method based on 2D partitioning and other methods such as communication compression and vertex sorting. Our optimized implementation can handle BFS of a large graph with 2^{36} (68.7 billion vertices) and 2^{40} (1.1 trillion) edges in 10.58 s while using 1366 nodes and 16,392 CPU cores on the TSUBAME 2.0 supercomputer at Tokyo Institute of Technology. This performance corresponds to 103.9 GE/s. We also studied the performance characteristics of our optimized implementation and reference implementations on a large distributed memory supercomputer with a Fat-Tree-based Infiniband network.

2. “NUMA-optimized Parallel Breadth-first Search on Multicore Single-node System” [32]

Previous studies [2, 3] have proposed hybrid approaches that combine a well-known *top-down* algorithm and an efficient *bottom-up* algorithm for large frontiers. This reduces some unnecessary searching of outgoing edges in the BFS traversal of a small-world graph, such as a Kronecker graph. In this paper, we describe a highly efficient BFS using column-wise partitioning of the adjacency list while carefully considering the nonuniform memory access (NUMA) architecture. We explicitly manage the way in which each working thread accesses a partial adjacency list in local memory during BFS traversal. Our implementation has achieved a processing rate of 11.15 billion edges per second on a 4-way Intel Xeon E5-4640 system for a scale-26 problem of a Kronecker graph with 2^{26} vertices and 2^{30} edges. Not all of the speedup techniques in this paper are limited to the NUMA architecture system. With our winning Green Graph500 submission of June 2013, we achieved 64.12 GTEPS per kilowatt hour on an ASUS Pad TF700T with an NVIDIA Tegra 3 mobile processor.

3. “Fast and Energy-efficient Breadth-first Search on a single NUMA system” [33]

Our previous NUMA-optimized BFS [32] above reduced memory accesses to remote RAM on a NUMA architecture system; its performance was 11 GTEPS (giga TEPS) on a 4-way Intel Xeon E5-4640 system. Herein, we investigated the computational complexity of the bottom-up, a major bottleneck in NUMA-optimized BFS. We clarify the relationship between vertex out-degree and bottom-up performance. In November 2013, our new implementation achieved a

Graph500 benchmark performance of 37.66 GTEPS (fastest for a single node) on an SGI Altix UV1000 (one-rack) and 31.65 GTEPS (fastest for a single server) on a 4-way Intel Xeon E5-4650 system. Furthermore, we achieved the highest Green Graph500 performance of 153.17 MTEPS/W (mega TEPS per watt) on an Xperia-A SO-04E with a Qualcomm Snapdragon S4 Pro APQ8064.

4. “NVM-based Hybrid BFS with Memory Efficient Data Structure” [14]

We introduce a memory-efficient implementation for the NVM-based hybrid BFS algorithm that merges redundant data structures to a single graph data structure, while off-loading infrequent accessed graph data on NVMs based on the detailed analysis of access patterns, and demonstrate extremely fast BFS execution for large-scale unstructured graphs whose size exceeds the capacity of DRAM on the machine. Experimental results of Kronecker graphs compliant to the Graph500 benchmark on a 2-way INTEL Xeon E5-2690 machine with 256 GB of DRAM show that our proposed implementation can achieve 4.14 GTEPS for a SCALE31 graph problem with 2^{31} vertices and 2^{35} edges, whose size is 4 times larger than the size of graphs that the machine can accommodate only using DRAM with only 14.99 % performance degradation. We also show that the power efficiency of our proposed implementation achieves 11.8 MTEPS/W. Based on the implementation, we have achieved the third and fourth position of the Green Graph500 list (2014 June) in the big data category.

5. “Evaluating the Impacts of Code-Level Performance Tunings on Power Efficiency” [10]

As the power consumption of HPC systems will be a primary constraint for exascale computing, a main objective in HPC communities is recently becoming to maximize power efficiency (i.e., performance per watt) rather than performance. Although programmers have spent a considerable effort to improve performance by tuning HPC programs at a code level, tunings for improving power efficiency is now required. In this work, we select two representative HPC programs (Graph500 and SDPARA) and evaluate how traditional code-level performance tunings applied to these programs affect power efficiency. We also investigate the impacts of the tunings on power efficiency at various operating frequencies of CPUs and/or GPUs. The results show that the tunings significantly improve power efficiency, and different types of tunings exhibit different trends in power efficiency by varying CPU frequency. Finally, the scalability and power efficiency of state-of-the-art Graph500 implementations are explored on both a single-node platform and a 960-node supercomputer. With their high scalability, they achieve 27.43 MTEPS/watt with 129.76 GTEPS on the single-node system and 4.39 MTEPS/watt with 1,085.24 GTEPS on the supercomputer.

6. “Efficient Breadth-First Search on Massively Parallel and Distributed-Memory Machines” [28]

There are many large-scale graphs in real world such as Web graphs and social graphs. The interest in large-scale graph analysis is growing in recent years. BFS is one of the most fundamental graph algorithms used as a component of many graph algorithms. Our new method for distributed parallel BFS can compute BFS for one trillion vertices graph within half a second, using large supercomputers

such as the K computer. By the use of our proposed algorithm, the K computer was ranked first in Graph500 using all the 82,944 nodes and achieved 38,621.4 GTEPS. Based on the hybrid BFS algorithm by Beamer (Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum, IPDPSW 13, IEEE Computer Society, Washington, 2013), we devise sets of optimizations for scaling to extreme number of nodes, including a new efficient graph data structure and several optimization techniques such as vertex reordering and load balancing. Our performance evaluation on K computer shows that our new BFS is 3.19 times faster on 30,720 nodes than the base version using the previously known best techniques.

We finally succeeded in coping with large-scale and complicated real data expected in the future and developing graph search software with the world’s highest performance. We combined highly advanced software technologies:

1. algorithms to reduce redundant graph searches
2. optimization of communication performance on massively parallel computers connected by thousands to tens of thousands of high-speed networks
3. optimization of memory access on multicore processors.

In 2014–2017, our project team has achieved about 38.6TeraTEPS on K computer and been a winner at the 8th and 10th to 15th Graph500 benchmark (Fig. 11.3).

Figure 11.4 shows an application of the Graph500 benchmark. We slightly modified the source code for the Graph500 benchmark, which was applied to



Fig. 11.3 Our project team were awarded the first place in the 8th and 10th to 15th Graph500 benchmark

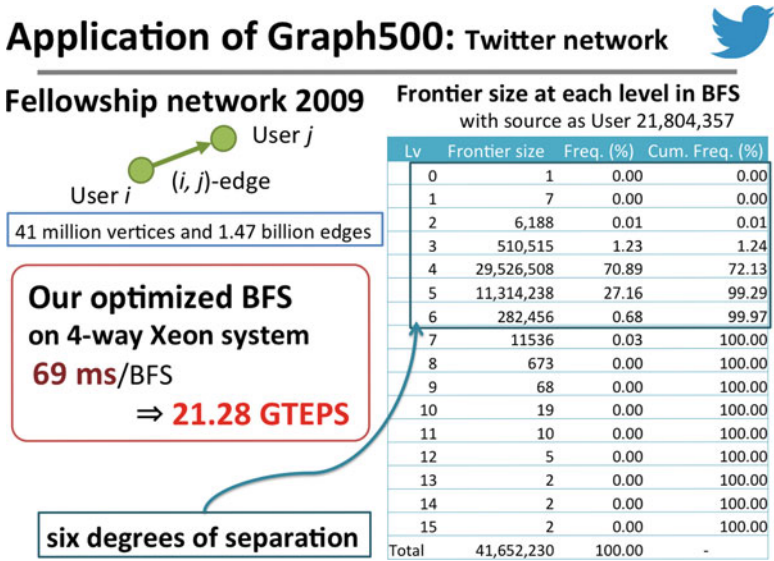


Fig. 11.4 Application of Graph500 benchmarks

making a BFS tree of the Twitter Fellowship Network 2009. It takes only about 70 ms to make a BFS tree from a root node, although this graph has 41 million vertices and 1.47 billion edges.

11.3 High-Performance Computing for Mathematical Optimization Problems

We also present our parallel implementation for large-scale mathematical optimization problems [4–6, 25, 29, 30]. In the last decade, mathematical optimization programming problems have been intensively studied in both their theoretical and practical aspect in a wide range of fields, such as combinatorial optimization, structural optimization, control theory, economics, quantum chemistry, sensor network location, data mining, and machine learning [1, 9, 14, 19]. The semidefinite programming (SDP) problem is a predominant problem in mathematical optimization. The primal-dual interior-point method (PDIPM) is one of the most powerful algorithms for solving SDP problems, and many research groups have employed it for developing software packages. However, two well-known major bottleneck parts (the generation of the Schur complement matrix (SCM) and its Cholesky factorization) exist in the algorithmic framework of PDIPM. These two parts where bottlenecks occur are called ELEMENTS and CHOLESKY, respectively. The standard-form SDP has the following primal-dual form.

$$\begin{aligned}
\mathcal{P} : & \text{minimize } \sum_{k=1}^m c_k x_k \\
& \text{subject to } X = \sum_{k=1}^m F_k x_k - F_0, \quad X \succeq O. \\
\mathcal{D} : & \text{maximize } F_0 \bullet Y \\
& \text{subject to } F_k \bullet Y = c_k \quad (k = 1, \dots, m), \quad Y \succeq O.
\end{aligned} \tag{11.1}$$

We denote by \mathbb{S}^n the space of $n \times n$ symmetric matrices. The notation $X \succeq O$ ($X \succ O$) indicates that $X \in \mathbb{S}^n$ is a positive semidefinite (positive definite) matrix. The inner-product between $U \in \mathbb{S}^n$ and $V \in \mathbb{S}^n$ is defined by $U \bullet V = \sum_{i=1}^n \sum_{j=1}^n U_{ij} V_{ij}$.

In most SDP applications, it is common for the input data matrices F_0, \dots, F_m to share the same diagonal block structure (n_1, \dots, n_h) . Each input data matrix F_k ($k = 1, \dots, m$) consists of sub-matrices in the diagonal positions as follows:

$$F_k = \begin{pmatrix} F_k^1 & O & O & O \\ O & F_k^2 & O & O \\ O & O & \ddots & O \\ O & O & O & F_k^h \end{pmatrix}$$

where $F_k^1 \in \mathbb{S}^{n_1}, F_k^2 \in \mathbb{S}^{n_2}, \dots, F_k^h \in \mathbb{S}^{n_h}$.

Note that $\sum_{\ell=1}^h n_\ell = n$ and the variable matrices X and Y share the same block structure. We define n_{\max} as $\max\{n_1, \dots, n_h\}$. For the blocks where $n_\ell = 1$, the constraints of positive semidefiniteness are equivalent to the constraints of the nonnegative orthant. Such blocks are sometimes called linear programming (LP) blocks.

The size of a given SDP problem can be approximately measured in terms of four metrics.

1. m : the number of equality constraints in the dual form \mathcal{D} (which equals the size of the SCM)
2. n : the size of the variable matrices X and Y
3. n_{\max} : the size of the largest block of input data matrices
4. nnz : the total number of nonzero elements in all data matrices

We denote the time complexities of ELEMENTS and CHOLESKY by $O(mn^3 + m^2n^2)$ and $O(m^3)$, respectively.

We have developed a new version of the semidefinite programming algorithm parallel version (SDPARA), which is a parallel implementation on multiple CPUs and GPUs for solving extremely large-scale SDP problems that have over a million constraints [4, 6, 25]. SDPARA can automatically extract the unique characteristics from an SDP problem and identify the bottleneck. When the generation of SCM becomes a bottleneck part, SDPARA can attain high scalability using a large quantity of CPU cores and some techniques for processor affinity and memory interleaving. SDPARA can also perform parallel Cholesky factorization using thousands of GPUs and techniques to overlap computation and communication

if an SDP problem has over two million constraints and Cholesky factorization constitutes a bottleneck.

We developed a new version of SDPARA 7.6.0-G, which is a parallel implementation on multiples CPUs and GPUs for solving extremely large-scale SDP problems. SDPARA is designed to execute PDIPM on parallel computers with distributed memory space. The speedup achieved by SDPARA is essentially attributable to its use of parallel computation to overcome the computational bottlenecks of ELEMENTS and CHOLESKY. Each process reads the input data and stores them and all variables in the process memory space, while the SCM data are divided between processes. We previously reported that SDPARA can compute each row of the SCM in parallel and applied the parallel Cholesky factorization provided by ScaLAPACK to the SCM. In our previous work [4], we developed SDPARA 7.5.0-G on TSUBAME 2.0,³ which is a high-performance GPU-accelerated supercomputer at the Tokyo Institute of Technology. We solved the largest SDP problem (which has over 1.48 million constraints) and created a new world record in 2012. In the same year, our implementation also achieved 533 TFlops in double precision for large-scale Cholesky factorization using 4,080 GPUs. We demonstrated that SDPARA is a high-performance general solver for SDPs in various application fields through numerical experiments at the TSUBAME 2.5 supercomputer, and we solved the largest SDP problem (which has over 2.33 million constraints), thereby creating a new world record. Our implementation also achieved 1.713 PFlops [6] and 1.774 PFlops [25] in double precision for large-scale Cholesky factorization using 2,720 CPUs and 4,080 GPUs (Fig. 11.5).

11.3.1 Extremely Large-Scale Parallel Cholesky Solver

As mentioned in the previous section, SDP has many applications that involve SDP problems with special structures. We initiated the SDPA project,⁴ which aims to develop high-performance software packages for SDP, and we have solved a large number of SDP problems since 1995; therefore, we can classify the various types of SDP problems into the following three cases:

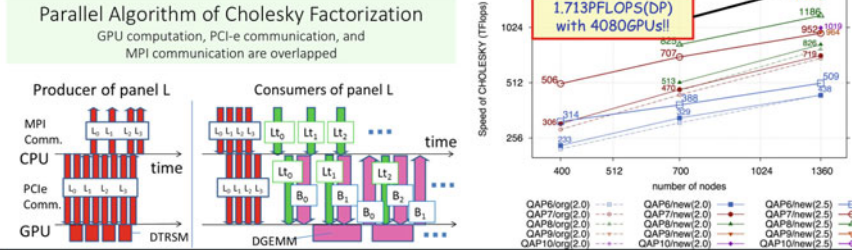
1. Case 1: SDP problems are sparse and satisfy the property of correlative sparsity; therefore, SCM tends to become sparse (e.g., the sensor network location problem and the polynomial optimization problem). In this case, CHOLESKY is the bottleneck part of PDIPM.
2. Case 2: m is less or not considerably greater than n , and SCM is fully dense (e.g., the quantum chemistry problem and the truss topology problem). In this case, ELEMENTS is the bottleneck part of PDIPM, so we can decrease the time complexity of ELEMENTS $O(mn^3 + m^2n^2)$ to $O(m^2)$ by exploiting the sparsity of the data matrix.

³<http://www.gsic.titech.ac.jp/en/tsubame>

⁴<http://sdpa.sourceforge.net/>

High-Performance General Solver for Extremely Large-scale Semidefinite Programming Problems (SDP)

1. **Mathematical Programming**: one of the most important mathematical programming
 2. **Many Applications**: combinatorial optimization, control theory, structural optimization, quantum chemistry, sensor network location, data mining, etc.



- **SDPARA** is a parallel implementation of the interior-point method for Semidefinite Programming Parallel computation for **two major bottlenecks**
 - **ELEMENTS** \Rightarrow Computation of Schur complement matrix (SCM)
 - **CHOLESKY** \Rightarrow Cholesky factorization of Schur complement matrix (SCM)
- **SDPARA** could attain high scalability using **16,320 CPU cores** on the TSUBAME 2.5 supercomputer and some techniques of processor affinity and memory interleaving when the computation of SCM (**ELEMENTS**) constituted a bottleneck.
- **With 4,080 NVIDIA GPUs** on the TSUBAME 2.0 & 2.5 supercomputer, our implementation achieved **1.019 PFlops(TSUBAME 2.0) & 1.774PFlops(TSUBAME 2.5: TSUBAME Grand Challenge 2015 Autumn)** in double precision for a large-scale problem (**CHOLESKY**) with over two million constraints.

Fig. 11.5 SDPARA and its performance on TSUBAME 2.0 & 2.5 supercomputer

3. Case 3: m is considerably greater than n , and SCM is fully dense (e.g., the combinatorial optimization problem and quadratic assignment problem(QAP) [4]). In this case, CHOLESKY is the bottleneck part of PDIPM.

We previously reported that SDPARA can certainly determine whether the SCM of an input SDP problem becomes sparse (Case 1) or not (Cases 2 and 3). In the present study, we mainly focused on parallel computation of ELEMENTS and CHOLESKY in Cases 2 and 3, respectively. We also demonstrated that SDPARA is a high-performance general solver for SDPs in various application fields through numerical experiments on the TSUBAME 2.5 supercomputer and solved the largest SDP problem (QAP10), which has over 2.33 million constraints [6]; and we created a new world record. Figure 11.6 and Table 11.1 show the speed of the CHOLESKY component in teraflops. New corresponds to the latest algorithm [6], while org denotes the original algorithm in our previous paper [4]. Our implementation also achieved 1.713 PFlops in double precision for large-scale Cholesky factorization using 2,720 CPUs and 4,080 GPUs. Table 11.2 shows the speed of the CHOLESKY component in teraflops on the CX400 supercomputer at Kyushu University. We have achieved 294.2 TFlops when using 384 GPUs.

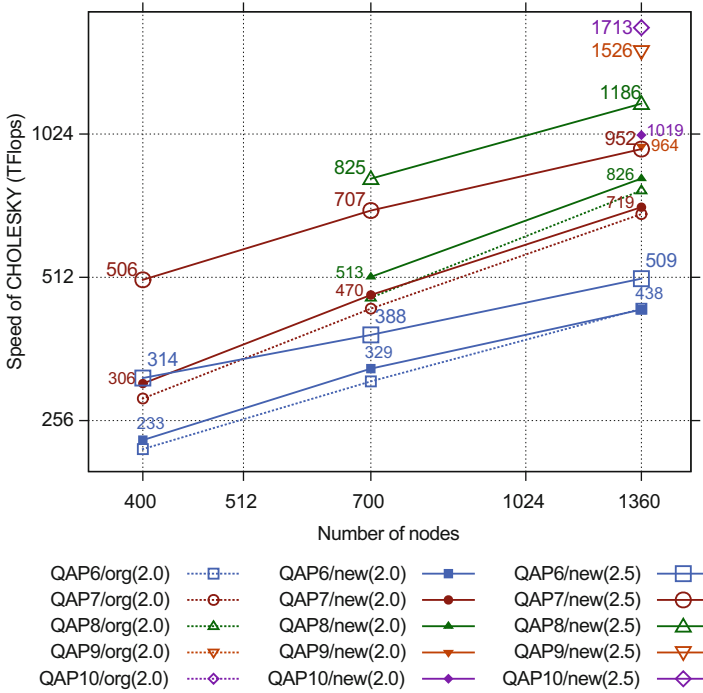


Fig. 11.6 Performance of GPU CHOLESKY obtained by using up to 1360 nodes (4080 GPUs) on TSUBAME 2.0 and 2.5

Table 11.1 Performance (teraflops) of GPU CHOLESKY obtained by using up to 1360 nodes (4080 GPUs) on TSUBAME 2.0 [4] and 2.5 [6]

(a) 400 nodes (1200 GPUs)					(b) 700 nodes (2100 GPUs)				
Name	m	org(2.0)	new(2.0)	new(2.5)	Name	m	org(2.0)	new(2.0)	new(2.5)
QAP6	709,275	223.0	233.0	314.5	QAP6	709,275	309.5	329.0	387.5
QAP7	1,218,400	248.8	306.2	505.8	QAP7	1,218,400	440.0	470.0	707.1
					QAP8	1,484,406	463.8	512.9	825.1

(c) 1360 nodes (4080 GPUs)				
Name	m	org(2.0)	new(2.0)	new(2.5)
QAP6	709,275	439.6	437.8	508.7
QAP7	1,218,400	695.2	718.8	952.0
QAP8	1,484,406	779.3	825.6	1186.4
QAP9	1,962,225	-	964.4	1526.5
QAP10	2,339,331	-	1018.5	1713.0

Table 11.2 Performance (teraflops) of GPU CHOLESKY obtained by using up to 384 nodes (384 GPUs) on CX400

(a) 128 nodes (128 GPUs)

Name	m	org(2.0)	new(2.0)	new(2.5)
QAP6	709,275	–	–	90.1
QAP7	1,218,400	–	–	98.7

(b) 384 nodes (384 GPUs)

Name	m	org(2.0)	new(2.0)	new(2.5)
QAP8	1,484,406	–	–	288.0
QAP8 – 1	1,495,602	–	–	294.2

11.4 Software Stacks for Extremely Large-Scale Graph Analysis System and Future Plans

In this paper, we finally propose new software stacks for an extremely large-scale graph analysis system (Fig. 11.7), which are based on our current ongoing research studies above.

1. **Hierarchical Graph Store:** We propose a hierarchal graph stores and process extremely large-scale graphs with minimum performance degradation by carefully considering the data structures of a given graph and the access patterns to both DRAM and NVM devices. We have developed an extended memory software stack for supporting extreme-scale graph computing. Utilizing emerging NVM devices as extended semi-external memory volumes for processing extremely large-scale graphs that exceed the DRAM capacity of the compute nodes, we design highly efficient and scalable data off-loading techniques, PGAS-based I/O abstraction schemes, and optimized I/O interfaces to NVMs.
2. **Graph Analysis and Optimization Library:** Large-scale graph data are divided between multiple nodes, and then, we perform graph analysis and search algorithms, such as the BFS kernel for Graph500, on multiple CPUs and GPUs. Implementations, including communication-avoiding algorithms and techniques for overlapping computation and communication, are needed for these libraries. Finally, we can make a BFS tree from an arbitrary node and find a shortest path between two arbitrary nodes on extremely large-scale graphs with tens of trillions of nodes and hundreds of trillions of edges.
3. **Graph Processing and Visualization:** We aim to perform an interactive operation for large-scale graphs with hundreds of millions of nodes and tens of billions of edges.

We focus on the graph analysis and optimization library, which are illustrated in Fig. 11.7. Figure 11.8 shows three algorithmic layers of graph analysis and optimization library. We classify many optimization algorithms into three layers according to both of the computation time that we need to solve and the data size of the optimization problem. We have developed parallel software packages for many

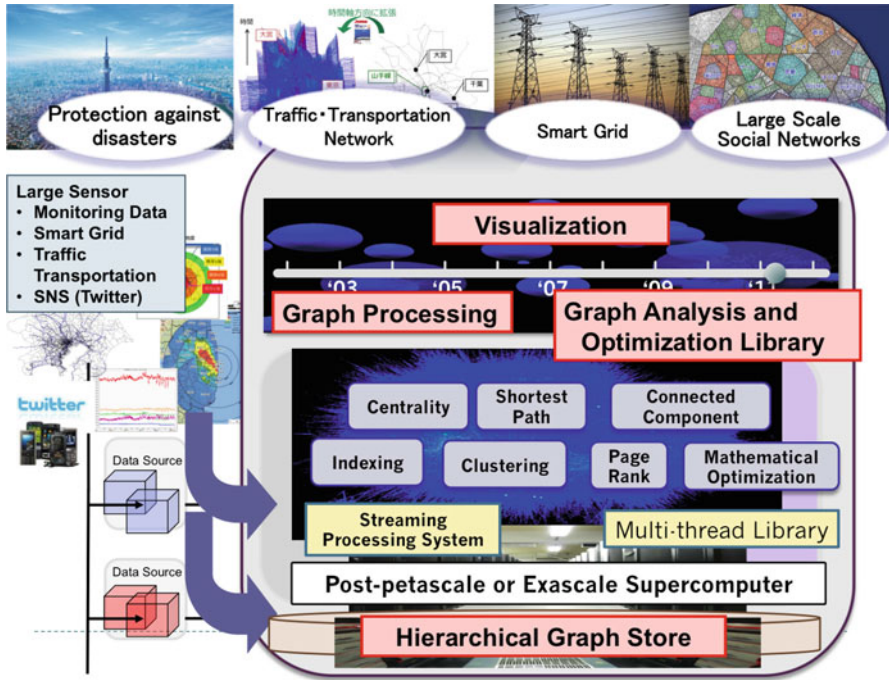


Fig. 11.7 Software stacks for extremely large-scale graph analysis system

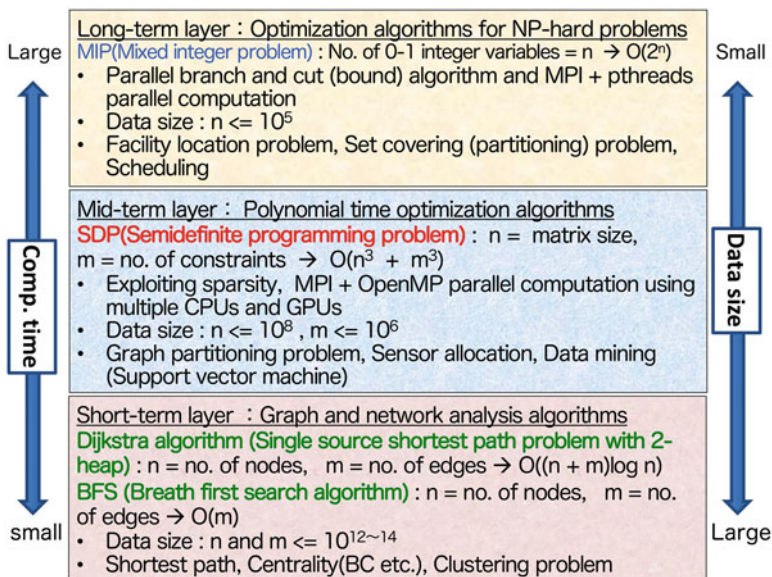


Fig. 11.8 Specification of each layer of HDAOS

optimization problems categorized into these three algorithmic layers. The upper layer contains optimization algorithms for NP-hard problems. The most typical and important optimization algorithm in this layer is a branch-and-cut (bound) algorithm for the mixed integer problem (MIP). We have collaborated with ZIB (Zuse Institute Berlin)⁵ in developing and evaluating parallel (MPI + pthread) software package for solving MIPs [17, 18, 20]. The middle and lower layer contains interior-point algorithms for SDP problems and BFS for graph analysis, respectively. We have released all the software packages developed in our projects until March 2017. We have started the research project for developing the Urban OS (Operating System) and implementing it on a large city (Fukuoka, Japan) from 2013.⁶ The Urban OS gathers big data sets of people and transportation movements by utilizing different sensor technologies and storing them to the cloud storage system. As mentioned in this paper, we have another research project whose objective is to develop advanced computing and optimization infrastructures for extremely large-scale graphs on post-peta-scale supercomputers. The Urban OS employs the graph analysis system developed by this research project and provides a feedback to a predicting and controlling center to optimize many social systems and services.

11.4.1 Hierarchical Data Analysis and Optimization System

In the cyber-physical system (CPS) (Fig. 11.9), it is possible to create new industries by optimizing and simulating the real-world data in social mobility. For this reason, we are attracting significant attention from a number of industries including social infrastructure, manufacturing industry, retail industry, and so on. We commenced our research project for developing the Urban Operating System (OS) for a large-scale city, in 2013. The Urban OS, which is regarded as one of the emerging applications of the cyber-physical system, gathers big data sets of people and transportation movements by utilizing different sensor technologies and storing them in the cloud storage system. As mentioned in our previous papers [7, 8], we have another research project whose objective is to develop advanced computing and optimization of infrastructures for extremely large-scale graphs on post-peta-scale supercomputers. For example, our project team was the winner at the 8th and 10th to 15th Graph500 benchmark⁷ [28]. The Urban OS employs the graph analysis system developed by this research project and provides a feedback to a predicting and controlling center to optimize many social systems and services.

Here, we focus on the HDAOS based on CPS, which are illustrated in Fig. 11.10. First, we gather a variety of data sets on a physical space and generate mathematical models for analyzing the social mobility of real worlds. In the next step, we apply

⁵<http://www.zib.de/>

⁶<http://coi.kyushu-u.ac.jp/en/>

⁷<https://.graph500.org/>

CPS(Cyber Physical System) and Urban OS (Operating System)

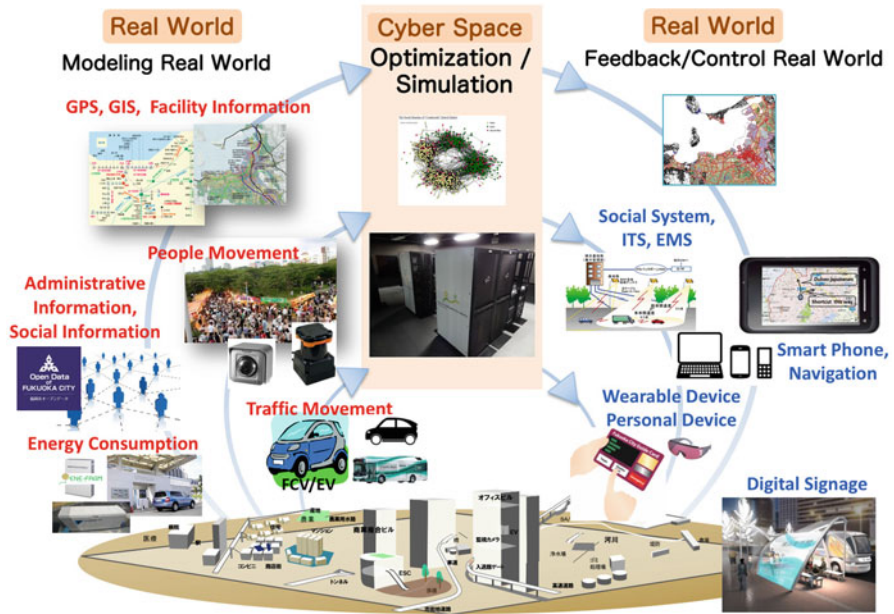


Fig. 11.9 CPS (cyber-physical system) and Urban OS (Operating System)

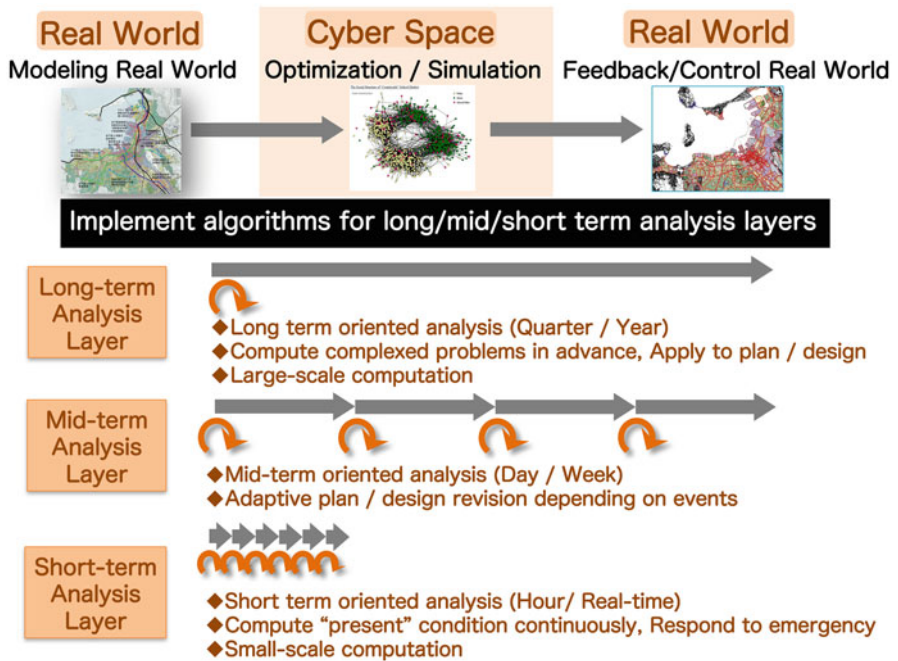


Fig. 11.10 Hierarchical data analysis and optimization system (HDAOS)

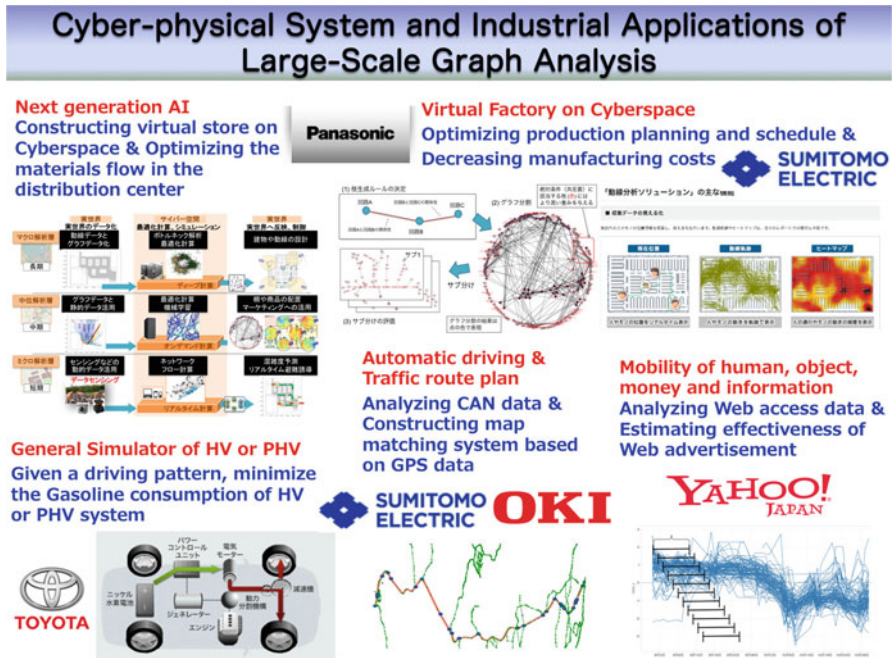


Fig. 11.11 Cyber-physical system and industrial applications of large-scale graph analysis

optimization and simulation techniques to solve them and check the validity of solutions obtained on the cyberspace. We finally feed these solutions into the real world.

Figure 11.8 shows three analysis layers, and we can choose the appropriate one according to a given time for the decision-making process. Figure 11.8 shows the algorithmic specifications of each layer of HDAOS. We classify many optimization algorithms into three layers according to both the computation time needed to solve problems and the data size of the optimization problem. We have developed parallel software packages for many optimization problems categorized into these three algorithmic layers. The long-term analysis layer contains optimization algorithms for NP-hard problems. The most typical and important optimization algorithm in this layer is a branch-and-cut algorithm for the mixed integer problem (MIP). We have collaborated with ZIB in developing and evaluating parallel (MPI + pthread) software packages for solving MIPs. The midterm and short-term analysis layers contain SDP problems [4, 6] and BFS for graph analysis [28], respectively. In the cyber-physical system (Fig. 11.11), it is possible to create new industries by optimizing and simulating the real-world data in social mobility. We gather a variety of data sets by utilizing different sensor technologies and storing them in the cloud storage system via the Internet. In the next step, we generate mathematical models

for analyzing the social mobility of real worlds. We finally feed these solutions into the real world. We will continue these activities toward real-world applications with many development partners.

Acknowledgements This research project was supported by the Japan Science and Technology Agency (JST), the Core Research for Evolutional Science and Technology (CREST), the Center of Innovation Science and Technology based Radical Innovation and Entrepreneurship Program (COI Program), JSPS KAKENHI Grant Number JP 16H01707, and the TSUBAME 2.0 & 2.5 Supercomputer Grand Challenge Program at the Tokyo Institute of Technology.

References

1. Anderson, J.S.M., Nakata, M., Igarashi, R., Fujisawa, K., Yamashita, M.: The second-order reduced density matrix method and the two-dimensional Hubbard model. *Comput. Theor. Chem.* **1003**, 22–27 (2013)
2. Beamer, S., Asanović, K., Patterson, D.A.: Searching for a parent instead of fighting over children: a fast breadth-first search implementation for Graph500. Eecs Department, University of California, UCB/EECS-2011-117, Berkeley (2011)
3. Beamer, S., Asanović, K., Patterson, D.A.: Direction-optimizing breadth-first search. In: *Proceedings of the ACM/IEEE International Conference on High Performance Computing, Networking, Storage and Analysis (SC12)*. IEEE Computer Society, Piscataway (2012)
4. Fujisawa, K., Endo, T., Sato, H., Yamashita, M., Matsuoka, S., Nakata, M.: High-performance general solver for extremely large-scale semidefinite programming problems. In: *2012 ACM/IEEE Conference on Supercomputing, SC12 (2012)*
5. Fujisawa, K., Endo, T., Sato, H., Yasui, Y., Matsuzawa, N., Waki, H.: Peta-scale general solver for semidefinite programming problems with over two million constraints, SC 2013 regular, electronic, and educational poster. In: *International Conference for High Performance Computing, Networking, Storage and Analysis (SC13)*, Denver (2013)
6. Fujisawa, K., Endo, T., Yasui, Y., Sato, H., Matsuzawa, N., Matsuoka, S., Waki, H.: Peta-scale general solver for semidefinite programming problems with over two million constraints. In: *The 28th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2014)*, Phoenix, pp. 1171–1180 (2014)
7. Fujisawa, K., Suzumura, T., Sato, H., Ueno, K., Yasui, Y., Iwabuchi, K., Endo, T.: Advanced computing & optimization infrastructure for extremely large-scale graphs on post peta-scale supercomputers. In: *Proceedings of the Optimization in the Real World – Toward Solving Real-World Optimization Problems–. Series of Mathematics for Industry*, pp. 1–13. Springer (2015)
8. Fujisawa, K., Endo, T., Yasui, Y.: Advanced computing & optimization infrastructure for extremely large-scale graphs on post peta-scale supercomputers. In: *Proceedings of Mathematical Software, ICMS 2016, 5th International Conference, Berlin, 11–14 July 2016. Lecture Notes in Computer Science*, vol. 9725, pp. 265–274. Springer (2016)
9. Gotoh, J.-y., Fujisawa, K.: Convex optimization approaches to maximally predictable portfolio selection. *Optim.: J. Math. Program. Oper. Res.*, vol. 63, pp. 1713–1735, (2014)
10. Imamura, S., Oka, K., Yasui, Y., Inadomi, Y., Fujisawa, K., Endo, T., Ueno, K., Fukazawa, K., Hata, N., Kakibuka, Y., Inoue, K., Ono, T.: Evaluating the impacts of code-level performance tunings on power efficiency. In: *2016 IEEE International Conference on BigData (IEEE BigData 2016)*, Washington, DC
11. Imamura, S., Yasui, Y., Inoue, K., Ono, T., Sasaki, H., Fujisawa, K.: Power-efficient breadth-first search with DRAM row buffer locality-aware address mapping. In: *HPGDMP16: High Performance Graph Data Management and Processing Workshop. In Conjunction with International Conference for High Performance Computing, Networking, Storage and Analysis (SC16)*. IEEE, Piscataway (2016)

12. Iwabuchi, K., Sato, H., Yasui, Y., Fujisawa, K.: Performance analysis of hybrid BFS approach using semi-external memory. SC 2013 regular, electronic, and educational poster. In: International Conference for High Performance Computing, Networking, Storage and Analysis (SC13), Denver (2013)
13. Iwabuchi, K., Sato, H., Mizote, R., Yasui, Y., Fujisawa, K., Matsuoka, S.: Hybrid BFS approach using semi-external memory. In: International Workshop on High Performance Data Intensive Computing (HPDIC 2014) in Conjunction with IEEE IPDPS, Phoenix (2014)
14. Iwabuchi, K., Sato, H., Yasui, Y., Fujisawa, K., Matsuoka, S.: NVM-based Hybrid BFS with memory efficient data structure. In: 2014 IEEE International Conference on BigData (IEEE BigData 2014), Washington, DC (2014)
15. Kakibuka, Y., Yasui, Y., Ono, T., Fujisawa, K., Inoue, K.: Performance evaluation of Graph500 considering CPU-DRAM power shifting, SC17 regular, electronic, and educational poster. In: International Conference for High Performance Computing, Networking, Storage and Analysis 17 (SC17), Denver (2017)
16. Kira, A., Iwane, H., Hirokazu, A., Kimura, Y., Fujisawa, K.: An indirect search algorithm for disaster restoration with precedence and synchronization constraints. *Pac. J. Math. Indus.* **9**, 7 (2017). Springer
17. Koch, T., Ralphs, T., Shinano, Y.: Could we use a million cores to solve an integer program?. *Math. Methods Oper. Res.* **76**, 67–93 (2012)
18. Koch, T., Martin, A., Pfetsch, M.E.: Progress in academic computational integer programming. In: Jünger, M. (eds.) *Facets of Combinatorial Optimization – Festschrift for Martin Grötschel*, pp. 483–506. Springer, Berlin/Heidelberg (2013)
19. Nakata, M., Fukuda, M., Fujisawa, K.: Variational approach to electronic structure calculations on second-order reduced density matrices and the N-representability problem. In: Siedentop, H. (ed.) *Complex Quantum Systems – Analysis of Large Coulomb Systems*, Institute of Mathematical Sciences, National University of Singapore, pp. 163–194 (2013)
20. Shinano, Y., Achterberg, T., Berthold, T., Heinz, S., Koch, T.: ParaSCIP: a parallel extension of SCIP. In: *Competence in High Performance Computing 2010*, pp. 135–148. Springer, Berlin/Heidelberg (2012)
21. Shirahata, K., Sato, H., Matsuoka, S.: Out-of-core GPU memory management for MapReduce-based large-scale graph processing. In: *Proceedings of the 2014 IEEE International Conference on Cluster Computing*, Madrid (2014)
22. Suzumura, T., Ueno, K.: ScaleGraph: a high-performance library for billion-scale graph analytics. In: *2015 IEEE International Conference on BigData (IEEE BigData 2015)*, Santa Clara, pp. 76–84 (2015)
23. Suzumura, T., Ueno, K., Sato, H., Fujisawa, K., Matsuoka, S.: A performance characteristics of Graph500 on large-scale distributed environment. In: *The Proceedings of the 2011 IEEE International Symposium on Workload Characterization*, Austin, pp. 149–158 (2011)
24. Tanaka, A., Hata, N., Tateiwa, N., Fujisawa, K.: Practical approach to evacuation planning via network flow and deep learning. In: *The Fourth International Workshop on High Performance Big Graph Data Management, Analysis, and Mining (BigGraphs 2017)*, to be held in Conjunction with the 2017 IEEE International Conference on Big Data (IEEE BigData 2017), in Boston (2017)
25. Tsujita, Y., Endo, T., Fujisawa, K.: The scalable petascale data-driven approach for the Cholesky factorization with multiple GPUs. In: *First International Workshop on Extreme Scale Programming Models and Middleware*. In Conjunction with International Conference for High Performance Computing, Networking, Storage and Analysis (SC15), Austin, pp 38–45 (2015)
26. Ueno, K., Suzumura, T.: Highly scalable graph search for the Graph500 benchmark. In: *The 21st International ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC 2012)*, Delft (2012)
27. Ueno, K., Suzumura, T.: Parallel distributed breadth first search on GPU. In: *IEEE International Conference on High Performance Computing (HiPC 2013)*, India (2013)
28. Ueno, K., Suzumura, T., Maruyama, N., Fujisawa, K., Matsuoka, S.: Efficient breadth-first search on massively parallel and distributed memory machines. *Data Sci. Eng.* **2**(1), 22–35 (2017). Springer

29. Yamashita, M., Fujisawa, K., Fukuda, M., Kobayashi, K., Nakata, K., Nakata, M.: Latest developments in the SDPA family for solving large-scale SDPs. In: Anjos, M.F., Lasserre, J.B. (eds.) *Handbook on Semidefinite, Conic and Polynomial Optimization*. International Series in Operations Research & Management Science, Chapter 24. Springer, Dordrecht (2011)
30. Yamashita, M., Fujisawa, K., Fukuda, M., Nakata, K., Nakata, M.: Parallel solver for semidefinite programming problem having sparse Schur complement matrix. *ACM Trans. Math. Softw.* **39**(12), 6:1–6:22 (2012)
31. Yasui, Y., Fujisawa, K., Goto, K., Kamiyama, N., Takamatsu, M.: NETAL: high-performance implementation of network analysis library considering computer memory hierarchy. *J. Oper. Res. Soc. Jpn.* **54**(4), 259–280 (2011)
32. Yasui, Y., Fujisawa, K., Goto, K.: NUMA-optimized parallel breadth-first search on multicore single-node system. In: 2013 IEEE International Conference on BigData (IEEE BigData 2013), Santa Clara (2013)
33. Yasui, Y., Fujisawa, K., Sato, Y.: Fast and energy-efficient breadth-first search on a single NUMA system. In: *Intentional Supercomputing Conference (ISC 14)*, (2014)

Chapter 12

Software Technology That Deals with Deeper Memory Hierarchy in Post-petascale Era



Toshio Endo, Hiroko Midorikawa, and Yukinori Sato

Abstract There is an urgent need to develop technology that realizes larger, finer, and faster simulations in meteorology, bioinformatics, disaster measures, and so on, toward post-petascale era. However, the “memory wall” problem will be the one of largest obstacles; the growth of memory bandwidth and capacity will be even slower than that of processor throughput. For this purpose, we suppose system architecture with memory hierarchy including hybrid memory devices, including nonvolatile RAM (NVRAM), and develop new software technology that efficiently utilizes the hybrid memory hierarchy. The area of our research includes new compiler technology, memory management, and application algorithms.

12.1 Introduction

With the existence of many-core accelerators including GPUs and Xeon Phi processors, exascale supercomputers will be realized in a few years to accommodate high-performance simulations in weather, medical, and disaster measurement area using big data. On the other hand, the scales of those simulations will be limited by *the memory wall problem* [9], which is that the improvement of capacity and/or bandwidth of memory is slower than that of processors. This is a well-known and traditional problem, and processor vendors have improved cache architecture to mitigate it for a long time. However, it is becoming even more critical because

T. Endo (✉)

Global Scientific Information and Computing Center, Tokyo Institute of Technology, Tokyo, Japan

e-mail: endo@is.titech.ac.jp

H. Midorikawa

Seikei University, Tokyo, Japan

e-mail: midori@st.seikei.ac.jp

Y. Sato

Toyohashi University of Technology, Aichi, Japan

e-mail: yukinori@cs.tut.ac.jp

© Springer Nature Singapore Pte Ltd. 2019

M. Sato (ed.), *Advanced Software Technologies for Post-Peta Scale Computing*,

https://doi.org/10.1007/978-981-13-1924-2_12

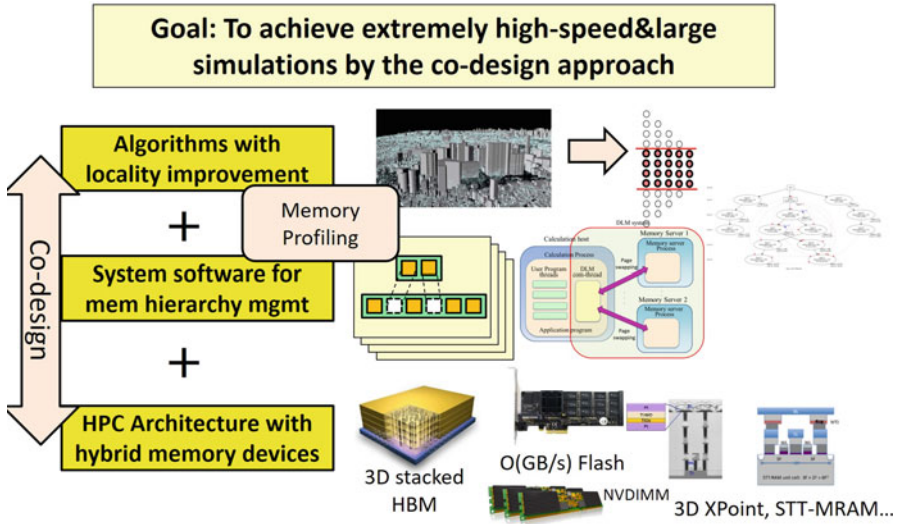


Fig. 12.1 A co-design approach that spans application algorithm, system software, and system architecture toward memory wall problem in post-petascale era

of both architectural tendency and demands from applications including larger and finer scale simulations, which would be coupled with emerging big data technologies.

Thus we require reactions toward memory wall problem, not only in processor architecture but also in system architecture including deeper memory hierarchy. Recently, high-performance flash SSDs with O(GB/s) bandwidth and O(us) access latency are expected to fill the gap between main memory and slow hard disk drives. We should take care of technology evolution such as 3D stacking memory (hybrid memory cube and high bandwidth memory) and next-generational non volatile memory. Next, application algorithms should be reconsidered to exploit memory hierarchy efficiently; the key direction is locality improvement and communication reduction. In stencil computation, which is a major computation in computational fluid dynamics, a technique called *temporal blocking* can largely improve memory access locality. Then system software including runtime libraries and compiler/transpiler have a role to combine such new architecture and new algorithms while reducing software development costs. This co-design approach that spans application algorithm, system software, and system architecture should be pursued toward post-petascale era (Fig. 12.1).

The following part of this section consists as follows. Section 12.2 describes out-of-core implementations of stencil computation that harness high-speed flash SSDs in order to achieve finer simulations that require larger memory capacity than main memory. The implementations adopt multi-level temporal blocking, and parameter tuning is done in a systematic method. Section 12.3 couples the temporal blocking technique with GPGPU architectures equipped with memory hierarchy that consists of GPU device memory, host memory, and high-speed flash SSDs. This

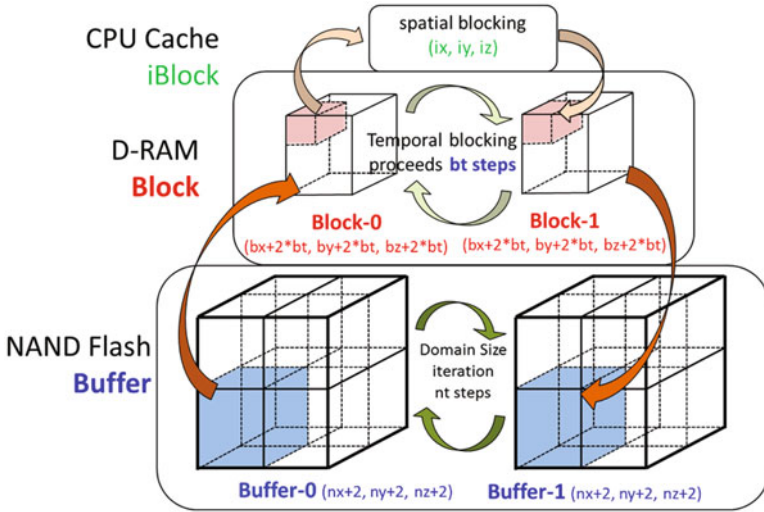


Fig. 12.2 The multi-level tiling in spatial and temporal spaces for three layers, cache, memory, and flash SSDs

section introduces a new runtime library designed to harness this memory hierarchy while reducing application programming costs. Section 12.4 describes a tool chain designed to help application programmers develop and improve user programs for efficient usage of memory hierarchy. The key technology is a memory profiler to capture memory access behaviors of applications. Finally, Sect. 12.5 concludes this chapter.

12.2 Horizontal and Vertical Memory Extensions for Large Data Applications (Midorikawa Group)

12.2.1 Flash-Based Out-of-Core Stencil Computations

The 1000-time latency gap between DRAM and flash is overcome by our advanced implementation using highly parallel AIO (asynchronous input/output) and a novel temporal blocking algorithm designed for flash as shown in Fig. 12.2 [15, 17].

Large number of AIOs issued by multiple threads boost flash I/O bandwidth at the maximum and achieve the highest performance compared to traditional schemes, such as page swap and file map, as shown in Fig. 12.3a.

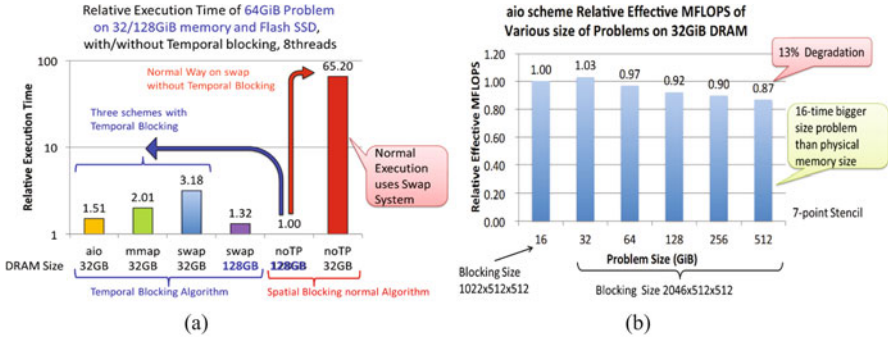


Fig. 12.3 (a) Relative times for various methods for 7-point stencil computation with 32 GiB DRAM, (b) AIO performance in various-size problems using 32 GiB DRAM

Figure 12.3b shows the relative effective Mflops of out-of-core stencil computations with 32 GiB of DRAM using the AIO method for various problems whose data sizes are between 16 and 512 GiB. In the 512-GiB problem, whose data size is 16 times larger than that of the DRAM, 87% of the Mflops execution performance is achieved with DRAM only. In other words, its performance degradation is limited to only 13% compared to that of the normal execution with sufficient DRAM, even if 94% of the program data exist in the flash SSD.

With our algorithm, the available maximum problem size for stencil computations is not limited by the capacity of main memory, but the capacity of local flash SSDs. Moreover, the optimal combinations of spatial and temporal block sizes are easily available in runtime by using Blk-Tune described in the next section.

12.2.2 Blk-Tune: Automatic Blocking Size Setting System

Blk-Tune automatically retrieves platform information and determines the globally optimal spatial/temporal blocking sizes to minimize the amount of I/O traffic to the flash SSD in runtime [14]. It realizes just-in-time selection by a search algorithm without any preliminary executions, which differentiates the Blk-Tune from existing other auto-tuning systems. What a user has to do is only input problem parameters as shown in Fig. 12.4.

Blk-Tune can work as the front end of stencil computations and finds global optimum blocking parameters for the particular problem and the platform in runtime. Blk-Tune is available not only for such online tunings but also for offline tunings for different platforms from the execution platform, with input of hardware parameters, such as capacities of memory/cache and number of CPU cores.

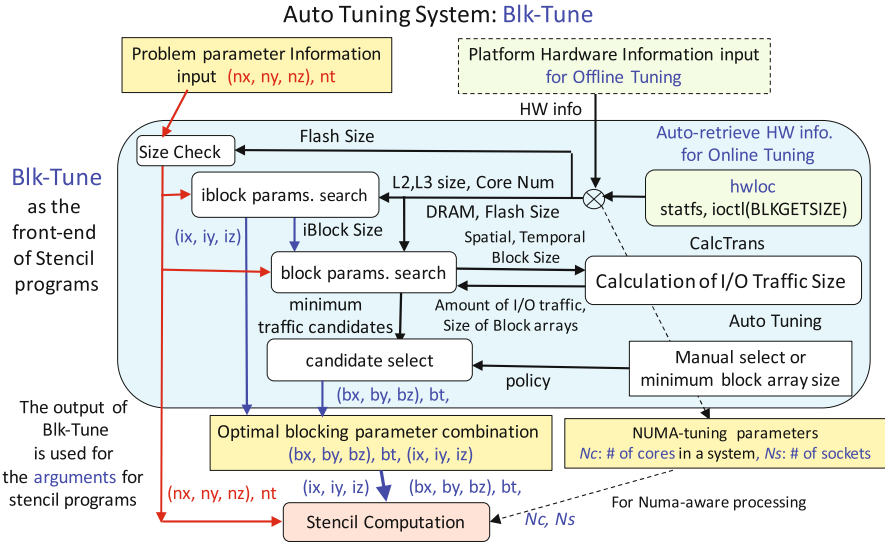


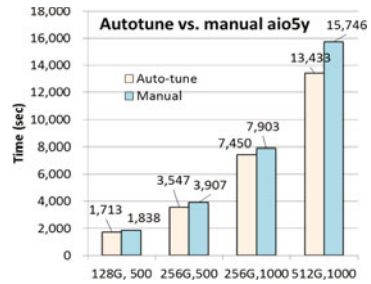
Fig. 12.4 The Blk-Tune internal overview

Blk-Tune outputs for 2 problems (128/256 GiB) and 3 platforms

Problem size	nx	ny	nz	nt			
256GiB-problem	2048	2048	4096	1000			
Server (DRAM, cores)	bx, ix	by	bz	bt	iy	iz	
crest6 (128GiB, 20)	2048	1025	2240	500	1	160	
crest4 (64GiB, 16)	2048	683	1536	334	1	128	
crest0 (32GiB, 8)	2048	513	832	250	1	64	

Problem size	nx	ny	nz	nt			
128GiB-problem	2048	2048	2048	500			
Server (DRAM, cores)	bx, ix	by	bz	bt	iy	iz	
crest6 (128GiB, 20)	2048	1025	2048	500	1	160	
crest4 (64GiB, 16)	2048	1025	1280	500	1	128	
crest0 (32GiB, 8)	2048	513	1152	250	1	64	

(a)



(b)

Fig. 12.5 (a) Blk-Tune output examples, (b) manual selection vs. Blk-Tune for various-size problems on crest6 (Haswell server) system

Figure 12.5a shows Blk-Tune output examples for two different problems in data size and time steps and for three platforms, crest6, crest4, and crest0. Even for the same problem, Blk-Tune outputs different spatial and temporal blocking parameters for each memory layer according to the platform hardware. Figure 12.5b compares the execution times by manual and Blk-tune parameter settings for various problem data sizes and time steps.

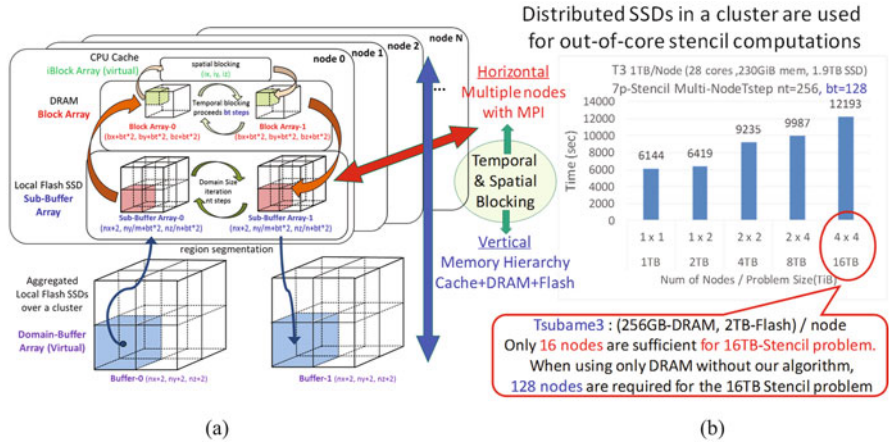


Fig. 12.6 (a) Large-scale stencil computations: vertical memory layers in a local node and a horizontal domain data distribution in a hierarchical temporal blocking computation for a cluster, (b) the execution times of stencil computations in TSUBAME3.0

12.2.3 Large-Scale Stencil Computations Using Distributed Flash SSDs and Memories in a Cluster

To solve large-scale stencil computation problems by using a cluster system, a new algorithm, which explores data access locality in both vertical and horizontal directions as shown in Fig. 12.6a, was proposed [16]. It utilized distributed flash SSDs over cluster nodes as an extension to the main memories of nodes in a cluster. A multi-level blocking scheme for cache, main memory, local flash, and remote node was introduced. The available maximum problem size is not limited by the total capacity of DRAMs in a cluster. It can be expanded to the total capacity of distributed flash SSDs in a cluster.

The results showed that large-scale stencil problems can be solved with a limited number of nodes and a moderate size of main memories, which reduces system cost and energy consumption. When using TSUBAME3.0 supercomputer [12, 31] (256 GiB-DRAM, 2 TB-Flash/node), only 16 nodes are sufficient for 16 TB stencil problem as shown in Fig. 12.6b, which usually requires 128 nodes when using only DRAM without our implementation.

12.2.4 mDLM: User-Level Remote Memory Paging System for Out-of-Core Multi-thread Applications

A user-level remote memory paging system, DLM (distributed large memory) [18], provides virtual large memory for one calculation node by using distributed node

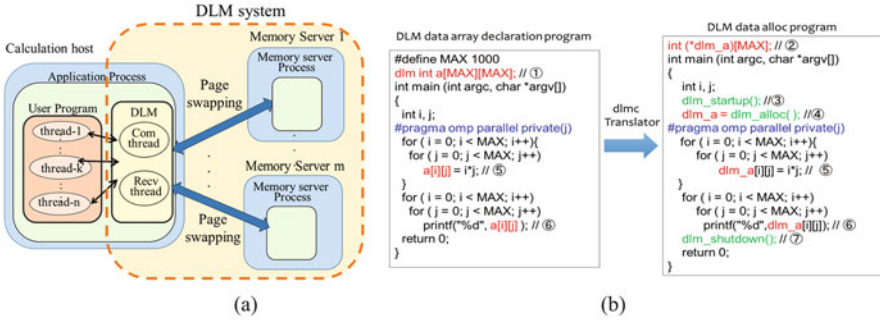


Fig. 12.7 (a) The DLM system: a calculation host and memory servers, (b) the DLM programs

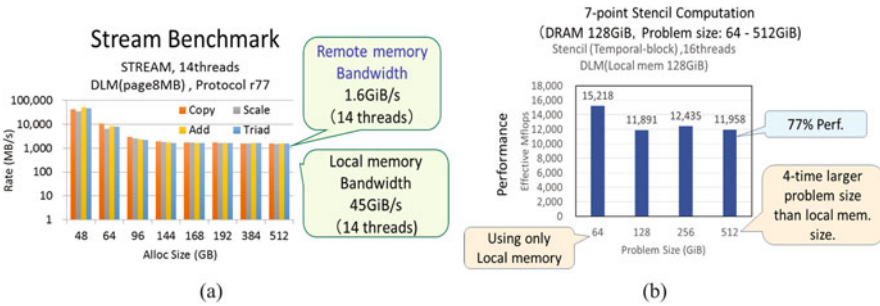


Fig. 12.8 DLM performances: (a) stream benchmark, (b) 7-point stencil computations

memories in a cluster, as shown in Fig. 12.7a. The mDLM [19] supports multi-thread C programs, such as OpenMP and pthread, as shown in Fig. 12.7b.

It was designed for users who need to solve large-size problems using existing algorithms and programs originally designed for shared-memory models for single node. They prefer and accept the extra execution time caused by partially using remote memory instead of the local memory, because converting existing complex algorithms to parallel MPI programs is not an easy task and requires substantial costs.

Figure 12.8a shows stream benchmark for DLM data, where calculation node memory is only 128 GiB. For 512 GiB DLM data, remote memory bandwidth achieves 1.6 GiB/s. However, the performance degradation caused by remote memory paging is limited by using a locality-aware algorithm. Figure 12.8b shows the performances of a temporal blocking 7-point stencil computation for problem sizes, from 64 to 512 GiB, when using only 128 GiB local memory. The performance of 512 GiB problem, whose data size is four times larger than the size of local memory, achieves 77% of the performance in the execution using only local memory.

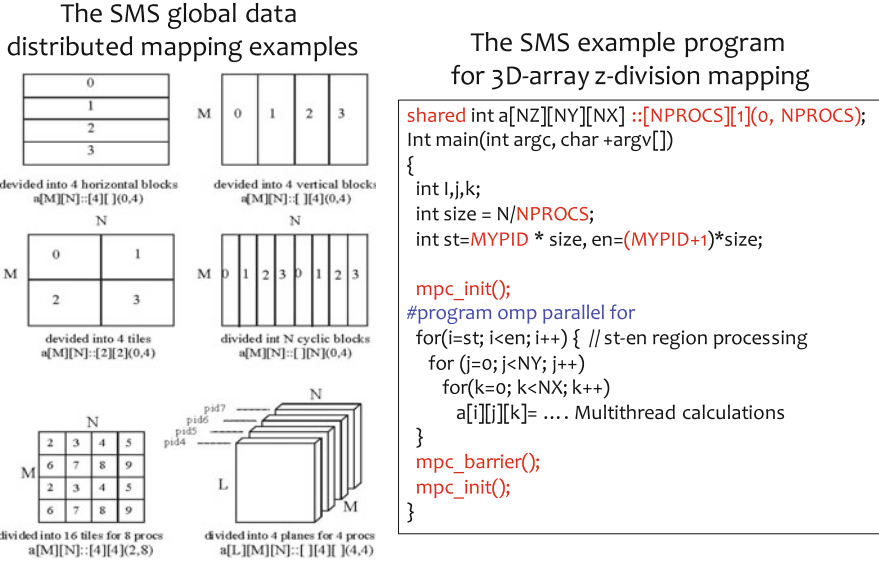


Fig. 12.9 Various SMS global data distributed mappings and a program example

12.2.5 mSMS: Software Distributed Shared Memory for Multi-node and Multi-thread Processing

mSMS, a newly designed software distributed shared memory (SDSM), realizes flexible and productive parallel programming environment for multi-node processing with OpenMP/threads/OpenACC for each node in a cluster. It provides a transparent full-accessible globally shared memory distributed over multiple nodes with the data distribution API shown in Fig. 12.9, which was developed in the traditional software distributed shared memory, SMS [13]. mSMS is very different from other existing PGAS languages and APIs, because the same complete full-accessible global address space is provided to each node, and C pointer-based programs are executable transparently for the global data that is actually distributed over nodes in a cluster with the data distribution interface.

mSMS employs an efficient node communication mechanism by limited number of threads dedicating to simultaneous node communications, and it achieved comparable or more efficient communication performance compared to that obtained by typical MPI + OpenMP programs

The performance of simple stencil computations on TSUBAME3.0 [12, 31] is shown in Fig. 12.10. Large-size (9.2 TiB) problems can be easily implemented on 72 nodes of the cluster by mSMS. The performance of mSMS with *preload* is comparable or even better than that of the MPI implementation in 7-point stencil. In more calculation-dominant 27-point stencil, the mSMS performance achieves 97–93% of the MPI performance without *preload*.

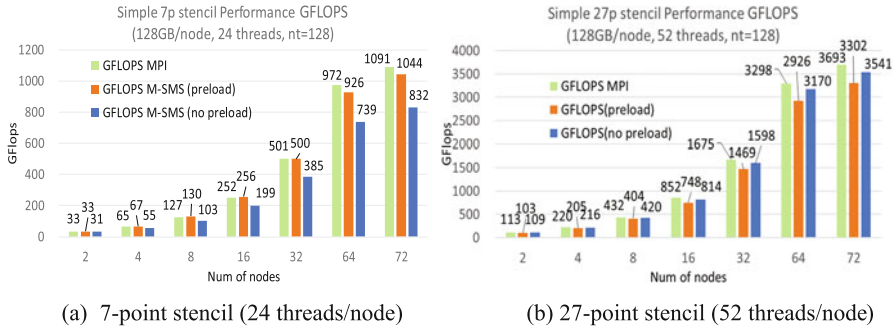


Fig. 12.10 The mSMS and MPI performances in 7-point and 27-point stencil computation (128 GiB–9.2 TiB data problems, 128 time steps, 128 GiB/node, 2–72 nodes in TSUBAME3.0)

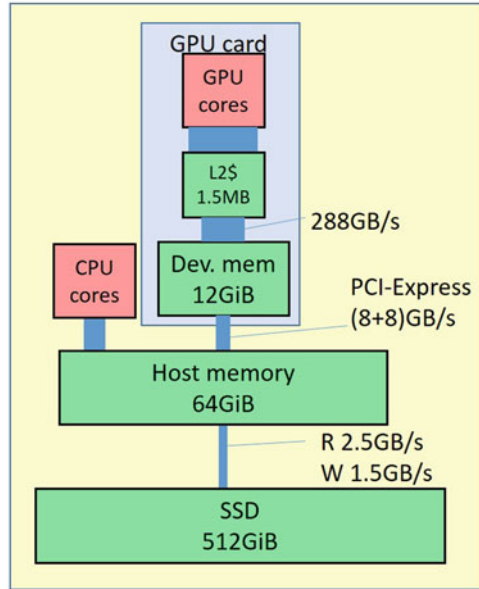
12.3 Out-of-Core Computations on Memory Hierarchy of GPGPU Clusters with a Co-design Approach (Endo Group)

Toward exascale in computational speed, many-core accelerators including GPUs and Xeon Phi processors are known as promising components, and many research projects have demonstrated their high performance. On the other hand, it is still harder to achieve “extreme problem scale” due to the memory wall problem mentioned in Sect. 12.1, especially on general purpose GPU (GPGPU) clusters. In current high-end products, while computation speed and memory bandwidth are high (around 1–2 TFlops in double precision and 200–400 GB/s per accelerator), memory capacity per accelerator is limited to 6–16 GiB. Owing to the advantage in performance of GPUs, many scientific applications have been executed successfully on GPGPU clusters; however, the problem sizes have been limited by capacity [2, 21, 22, 30].

In order to realize extremely fast and large-scale applications, we need properly designed approaches to harness deeper memory hierarchy. An example of architecture of a GPGPU computing node is shown in Fig. 12.11. If we harness both of high performance of upper memory layer and large capacity of lower layer, fast and large-scale simulations could be realized. Toward this direction, this section describes a co-design approach of application programs with locality improvement techniques and underlying runtime library to perform data swapping between memory layers.

This section assumes that the target applications are designed for GPGPU clusters and written in MPI and CUDA. Those applications are executed on top of a runtime library called *hybrid hierarchical runtime (HHRT)* in order to enable larger problem scales that surpass the GPU device memory capacity [3, 4, 6]. By using proper system architecture and a runtime library, scale expansion is achieved; however, speed performance is still insufficient “as is,” due to overhead of data movement among memory hierarchy. This cost is largely mitigated by locality

Fig. 12.11 Memory hierarchy of a GPGPU machine from the viewpoint of GPU cores. Here an SSD with bandwidth of >1 GB/s is equipped



improvement in the application algorithm layer. In stencil computation, temporal blocking [20, 32], also used in Sect. 12.2, is applied.

12.3.1 HHRT

The objective of hybrid hierarchical runtime (HHRT) library is *to extend applications' supportable problem scales*.¹ The main targets of HHRT are applications whose problem scales have been limited by the capacity of upper memory layer, such as GPU device memory in GPU clusters. For instance, such applications include simulation software based on stencil computations written for GPUs. They have been enjoyed high computing speed and memory bandwidth of GPUs. However, most of those applications are designed as “in core,” and supportable problem sizes are determined by device memory capacity as shown in Fig. 12.12 (A), in spite of the existence of larger memory (storage) layers, including host memory and file systems. While the problem sizes are expanded by using multiply GPUs and compute nodes, they are still limited by the aggregated amount of used device memory capacity.

¹Available at <https://github.com/toshioendo/hhrt>

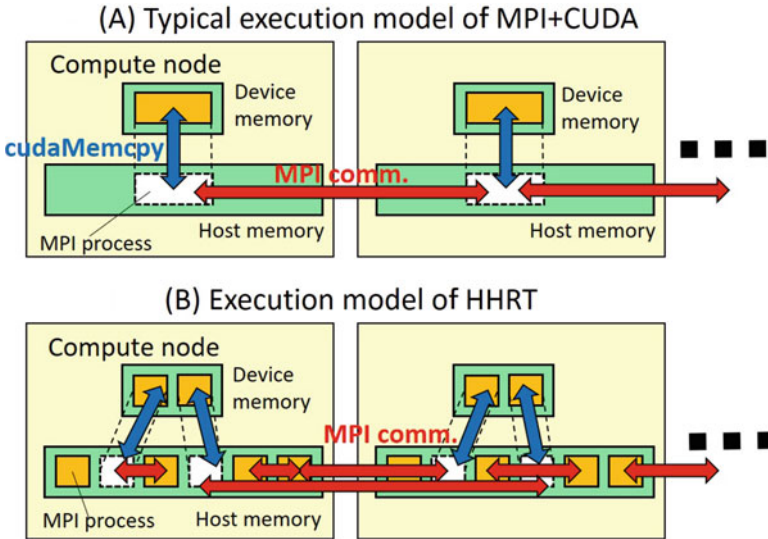


Fig. 12.12 Execution model on typical MPI/CUDA and execution model on HHRT library

The problem scales of such applications are expanded by executing them with only slight modifications on top of HHRT library. Basically we assume that the target applications of HHRT have the following characteristics:

- The applications consist of multiple processes working cooperatively.
- Data structure that is frequently accessed by each process is put on upper memory layer.

Many stencil applications on GPU clusters described above have already these characteristics, since they are written in MPI to support multiple nodes, and regions to be simulated are distributed among processes so that each process has smaller local region than device memory capacity.

On the execution model of HHRT, each GPU is shared by multiple MPI processes as illustrated in Fig. 12.12 (B), unlike in (A) (this figure shows only two-layer hierarchy; however, if flash SSDs are available, they are visible as the third layer).

When users execute their application on top of HHRT, they would typically adjust the number of MPI processes so that *the data size per each process is smaller than the capacity of device memory*. Hereafter P_s denotes the number of processes sharing a single GPU, which is 6 in the figure. By invoking plenty number of processes per GPU, we can support larger problem sizes than device memory in total.

This *oversubscribing* model itself, however, does not support larger problem sizes. We cannot hold all the data of P_s processes on the device memory at once, when P_s is large enough. Instead, we execute swapping out of memory regions of some processes from the device memory (process-wise swapping).

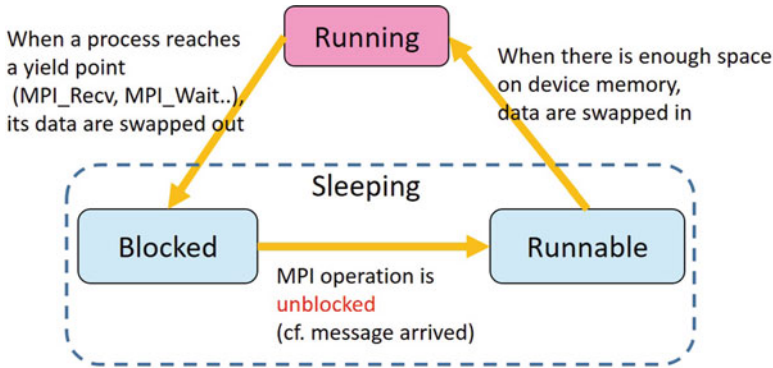


Fig. 12.13 State transition of each process on HHRT

On HHRT, swapping is tightly coupled with process scheduling. Swapping out may occur at *yield points*, where process may start sleeping, instead of individual memory accesses. In current library, based on CUDA and MPI, yield points correspond to blocking operations of MPI, such as `MPI_Recv`, `MPI_Wait`, and so on.

Figure 12.13 illustrates state transition of each MPI process. Each process is in one of states, “running,” “blocked,” or “runnable.”²

When a running process p reaches a yield point, it starts swapping out contents of all the regions that the process holds on the device memory into some dedicated place (called swap buffer hereafter) on the lower memory layer, such as host memory or flash SSDs. Then the process releases the capacity of device memory so that it can be reused by other processes, and the process starts sleeping. While the MPI operation that have let the process p start to sleep is still blocked, the process p remains in “blocked” state. Even when the operation is unblocked (e.g., a message has arrived), the process p may not start running immediately if the capacity of device memory is insufficient; here p is in the “runnable” state. Afterward, when the size of free space in device memory becomes sufficient, the sleeping process p can start swapping in; it allocates the heap region again on device memory, copying user data from swap buffer to the heap on device memory. Then p can exit from the yield point, which is an MPI blocked operation function. Now p is in the “running” state.

With this swapping mechanism, P_s processes share the limited capacity of device memory in a transparent fashion from application programs.

Generally, we can obtain better performance if more than one process out of P_s processes can be in “running” state, since such a situation enables overlapping of swapping processes and running processes. This can be done by configuring the data

²In the actual implementation, there are two transient states, “swapping-in” and “swapping-out.”

size of each process to be less than half of device memory capacity. Figure 12.12 shows such a situation where two processes are running and other four processes are sleeping.

12.3.2 Performance Evaluation

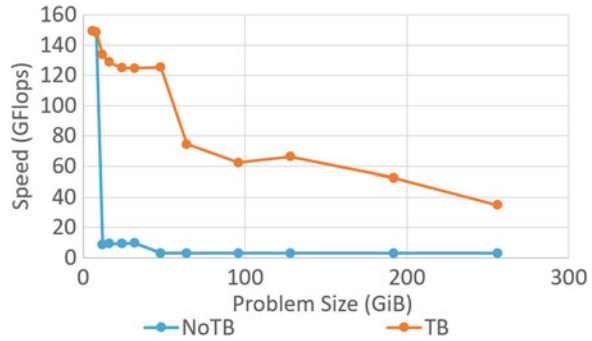
We evaluate performance of out-of-core execution by executing a “seven-point” stencil benchmark on top of HHRT library. The benchmark is written with CUDA and MPI, and *temporal blocking* technique is implemented in order to improve locality. The temporal blocking size has been already tuned through preliminary evaluations.

The evaluation has been conducted on two GPGPU platforms shown in Table 12.1. One is a PC server equipped both with a NVIDIA K40 GPU and a high-speed m.2 flash SSD, which is used for a single GPU evaluation. For multiple GPU/node evaluation, we use another platform, a 40-node GPGPU cluster named *TSUBAME-KFC/DL* [5]. Each node has four NVIDIA K80 gemini boards; thus eight GPUs are available. In this paper, a single GPU per node is used to evaluate the effect of memory hierarchy. Note that the application performance is significantly affected by SSD access performance in “out-of-core” cases. Each *TSUBAME-KFC/DL* node is equipped with two SATA SSDs, whose bandwidth is 0.4–0.5GB/s and severely lower than the m.2 SSD. To alleviate this lower performance, we use two SSDs in parallel.

Table 12.1 Platforms used for evaluation

	PC server	TSUBAME-KFC/DL
# of nodes	1	40
GPU	NVIDIA Tesla K40	NVIDIA Tesla K80
SP peak perf. (GFlops)	4.29 (5.0 w/ boost)	2.8 (4.37 w/ boost)
Device memory BW (GB/s)	288	240
Device memory size (GiB)	12	12
# of GPUs/node	1	8 (4 boards)
CPU	Intel Core i7-6700K	Intel Xeon E5-2620 v2
# of CPUs/node	1	2
CPU-GPU connection	PCIe gen3 x8	PCIe gen3 x16
Host memory size (GiB)	64	64
SSD	Samsung 950PRO m.2	Intel DC S3500
Read/Write BW (GB/s)	2.5/1.5	0.50/0.41
Capacity (GB)	512	480
# of SSDs/node	1	2
Network interface	Gigabit Ethernet	4x FDR InfiniBand

Fig. 12.14 Performance evaluation with various problem sizes on a single GPU



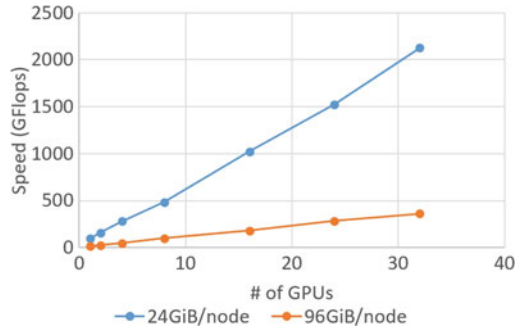
The graphs in Fig. 12.14 show the performance of the stencil program on a single GPU on the PC server. “TB” in graphs corresponds to the cases with temporal blocking, and “NoTB” does not use temporal blocking (thus $k = 1$). In TB cases, we show the fastest cases with varying k . The x-axis shows the aggregated size of stencil grids, which correspond to the problem scales.

We have successfully executed the program with problem sizes of up to 256 GiB, which is 4 times larger than host memory capacity and 20 times larger than device memory. This is owing to HHRT’s facility for oversubscription and swapping. While NoTB is impractically slow with 12 GiB problem sizes or larger for too heavy swapping cost, the situation is significantly better on TB for better locality. In the TB case, the speed performance is around 80–90% with 12–48 GiB problems, compared with in-core execution. In these cases, host memory layer is used for swapping out. With even larger sizes, the overhead becomes larger for swapping data to flash SSDs; the speed is 30–50%. This situation shall be alleviated in future improvement of HHRT, flash devices, and temporal blocking algorithm; however, we can say it is realistic to achieve extremely large problem sizes in scientific applications by harnessing deeper memory hierarchy.

Figure 12.15 demonstrates weak scalability of multiple GPU cases on TSUBAME-KFC cluster. The graph shows the case of 24 GiB problem size per node and 96 GiB per node. Both cases use larger problem sizes than GPU device memory capacity; the former case uses host memory as swapping devices, while the later uses flash SATA SSDs.

We observe the scalability is pretty good in both cases. Compared with a single GPU performance (97.6 GFlops in 24 GiB case and 15.2 GFlops in 96 GiB case), we got 21.8 times and 23.9 times speedup on 32 GPUs. In the latter case, we have successfully executed the problem of $96 \times 32 = 3072$ GiB size by using limited computing resources. Currently, the resultant speed 363 GFlops is not so high due to the insufficient speed of SATA SSDs. If each node were equipped with fast m.2 SSDs as the PC server we used, we could obtain around 1 TFlops for 3 TiB problem.

Fig. 12.15 Weak scalability evaluation on TSUBAME-KFC/DL



12.3.3 Summary of This Section

This section described a co-design approach toward extremely large-scale applications on top of GPGPU clusters. The approach consists of (1) a runtime library, HHRT, which provides data swapping mechanism among memory layers, and (2) locality improvement technique, temporal blocking for stencil computations. As a result, this section showed that it is realistic to achieve extremely large problem sizes in scientific applications with properly designed deeper memory hierarchy.

12.4 Tool Chains for Memory Locality Profiling and Performance Tuning (Y. Sato, Endo Group)

In this project, we have developed tool chains for accelerating system with deeply hierarchical memory. Starting with source code or pre-compiled executable code, these tools contribute to evolving application software into the underlying memory subsystems. The objective of these is to enable automatic/semiautomatic performance tuning and optimizations for deeply hierarchical memory and contribute to productive software performance engineering. To realize such performance tuning and optimizations, we investigate a wide range of techniques to profile, estimate, translate, and switch application code. Figure 12.16 shows an overview of our tool chains for memory locality profiling and performance tuning developed in this project. In the following, we briefly introduce key points of these tools.

To profile the actual application execution, we use executable binary code as an input and transparently analyze code based on dynamic binary instrumentation technique. Here, we build static and dynamic analysis routines for binary code on the top of Pin tool set [10]. To assist performance tuning process, we feed back the obtained profiling results to source code as key clues for the performance under the current execution environment. In this project, we focus on the ways to profile memory locality and cache-line conflicts.

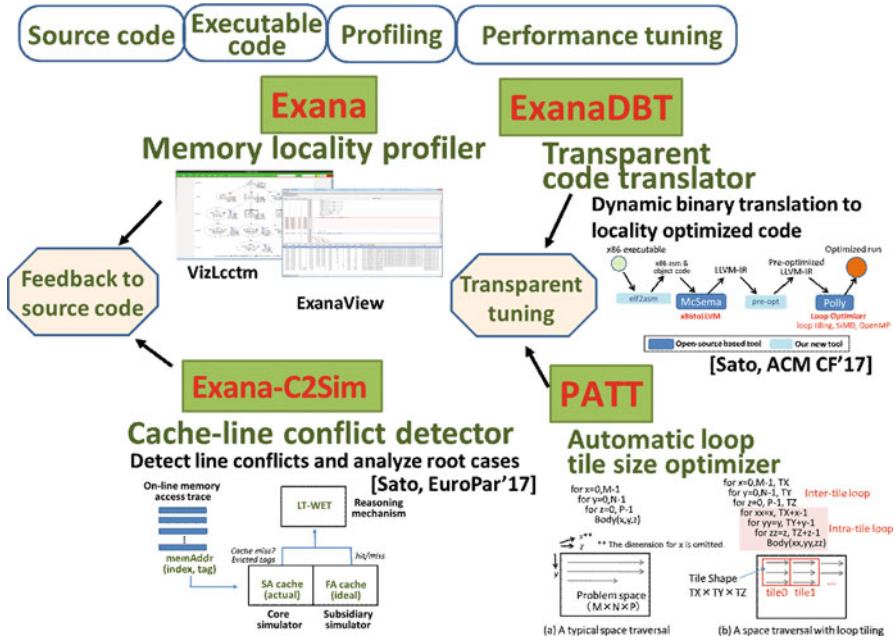


Fig. 12.16 Overview of our tool chains for memory locality profiling and performance tuning

To estimate the performance of execution in systems with deeply hierarchical memory, we build performance models that focus on memory locality. Here, we deeply explore sensitivity of loop tile size selection for performance and develop an automatic loop tile size optimizer called PATT. Using PATT, we can perform transparent tuning-based iterative compilation of code.

To translate and switch application code, we attempt to apply dynamic binary translation technique to transparent code optimization for memory locality. Here, we implement transparent code translation based on the existing open source tool chains and extend them to compensate their weaknesses for applying optimization from the executable binary code. Then, we package these tools as ExanaDBT, which realizes performance gain from transparent binary translation.

In the following subsections, we describe outlines of these tool chains we developed in this project.

12.4.1 Memory Locality Profiler

As modern memory subsystems have become complex, performance tuning of application code targeting for their deeper memory hierarchy becomes time-consuming and empirical tasks and often depends on the hands of skilled program-

mers or domain experts. These are sometimes called *Ninja gap* [23] and known to hinder productivity for software and system development cycles. To assist such burdens on performance tuning process, we have developed an application analysis tool called *Exana* and attempted to automate some parts of it [28].

Exana provides the ability to transparently analyze program structures, data dependences, memory access characteristics, and cache hit/miss statistics across program execution. For program structures, it can monitor all the dynamic control flows during the actual execution and especially focuses on dynamic loop and call nests appearing at runtime which are sometimes sensitive scenarios of input data [25, 27]. Here, we present **LCCT (loop-call context-tree)** representation, which is an extension of CCT (Call Context Tree) representation [1]. The LCCT representation, where loop nodes are added into the original CCT, can depict loop nests across multiple procedure calls effectively.

For data dependence analysis via memory, we present **LCCT+M (Loop-Call Context Tree with Memory)** representation that combines dynamic data dependencies with the LCCT representation [26]. Using LCCT+M, we can visualize data dependencies via memory reference together with dynamic program context based on regions composed of function calls, nested loops, innermost loops and bodies of loop iterations. This provides a new methodology that enables understanding and characterization of actual dynamic execution of workloads in terms of data dependencies and dynamic program contexts. Using LCCT+M representations, we can uncover various types of parallelisms such as loop-, task- and pipeline-parallelisms appeared in the actual executions.


Exana also provides ability to profile memory access characteristics and cache hit/miss statistics across program execution by feeding online memory trace to a pattern analyzer or a cache simulator directly. For memory access pattern analysis, *Exana* organizes a series of memory accesses as patterns and formulates whole patterns during the execution [11]. Here, *Exana* is seen as performing online loss-less compression of memory access behaviors corresponding to each memory instruction. *Exana* also provides the functionality for working set analysis in the granularity of loop region [28].

Exana is not a loose collection of these individual analyses, but all of analyses are packaged as an integrated tool. This enables us to activate any of analyses implemented on *Exana* at the same time in a single analysis run. This feature also contributes to providing simple interface for users to obtain application profiling results. Figure 12.17 shows an overview of how to run *Exana*. Here, all the users needed is to select the functionality for profiling and feed the command to be executed. After the execution with profiling code complete, *Exana* outputs their result files, which can be visualized using GUI-based tools. Hence, the users can intuitively find clues for performance gains and easy to feed back to their source code.

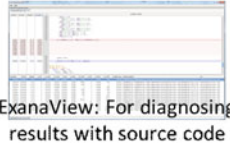
How to run Exana:

% Exana [option] -- ./a.out

	options	output file
Loop and call nests analysis [CF'11]	-mode LCCT	lcct.dat
Data dependence analysis [IISWC'12]	-mode LCCT+M	lcctm.dat
Memory access pattern analysis [WANC'14]	-mempat 1	mempat.dat
Memory hierarchy performance simulation	-cacheSim 1	cacheSim.dat
Working set analysis	-workingSetAna 1	lcct.dat



VizLcctm: Visualize loop-call nests



ExanaView: For diagnosing results with source code

Fig. 12.17 Interface of Exana for users

12.4.2 Cache-Line Conflict Detector

We have developed a cache-line conflict profiling method called **C2Sim** on the top of Exana tool [24]. The basic idea behind this is the use of cache simulators as a diagnosis tool for cache-line conflicts. Further, Exana-C2Sim pushes forward the state-of-the-art performance tuning workflow by accurately highlighting the sources of conflicts. We also propose a mechanism that enables to identify where line conflict misses are incurred and the reasons why the conflicts occur.

The background behind this paper is that performance degradation or performance variability due to line conflict misses still found occasionally in modern CPU architectures, so we strive to eliminate it based on its accurate detection technique. Modern CPUs typically have highly associative cache structures to avoid conflict misses as much as possible. One example seen in Intel Sandy Bridge CPU is that the L3 cache is organized as a 20-way associative cache. Even in the lower L1 and L2 caches, their associativity is 8-way. However, in some of applications that intensively access a particular set in the associative cache, the number of elements mapped onto the same set can easily exceed the degree of associativity [8] and cause conflict misses. Since this often impacts on performance seriously, we should avoid it by refactoring the source code.

We evaluate our conflict simulator using some of the benchmark codes used in the HPC field. From the results, we confirm that our simulator can accurately model the cache behaviors that cause line conflicts and reveal the sources of them during the execution. We also demonstrate that optimizations assisted by our mechanism contribute to improving performance for both of serial and parallel executions.

12.4.3 Automatic Loop Tile Size Optimizer

On modern CPU architectures, performance tuning against complexity within a hierarchical memory system and scalability for parallelism is inevitable for achieving their potential. As many-core processors become popular more and more in high performance computing domain, these trends are becoming much significant than ever before. Here, we focus on loop tiling that plays an important role in performance tuning process and strive to find its optimal parameters for scalable parallel executions. Then, we have developed a novel auto-tuning framework called **PATT**, which analytically models load balance and empirically auto-tunes tricky cache behaviors based on iterative polyhedral compilation using an LLVM-based polyhedral optimizer, Polly [7].

We compare our method with two of the existing general purpose heuristics: simulated annealing and Nelder-Mead [33]. From the result, we demonstrate that our method obtains good tile sizes with shorter search steps and higher accuracy than the other methods.

12.4.4 Transparent Code Optimizer

To fully automate performance tuning especially for deeper memory hierarchy, we have developed a dynamic compilation system called **ExanaDBT** on the top of dynamic code translation technique [29]. Here, ExanaDBT transparently optimizes and parallelizes binaries at runtime. To realize advanced loop-level optimizations beyond trace or instruction level, ExanaDBT uses a polyhedral optimizer [7] and performs loop transformation for rewarding sustainable performance gain on systems with deeper memory hierarchy. Especially for successful optimizations, we reveal that a simple conversion from the original binaries to LLVM IR will not be enough for representing the code in polyhedral model and then investigate a feasible way to lift binaries to the IR capable of polyhedral optimizations.

We implement a proof-of-concept design of ExanaDBT and evaluate it. Starting from hot spot detection of the execution, it dynamically estimates gains for optimization, translates the target region into highly optimized code, and switches the execution of original code to optimized one. From the evaluation results, we confirm that ExanaDBT realizes dynamic optimization in a fully automated fashion. The results also show that ExanaDBT can contribute to speeding up the execution in average 3.2 times from unoptimized serial code in a single-thread execution and 11.9 times in a 16-thread parallel execution.

12.4.5 *Contribution to Open-Source Community*

We have released Exana and C2Sim to the public, and these are available at GitHub.³ We encourage researchers and developers to download it as a basis for productive performance tuning.

12.5 Conclusion

This chapter summarized our efforts to efficient usage of deeper memory hierarchy in post-petascale era. The topics span system architecture, system software, and application algorithm areas; we demonstrated extremely high performance and large-scale applications are feasible by combining the proposed technologies. The knowledge obtained in this project has affected the system architecture design of TSUBAME3.0 supercomputer introduced at Tokyo Institute of Technology in 2017; each compute node is equipped with NVMe flash SSD of 2 TB capacity and Read 2.7 GB/s and Write 1.8 GB/s bandwidth, which are expected to be utilized for extremely high-speed and large computations. In the future, we will pursue this co-design approach to harness deeper memory hierarchy with new types of nonvolatile memory devices those appear in next-generation large-scale computer systems.

References

1. Ammons, G., Ball, T., Larus, J.R.: Exploiting hardware performance counters with flow and context sensitive profiling. In: Proceedings of the ACM SIGPLAN 1997 conference on programming language design and implementation, pp. 85–96 (1997)
2. Bernaschi, M., Bisson, M., Endo, T., Fatica, M., Matsuoka, S., Melchionna, S., Succi, S.: Petaflop biofluidics simulations on a two million-core system. In: IEEE/ACM SC'11, 12p. (2011)
3. Endo, T.: Realizing out-of-core stencil computations using multi-Tier memory hierarchy on GPGPU clusters. In: IEEE Cluster Computing (CLUSTER2016), pp. 21–29 (2016)
4. Endo, T., Jin, G.: Software technologies coping with memory hierarchy of GPGPU clusters for stencil computations. In: IEEE Cluster Computing (CLUSTER2014), pp. 132–139 (2014)
5. Endo, T., Nukada, A., Matsuoka, S.: TSUBAME-KFC: a modern liquid submersion cooling prototype towards exascale becoming the greenest supercomputer in the world. In: IEEE International Conference on Parallel and Distributed Systems (ICPADS 2014), pp. 360–367 (2014)
6. Endo, T., Takasaki, Y., Matsuoka, S.: Realizing extremely large-scale stencil applications on GPU supercomputers. In: IEEE International Conference on Parallel and Distributed Systems (ICPADS 2015), pp. 625–632 (2015)
7. Grosser, T., Groesslinger, A., Lengauer, C.: Polly – performing polyhedral optimizations on a low-level intermediate representation. *Parallel Process. Lett.* **22**(04), 1–28 (2012)

³<https://github.com/YukinoriSato/ExanaPkg>

8. Hong, C., et al.: Effective padding of multidimensional arrays to avoid cache conflict misses. In: Proceedings of the 37th ACM Conference on Programming Language Design and Implementation, PLDI '16, pp. 129–144 (2016)
9. Lucas, R., et al.: Top ten exascale research challenges, DOE ASCAC Subcommittee Report (2014)
10. Luk, C.K., et al.: Pin: building customized program analysis tools with dynamic instrumentation. In: Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 190–200 (2005)
11. Matsubara, Y., Sato, Y.: Online memory access pattern analysis on an application profiling tool. In: International Workshop on Advances in Networking and Computing, 2014 (WANC2014), pp. 602–604 (2014)
12. Matsuoka, S., Endo, T., Nukada, A., Miura, S., Nomura, A., Sato, H., Jitsumoto, H., Sandr Drozd, A.: Overview of TSUBAME3.0, green cloud supercomputer for convergence of HPC, AI and big-data, GSIC, Tokyo Institute of Technology. *e-Sci. J.* **16**, 2–9 (2017)
13. Midorikawa, H.: The performance analysis of portable parallel programming interface MpC for SDSM and pthread. In: Proceedings of IEEE/ACM International Symposium on Cluster Computing and the Grid CCGRID2005. Fifth International Workshop on Distributed Shared Memory (DSM2005), vol. 2, pp. 889–896 (2005). <https://doi.org/10.1109/CCGRID.2005.155865>
14. Midorikawa, H.: Blk-Tune: blocking parameter auto-tuning to minimize input-output traffic for flash-based out-of-core stencil computations. In: Proceedings of IEEE International Parallel and Distributed Processing Symposium 2016 Workshop, IPDPSW2016, pp. 1516–1526 (2016). <https://doi.org/10.1109/IPDPSW.2016.48>
15. Midorikawa, H., Tan, H.: Locality-aware stencil computations using flash SSDs as main memory extension. In: Proceedings of IEEE/ACM International Symposium on Cluster, Cloud and the Grid Computing CCGRID2015, pp. 1163–1168 (2015). <https://doi.org/10.1109/CCGrid.2015.126>
16. Midorikawa, H., Tan, H.: Evaluation of flash-based out-of-core stencil computation algorithms for SSD-equipped clusters. In: The 22nd IEEE International Conference on Parallel and Distributed Systems ICPADS2016, pp. 1031–1040 (2016). <https://doi.org/10.1109/ICPADS.2016.0137>
17. Midorikawa, H., Tan, H.: A highly efficient I/O-based out-of-core stencil algorithm with globally optimized temporal blocking. In: Proceedings of 2017 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, pp. 1–6 (2017). <https://doi.org/10.1109/PACRIM.2017.8121909>
18. Midorikawa, H., Saito, K., et al.: Using a cluster as a memory resource: a fast and large virtual memory on MPI. In: Proceedings of IEEE International Conference on Cluster Computing Cluster2009, pp. 1–10 (2009). <https://doi.org/10.1109/CLUSTER.2009.5289180>
19. Midorikawa, H., Kitagawa, K., Ohura, H.: Efficient swap protocol for remote memory paging in out-of-core multi-thread applications. In: Proceedings of 2017 IEEE International Conference on Cluster Computing Cluster2017, pp. 637–638 (2017). <https://doi.org/10.1109/CLUSTER.2017.55>
20. Nguyen, A., Satish, N., Chhugani, J., Kim, C., Dubey, P.: 3.5-D blocking optimization for stencil computations on modern CPUs and GPUs. In: IEEE/ACM SC'10, 13p. (2010)
21. Onodera, N., Aoki, T., Shimokawabe, T., Miyashita, T., Kobayashi, H.: Large-Eddy simulation of fluid-structure interaction using lattice Boltzmann method on multi-GPU clusters. In: 5th Asia Pacific Congress on Computational Mechanics and 4th International Symposium on Computational Mechanics (2013).
22. Phillips, E.H., Fatica, M.: Implementing the Himeno benchmark with CUDA on GPU clusters. In: IEEE International Symposium on Parallel and Distributed Processing (IPDPS), pp. 1–10 (2010)
23. Satish, N., Kim, C., Chhugani, J., Saito, H., Krishnaiyer, R., Smelyanskiy, M., Girkar, M., Dubey, P.: Can traditional programming bridge the ninja performance gap for parallel computing applications? *Commun. ACM* **58**(5), 77–86 (2015)

24. Sato, Y., Endo, T.: An accurate simulator of cache-line conflicts to exploit the underlying cache performance. In: Proceedings of 23rd International European Conference on Parallel and Distributed Computing (Euro-Par 2017), pp. 119–133 (2017)
25. Sato, Y., Inoguchi, Y., Nakamura, T.: On-the-fly detection of precise loop nests across procedures on a dynamic binary translation system. In: Proceedings of the 8th ACM International Conference on Computing Frontiers, pp. 25:0–25:10 (2011)
26. Sato, Y., Inoguchi, Y., Nakamura, T.: Whole program data dependence profiling to unveil parallel regions in the dynamic execution. In: Proceedings of 2012 IEEE International Symposium on Workload Characterization (IISWC2012), pp. 69–80 (2012)
27. Sato, Y., Inoguchi, Y., Nakamura, T.: Identifying program loop nesting structures during execution of machine code. *IEICE Trans. Inf. Syst.* **E97-D(9)**, 2371–2385 (2014)
28. Sato, Y., Sato, S., Endo, T.: Exana: an execution-driven application analysis tool for assisting productive performance tuning. In: Proceedings of the 2nd International Workshop on Software Engineering for Parallel Systems, SEPS 2015, pp. 1–10 (2015)
29. Sato, Y., Yuki, T., Endo, T.: ExanaDBT: a dynamic compilation system for transparent polyhedral optimizations at runtime. In: ACM International Conference on Computing Frontiers 2017 (CF'17), p. 10 (2017)
30. Shimokawabe, T., Aoki, T., Takaki, T., Yamanaka, A., Nukada, A., Endo, T., Maruyama, N., Matsuoka, S.: Peta-scale phase-field simulation for dendritic solidification on the TSUBAME 2.0 supercomputer. In: *IEEE/ACM SC'11*, 11p. (2011)
31. TSUBAME3.0: The super computer in Global Scientific Information and Computing Center, Tokyo Institute of Technology. <http://www.gsic.titech.ac.jp/en>. Online: 26 Mar 2018
32. Wolf, M.E., Lam, M.S.: A data locality optimizing algorithm. *ACM PLDI* **91**, 30–44 (1991)
33. Yuki, T., Sato, Y., Endo, T.: Evaluating autotuning heuristics for loop tiling. In: International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2018), p. 2 (2018)

Chapter 13

Power Management Framework for Post-petascale Supercomputers



Masaaki Kondo, Ikuo Miyoshi, Koji Inoue, and Shinobu Miwa

Abstract Power consumption is a first class design constraint for developing future exascale computing systems. To achieve exascale system performance with realistic power provisioning of 20–30 MW, we need to improve power-performance efficiency significantly compared to today’s supercomputer systems. In order to maximize effective performance within a power constraint, investigating how to optimize power resource allocation to each hardware component or each job submitted to the system is necessary. We have been conducting research and development on a software framework for code optimization and system power management for the power-constraint adaptive systems. We briefly introduce the research efforts for maximizing application performance under a given power constraint, power-aware resource manager, and power-performance simulation and analysis framework for future supercomputer systems.

13.1 Introduction

Power consumption is expected to be a first class design constraint for developing future exascale computing systems. Figure 13.1 which illustrates power consumption of the top ten fastest supercomputers in the world over a decade clearly shows

M. Kondo (✉)
The University of Tokyo, Tokyo, Japan
e-mail: kondo@hal.ipc.i.u-tokyo.ac.jp

I. Miyoshi
Fujitsu Limited, Kawasaki, Kanagawa, Japan
e-mail: miyoshi.ikuo@jp.fujitsu.com

K. Inoue
Kyushu University, Fukuoka, Japan
e-mail: inoue@ait.kyushu-u.ac.jp

S. Miwa
The University of Electro-Communications, Chofu, Tokyo, Japan
e-mail: miwa@hpc.is.uec.ac.jp

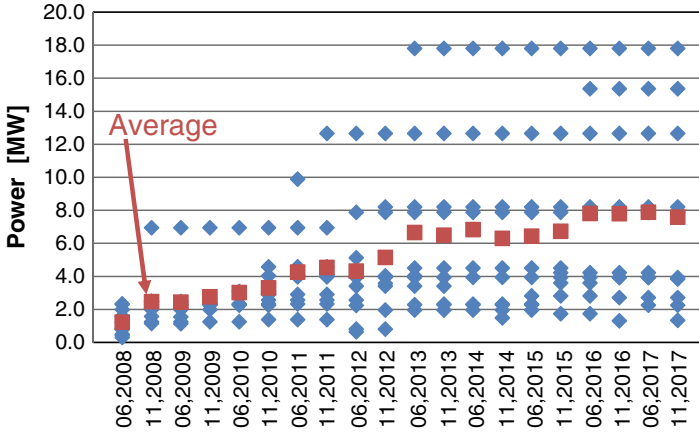


Fig. 13.1 Power consumption of the top ten fastest supercomputers in the world

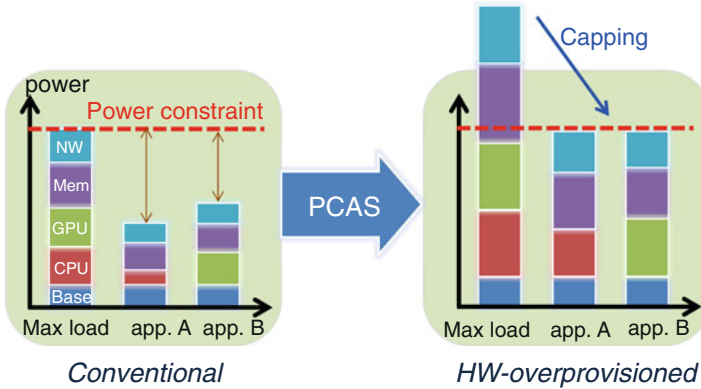


Fig. 13.2 Concept of the power-constraint adaptive system (PCAS)

that power consumption of large-scale supercomputer systems gradually increases. To achieve exascale computing performance with realistic power provisioning of 20–30 MW, significant power efficiency improvement over today’s supercomputers is necessary. In order to maximize effective performance within a power constraint, we need a paradigm shift from the worst-case design strategy to the power-constraint adaptive system (PCAS) design, which allows the system’s peak power to exceed maximum power provisioning with adaptively controlling power-knows equipped in hardware components so that effective power consumption at runtime is under the power constraint (Fig. 13.2). This concept is recently known as hardware overprovisioning.

To realize an exascale system with the PCAS concept, we have been conducting a research project called *PomPP* (*Power Management framework for Post Peta-*

scale systems) and developing a software framework for code optimization and system power management which adaptively controls power-performance knobs under a given power constraint. There are several key challenges that need to be addressed, for example, (1) framework to maximize application performance under a given power constraint, (2) power-aware job scheduling to maximize system throughput and to minimize underutilized power resources, and (3) power-performance simulation and analysis framework for exascale applications. In the following sections, we briefly discuss each of the key challenges and introduce our research efforts for them.

13.2 Power-Performance Optimization Framework

There is a strong demand for maximizing application performance under a given power constraint. In fact, power demand of each hardware component is very diverse among applications. For example, compute-intensive applications prefer allocating much power to CPUs, while memory-intensive applications prefer investing power resource in memory bandwidth. It is necessary to investigate how to optimize power resource allocation to each hardware component such as CPUs, memory subsystems, and network subsystems within a job execution. In this section, we describe our research efforts of optimizing power-performance behavior of an application execution.

13.2.1 Variation-Aware Power Allocation Among Compute Nodes

Due to the advances in VLSI manufacturing processes, modern processors suffer from increasingly large power variations, and it causes a critical issue on power-constrained large-scale computer systems. Because of this manufacturing variability, *modules* that include individual processors and associated DRAMs in current HPC systems already have *inhomogeneity* from the viewpoint of power consumption. In power-limited systems with hardware power cap management, this power variation turns into CPU frequency variation, causing performance inhomogeneity in computing nodes and degrading the effective system performance on the large-scale parallel computing. This section discusses the impact of manufacturing variability in power-constrained supercomputing and introduces a power assignment technique to mitigate the negative effects of the inhomogeneity. The detail of this section has been reported in [5].

Table 13.1 Module power and performance variation (*DGEMM)

Module-level power constraint (C_m)	No power cap	110 W	100 W	90 W	80 W	70 W
CPU power cap (C_{cpu})	Non	97.4 W	88.1 W	78.8 W	69.5 W	60.1 W
Worst-case CPU frequency variation (V_f)	1.00	1.20	1.32	1.35	1.42	1.40
Worst-case power variation (V_p)	1.30	1.16	1.14	1.16	1.18	1.21
Worst-case execution time variation (V_t)	1.00	1.31	1.27	1.28	1.40	1.64

13.2.1.1 Impact of Variation on Power-Constrained HPC Systems

Table 13.1 shows power-performance variation for *DGEMM benchmark program that is a compute-bound, embarrassingly parallel matrix multiplication subroutine from the BLAS library and is also the main kernel for the High Performance Linpack (HPL) benchmark. We used a thread-parallelized version of this code in the Intel Math Kernel Library with a matrix size of $12,288 \times 12,288$. We measured the power and performance on a supercomputer called HA8000, which is a large-scale production system at Kyushu University, with 960 computing nodes, each of which includes two Intel E5-2697v2 Ivy Bridge processors and 256 GB memory. The total number of modules is 1,920. We study the impact of module-level power variation on application performance in power-constrained scenarios with the help of RAPL power measurements and caps. RAPL (Running Average Power Limit) [6] is the interface to monitor and limit power consumption of the CPU and DRAM for Intel CPUs. Table 13.1 presents results of the 1,920-module experiments with and without power capping.

Although the specification of RAPL covers DRAM power capping, we restrict power capping to the CPU domain. V_p in the table is calculated by dividing the maximum power value with the minimum power value in the appropriate module set. Without power constraints, it is observed that V_p values at the module level are about 1.3, i.e., there is a 30% difference in power consumption across modules even when they are running identical codes. We see that variation in power (when no power capping is enforced) translates to variation in CPU frequency under a power cap, e.g., applying the 70 W power constraint increases V_f , which is the worst-case variation in CPU frequency across the modules, from 1.00 to 1.40, indicating a 40% difference in CPU frequencies across modules. Eventually, the frequency variation causes the worst-case execution time variation across all MPI ranks represented as V_t . For *DGEMM, power capping results in up to 64% variation in per-rank performance ($C_m = 70$ W), resulting in poor application performance. Note that this scenario makes a perfectly load-balanced application exhibit load imbalance under a power constraint. This is a serious issue for HPC applications and future power-constrained systems.

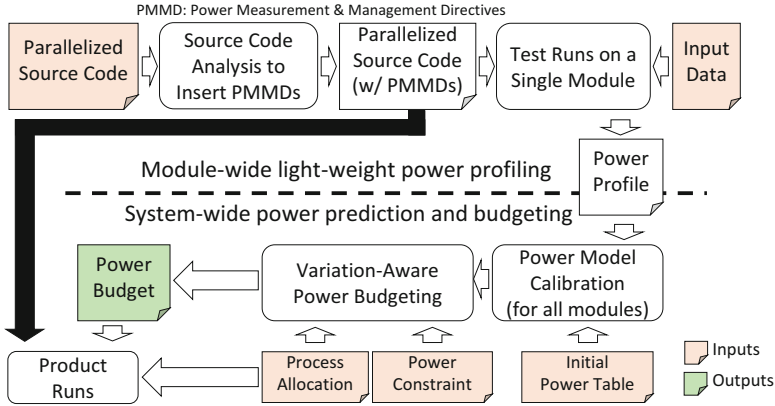


Fig. 13.3 Framework of variation-aware power budgeting

13.2.1.2 Variation-Aware Power Budgeting

To mitigate the negative effects of the manufacturing variability, we have proposed a variation-aware power budgeting scheme that attempts to detect and balance power to even out such variations. Figure 13.3 depicts the framework of our variation-aware power budgeting. This framework requires to input an HPC application, its associated data, an application-level power constraint, a list of modules (i.e., physical processors), and a Power Variation Table (PVT) that serves as the basis to estimate power variations in any target application. The PVT is constructed once per system, e.g., when the system is installed by using some application-independent standard benchmarks. The framework works as follows:

1. Inserting power directives: Power Measurement and Management Directives (PMMDs) are inserted in the target HPC application for profiling and management. We use the compiler-based instrumentation system from the TAU toolkit and define the region of interest by inserting PMMDs.
2. Lightweight single-module test run: Two low-cost, single-module test runs of the application are performed, one at the maximum CPU frequency and the other at the minimum CPU frequency. CPU and DRAM power are measured.
3. Power model calibration: The information from the single-module test runs is used to obtain power model of all modules by using the pre-computed system-level PVT. Then, an application-dependent variation-aware Power Model Table (PMT) is created. The detail of system-wide power estimation based on PVT/PMT is explained in Sect. 13.4.
4. Power budgeting algorithm: We use the application PMT and the given module list to determine the module-level power allocations to maximize the application performance under the specified application-level power constraint. This includes determining module-level CPU frequencies and deciding a power constraint for each module in order to realize that frequency.

- Product run: The HPC application annotated with the PMMDs is executed on the given module list by using the module-level power allocations determined by the variation-aware power budgeting algorithm. Two implementation strategies, power capping (PC) with RAPL and frequency selection (FS) with *cpufrequtils*, are supported.

13.2.1.3 Evaluation Results

In this evaluation, we consider HPC application executions on HA8000 with 1,920 modules under different global power constraints without any other interference. We used several HPC benchmarks such as **DGEMM*, **STREAM* that is used to measure sustainable memory bandwidth and executes simple vector operations, *NPB-BT/SP* from NAS Parallel Benchmark suite, *MHD* for magneto-hydrodynamics simulations, and *mVMC* that is a mini-application included in the FIBER benchmark suite. We compare four power budgeting schemes, an application-independent variation-unaware power budgeting scheme that distributes power uniformly across all modules (*Naïve*), an application-dependent variation-unaware scheme implemented via RAPL power capping (*Pc*), our application-dependent variation-aware scheme implemented with PC using RAPL (*VaPc*), or with frequency selection using *cpufrequtils* (*VaFs*). Figure 13.4 reports the speedup of all the aforementioned schemes when compared to *Naïve*. It is observed that *VaFs* achieves a maximum speedup of 5.40X (for the *NPB-BT* benchmark with 96 KW power constraint) and an average improvement of 1.86X across all benchmarks. For *VaPc*, which exploits the RAPL CPU power capping functionality, the maximum performance improvement is 4.03X (for *NPB-SP* at 96 KW power constraint), and the average improvement across all benchmarks is 1.72X. From these results, it is clear that variation-aware power management is a promising technique to improve the performance of power-constrained supercomputing.

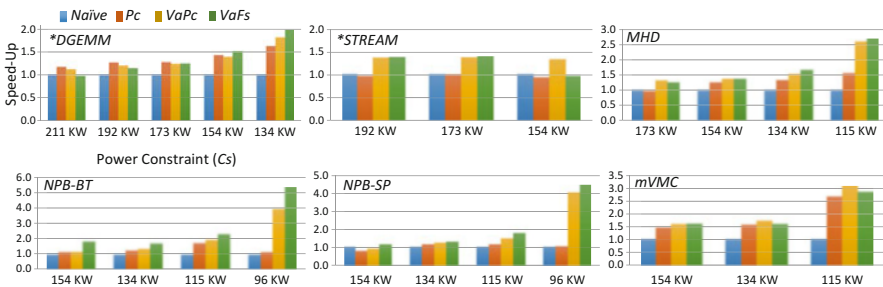


Fig. 13.4 Evaluation results

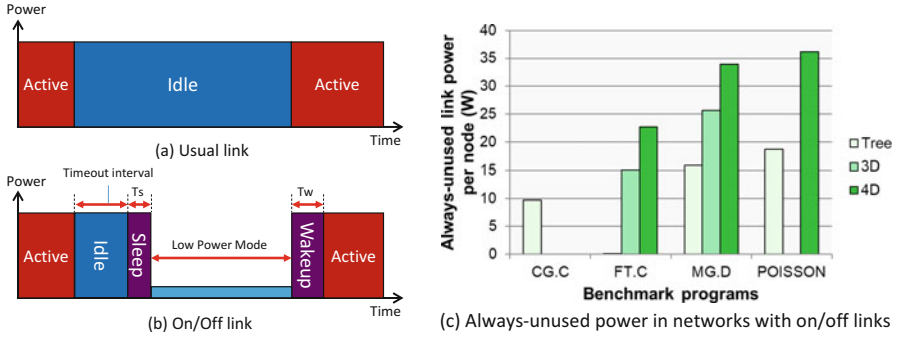


Fig. 13.5 Mechanism of on/off links and their impact on network power

13.2.2 Power Shifting in Interconnection Networks with On/Off Links

Modern supercomputing systems consume considerable amounts of power in the form of network power (up to 33% of the overall system power) [7]. For power savings of networks, many HPC engineers lately focus on the state-of-the-art technology known as on/off links due to the great capability of power saving (up to 10.8 W per link) [13] and explore its applicability in supercomputing systems [8, 13, 14, 16].

Figure 13.5 shows a power-saving mechanism of on/off links. As shown in Fig. 13.5a, usual links consume an almost constant amount of power regardless that they are used for data transfer or not (i.e., active or idle state). This is because PHYs at both ends of a network cable are always activated and frequently communicate with each other to check the link connectivity. In contrast to this, on/off links shown in Fig. 13.5b shutdown in idle state and therefore save up to 90% of link power, accompanying with the time overheads of a few microseconds (denoted as T_s and T_w) [13]. Saravanan et al. report that on/off links can save 70% of network power with the performance degradation of 2% for various HPC applications, and the power saved by on/off links accounts for 7.3% of the overall system power [13].

On/off links are very hopeful to power savings in supercomputing systems, but how should we use these technologies in a power-constrained scenario? To answer this question, we first identified the power that is always unused in networks with on/off links during the execution of applications. Figure 13.5c shows the always-unused power of various networks connected to 64 compute nodes. The figure represents that the radix of networks highly affects the always-unused power. In particular, a power of 36.1 W per node keeps unused on the 4D-torus system at runtime of POISSON, which is a handmade application that solves Poisson's equation with the Jacobi method. There exists the significant amount of power remaining even on the tree-topology system, e.g., power of 18.7 W per node (POISSON), which is comparable to the power required for boosting a 6-core CPU by 0.3 GHz.

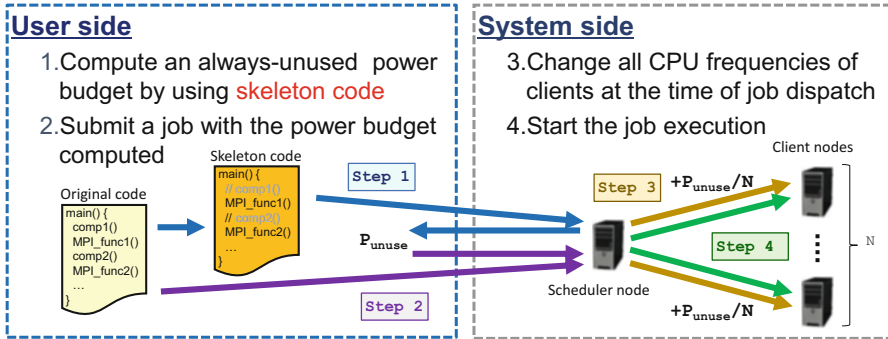


Fig. 13.6 Profile-based power shifting by using skeleton code

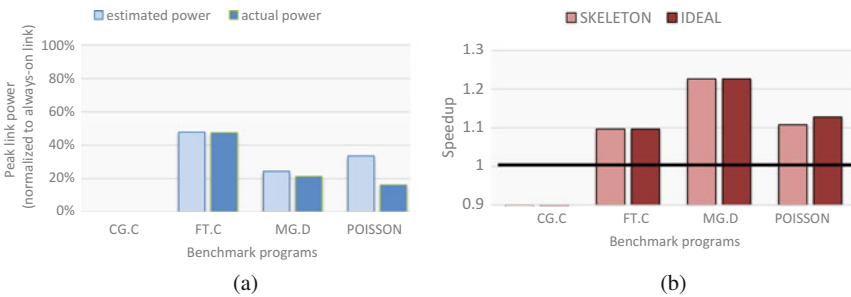


Fig. 13.7 Impact of profile-based power shifting on a 64-node 4D-torus system. (a) Power estimated with skeleton code. (b) Speedup achieved by power shifting

Based on this observation, we proposed profile-based power shifting, which enables end users to accelerate their applications by leveraging the always-unused power of networks with on/off links [9]. Figure 13.6 shows the overview of profile-based power shifting. Our profile-based power shifting first estimates the impact of a given application on power of a network with on/off links. This power profiling is done by an end user with the skeleton code of the application, which quickly reproduces the spatial and volume attributes of communication included in the original program by skipping many computations. The end user submits the above job with the estimated power P_{unuse} to a job scheduler. Our job scheduler changes the power budget of each device within a client according to P_{unuse} and then launches the job to clients.

Here we introduce the effect of profile-based power shifting, which is a part of our experimental results shown in the paper [9]. Figure 13.7a shows the accuracy of the always-unused link power estimated by using skeleton code. We conducted this experiment on a 64-node 4D-torus system emulated by the modified SimGrid-3.11 [3]. The power shown in the figure is normalized to the power of usual links. The figure shows that power profiling with skeleton codes has significant accuracy in estimating always-unused link power (an error of up to 17.3% of usual link power).

Note that skeleton codes somewhat overestimate always-unused power of network links, because skipping computations causes higher link activities than original codes.

Figure 13.7b shows the speedup achieved by two power-shifting schemes. SKELETON represents power shifting based on the always-unused power estimated by using skeleton codes, while IDEAL represents power shifting based on the always-unused power measured by using original codes. Exploiting the large amount of always-unused power on network links, SKELETON can achieve the significant speedup (up to $1.23\times$), which is close to IDEAL. Our technique has the good capability of detecting the power budget remaining on networks with on/off links.

A few challenges to implementing profile-based power shifting on production systems still remain, but we believe that our profile-based power shifting will help end users get performance gains from power-constrained supercomputing systems with on/off links.

13.2.3 A Runtime Performance-per-Watt Optimization Methodology

In the use of hardware-overprovisioned systems, since job throughput is restricted by power limit, “performance-per-watt” becomes a more important indicator of efficient system use. Because application developers are not expected to pay much attention to performance-per-watt, power optimizations should be done by system providers and system administrators. Although optimization techniques for power and energy, such as dynamic frequency scaling (DFS) and dynamic concurrency throttling (DCT), have been studied well, practical use is limited to a DB-based approach, which records and reuses the relationship between a job’s execution time and its locked CPU frequency. Conducting DFS/DCT optimization during the execution of a job is simple to use, adaptive to the environment of executing systems, and possibly robust in terms of a CPU’s manufacturing variability. Therefore we developed a runtime library which conducts power optimization during the execution of a job.

In order to design the algorithm of runtime power optimization, we made the following assumptions:

- Target application does iterative computation, such as time integration and iterative solver
- The application can be divided into several regions based on computational characteristics, such as ComputeMG and ComputeSPMV in HPCG benchmark
- From the application users’ viewpoint, performance fluctuation by up to 10% is ignored or acceptable

The optimization target is a DFS/DCT configuration of maximal performance-per-watt for each region in the range of acceptable performance degradation. Optimization proceeds by the following steps:

1. For use as reference performance, observe the performance during initial iteration(s)
2. Under the “Turbo” frequency, search the number of threads in decreasing order for the best efficiency
3. Under the base frequency, search the number of threads in the same way as Step 2
4. With the number of threads chosen by Steps 2 and 3 as the best efficiency in the range of acceptable performance degradation, search CPU frequency in decreasing order for the best efficiency
5. When performance degradation exceeds the acceptable range, continue the search after adding one more thread

The optimization procedure works based on the “efficiency” value, as described above. Since we are assuming that the amount of computation in each region is constant (for simplicity’s sake), relative performance-per-watt, or efficiency, equals the inverse of relative energy consumption. RAPL on Intel Xeon processors can measure the energy consumption; however, the measurement interval is 1ms and affects the accuracy of the efficiency value for short regions. Therefore we adopt “estimated energy” for the calculation of efficiency. Those calculations are defined as follows:

- When measuring by RAPL, efficiency = “measured energy for the reference performance”/“measured energy for a configuration”
- When estimating without RAPL, measured energy is replaced with “estimated energy”; estimated energy = “CPU cycles” \times (1 + (“the number of threads” – 1) \times “correction factor”)

The “correction factor,” which is 0.13 for Xeon E5-2680, 0.06 for Xeon E5-2698v3, and 0.065 for Xeon Platinum 8168, was derived from the results of STREAM benchmark by regression analysis.

Figure 13.8 shows an example of optimization behavior for ComputeSPMV region of HPCG benchmark on Xeon E5-2698v3. Although ranges of relative efficiency by RAPL measurement and estimation without RAPL are different, shapes of those behaviors are similar, and the same DFS/DCT configuration is chosen.

Figure 13.9 shows optimization results for HPCG benchmark by the algorithm described above. For all of three generations of Xeon processors, “performance-per-watt” is improved, and the results by RAPL measurement and estimation without RAPL are equivalent. On the latest Xeon processor, 45% improvement is achieved.

Figure 13.10 shows the difference of power consumption with and without the optimization. Target program is NICAM-DC-MINI which is a proxy application of climate models for Post-K development. As the result of optimization, average power consumption is reduced by 28% in exchange for 13% slowdown of its execution. It indicates the possibility to increase headroom for power shifting between jobs.

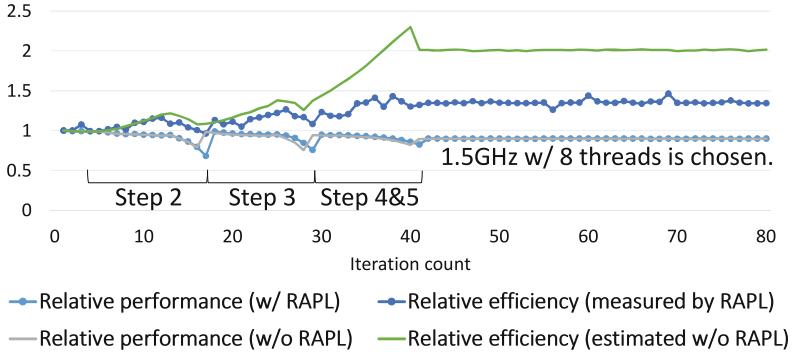


Fig. 13.8 Optimization behavior for ComputeSPMV region of HPCG on Xeon E5-2698v3 (2.3 GHz, 16 cores)

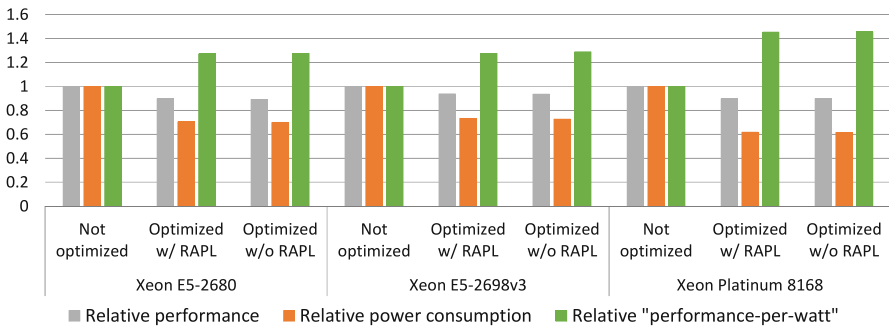


Fig. 13.9 Results of the runtime optimization for HPCG

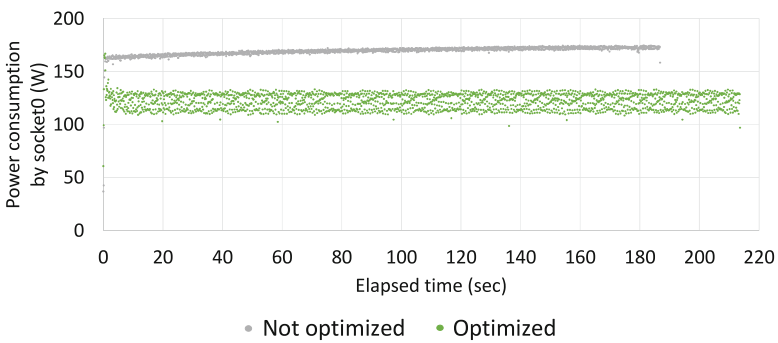


Fig. 13.10 Difference of power consumption with and without the optimization

As a result of evaluation with HPCG and other benchmark programs, effectiveness and wide applicability of the “on-the-fly” optimization were validated. As for the applicability, the optimization algorithm is adapted to three generations of Intel Xeon processors by only one parameter, as well as showing the improvement of performance-per-watt in many cases.

13.2.4 Automating Workflow of Power-Performance Optimization

To optimize power allocation among different hardware components by application developers, we need a framework which assists collecting power-performance information with various power settings, optimizing power allocation with power-performance knobs, and modifying the application code to control power-performance knobs appropriately. We have been developing a versatile power management framework which applies power-performance optimization to the application automatically. The main objective of the framework is to make the power management and power-performance optimizations processes more facilitating and flexible for both users and system administrators.

For power-performance efficiency optimization and power management for HPC applications, we have to take care of many things including (1) what kinds of hardware components the system has and how much power is consumed in them, (2) what kinds of power-knobs are available and how to control them, (3) how the applications behave at runtime, and (4) what is the relationship between performance and power consumption of the application. Based on these information, (5) we have to design a power-performance optimization algorithm and finally (6) assemble and utilize existing tool sets for collecting the necessary information and actually controlling power-knobs. Our framework is designed to provide or to support the following functionalities which help users and administrators carry out power management/control effectively without taking care of the abovementioned issues:

- Analyzing source code and applying automatic instrumentation
- Measuring and controlling application power consumption and performance
- Optimizing an application under given power budget
- Specifying and defining the target machine specification
- Calibrating hardware power consumption of the system

Figure 13.11 presents the outline of the framework. One of the benefits of using this framework is that workflow of power-performance optimization and control can be specified in higher abstraction level. Details of how to use libraries for controlling power-knobs, how to profile and analyze the application code, and how to instrument power management pragmas in the code or the job submission script are hidden from users. Moreover, the framework provides high modularity and flexibility so that libraries or tool set used in the framework and power optimization algorithms are customizable.

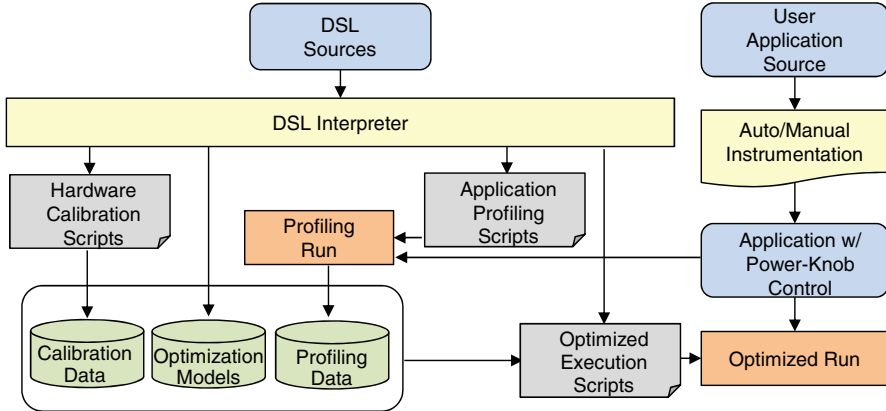


Fig. 13.11 Overview of the power-performance optimization framework

In order to provide customizability and flexibility, a simple DSL is used as the front-end for supported tools and for selecting the power optimization algorithm. In current version of the framework, we support RAPL/cpufreq utils for accessing power-knobs and TAU/PDT [10, 15] for code instrumentation and profiling. However, these are extensible and other tool sets are easily supported. More details of this framework are explained in [17], and the source code of it is available on our GitHub repository [4].

13.3 Power-Constraint-Aware Resource Management

In overprovisioned systems, power budget allocated for each node should be controlled by power-knobs such as dynamic voltage and frequency scaling (DVFS) or a power capping mechanisms. One of the key challenges is to develop a runtime system for optimizing and controlling power budget of executing jobs based on the available power budget of the entire system to maximize the total system throughput. To this end, a power-aware resource manager and several power allocation or job scheduling algorithms are necessary.

13.3.1 Development of Power-Constraint-Aware Resource Manager

Though there have been several studies of hardware overprovisioned HPC systems, the impact of hardware overprovisioning in production environments at large scale has not yet been examined intensively. In order to provide a software stack which

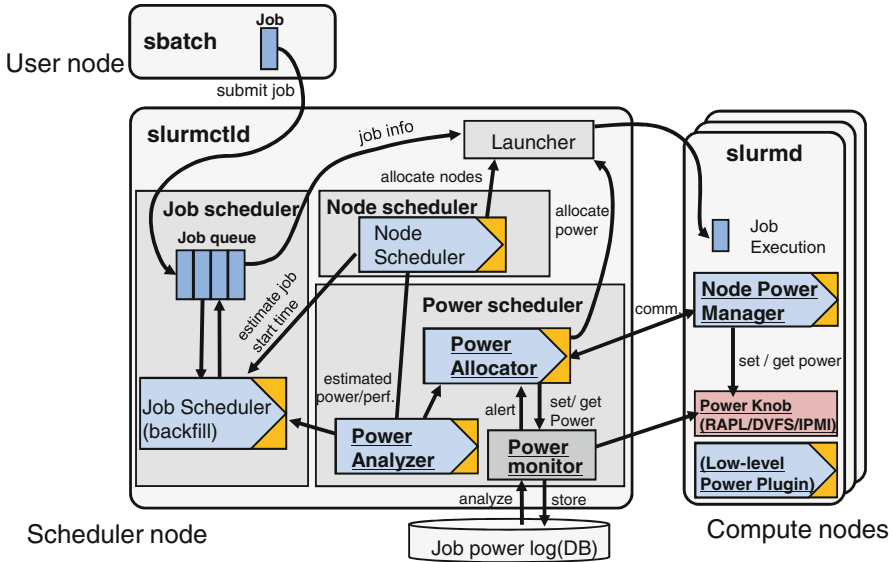


Fig. 13.12 Design of power-aware framework in SLURM resource manager

makes such systems accessible to users or administrators, we have been developing an extensible power-aware resource management and job scheduling framework based on the widely used SLURM resource manager [18].

Figure 13.12 shows the design of our proposed power-aware resource manager, which is implemented as a SLURM extension. It enables the implementation of a wide spectrum of power management algorithms by flexible plugin interfaces that can be used by different HPC centers to enforce site-specific power management policies. We add a *power scheduler*, a *node power manager*, and a *low-level power plugin interface* to the original SLURM code. The power scheduler schedules all of the system power by monitoring and distributing the compute node power. It consists of three components: power monitor, power analyzer, and power allocator. The power allocator and power analyzer have extensible plugin interfaces. This allows other plugin developers to implement and test their own power management algorithms. More details of this power-aware resource manager are explained in [11].

We validated the power management functionality of the extended SLURM using all the nodes in the HA8000 system. The system contains 965 compute nodes, and each node has two Xeon processors with 128-GB DDR3 memory. We evaluated the power consumption of the system varying the total power budget. We evaluate three power budgets, 100, 110, and 120 kW. Figure 13.13 shows the aggregate power consumption measured by RAPL for all the compute nodes. As shown in the figure, the actual power is almost the same as the allocated power budget when the system is fully loaded (middle section of each curve). This indicates that our power-aware SLURM successfully controls power usage of executing jobs.

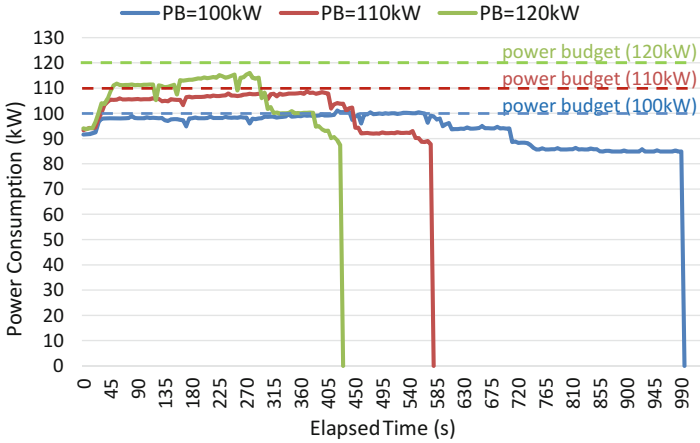


Fig. 13.13 Verification of system power cap capability

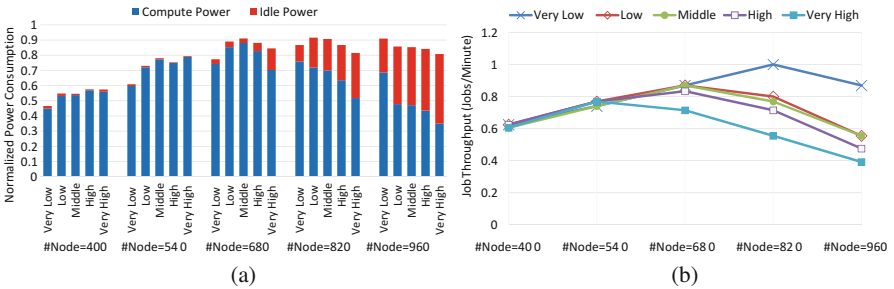


Fig. 13.14 Evaluation of overprovisioning in a real 960-node system. (a) Power resource usage. (b) Total system performance

With the extension of SLURM, we investigated the impact of varying the number of extra compute nodes or the degree of overprovisioning. We assume that the base system is a non-overprovisioned HPC system with 400 compute nodes whose system power budget is the peak power consumption of 400 nodes of the HA8000 system. Then, we evaluate several cases of overprovisioned systems: 540, 680, 820, and 960 compute nodes with the same system power budget.

Figure 13.14a, b present average power resource utilization and system throughput, respectively, under several overprovisioned system configurations. We evaluated five job mixes varying average power demand of jobs. From Fig. 13.14a, it can be observed that power resource utilization increases if we increase the number of overprovisioned nodes to 680 nodes. However, utilization decreases when the node count increases any further. The node resource is exhausted in cases of small numbers of overprovisioned nodes and power resource is not fully consumed. On the other hand, the power resource is exhausted in cases of large numbers of overprovisioned nodes, and the scheduler must leave some of the nodes being idle to maintain the power budget.

In Fig. 13.14b, system throughput tends to improve if we increase the number of nodes since overprovisioning provides more computational resources. However, the performance trends are more complex because utilization decreases in the case of 960-node and depends on job mixes. Power-hungry job mixes with many CPU-intensive jobs prefer a relatively small number of overprovisioned nodes because the average node power consumption is high and power resource is easily exhausted by small number of nodes. On the other hand, low-power job mixes with more memory-intensive jobs prefer relatively large numbers of overprovisioned nodes as the power resource does not tend to be exhausted and a larger number of nodes helps increase the number of jobs executing concurrently.

Through experimentation on a large-scale production system, we have shown that our power management framework is safe for deployment on production systems. The experiment for overprovisioning presented above could not be possible without the production level power-aware resource management framework. Our developed SLURM extension is very useful in this point. The source code of the SLURM extension is available on a GitHub repository [4].

13.3.2 Strategies for Power-Aware Resource Management

Besides a software stack for power-aware resource management, strategies or algorithms for power allocation and job scheduling need to be developed. For the PCAS design to be effective, we have also investigated several power allocation and job scheduling strategies, for example, the strategies of job demand-aware power management [1], cooling-aware job scheduling/node allocation [2], and node active state control [12]. In this subsection, we briefly introduce the job demand-aware power management strategy.

An easy way to allocate power for each job is to set power cap statically so that the total power consumption is within the power constraint of the system. This static approach is not optimal since runtime power of a job is not constant for its entire execution. Moreover, the QoS demand, that is, the level of allowable performance degradation, is not identical for all the jobs. Guaranteeing performance is indispensable for high-QoS jobs, whereas a part of power resource allocated for low QoS jobs can be distrained and used for other jobs to execute, so that total system performance increases. The proposed demand-aware adaptive power capping scheme dynamically controls power budget of running jobs according to their power demand.

The demand-aware adaptive power capping scheme (hereinafter referred to as adaptive power capping) determines the power cap value of each CPU adaptively by a power manager for every presumed time interval. We assume that jobs are submitted to the system with their performance requirements or required QoS levels, which is defined by the maximum acceptable performance loss. Periodically, the power manager receives power usage information from all of the compute nodes, predicts the current performance levels of all executing jobs, and then optimizes

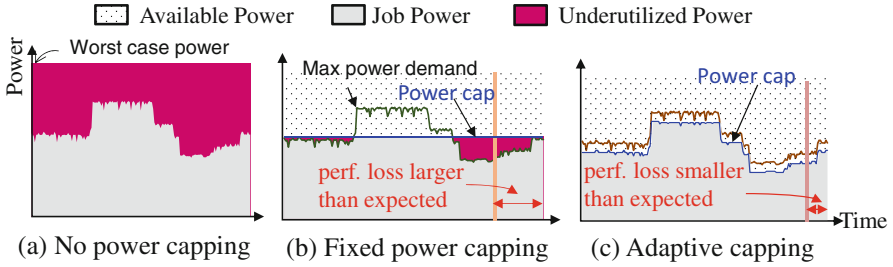


Fig. 13.15 Static and adaptive power capping

power cap values for executing jobs so that their QoS levels are satisfied. To control and restrict power usage of each job, we use RAPL for CPU power capping. Since more than half of the power of compute nodes is usually consumed by CPUs and power fluctuation of CPUs is much larger than that of the other subsystems, the system power is fairly manageable by controlling CPU power.

Figure 13.15 shows power consumption versus execution time when (a) no power capping, (b) static fixed power capping, and (c) adaptive power capping are applied. Using TDP as the power allocation will leave a certain amount of underutilized power resource, but no performance degradation is expected as shown in Fig. 13.15a. If power cap value is statically determined and fixed at, for example, average power consumption of the job throughout the execution as illustrated in Fig. 13.15b, underutilized power becomes smaller, but performance of the job may be unexpectedly degraded. Figure 13.15c shows an example of power draw of the adaptive power capping for a job whose power cap value is set such that performance loss is smaller than expected. Because the power cap is set to the same as or below the maximum power demand of the job, no underutilized power is reserved for it, and the adaptive power capping scheme saves a certain amount of power budget, and hence the scheduler can submit additional jobs to the system to improve the system throughput.

We evaluated our adaptive capping scheme on a subsystem of HA8000 with 32 compute nodes. We assumed that the power budget of CPUs and DRAMs is 4,956 W which corresponds to the sum of the maximum power of PKG and DRAM domains of 14 compute nodes. We created a job set which consists of 90 jobs from NPB. The QoS and the number of compute nodes for each job are assigned randomly. The time interval of the power management is set to one second.

The adaptive capping scheme (Adaptive) was compared with four fixed power capping strategies, namely, NoCap, MaxReq, Linear, and Nominal. In NoCap, the power manager allocates TDP (130 W) to each CPU. In MaxReq, power allocated to a CPU is the maximum power which satisfies the performance requirement and is determined empirically by profiling. Linear uses a power cap value which is calculated from a linear function of the performance requirement value based on the max-min CPU power. In Nominal, the power cap value is the average power value which satisfies performance requirement.

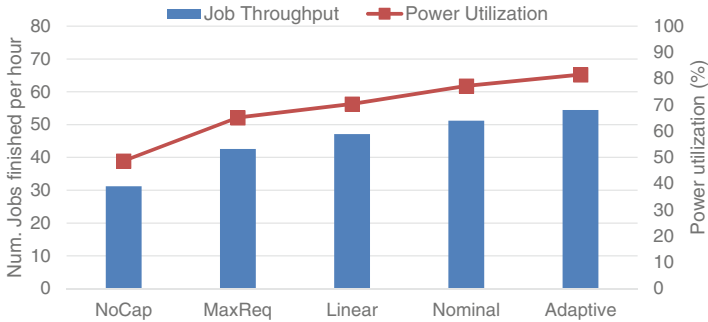


Fig. 13.16 Job throughput and relative power utilization

Figure 13.16 shows job execution throughput of the system (number of jobs finished per hour) and the ratio of the average utilized power to the power constraint of the system. NoCap has the worst power resource utilization since TDP is allocated to each CPU and that is much higher than its effective power usage. A job in the queue needs to wait for execution until the power budget of TDP for all the CPUs and DRAMs necessary for it is available, leading to poor job throughput. The job throughput and power resource utilization gradually increase from MaxReq to Adaptive. Adaptive achieves the best throughput and power resource utilization since it can afford much of power budget to waiting jobs in the queue and execute more number of jobs simultaneously. The result indicates that the adaptive capping scheme is very useful for power-constrained HPC systems.

13.4 Power-Performance Estimation for Large-Scale Systems

As discussed in Sect. 13.2.1, manufacturing variation strongly affects the power characteristics of large-scale systems. This means that considering the variability is an essential challenge for accurate estimation of system-wide power consumption. To address this challenge, we introduce a power model that assumes liner function of clock frequency and a scheme to calibrate it to solve the variation issue. Due to the manufacturing variability, power consumption with minimum and maximum clock frequencies are not always identical in all modules even if they have the same hardware specification. Because it is impractical to execute the application on all installed modules to obtain the power values, we use a system-level Power Variation Table (PVT) and perform two **single-module** application runs for lightweight profiling.

Figure 13.17 shows the steps of our power model calibration. The PVT consists of N (the number of total modules in the system) entries, each of which stores variation scales associated with the CPU and DRAM in each module to represent the degree of inter-module variation. The PVT is generated when the system is installed

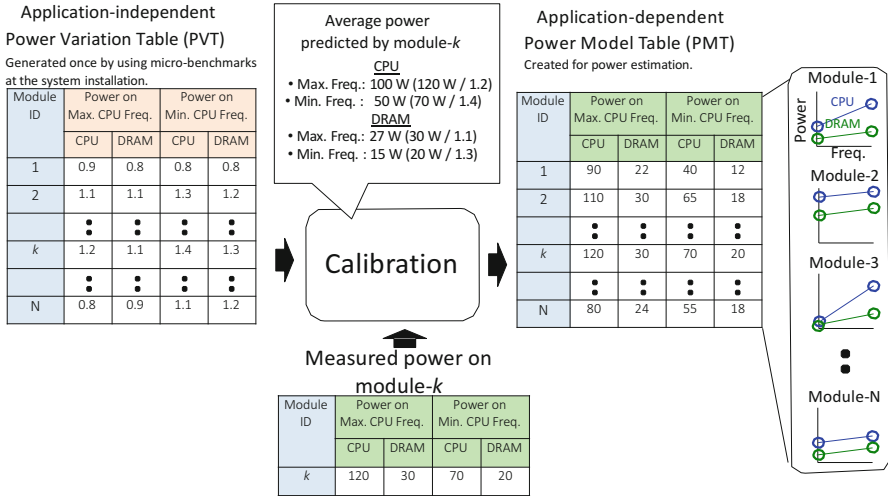


Fig. 13.17 PVT and PMT for power model calibration

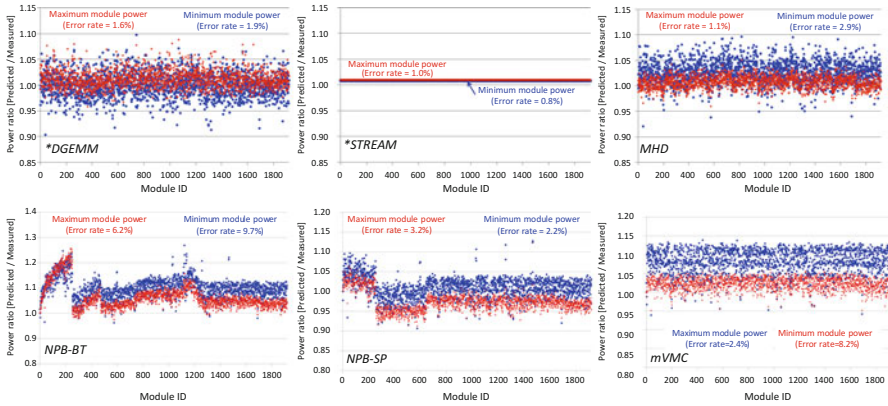


Fig. 13.18 Power estimation results

by executing representative microbenchmarks on each module. The power values at the maximum and minimum CPU frequencies are measured for each module, and the variation scales are obtained by dividing each of these module power values by the respective average. By performing two single-module test runs at the maximum and minimum CPU frequencies, we measure the power values on module-k and then generate the application-dependent Power Model Table (PMT) that includes the calibrated power models for all modules as shown in Fig. 13.17.

Figure 13.18 reports the power estimation results obtained by performing our power model calibration. The x-axis shows the module ID, and the y-axis presents the power ratio obtained by dividing the predicted power value by the measured one

on each module. For instance, 1.0 of the power ratio means the perfect estimation. Here, we show the results for six benchmark programs explained in Sect. 13.2.1. From the results, it is observed that our model calibration scheme works well. For all benchmarks, both the maximum and minimum module power can be estimated with less than 10% error (in average).

We have also developed power-performance estimation methodologies for computing nodes and interconnection networks. Our plan is to integrate them to make it possible a full system power-performance estimation.

13.5 Conclusion

The power is becoming a most precious resource in supercomputer systems. The power-constraint adaptive system design or hardware overprovisioning is a viable approach for increasing power utilization and performance for power-limited HPC systems at scale. In the *PomPP* project, we have developed several power-performance optimization strategies and frameworks including the framework to maximize application performance under a given power constraint, power-aware resource manager, and power-performance simulation and analysis framework for exascale applications. Tools developed in this project is freely available on a GitHub open source repository [4]. We continue to make an effort of research and development of the tool set for effectively utilizing power budget in future post petascale supercomputers.

References

1. Cao, T., He, Y., Kondo, M.: Demand-aware power management for power-constrained HPC systems. In: Proceedings of the 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid2016), Cartagena, pp. 21–31 (2016)
2. Cao, T., Huang, W., He, Y., Kondo, M.: Cooling-aware job scheduling and node allocation for overprovisioned HPC systems. In: Proceedings of 31st IEEE International Parallel & Distributed Processing Symposium (IPDPS 2017), Orlando (2017)
3. Casanova, H., Giersch, A., Legrand, A., Quinson, M., Suter, F.: Versatile, scalable, and accurate simulation of distributed applications and platforms. *J. Parallel Distrib. Comput.* **74**(10), 2899–2917 (2014)
4. <https://github.com/pompp/>
5. Inadomi, Y., Patki, T., Inoue, K., Aoyagi, M., Rountree, B., Schulz, M., Lowenthal, D., Wada, Y., Fukazawa, K., Ueda, M., Kondo, M., Miyoshi, I.: Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC15), Austin (2015)
6. Intel 64 and IA-32 architectures software developers manual. Intel, vol. 3, Mar 2013
7. Kogge, P.M.: Architectural challenges at the exascale frontier. In: *Simulating the Future: Using One Million Cores and Beyond* (invited talk) (2008)

8. Miwa, S., Aita, S., Nakamura, H.: Performance estimation for high performance computing systems with energy efficient ethernet technology. *J. Comput. Sci. Res. Dev.* **29**(3–4), 161–169 (2014)
9. Miwa, S., Nakamura, H.: Profile-based power shifting in interconnection networks with on/off links. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC15)*, Austin (2015)
10. PDT. <https://www.cs.uoregon.edu/research/pdt/home.php>
11. Sakamoto, R., Cao, T., Kondo, M., Inoue, K., Ueda, M., Patki, T., Ellsworth, D., Rountree, B., Schulz, M.: Production hardware overprovisioning: real-world performance optimization using an extensible power-aware resource management framework. In: *Proceedings of the 31st IEEE International Parallel & Distributed Processing Symposium (IPDPS 2017)*, Orlando (2017)
12. Sakamoto, R., Patki, T., Cao, T., Kondo, M., Inoue, K., Ueda, M., Ellsworth, D., Rountree, B., Schulz, M.: Analyzing resource trade-offs in hardware overprovisioned supercomputers. In: *Proceedings of the 32nd IEEE International Parallel & Distributed Processing Symposium (IPDPS2018)*, Orlando (2017)
13. Saravanan, K.P., Carpenter, P.M., Ramirez, A.: Power/performance evaluation of energy efficient ethernet (EEE) for high performance computing. In: *Proceedings of the 2013 IEEE International Symposium on Performance Analysis of Systems and Software*, Austin, pp. 205–214 (2013)
14. Saravanan, K.P., Carpenter, P.M., Ramirez, A.: A performance perspective on energy efficient HPC links. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC14)*, New Orleans, pp. 313–322 (2014)
15. Shende, S.S., Malony, A.D.: The TAU parallel performance system. *Int. J. High Perform. Comput. Appl.* **20**(2) (2006). <https://www.cs.uoregon.edu/research/tau/home.php>
16. Totoni, E., Jain, N., Kale, L.: Power management of extreme-scale networks with on/off links in runtime systems. *ACM Trans. Parallel Comput.* **1**(2), 16 (2015)
17. Wada, Y., He, Y., Cao, T., Kondo, M.: A power management framework with simple DSL for automatic power-performance optimization on power-constrained HPC systems. In: *Proceedings of Supercomputing Asia (SCA18)* (2018)
18. Yoo, A., Jette, M., Grondona, M.: SLURM: simple linux utility for resource management. In: *Job Scheduling Strategies for Parallel Processing*, Seattle. *Lecture Notes in Computer Science*, vol. 2862, pp. 44–60 (2003)

Chapter 14

Project CASSIA —Framework for Exhaustive and Large-Scale Social Simulation—



**Itsuki Noda, Yohsuke Murase, Nobuyasu Ito, Kiyoshi Izumi,
Hiromitsu Hattori, Tomio Kamada, Hideyuki Mizuta, and Mikio Takeuchi**

Abstract Project CASSIA (Comprehensive Architecture of Social Simulation for Inclusive Analysis) aimed to develop a framework to administer execution of large-scale multiagent simulations exhaustively to analyze socially interactive systems on high-performance computing infrastructures. The framework consists of two parts, MASS Planning Module and MASS Parallel Middleware. MASS Planning Module is a manager module conducts effective execution plans of simulations among massive possible conditions according to available computer resources. It consists of OACIS/CARAVAN and their intelligent modules. MASS Parallel Middleware is an execution middleware which provides functionality to realize distributed multiagent simulation on many-core computers. It is a collection of X10 libraries, scheduler, and XASDI, a platform to program multiagent simulations. The CASSIA framework was applied to various real applications in pedestrian, traffic, and economics simulation domains and provided practical results and suggestions for real-world problems. We also discussed road maps of social simulation and

I. Noda (✉)

AIST, Tsukuba, Japan

e-mail: i.noda@aist.go.jp

Y. Murase

Center for Computational Science, RIKEN, Kobe, Japan

N. Ito

University of Tokyo, Tokyo, Japan

Riken, Tokyo, Japan

K. Izumi

University of Tokyo, Tokyo, Japan

H. Hattori

Ritsumei University, Kyoto, Japan

T. Kamada

Kobe University, Kobe, Japan

H. Mizuta · M. Takeuchi

IBM, Tokyo, Japan

high-performance computing to attack real and huge issues on social systems. This discussion indicates the possibility of CASSIA framework and multiagent simulations to realize engineering environment to design and synthesize social systems like traffics, economy, and politics.

14.1 Overview

Project CASSIA (Comprehensive Architecture of Social Simulation for Inclusive Analysis) aims to develop a framework to administer execution of large-scale multiagent simulations exhaustively to analyze socially interactive systems. The framework will realize engineering environment to design and synthesize social systems like traffics, economy, and politics.

The purpose of multiagent social simulation is to provide tools to design social systems. It is impossible or quite difficult to carry out experiments of social phenomena in the real world by the similar way as experiments in physics or chemistry. Therefore, computational social simulations are indispensable means for social science.

Fortunately, progress of computational power has a potential to realize wider applications of computer simulation not only in physical phenomena but also in social problems. High-performance computing (HPC) has been enabling several simulation researches on large-scale weather, molecular dynamics, structures and architectures, and disasters. In addition to these physical phenomena, recently, social phenomena like economics, traffics, or information flow on networks attract many attentions as applications of HPC (Fig. 14.1).

In order to make such social simulations on HPC available for wide-range users, the CASSIA framework consists of:

- MASS Planning Module: a manager module conducts effective execution plans of simulations among massive possible conditions according to available computer resources.
- MASS Parallel Middleware: an execution middleware provides functionality to realize distributed multiagent simulation on many-core computers.

14.2 MASS Planning Module

14.2.1 *Categories of Simulation Class in Computational Cost*

In this section, we give an overview on the two frameworks for parameter-space exploration, OACIS and CARAVAN. Although both of these frameworks are designed in order to conduct parameter-space exploration making full use of HPC resources, they differ in the target scale of each job. On the one hand, a certain

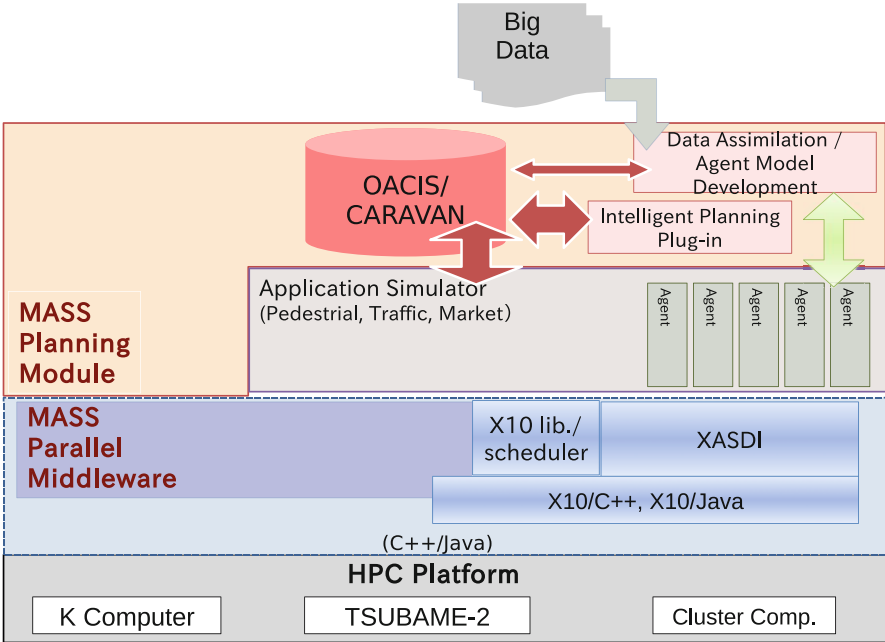


Fig. 14.1 Cassia framework

class of research issues requires to use the maximum computing power to solve a single large problem, so-called capability computing. On the other hand, other researches require to do many small- or medium-scale simulations for parameter-space explorations, i.e., capacity computing. We categorized these problems into four classes depending on the scale of a single simulation job as summarized in Table 14.1. In this table, it is assumed that the total amount of computation is order of ten exa-floating-point operations. The left most column, which we call class A, corresponds to typical capability computing. The number of independent jobs is at most 10^2 . In class B, a typical single job is an MPI-parallel program using $10^3 \sim 10^5$ CPU cores. The number of jobs amounts to $10^3 \sim 10^5$. In this class, a naive manual management of jobs is no longer possible, and a framework for managing jobs is necessary. When typical job scale is serial or shared-memory parallel application, as labeled class C, the number of jobs expands up to $10^6 \sim 10^9$, which requires an even harder job management. In this class, the parameter selection and interpretation of the results for each job must be done algorithmically. Finally, on the right most class, where a single job becomes a function level, the number of jobs is more than 10^{10} . Thus, for capacity computing ranging from class B to D, the demands for frameworks can be totally different depending on the granularity of jobs. This is why we developed two frameworks, OACIS and CARAVAN for classes B and C, where majority of the social simulations are found.

Table 14.1 Categorization of the problems according to the scale of single simulation job. To calculate the required number of operations, FLOPS, and the number of CPU cores for each job, we made assumptions that an exa-flops computer is available, the efficiency of each job is 10%, and the duration of each job is order of 10^2 s. From left to right, the typical scale becomes finer, while the typical number of jobs gets greater. In the last two lines, we showed an example of social simulations and a framework used for parallel job execution

Class	A	B	C	D
# of jobs	$10^0 \sim 10^2$	$10^3 \sim 10^5$	$10^6 \sim 10^9$	$10^{10} \sim$
# of operations/ job	$10^{19} \sim 10^{17}$	$10^{16} \sim 10^{14}$	$10^{13} \sim 10^{10}$	$10^9 \sim$
FLOPS/job	$10^{18} \sim 10^{16}$	$10^{15} \sim 10^{13}$	$10^{12} \sim 10^9$	$10^8 \sim$
# of cores/job	$10^8 \sim 10^6$	$10^5 \sim 10^3$	$10^2 \sim 10^{-1}$	$10^{-2} \sim$
Typical job scale	Large-scale MPI	Medium-scale MPI	SMP or serial	Function
Parameter selection	Manual	Manual or auto	Auto	Auto
Social simulation application	–	Traffic in metropolitan area	Traffic in a city	Data-driven model
Frameworks		OACIS	CARAVAN	Map-reduce

14.2.2 OACIS

OACIS, which stands for Organizing Assistant for Comprehensive and Interactive Simulations, is a job management framework for problems in class B [12]. It is available as an open-source software under the MIT license. (<http://github.com/crest-cassia/oacis>). This class of problems require researchers to carry out many simulation jobs changing models and parameters by trial and error. This kind of trial-and-error approach often causes a problem of job management because of a large amount of repetitive works. Such repetitions are not only troublesome and tedious but prone to human errors. OACIS is designed to let researchers conduct their research in an efficient, reliable, and reproducible way, helping management of simulation jobs and results.

The system architecture of OACIS is depicted in Fig. 14.2. It is a web application developed based on the Ruby on Rails framework, which provides an interactive user interface. The application server is responsible for handling requests from users. When a user creates a job using a web browser, the record of the job is created in the database. Another daemon process, which we denote as “worker,” periodically checks whether a job is ready to be submitted to a remote host. If a job is found, the worker generates a shell script to execute a job and submits it to the job scheduler on the remote host (which we call “computational hosts”) by SSH connection. The worker process then periodically checks the status of the submitted jobs, and, when the jobs are finished, it downloads the results and stores them into designated storage and database appropriately. Hence, users do not have to check the job status by themselves, and the simulation results are kept in an organized and traceable way. Various logs, including the values of parameters, executed dates, elapsed times, and

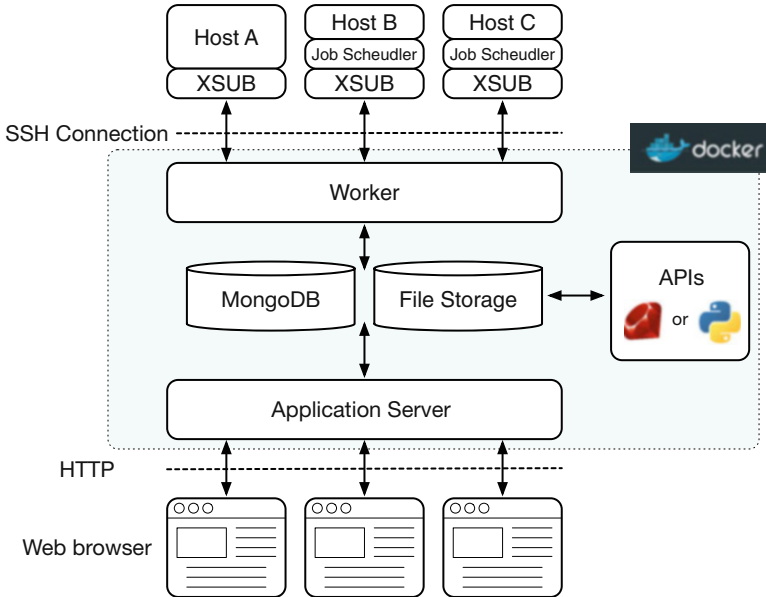


Fig. 14.2 A system overview of OACIS

the version number of the simulator, are automatically kept as well. A simulator on OACIS is registered as a command line string to execute the simulation, not as the execution program itself. By this design, OACIS can run simulators in various research fields, which may be written in different programming language.

In addition to an interactive user interface, OACIS provides application programming interfaces (APIs) in Ruby and Python programming languages. Any set of operations on OACIS is programmable using the APIs, which can be used for various types of parameter-space explorations including parameter sweeps, sensitivity analysis, and optimization of parameters.

14.2.3 CARAVAN

CARAVAN is another framework designed for class C jobs. It is also available as an open-source software under the MIT license (<https://github.com/crest-cassia/caravan>).

Figure 14.3 shows the whole architecture of CARAVAN. It consists of three parts: search engine, scheduler, and simulator. “Simulator” is an executable program prepared for each use case. Once a user integrate a simulator into CARAVAN, it is executed in parallel. Since a simulator is executed as an external process, a simulator may be implemented in any language as in OACIS. “Scheduler” is a part which is

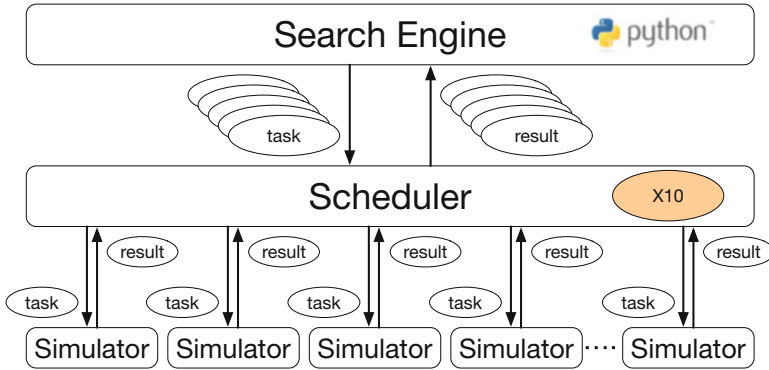


Fig. 14.3 A system overview of CARAVAN

responsible for parallelization. It receives the commands to execute simulators from the search engine, distributes them to available nodes, and executes the simulator in parallel. This part is implemented in X10 programming language using MPI for a communication layer. “Search engine” is a part which determines the policy on how parameter-space is explored. More specifically, it generates a series of commands to be executed in parallel and sends them to scheduler. It also receives the results from the scheduler when these tasks are done. Based on the received results, search engine can generate other sets of tasks repeatedly as many as a user wants. This part is written in Python. A simulator and a search engine must be prepared by each user, while the scheduler does not have to be modified once it is built.

When writing a simulator and a search engine, users do not have to explicitly take care of the parallelization. The scheduler is designed so that the whole application can scale up to tens of thousands of processes. To evaluate the performance of the scheduler on the K computer, we tested an embarrassingly parallel problem, in which each task takes about 20 s. We obtained a result that the efficiency of the task scheduling remains more than 99% even when the number of MPI processes is scaled up to 18,432.

14.3 MASS Parallel Middleware

14.3.1 X10 Extensions and Plham

To realize distributed multiagent simulation on large-scale distributed computers, we developed parallel middleware on X10 [17], which is an object-oriented parallel programming language developed by IBM. X10 adopts partitioned global address space (PGAS) model and features asynchronous and fork-join-style programs. Using X10 and our parallel middleware, we developed Plham [4, 22], a platform for

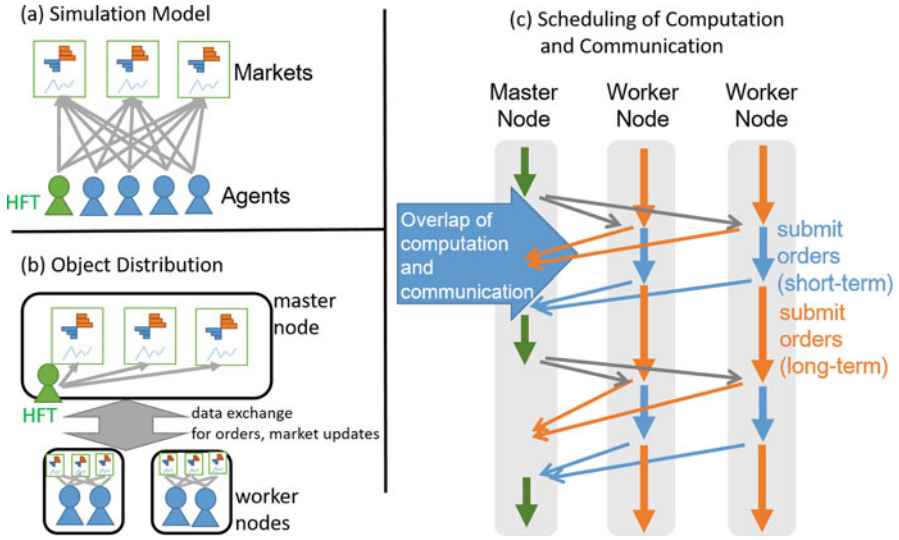


Fig. 14.4 Plham (simulation model and its parallel execution)

large-scale and high-frequency artificial market simulation. In this section, we first summarize the design and implementation of Plham to enable large-scale and high-frequency artificial market simulation. Then we show two libraries, a distributed collection library and a global load balancing library with multistage facility [24], that are designed for large-scale multiagent simulations.

Artificial market agent-based simulations have potential to be a strong tool for studying rapid and large market fluctuation and designing financial regulations. High-frequency traders, which exchange multiple assets simultaneously within a millisecond, are said to be a cause of rapid and large market fluctuation. Plham enables modeling financial markets composed of various brands of assets and a large number of agents trading on a short timescale. The design feature of Plham is the separation of artificial market models (simulation models) from their execution (Fig. 14.4). The primary components of a simulation model are “agents” and “markets” as shown in (a). The term *agent* is used to represent a trader, including high-frequency traders. The term *market* is used to represent a system or place where agents can buy or sell a specific asset (here, we assume one market is for one asset). A certain type of assets depends on a collection of assets and is itself a tradable financial instrument (e.g., index futures, exchange-traded funds). Users can employ ready-made agent/market classes provided by Plham or define their original ones by extending these classes and construct their simulation models without parallel computing expertise.

Plham provides multiple runners that implement respective execution models. To execute a parallel simulation, the users can employ a parallel runner implementing its parallel execution model that allocates agents over multiple computing nodes, changes the scheduling frequency of agents depending on the class of agents, and accepts delayed propagation of market information depending on the class. In the current implementation, HFT agents are assigned at the master node, in which the order handling of the markets is processed, and the computation for ordinal traders is executed at the worker nodes, as shown in Fig. 14.4b. For large-scale simulations, nonHFT agents must be forced to make a decision by using delayed or out-of-date market information, whereas HFT agents are given access to latest or relatively up-to-date market information.

The current parallel runner allows concurrent execution of order submission and handling as shown in (c) to reduce CPU idle time at worker nodes and achieve simulation scalability. To increase the parallelism of the simulation, the runner introduces three agent ranks: high-frequency traders, short-term traders, and long-term traders. It then concurrently executes the order-submission stage of long-term traders and the order-handling stage of markets. In addition, it succeeded in overlapping communication and computation. The scalability of the current parallel execution model was evaluated in weak scaling to a maximum of 2048 computing nodes on the K computer [4]. Plham is available as an open-source software under Eclipse Public License (<http://github.com/crest-cassia/plham>).

To realize distributed multiagent simulation on large-scale distributed computers, developers have to implement proper agent distribution, data transportation for agent communication, parallel execution of agent computation using multi-core CPUs, and efficient scheduling of respective components of computation and communication. We developed a distributed collection library to easily manage collection of object elements distributed over multiple computing nodes. It offers methods for element relocation (communication) and data parallel processing for the elements (computation). Our library provides a series of distributed collections, such as `DistCol [T]`, `DistBag [T]`, and `DistMap [K, V]`, each of which manages a collection of objects that spreads over worker nodes. Our library also prepared another type of distributed collection, `ColDist [T]`, that holds a list of objects and allocates the cache proxies of the list at all worker nodes. In the implementation of Plham, `ColDist` is used to manage markets and their proxies and `DistCol` is used for agents. Orders and contracted orders are stored into `DistBag` and `DistMap` and relocated at each calculation step. Our library allows object transportation by using collective communication of MPI, and the elapsed time for the communication can be reduced. These classes also offer an asynchronous method, `asyncEach`, that receives a function and applies it to all the local elements in the called node using thread pools. Using these features, the overlapping of communication and computation in Plham was briefly described as shown in Fig. 14.5a.

We also conducted research on global load balancing for multiagent simulation having multiple calculation steps. Load balancing is a major concern in massively parallel computing. X10 provides a global load balancing (GLB) library that shows

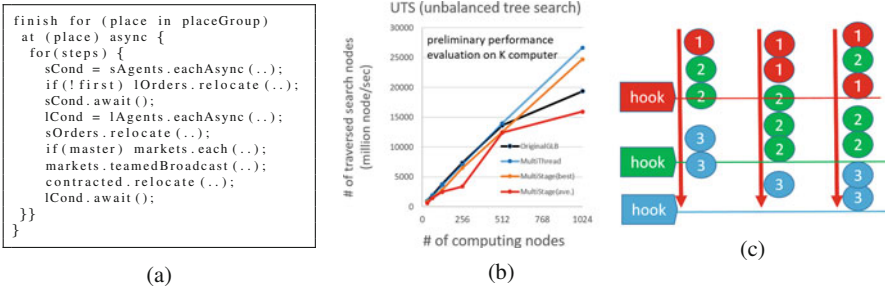


Fig. 14.5 Parallel middleware for large-scale multiagent simulations. (a) Sample program of distributed collections. (b) Performance evaluation of global load balancing. (c) Multistage features for GLB

high scalability over 10,000 CPU cores. GLB features a lifeline-based scalable work-stealing algorithm. Results of completed tasks are gathered by means of reduction operations. We introduced a multithread design into GLB to allow efficient data sharing between CPU cores. The multithread version showed high scalability than the original one (Fig. 14.5b). In addition, we proposed a multistage mechanism for GLB to assign execution stages to tasks. The system gives high priority to tasks that are assigned to earlier stages and then proceeds with subsequent stage tasks (Fig. 14.5c). When a computing node runs out of tasks at the earliest stage, it requests tasks at the earliest stage from other nodes and awaits responses by processing subsequent stage tasks. When the system identifies the task termination at a certain stage, it executes a reduction operation over nodes. The multistage version has performance problems that appear to be caused by network message scheduling or thread scheduling. After fixing the problems, we plan to introduce this dynamic load balancing features into our distributed collection library to treat dynamic load imbalance of tasks.

14.3.2 XASDI

In this section, we describe the overview of X10-based Agent Simulation on Distributed Infrastructure (XASDI). This framework is published as an open-source software (<https://github.com/x10-lang/xasdi>) under the Eclipse Public License (EPL).

XASDI is large-scale agent-based social simulation framework with enormous number (billions) of agents to represent citizens in cities or countries. XASDI enables distributed simulation with the X10 language for post-peta-scale machines. The X10 programming language (<http://x10-lang.org/>) is the APGAS (Asynchronous Partitioned Global Address Space) language that provides highly parallel and distributed functionalities with Java-like syntax [2]. X10 application can be

Fig. 14.6 XASDI software stack

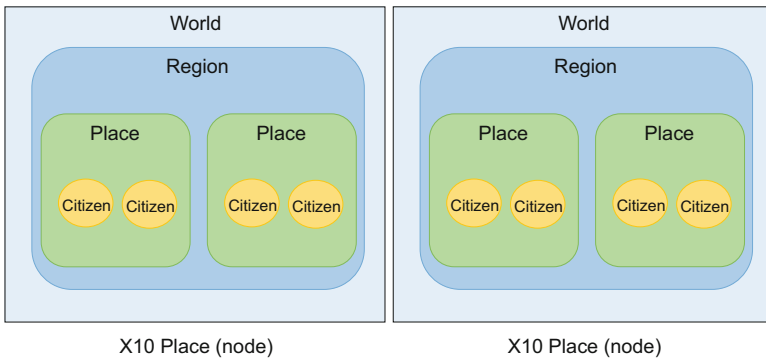
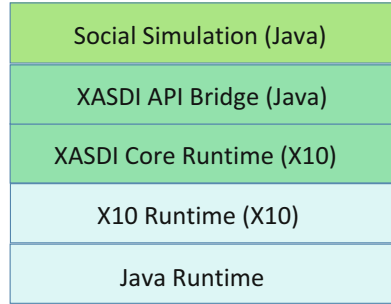


Fig. 14.7 XASDI hierarchical structure to manage agents

compiled and executed on Java environment [19, 20]. With this feature, XASDI provides easy-to-use API with Java that is familiar to application programmer of social simulations and can be developed with powerful IDE functionalities (e.g., Eclipse refactoring and debugger).

XASDI software stack contains core runtime written in X10 language for distributed agent and execution management and Java API bridge to enable application programmer to utilize familiar Java languages (Fig. 14.6). By utilizing XASDI framework, users can easily develop their social simulator with Java on distributed parallel environment without studying the new X10 language.

The agent in XASDI is referred to as Citizen and Citizen has corresponding CitizenProxy that is managed in the simulation environment to exchange messages. To manage CitizenProxy, XASDI provides a hierarchical container structure called Place, Region, and World (see Fig. 14.7). CitizenProxies belong to a Place and Places belong to a Region. World can contain several Regions, but usually there is only one Region in the World.

Here, we need to note that the confusing terminology of the X10 programming language and the framework. The X10 use the term “Place,” too, but the meaning of the term is different. The Place of X10 is used to denote the distributed execution environment for multi-core or multi-node. For this meaning, we will use “X10 Place

(node)” in distinction from the Place container of agents. Only one World instance exists in one X10 Place and manages lists of entities in the world including Region and Citizen. The world can also contain IDs of Citizens in other nodes.

Other important classes in XASDI are Message, MessageRepository and Driver. MessageRepository manages Message exchange among CitizenProxy and environment, and this class also works as interface between Java environment and X10 environment to exchange Messages in distributed X10 Places. Driver manages execution of the simulation with a corresponding thread. Each Driver is related to Places (and Citizens in the Places) where it has a responsibility for execution.

Finally, XASDI provides a logging mechanism. By preparing log definitions for the application, it can output the simulation log at each X10 Places.

Application users of XASDI need to develop their own simulation application by utilizing these classes and execute the application with XASDI library on Java and X10. We describe the execution process on XASDI and the simple customization for the sample application bundled with XASDI.

A user starts the simulation by executing the shell script to invoke the X10 core runtime. After the preparation of X10 environment, Launcher for the simulation written by Java is called at each X10 Place. Launcher reads the initialization file and generates a Region, Places and Drivers. If agents are needed to exist from the beginning of the simulation, Launcher or Driver generates Citizens. Citizens can also be generated by Driver through the method of the Region during the simulation at given time or randomly. For example, consumer agents are generated when they enter a shopping mall.

The main simulation process is executed through Drivers. Simulation managers in the X10 core environment generate threads and invoke call back method in each Driver in parallel. One simulation time step can be divided into phases. The number of phases is determined at the beginning of the simulation with initialization file. The method of the Driver looks at the time and phase given by the environment and determines the action that the corresponding Places and Citizens should perform.

In one node, there is one Region instance that manages the execution of the simulator. The core framework written using X10 manages the Regions and Message Repositories of distributed nodes. The Message Repository supports several kinds of messages such as individual message, broadcast message, and control message to move agents between X10 Places. An individual message is a standard message from one agent to another. A broadcast message is sent to all nodes and received by the Region or agents corresponding to the type of message. A move message is a control message for X10 core runtime to remove the Citizen at the source node and restore it at the destination node with serialized field data stored in the message.

As applications for large-scale simulation on XASDI, we developed a large-scale traffic simulator for city [15] and a shopping mall simulator with integrated model of walking and purchasing behavior [8]. Figure 14.8 shows the weak scaling performance of XASDI by using our application for shopping mall simulation with distributed consumer agents [18].

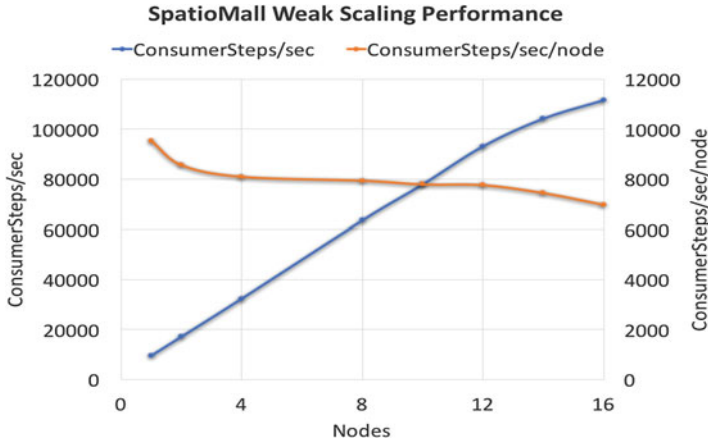


Fig. 14.8 Weak scaling performance of XASDI with shopping mall application

14.4 Applications

14.4.1 Financial Market Simulation

We developed the platform software Plham (platform for large-scale and high-frequency artificial market), which can simulate financial markets with thousands of stocks and hundreds of thousands of market participants [22]. By joint research with Tokyo Stock Exchange, the market simulation result using Plham supported a new rule determination in the actual financial market. Simulation execution and management utilized the execution control system OACIS developed in this project. The information about the main part of OACIS, improvement of a user interface, and the decision of analysis/visualization application acquired by simulation execution was fed back to the OACIS development team. The result shown below was obtained about the market simulation.

14.4.1.1 Policy-Making Support of Tick Size Reduction of Tokyo Stock Exchange

About 100 kinds of parameter study were performed using the prototype program of the small-scale financial market simulation in which 1,000 trading agents trade one kind of stocks (Fig. 14.9) [9–11]. As compared with a simulation result and the actual data of Tokyo Stock Exchange, the threshold of proper tick size (the minimum price unit) was calculated from the fluctuation of a stock price. This simulation result was reflected in the plan of selection of the target stocks in the tick size reduction of Tokyo Stock Exchange in 2014. Furthermore, while developing an analytical theory which reproduces a simulation result in good accuracy and

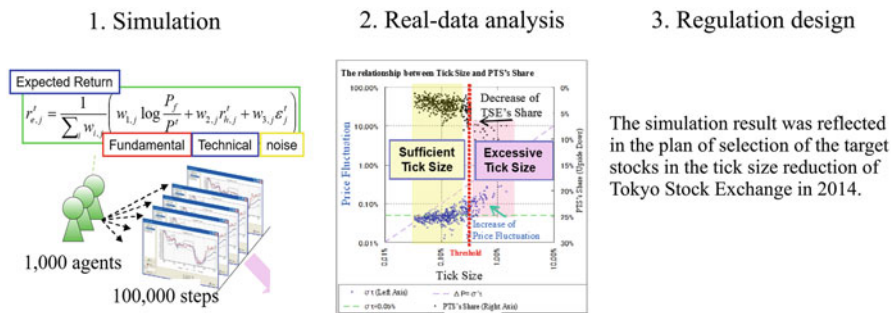


Fig. 14.9 Impact analysis of the tick size reduction by a financial market simulation

helping an intuitive understanding of the simulation result, the simulation model which reduces a computational complexity was developed.

14.4.1.2 Simulation of Flash Crash

The prototype model was extended using the parallel processing language X10 to the larger-scale simulation which consists of 100 individual stocks and 1 index security coupled to a stock price average [21]. We investigated the conditions to which the high-frequency arbitrage trading propagates a rapid price change to other stocks using this model. About 5000 kinds of parameter sets were analyzed using social simulation management module OACIS, and the combination of the trading strategies which a price decline propagates to other stocks and the flash crash may occur was specified (Fig. 14.10). We introduced the circuit breaker into the model as a financial regulation which prevents such fluctuation propagation and investigated the conditions where a circuit breaker prevents propagation effectively, and the conditions where it accelerated the propagation.

14.4.1.3 Analysis of the Influence of Risk Management Regulation

By the joint research with a financial institution, we developed the support method of the institutional design financial regulation such as capital adequacy requirements [30]. As a result, the market became unstable when the market participants who manage a market risk based on capital adequacy requirements increased in number (Fig. 14.11). An increase of the kind of securities in which it trades showed that the instability by regulation was eased (Fig. 14.12).

This artificial market simulation software Plham is released from April 2016 (<https://github.com/plham/plham>). Since it is implementation by an agent model, it can reproduce the mixed environment of various investment strategies seen in actual markets, such as an HFT and an automatic transaction. Since it is implemented with the parallel processing language X10, it can run on various scales from

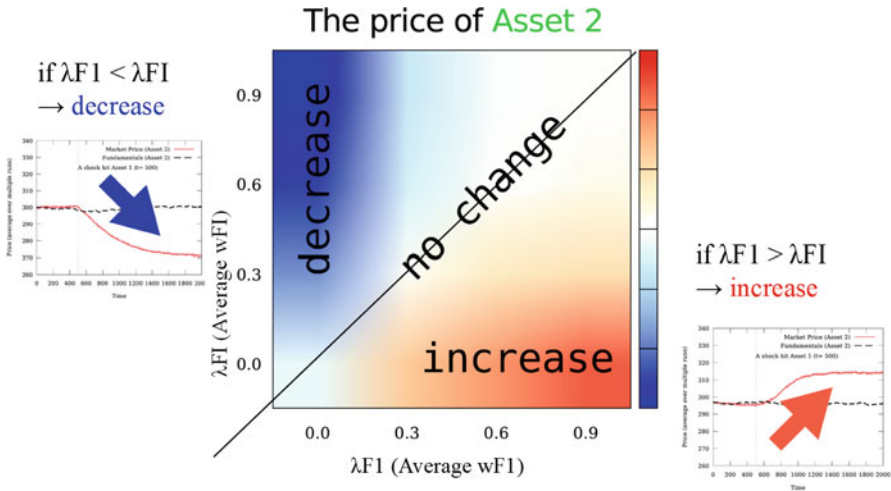


Fig. 14.10 Analysis of the price decline propagation by a simulation

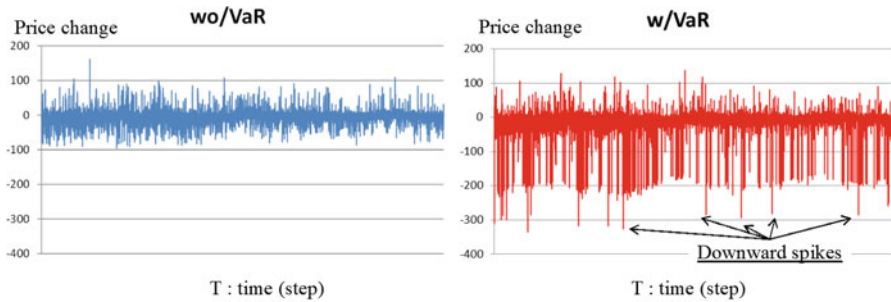


Fig. 14.11 The example of price fluctuation in having no capital adequacy requirements (left figure) and a simulation with regulation (right figure)

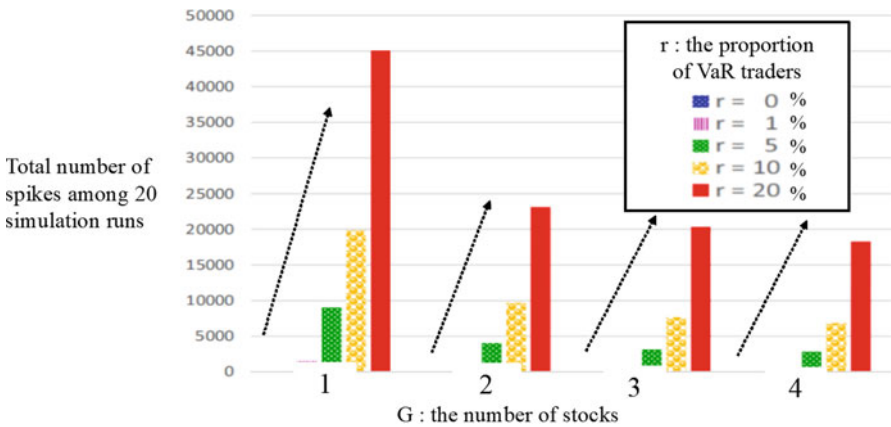


Fig. 14.12 The ratio r of the agents imposed by regulation, and the relation between the number G of trading capital, and the instability (vertical axis) of a market

stand-alone PC to large-scale parallel execution environment according to the scale of a simulation. We are targeting a researcher or not only an engineer but the financial working member and the policy-maker as a user.

14.4.2 Pedestrian Simulation

CASSIA Framework can illustrate a trade-off structure of social problems. The most of social problems like planning of evacuations from disasters are not simple optimization problems but dilemmas among multiple objective functions. We will show an example to apply CASSIA framework to find such trade-off structures using evacuation planning for Nishiyodogawa-ku, Osaka, which includes over 300 control parameters [7]. Because of the large degrees of freedom, the search space of this problem is so huge that the solution of this problem require high-performance computing like K computers.

14.4.2.1 Pedestrian Simulator: CrowdWalk

In order to simulate evacuation situations, we employ CrowdWalk [27, 28]. CrowdWalk is a pedestrian simulator that limits human movement to one-dimensional movement on the road network. Road network is composed of nodes and links, on which CrowdWalk controls large number of pedestrian movements at high speed.

We use a road map of Nishiyodogawa-ku, which consists of 7,624 nodes (intersections) and 10,707 road links (Fig. 14.13). This area has 86 official refuges and 54,909 of the population, which are distributed in 146 small zones. We suppose that every people in each zone follows one guidance rule that requests them to go to a certain destination with one via point in a map. The destination and the via point are selected from the 86 official refuges and from 533 major intersections, respectively.

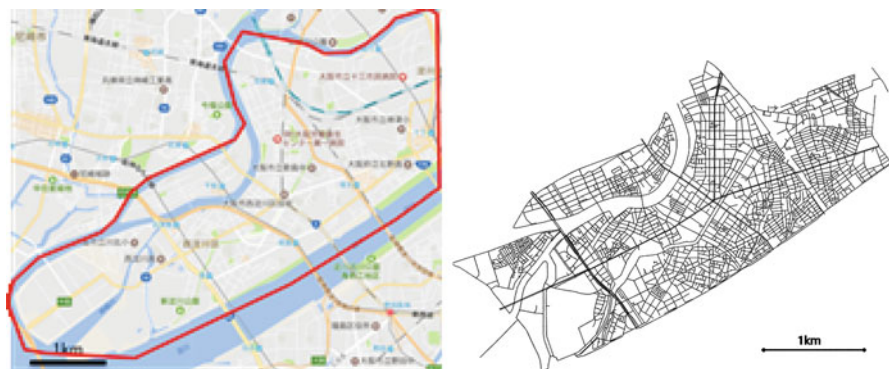


Fig. 14.13 Nishiyodogawa area (left) and road map (right) used in pedestrian simulation

14.4.2.2 Multi-objective Optimization

In general, evacuation planning includes a dilemma between evacuation time and simpleness of evacuation guidance. From the viewpoint of the minimization of evacuation time, it is better to use the result of mathematical optimization like maximum network-flow [6]. However, we need to guide large number of people that include persons who are not acquainted with the place like visitors. So, the guidance should be simple enough to understand and to follow easily.

In order to know the relationship of these two objectives, we apply NSGA-II (Elitist Non-Dominated Sorting Genetic Algorithm) [3], a genetic algorithm for multiple objective optimization.

For the first objection function, the evacuation time, is estimated by simulation using CrowdWalk for each guidance plan.

For the second objective function, the simpleness of evacuation guidance, we introduce “entropy” of the plan as follows. Suppose two connecting zones, z_i and z_j in the area, whose populations are n_i and n_j , respectively. If the two rules for these zones has same via point and destination, then the entropy is zero. Otherwise, the entropy of this pair is defined by:

$$H(z_i, z_j) = -(n_i/(n_i + n_j)) \log(n_i/(n_i + n_j)) - (n_j/(n_i + n_j)) \log(n_j/(n_i + n_j)).$$

Finally, we use total entropy $H = \sum_{z_i, z_j} H(z_i, z_j)$ for the index of the complexity of the guidance (opposite value of the simpleness).

14.4.2.3 Experimental and Discussion

In order to run NSGA-II for this guidance plan, we utilized OACIS described in Sect. 14.2.2 to manage the large number of runs. The search space of this problem is so huge ($R^{73} \times 533^{146} \times 86^{146}$), and NSGA-II requires large number of populations (about 100–1,000). So we need to run so many runs for the optimization. In the experiment, we run 500 generations with 100 population for the optimization, which means we run 500,000 simulations¹ for this experiment.

To control we implemented NSGA-II procedure using Ruby API of OACIS. In the actual GA procedure, we use “simulated binary crossover” and “polynomial mutation” for creating new generations and Paeto ranking mechanism to determine the selection.

Figure 14.14 shows the result of the experiment. In this graph, vertical and horizontal axes indicate evacuation time and the complexity of plan (total entropy scaled by 100), respectively. The color of the dot indicates the generations. From this result, we can see that the evacuation plans are improved by progress of generations and almost saturate to boundary of 3000 for evacuation time and 2100

¹We runs 10 simulations for one guidance plan to calculate average evacuation time.

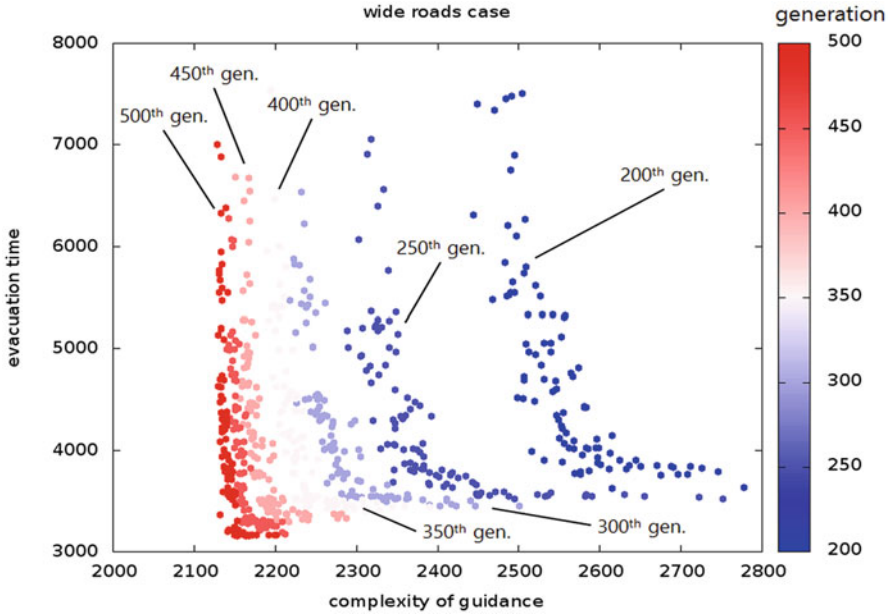


Fig. 14.14 Result of evacuation simulation (wide road)

for complexity of guidance. In order to minimize the evacuation time, we need to choose relatively complex guidance (the complexity is about 2200 rather than 2100). On the other hand, if we consider simplify the complexity, the evacuation time increase drastically up to 7000. And, we can see reasonable guidance will exist the most left-bottom area of the Pareto front in this graph.

Figure 14.15 also shows the result of the case that the people use only pedestrian road. In this case, the boundary of the evacuation time increase to 4500, but the complexity of guidance is similar to the previous case. Anyway, in both cases of Figs. 14.14 and 14.15, we can see a clear trade-off structure of Pareto solution sets.

This result implies that OACIS with evolutionary methods have a great potential to investigate such social problems. We also succeeded to apply CARAVAN to run the same procedure on K computer. This combination enables to run larger scale of simulations and search spaces.

14.4.3 Traffic Simulation

14.4.3.1 Simulation Cost and Benchmark

Traffic simulations are comprised of three parts: road map with traffic rules and regulations, origin and destination (OD) sets of cars with travel routes, and movements

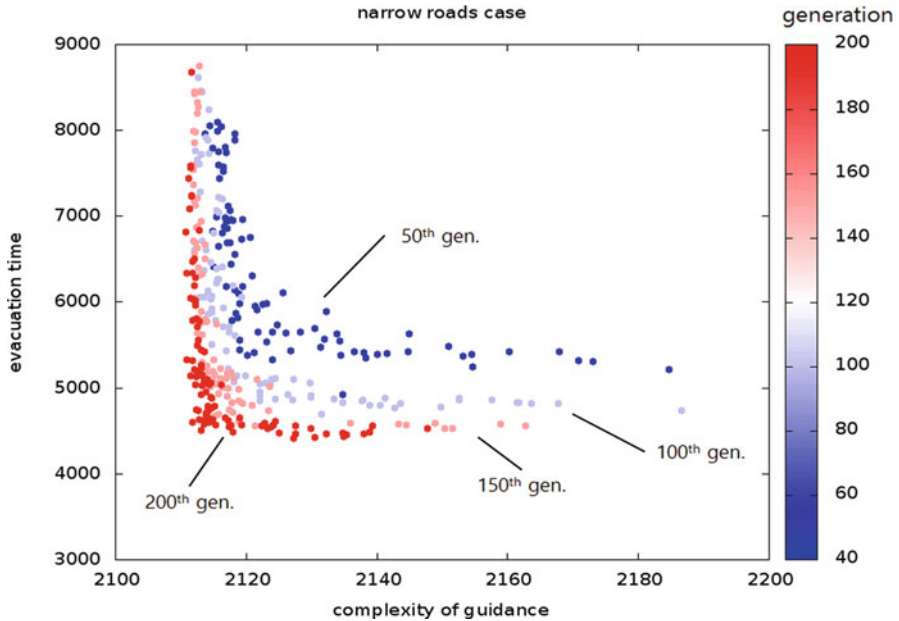


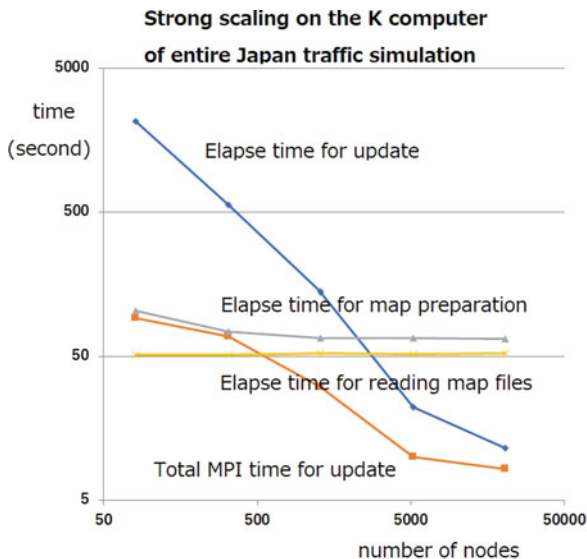
Fig. 14.15 Result of evacuation simulation (narrow road)

of cars on the map along their routes. There are two methods to parallelize traffic simulations. One is to parallelize geometrically, and the other car- or trip-wisely. Geometrical parallelization is advantageous on modern supercomputers, because necessary data transfer is states of cars which move out from an area on one node and go to an area on other node, and only nearest-neighbor communications appear.

Parallelization of trip routings is more complicated, but most trips are short range and are executed on one node using local map. Time-consuming long trips are few, and their routing cost can be made negligible.

Parallelization feature of car-traffic simulations with the most parallelization demanding case, that is, simulations using the simplest car movement, was measured in this CASSIA project. Each car continues to move along road and turn randomly at each crossing. This means that no routing is done. This simulation model will require the lightest processing in each computer node and the most frequent internode communication. Results of strong scaling of parallelized performance using the K computer up to its quarter configuration (20,736 nodes) for the traffic simulation on entire road of Japan are shown in Fig. 14.16. Simulations were up to 100 s and time step was 0.01 s. So totally 10,000 steps were done for about 11.7 million cars on totally 1.28 million kilometer-long roads using the “Open Street Map.” Using the quarter nodes, simulations of 10,000 steps were executed in 11.5 s apart from the initial preparation (66.1 s), and efficient parallelization is confirmed. For another simulation of 100 million car on all-over the world with 30.9 million kilometer took 116 s using quarter nodes of the K computer.

Fig. 14.16 Results of strong scaling using the K computer for entire Japan traffic simulation with the simplest movements without routings are plotted



This result implies that real-time or faster car-traffic simulation will be executed on the K and the future supercomputers.

14.4.3.2 Traffic Factors

In traffic simulation models, numbers of input parameters and output results are enormous. Each car has several input parameters, for example, its origin, destination, starting time, and driving preference, and also several output parameters, for example, travel duration and gas consumption. Each traffic signal has its control parameters. Each road segment has its speed limit and car flux. Such parameter-number explosion is quite common in various social models. In Kobe city, as an example, there are more than 30,000 road segments and about 100,000 trips (OD pairs) daily. So the number of input parameters and output results are order of 10^5 at least. Without knowing behavior of simulation model, we cannot apply the model to describe, predict, and design the real phenomena. But it is impossible to tame a model with 10^5 input and output. So the first step is to eliminate unimportant parameters and to extract relevant parameters. A standard method for this purpose is the multivariate statistical analysis.

In this CASSIA project, a car-traffic model and simulator of Kobe city was developed [1]. Using this simulator, Monte Carlo simulations for various OD sets satisfying a constraint that daily trips be 100,000 and 70% of the trips are coming from peripheral cities and just passing through Kobe city. A factor analysis was applied to the output flux of road segments [23]. Numbers of estimated factors are

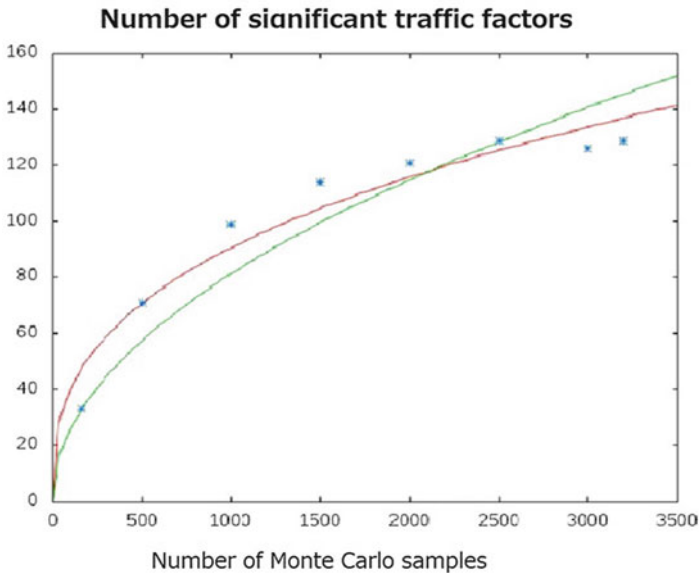


Fig. 14.17 Number of factors estimated using a factor analysis are plotted. Green curve shows square root of number of Monte Carlo samples, which is expected in the case of uniform distribution of eigenvalues of correlation matrix, but this plot is interpolated by slower red curve

plotted in Fig. 14.17. It is observed in this Fig. 14.17 that thousands of samples finally clarifies hundreds of factors. Hundreds are still large, but anyway we succeeded to extract relevant factors.

This is still a model analysis, but this result will suggest that we will need to measure traffic thousands times to extract the relevant factors from real traffic phenomena, but real traffic will never repeat the same traffic thousands times. So we can use plausible computer simulation models to get better description of the real social phenomena.

14.4.3.3 Behavior Analysis of Taxi Drivers and Model Construction

Real-life urban traffic flows consist of different types of vehicles, each of which has its own properties of behaviors. In the project, taxi is our primary target.

We use probe-car data provided by a taxi company in Kyoto City for a 1-month period including approximately 700,000 location data points recorded per day. In general, the driver's behavior in VACANT state is driver-determined. That is, after dropping off passenger(s), a driver can freely select next destination and mode (queuing or cruising) for picking up the next passenger(s). In contrast,



Fig. 14.18 Visualization of origins for each cluster: red (high ratio) \Leftrightarrow blue (low ratio). (a) Cluster 1. (b) Cluster 4. (c) Uniform

the destination in OCCUPIED state is passenger-determined. Given this fact, we hypothesize that the tendency of the location that a taxi driver got passenger(s) would be one of the main characteristics of the driver. For considering that, we focused on the preferred location to pick up passenger(s) and try to classified drivers based on that.

To extract popular regions to get passenger(s), we apply kernel density analysis and the mean shift clustering algorithm which are widely used for extracting POI (Point of Interest) from the two-dimensional information provided by photo geotagging and GPS. From the results of Kernel density analysis, we can assume that the activity range defining the average driver's interest point is 50 s of longitude and latitude. We use these values as the bandwidth for the mean shift clustering algorithm so that we got 15 regions. We then calculated the ratio of picking up passenger(s) at each region. We use the ratios and the affinity propagation clustering algorithm to classify all taxi drivers. This yielded 10 clusters of driver type. Figure 14.18 shows the departure points for the clusters. These figures clearly indicate drivers tend to favor a specific territory for picking up passenger(s). For example, 112 drivers for Cluster 1 tend to operate on the north side of Kyoto City. On the other hand, 33 drivers Cluster 4 are apt to go to the south side.

In simulations each taxi agent has two states: OCCUPIED and VACANT. When the state changes from OCCUPIED to VACANT, taxi agents stochastically decide next destination and mode according to current time zone and current location. To analyze this decision-making process, we construct 10 driver models based on the clustering results shown in Fig. 14.18. Further, if the taxi picks up a passenger(s) and changes their state to OCCUPIED, the destination is given by the passenger agent. Each passenger agent has an OD matrix whose contents stochastically mirror the probe-car data. In this taxi model, the driver type is given by (1) ratio of queuing/cruising and (2) selection of area used in searching for passenger(s). We conduct traffic simulations in the city of Kyoto with constructed taxi agents. Figure 14.19 shows a visualization of agents whose model followed taxi driver cluster ID 1 or ID 4. ID 1 is a driver who tends to work in the northern part of Kyoto City, while ID 4 is in the south. The characteristic behavior model confirmed that there is a deviation in the preferred areas for each taxi driver class as expected;

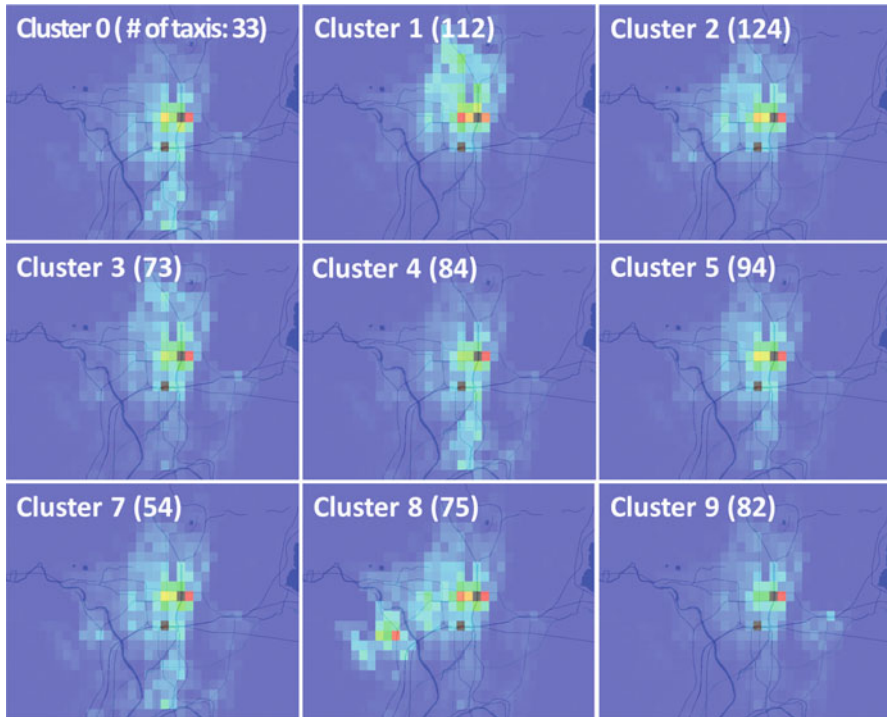


Fig. 14.19 Comparison of characteristic behavior model and uniform model

see Fig. 14.19a, b. Table 14.2 lists the ratios of staying time in each region. Comparing probe-car data, it seems that the tendencies of agents follow the clustering result.

14.5 Computational Road Maps of Social Simulations and Future

In this project, we also try to determine how HPC contributes to the advancement of research on social simulation or to clarify the computational power required for real applications of social simulation. In this section, we focus on three applications and try to develop road maps for them [13].

In the development of these road maps, we adopted two indexes to measure the computational cost, “number of situations” and “complexity of one simulation session.” We considered exhaustive evaluation by simulation as a key methodology of social simulation. Therefore, to evaluate the model, examining many conditions and models is important. The index of “number of situations” indicates this number.

Table 14.2 Ratio of staying time in each region

	Probe-data		Behavior model	
	Cluster1	Cluster4	Cluster1	Cluster4
Region3	0.263	0.056	0.217	0.095
Region4	0.016	0.196	0.054	0.165

Meanwhile, ordinal computational cost of a simulation, which is determined by the number of entities and the number of interactions among the entities, is important. In addition, in multiagent simulation, the computational cost of thinking of each agent is significant. In the following discussion, we integrate these complexities as “complexity of one simulation session.”

14.5.1 Evacuation/Pedestrian Simulation

The main target of evacuation simulation is not to find an optimal plan of evacuation for a given disaster situation, but to evaluate the feasibility and robustness of executable candidates of evacuation plans or guidance policies.

Several simulations have been performed for evaluating such evacuation plans [14, 25, 26, 29]. For example, a simulation of an evacuation from a Tsunami struck city in Tokai area in Japan was performed. We conducted the following exhaustive simulations considering various sizes and evacuation policies (evacuee’s origin-destination (OD) plans). The simulation results tell that the scale of evacuation can be grouped into two categories, namely, “large” (>3,000 evacuees) and “small” (<3,000 evacuees), and that citizens and local governments should consider at least two plans for large- and small-scale evacuations.

We execute the evacuation simulation described above to arrive at a reference point for illustrating computational costs of various actual applications. In the above simulation, we considered the following scenarios:

- 2,187 OD plans
- 8 cases of evacuation population (70–10,000 agents).

Therefore, in total, 17,497 simulation scenarios were executed over about 30 days when using a single process on Xeon E5 CPU (2.7 GHz). We denote this reference point as the rectangle “city zone, TSUNAMI” in Fig. 14.20.

We can easily extend the simulation scale. Although a population of only 10,000 is considered in “city zone, TSUNAMI,” we can extend the simulation to a more densely populated area such as in Tokyo. For example, we performed a similar simulation analysis in the Kanazawa area, which is located on the coast along the Japan Sea and experiences snowfall in the winter. In this case, the population size is similar (about 6,000 agents), but the number of combinations of scenarios increases to 4,194,304 (2^{22}). The rectangle “city zone, TSUNAMI and HEAVY SNOW” in Fig. 14.20 denotes this calculation cost.

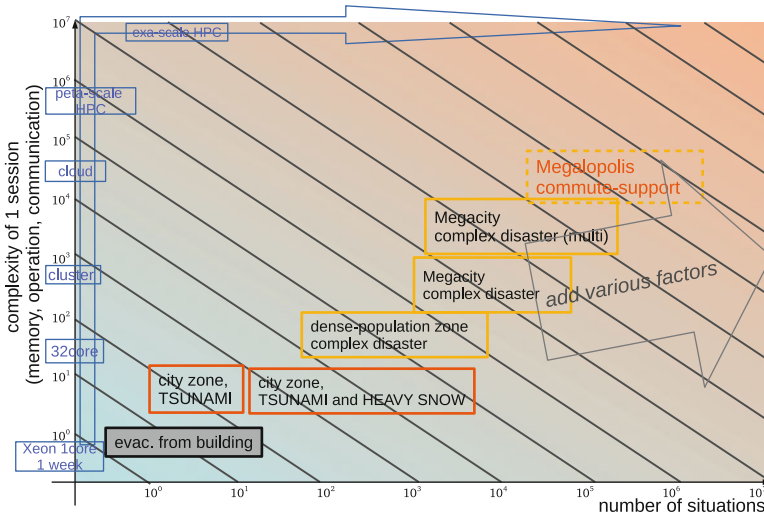


Fig. 14.20 Road map of evacuation simulation

We can further extend the simulation to a large scale with a larger number of scenarios. Kitasenju area, a large transfer station surrounded by rivers, has a population of 70,000, and the computational cost of simulating this area is denoted by “dense-population zone, complex disaster” in Fig. 14.20. Because this area is densely populated and complex, we have combinations of 44 policy candidates, that is, 2^{44} scenarios. In the case of Tokyo, we need additional computational power. In Fig. 14.20, “megacity” corresponds a huge city such as Tokyo. In this case, the size of evacuation and the number of possible scenarios is very large. Therefore, peta- or exa-scale HPC is required to handle such simulations.

14.5.2 Traffic Simulation

To create a reference point for the road map of the traffic simulation, we considered the case of evaluating road restriction policies for road construction in the Hiroshima area [16]. In this case, we performed simulations of the following scales:

- 70,000 agents (trips), 120,000 road links, and 15 h
- 20 cases

In this case, the calculation required about one day when using a single process on Xion E5 CPU. We denote this reference point as “million city, road plan” in Fig. 14.21.

We can draw out the road map from this reference point. When considering the Tokyo area, the number of agents increases up to about two million and the number

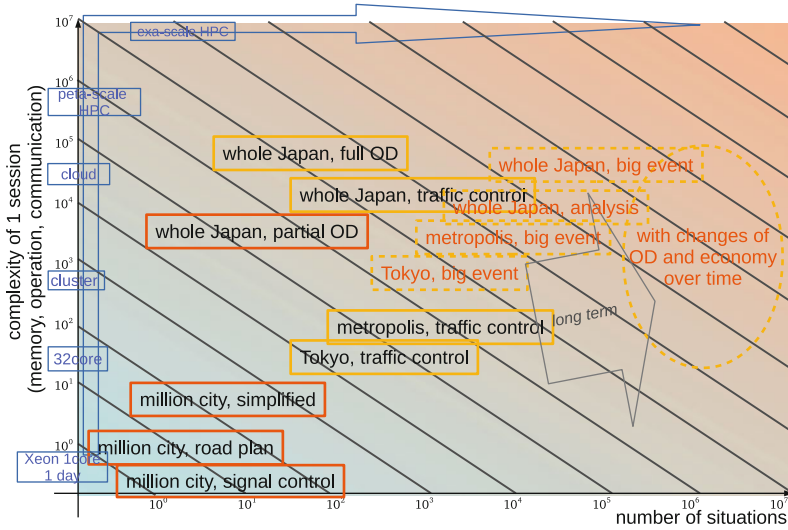


Fig. 14.21 Road map of traffic simulation

of road links increases to about 610,000. Moreover, if we consider a larger area such as the Tokyo metropolitan area, the population increases about four million, and the number of road links increases to 2.5 million. These calculation costs are plotted as “Tokyo, traffic control” and “metropolis, traffic control” in Fig. 14.21.

When we consider a big event, we must list a large number of cases to evaluate the robustness of road traffic to accidents, whereas the scenarios mentioned above pertain to normal situations that are repeated every day. Because various situations affect traffics, the number of situations increases quickly. These costs are plotted as “Tokyo, big event,” “metropolis, big event,” and “whole Japan, big event” in Fig. 14.21, and they require exa-scale computational power.

14.5.3 Market Simulation

Market simulations are another important application of multiagent simulations, in which agents directly affect each other by selling/buying stocks and/or currencies [5]. Compared with evacuation and traffic simulations, market simulations are not constrained by physical space. Therefore, the time cycles of agents’ interactions may be quite short. Moreover, the ways of thinking of agents show large variations. This means that the market simulations also require huge computational cost.

As the reference point of the calculation cost in market simulations, we present the case of “tic size” evaluation. In this scenario, we conducted a simulation of multiple markets having different tic sizes, which is the minimum price unit for trading stocks. Market companies such as Japan Exchange Group internationally

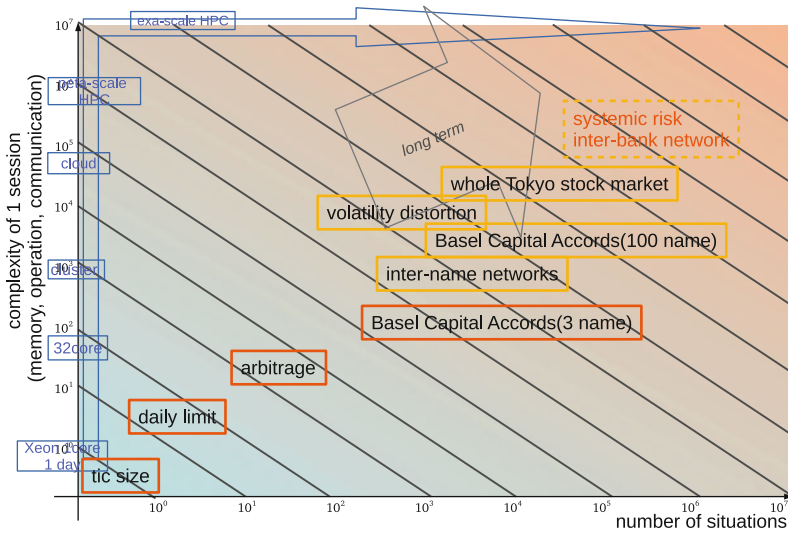


Fig. 14.22 Road map of market simulation

compete with each other by providing attractive services to traders. A small tic size is one of such services that considerably increases cost. Therefore, such organizations need evaluations of changes to such services in advance. In collaborative works with Japan Exchange Group, we conducted a simulation experiment to find key conditions that determine market share among markets. In the simulation, we considered the following scenario:

- one good in two markets, 1,000 agents, and 10 million cycles
- five cases of tic size and 100 simulation runs per case

This simulation takes about one day when using a single thread on a Xeon E5 CPU. As the reference point, we plot this as “tic size” in Fig. 14.22.

We are considering extending the market simulations to various applications used for stock market analyses. For example, it is in the interest of market companies to determine “daily limit” and “cut-off” prices [9]. In this case, the simulation must handle 10–20 goods. Moreover, evaluating the effects of “arbitrage” [5], which involves trading rather quickly in intervals of milliseconds, is important from the viewpoint of maintaining sound market conditions. This will increase the computational cost, as plotted in Fig. 14.22. Another topic is the evaluation of “Basel Capital Accords,” which deal with the soundness of banks in markets. In the present study, we executed the case of three names for the Basel Accords, but we will extend it to 100 names in the real application.

The evaluation of “systemic risks of inter-bank network” is an important issue in market evaluation. However, currently, the computational cost of a naive simulation exceeds exa-scale HPC.

14.6 Toward Smart Society

In the road map discussion, we count the number of scenarios naively based on the actual numbers used in our works in this project. We suppose that we simply apply exhaustive search on these scenarios. Of course, we can apply several methods based on design of experiments or other optimization/learning methods to reduce the number of scenarios we should run. OACIS and CARAVAN also provide facilities to realize such intelligent and effective functions.

We also need to investigate the cost of thinking part of each agent. In the evaluation above, we assume that the intelligence of each agent will not change, so that the complexity of the thinking in each agent is constant. But, for further simulation researches, we need to introduce more sophisticated and complex thinking engine to realize more intelligent and adaptive behaviors like human. This is still open issues.

The multiagent social simulation is an evolving research domain to realize smart societies by IT and AI and is still under establishing phase. However, requests from application fields become stronger and wider. So, it is important to determine a measure to know achievements will be important. The road maps shown in this article will become a testbed to provide such measures. Also, the CASSIA framework shown in this chapter will provide powerful tool to push forward the research on these road maps.

References

1. Asano, Y., Ito, N., Inaoka, H., Imai, T., Uchinane, T.: Traffic simulation of Kobe-City. In: Proceedings of the International Conference on Social Modeling and Simulation, Plus Econophysics Colloquium 2014, pp. 255–264. Springer, Cham (2014)
2. Charles, P., Grothoff, C., Saraswat, V., Donawa, C., Kielstra, A., Ebcioğlu, K., von Praun, C., Sarkar, V.: X10: an object-oriented approach to non-uniform cluster computing. In: Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA '05, pp. 519–538. ACM, New York (2005). <http://doi.acm.org/10.1145/1094811.1094852>
3. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In: Schoenauer, M., et al. (eds.) PPSN VI Proceedings of the 6th International Conference on Parallel Problem Solving from Nature. Lecture Notes in Computer Science, vol. 1917, pp. 849–858. Springer, Berlin/Heidelberg (2000). https://doi.org/10.1007/3-540-45356-3_83
4. Fujishima, D., Kamada, T., Torii, T., Izumi, K.: Overlapping communication and computation for large-scale artificial market simulation. In: Proceedings of 22nd International Symposium on Artificial Life and Robotics (AROB 2017), Beppu, pp. 708–713 (2017)
5. Kawakubo, S., Izumi, K., Yoshimura, S.: Analysis of an option market dynamics based on a heterogeneous agent model. *Int. J. Intell. Syst. Acc. Financ. Manag.* **21**, 105–128 (2014). <https://doi.org/10.1002/isaf.1353>
6. Kobayashi, K., Narisawa, R., Yasui, Y., Fujisawa, K.: Experimental analyses of the evacuation planning model using lexicographically quickest flow (in Japanese). *Trans. Oper. Res. Soc. Jpn.* **59**, 86–105 (2016). <https://doi.org/10.15807/torsj.59.86>

7. Matsushima, H., Noda, I.: Analysis of trade-off in evacuation plan using evolutionarily exhaustive simulation. In: Proceedings of the 23rd International Symposium on Artificial Life and Robotics (AROB 23rd 2018), pp. OS15–4. International Society of Artificial Life and Robotics (2018)
8. Mizuta, H.: Large-scale distributed agent-based simulation for shopping mall and performance improvement with shadow agent projection. In: Chan, W.K.V., D’Ambrogio, A., Zacharewicz, G., Mustafee, N., Wainer, G., Page, E. (eds.) Proceedings of the 2017 Winter Simulation Conference, pp. 3925–3936. Institute of Electrical and Electronics Engineers, Inc., Piscataway (2017)
9. Mizuta, T., Izumi, K., Yagi, I., Yoshimura, S.: Design of financial market regulations against large price fluctuations using by artificial market simulations. *J. Math. Financ.* **3**(2A), 15–22 (2013). <https://doi.org/10.4236/jmf.2013.32A003>
10. Mizuta, T., Kosugi, S., Kusumoto, T., Matsumoto, W., Izumi, K.: Effects of dark pools on financial markets’ efficiency and price discovery function: an investigation by multi-agent simulations. *Evol. Inst. Econ. Rev.* **12**(2), 375–394 (2015). <https://doi.org/10.1007/s40844-015-0020-3>
11. Mizuta, T., Kosugi, S., Kusumoto, T., Matsumoto, W., Izumi, K., Yagi, I., Yoshimura, S.: Effects of price regulations and dark pools on financial market stability: an investigation by multiagent simulations. *Intell. Syst. Account. Financ. Manag.* **23**(1–2), 97–120. <https://onlinelibrary.wiley.com/doi/abs/10.1002/isaf.1374>
12. Murase, Y., Uchitane, T., Ito, N.: A tool for parameter-space explorations. *Phys. Proc.* **57**, 73–76 (2014)
13. Noda, I., Ito, N., Izumi, K., Mizuta, H., Kamada, T., Hattori, H.: Roadmap and research issues of multiagent social simulation using high-performance computing. *J. Comput. Soc. Sci.* **1**(1), 155–166 (2018)
14. Nurdin, Y., Yuliana, D.K., Noda, I., Soeda, S., Yamashita, T.: Disaster evacuation simulation with multi-agent system approach using netmas for contingency planning (meulaboh case study). In: Proceedings of 5th Annual International Workshop & Expo on Sumatra Tsunami Disaster & Recovery 2010, AIWEST-2010 (2010)
15. Osogami, T., Imamichi, T., Mizuta, H., Suzumura, T., Ide, T.: Toward simulating entire cities with behavioral models of traffic. *IBM J. Res. Dev.* **57**(5), 6–1 (2013)
16. Osogami, T., Mizuta, H., Ide, T.: Identifying the optimal road closure with simulation. In: Proceedings of the 20th ITS World Congress Tokyo 2013 (2013)
17. Saraswat, V., Bloom, B., Peshansky, I., Tardieu, O., Grove, D.: X10 language specification version 2.6.1 (2017). <http://x10-lang.org/>
18. Takeuchi, M., Mizuta, H.: X10: performance and productivity at scale. In: Poster presentation at 20th Workshop on Programming and Programming Languages (2018)
19. Takeuchi, M., Makino, Y., Kawachiya, K., Horii, H., Suzumura, T., Suganuma, T., Onodera, T.: Compiling x10 to java. In: Proceedings of the 2011 ACM SIGPLAN X10 Workshop, X10 ’11, pp. 3:1–3:10. ACM, New York (2011). <http://doi.acm.org/10.1145/2212736.2212739>
20. Takeuchi, M., Cunningham, D., Grove, D., Saraswat, V.: Java interoperability in managed x10. In: Proceedings of the Third ACM SIGPLAN X10 Workshop, X10 ’13, pp. 39–46. ACM, New York (2013). <http://doi.acm.org/10.1145/2481268.2481278>
21. Torii, T., Izumi, K., Yamada, K.: Shock transfer by arbitrage trading: analysis using multi-asset artificial market. *Evol. Inst. Econ. Rev.* **12**(2), 395–412 (2016)
22. Torii, T., Kamada, T., Izumi, K., Yamada, K.: Platform design for large-scale artificial market simulation and preliminary evaluation on the k computer. *Artif. Life Robot.* **22**(3), 301–307 (2017). <https://doi.org/10.1007/s10015-017-0368-z>
23. Uchitane, T., Ito, N.: Applying factor analysis to describe urban scale vehicle traffic simulation results (in Japanese). *J. Soc. Instrum. Control Eng.* **52**(10), 545–554 (2016)
24. Yamashita, K., Kamada, T.: Introducing a multithread and multistage mechanism for the global load balancing library of x10. *J. Inf. Process.* **24**(2), 416–424 (2016). <https://doi.org/10.2197/ipsjip.24.416>

25. Yamashita, T., Soeda, S., Noda, I.: Evacuation planning assist system with network model-based pedestrian simulator. In: Yang, J.J., Yokoo, M., Takayuki Ito, Z.J., Scerri, P. (eds.) *Principles of Practice in Multi-agent Systems*. Proceedings of 12th International Conference, PRIMA 2009, pp. 649–656. Springer, Heidelberg (2009)
26. Yamashita, T., Soeda, S., Noda, I.: Assistance of evacuation planning with high-speed network model-based pedestrian simulator. In: *Proceedings of Fifth International Conference on Pedestrian and Evacuation Dynamics (PED 2010)*, p. 58 (2010)
27. Yamashita, T., Soeda, S., Onishi, M., Noda, I.: Development and application of high-speed evacuation simulator with one-dimensional pedestrian model. *J. Inf. Process. Soc. Jpn.* **53**(7), 1732–1744 (2012)
28. Yamashita, T., Okada, T., Noda, I.: Implementation of simulation environment for exhaustive analysis of huge-scale pedestrian flow. *SICE JCMSI* **6**(2), 137–146 (2013)
29. Yamashita, T., Matsushima, H., Noda, I.: Exhaustive analysis with a pedestrian simulation environment for assistant of evacuation planning. In: *Prof. of PED 2014*, pp. SE05–3 (2014). <https://doi.org/https://doi.org/10.1016/j.trpro.2014.09.047>
30. Yonenoh, H., Izumi, K.: Destabilization effect of var-based risk management on a multiple-asset market: an artificial market approach. In: *23rd International Symposium on Artificial Life and Robotics (AROB2018)* (2018)

Chapter 15

GPU-Accelerated Language and Communication Support by FPGA



Taisuke Boku, Toshihiro Hanawa, Hitoshi Murai, Masahiro Nakao,
Yohei Miki, Hideharu Amano, and Masayuki Umemura

Abstract Although the GPU is one of the most successfully used accelerating devices for HPC, there are several issues when it is used for large-scale parallel systems. To describe real applications on GPU-ready parallel systems, we need to combine different paradigms of programming such as CUDA/OpenCL, MPI, and OpenMP for advanced platforms. In the hardware configuration, inter-GPU communication through PCIe channel and support by CPU are required which causes large overhead to be a bottleneck of total parallel processing performance. In our project to be described in this chapter, we developed an FPGA-based platform to reduce the latency of inter-GPU communication and also a PGAS language for distributed-memory programming with accelerating devices such as GPU. Through this work, a new approach to compensate the hardware and software weakness of parallel GPU computing is provided. Moreover, FPGA technology for computation and communication acceleration is described upon astrophysical problem where GPU or CPU computation is not sufficient on performance.

T. Boku (✉) · M. Umemura
Center for Computational Sciences, University of Tsukuba, Tsukuba, Japan
e-mail: taisuke@cs.tsukuba.ac.jp; umemura@ccs.tsukuba.ac.jp

T. Hanawa · Y. Miki
Information Technology Center, The University of Tokyo, Tokyo, Japan
e-mail: hanawa@cc.u-tokyo.ac.jp; ymiki@cc.u-tokyo.ac.jp

H. Murai · M. Nakao
Center for Computational Science, RIKEN, Kobe, Japan
e-mail: h-murai@riken.jp; masahiro.nakao@riken.jp

H. Amano
Department of Information and Computer Science, Keio University, Tokyo, Japan
e-mail: hunga@am.ics.keio.ac.jp

15.1 Introduction

We started the project named “Accelerator and Communication Unification for Scientific Computing” where we utilize the FOG technology to realize a short latency communication between accelerators such as GPUs for strong scaling on accelerated parallel computing. Today’s GPUs such as NVIDIA CUDA devices are equipped with a feature for device-to-device direct memory access within a computation node. Our goal was to develop a special hardware technology as well as system software to make over-node direct communication among GPUs. This concept is named “TCA (Tightly Coupled Accelerators).” We also implemented a prototype system to realize this concept with external link of PCIe (PCI Express) to enable GPU-GPU direct memory access over nodes. We implemented it on an FPGA system named PEACH2 (PCI Express Adaptive Communication Hub ver.2).

While PEACH2 provides very short latency of communication among GPUs on different nodes, the system software stack to support application level coding is required. We developed an API library to drive PEACH2 in a similar style of GPU Direct access feature by NVIDIA to program this system based on CUDA-style coding where we can call GPU-GPU direct access instead of MPI communication over GPUs. However, this level of coding is still difficult for application users such as advanced computational scientists. To support them, we developed a new language named XcalableACC (XACC) for higher level coding in an incremental manner. In an implementation of XcalableACC, we developed a special version to support PEACH2 communication as well as ordinary MPI communication with InfiniBand.

Finally, we stepped into a new method to utilize FPGA for PEACH2 not only for PCIe base communication but also for sub-computation of the entire scientific algorithm. It is a brand-new challenge to apply FPGA both for communication and computation where a class of tightly coupled parallel computing can be implemented to partially off-load the computation to the function of internode communication. This concept is named “AiS (Accelerator in Switch).” We demonstrated this new feature on an astrophysics application on enhanced version of PEACH2.

In this chapter, we introduce PEACH2 technology at first for the realization of TCA concept and then briefly introduce the feature and implementation of XcalableACC. Finally, we describe the AiS implementation for an astrophysics code.

15.2 PEACH2

15.2.1 Realizing TCA Concept by PCIe

Recent GPUs such as NVIDIA CUDA devices are equipped with functions to apply DMA (Direct Memory Access) through PCIe where these devices are connected

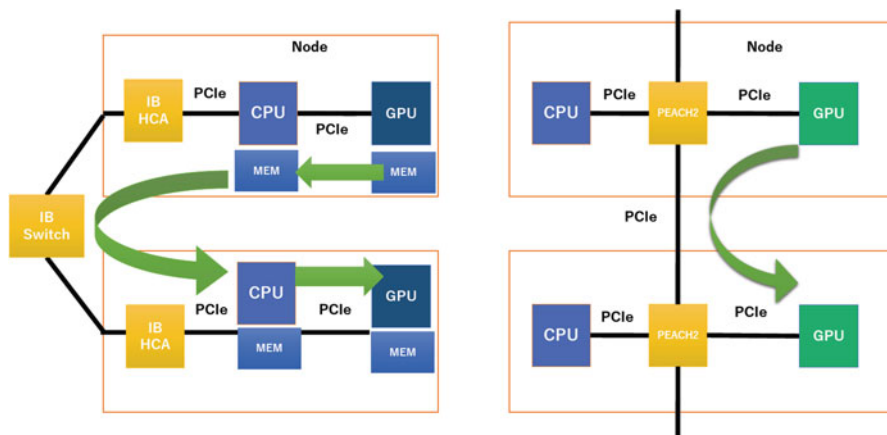


Fig. 15.1 TCA implementation by PCIe bus (left, ordinary method; right, TCA by PCIe)

with other devices including the host CPU. For example, it is available to directly access the global memory of GPU from InfiniBand HCA through PCIe where the technology is called GPU Direct.

On the other hand, PCIe is possible to extend its communication link not just on the motherboard of computation node but also to external link to connect it to another node's PCIe interface. Thus, it is theoretically possible to extend GPU Direct to another node. However, there is a problem of PCIe device for master/slave relationship. There is only one RootComplex that is allowed on PCIe bus, and all other devices must be in EndPoint mode. If we can solve this problem with some appropriate circuit with both sides of interface which is compatible with PCIe specification, we can use PCIe for interconnection among nodes where GPU Direct is possible to operate. Since all the communication is performed just within simple PCIe protocol, it is very fast with short latency. It is one of the simplest implementation of TCA (Fig. 15.1).

To realize above concept under TCA model, we implemented this feature to FPGA. Here, Altera Stratix IV FPGA was used as the latest technology at that time. This device is named as PEACH2 (PCI Express Adaptive Communication Hub ver.2).¹ A PEACH2 chip (FPGA) is equipped with four ports of PCIe gen2 x8 interfaces to be connected to host CPU or external link to other nodes. Figure 15.2 shows the block diagram of PEACH2. The port to the host CPU must be EndPoint of course, but other three ports which are configured as RootComplex, EndPoint, or the selection of them. The last port can be configured either RootComplex or EndPoint. Theoretically, we can make any combination including ring/torus network with the routing function inside PEACH2 chip.

¹Before we started this research, we had made another PCIe base communication. Then it is named as the second version.

Fig. 15.2 Block diagram of PEACH2 chip

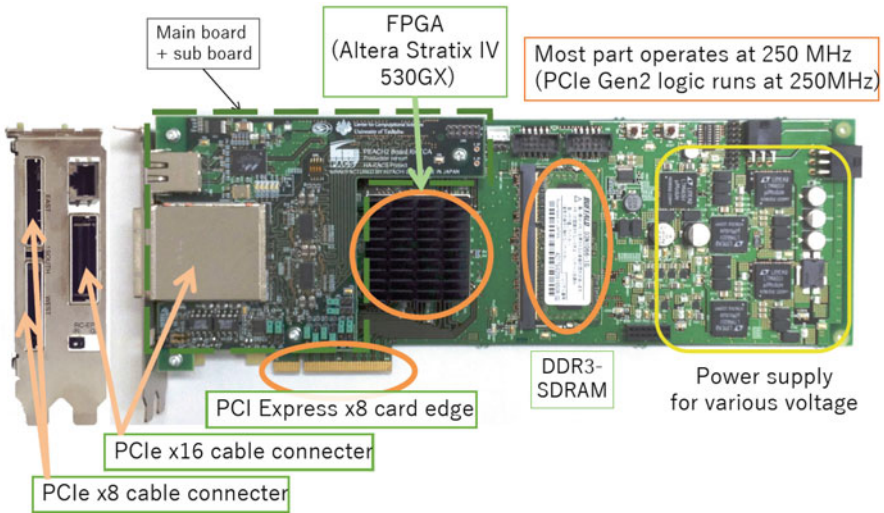
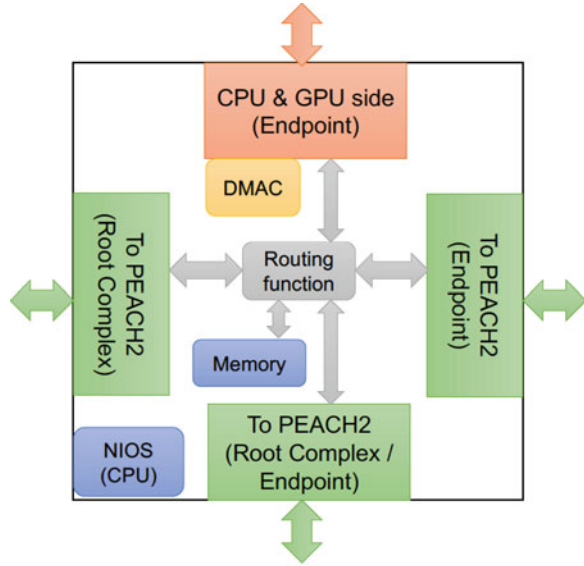
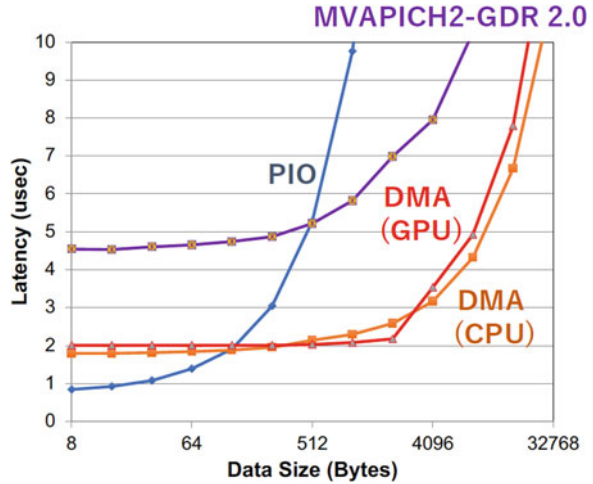


Fig. 15.3 PEACH2 board

The PEACH2 FPGA chip is mounted on an PCIe board to be inserted to the motherboard as like as ordinary PCIe devices. This board is called PEACH2 board (Fig. 15.3).

Fig. 15.4 PEACH2 latency

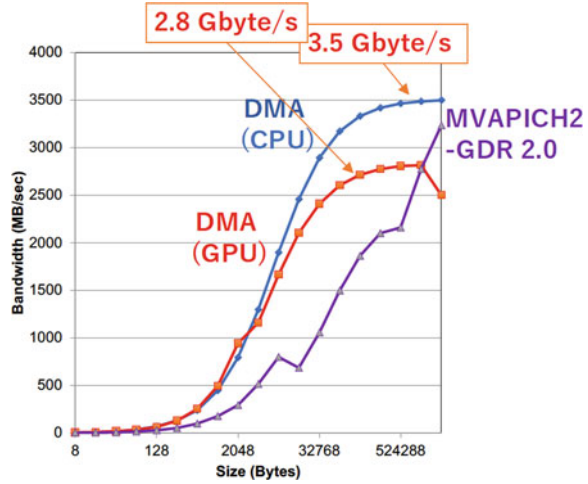


15.2.2 PEACH2 Performance

Since PEACH2 enables the simplest communication protocol on PCIe to connect multiple GPUs over multiple nodes, it can achieve very low latency in the communication for GPU-GPU remote memory copy. Figure 15.4 shows the point-to-point communication latency comparison between PEACH2 and MVAPICH2 on InfiniBand (QDR). “MVAPICH2-GDR 2.0” shows the latency of MVAPICH2 at that date, while three lines “PIO,” “DMA(GPU),” and “DMA(CPU)” show that of PEACH2. The actual use case of GPU-GPU communication is represented by “DMA(GPU),” and it shows 2.1 μ s of latency up to 2 KB of message. It is quite faster than MVAPICH2.

For the bandwidth, the situation is different. Since our PEACH2 implementation with Stratix IV allows to be interfaced by PCIe gen2 x8 lanes, its maximum bandwidth is up to 4 GB/s. On the other hand, InfiniBand QDR can be connected by PCIe gen3 x8 lanes where the maximum bandwidth reached to the double of PEACH2. Figure 15.5 shows the ping-pong bandwidth of PEACH2 and MVAPICH2 over InfiniBand(QDR). Since the latency of PEACH2 is much shorter than MVAPICH2, the bandwidth is higher than it for short messages; then MVAPICH2 performance overcomes PEACH2. It is caused by the physical performance difference on PCIe technology, but still we can demonstrate that PEACH2 provides higher performance when the message size is relatively short, and it is a better solution for strong scaling.

Fig. 15.5 PEACH2 bandwidth



15.2.3 Conclusion

The basic research for development of PEACH2 to realize TCA concept shows the possibility to reduce the communication latency between GPUs over multiple nodes. PEACH2 technology is based on PCIe external link extension which provides a very simple and flat communication protocol over remote GPU communication. The FPGA implementation is just a prototyping method for easy and cost-effective way, and we developed the PEACH2 chip only for the communication functionality with intelligent PCIe controlling. Since we could utilize PCIe gen2 technology on that date of FPGA (Altera Stratix IV), the absolute performance of following generations such as InfiniBand FDR or EDR overcame the performance of PEACH2 later.

After this basic research on PEACH2 implementation, we expanded its utilization to language level, introducing a new parallel language with PGAS model named XcalableACC. The programmability and productivity of the scientific code for large-scale parallel GPU clusters are enhanced by this research. We will describe it in the next section. Another new challenge was to utilize FPGA not only for communication but also for partial computation which is not suitable for GPU. It is a unique solution to speed up the application by FPGA to be unified computation with communication. This new concept is named as AiS (Accelerator in Switch). We will describe this feature and actual application on this concept in the following section.

15.3 XcalableACC: A Directive-Based Language for Accelerated Clusters

15.3.1 Introduction

A type of parallel computer that is composed of multiple nodes equipped with accelerator devices (e.g., Graphics Processing Unit (GPU)) has become a popular HPC platform. In fact, many supercomputers in the recent TOP500 lists are of this type. We call it *accelerated clusters*.

To program accelerated clusters, the combination of Message Passing Interface (MPI) for distributed-memory parallelism among nodes and a dedicated language or application programming interface for off-loading works to accelerator devices within a node (e.g., CUDA for NVIDIA's GPU and OpenACC [15]) is usually adopted. However such a style of programming is quite complicated and difficult for most of application programmers, and an easier way to program accelerated clusters is strongly demanded.

To meet this demand, some PGAS languages [3, 18] have already been extended to support accelerators. On the other hand, there have been other approaches based on C++ template library, such as Kokkos [4], RAJA [7], Alpaka [25], and Phalanx [5], for heterogeneous architectures including accelerated clusters.

In this project, we propose a new language named *XcalableACC* [17], which is a diagonal integration of two existing directive-based language extensions: XcalableMP and OpenACC.

XcalableMP (XMP) [24], developed by the XMP Specification Working Group of the PC Cluster Consortium, is a directive-based language extension for C and Fortran to program distributed-memory parallel computers. Using XMP, programmers can obtain parallel programs just by inserting simple directives into their serial programs.

OpenACC is another directive-based language extension designed to program heterogeneous CPU/accelerator systems. It targets off-loading works from a host CPU to attached accelerator devices and has an advantage of portability across operating systems and various types of host CPUs and accelerators.

XcalableACC (XACC) has features for handling distributed-memory parallelism, derived from XMP, and off-loading works to accelerators, derived from OpenACC, and two additional functions: direct communication between accelerators and data/work mapping among multiple accelerators. These two functions are the advantages of XACC against the previous works.

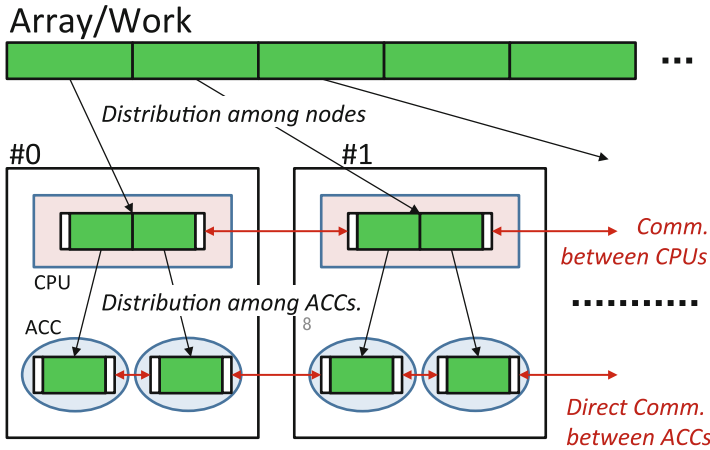


Fig. 15.6 Execution model of XACC for data distribution, off-loading, and communication

15.3.2 XcalableACC Language

XACC consists of three components: the XMP directives, the OpenACC directives, and the XACC extensions, which have the following functions, respectively.

- XMP directives for distributed-memory parallelism
- OpenACC directives for off-loading works to accelerator
- XACC extensions for handling direct communication between accelerators and multiple accelerators

15.3.2.1 Execution Model

Figure 15.6 shows the execution model of XACC for data distribution, off-loading, and communication. On this model, data or works are distributed onto nodes and then off-loaded onto accelerators within a node; communication of the data in accelerator memory might be performed via the direct interconnect between accelerators, if available.

An example code of XACC is given in Fig. 15.7.

15.3.2.2 XACC Extensions

The XACC extensions in the XACC language have specifically the following two functions:

- Direct communication between accelerators

```

1  #pragma xmp nodes p[*]
2  #pragma acc device d(*)
3
4  #pragma xmp template t[100]
5  #pragma xmp distribute t(block) onto p
6
7  float a[100][100];
8  #pragma xmp align a[i][*] with t[i]
9  #pragma xmp shadow a[1:1][0]
10
11 #pragma acc declare copy(a) layout([*][block]) \
12     shadow([0][1:1]) on_device(d)
13
14 #pragma xmp reflect (a) acc
15
16 #pragma xmp loop (i) on t[i]
17 for (int i = 0; i < 100; i++){
18 #pragma acc kernels loop layout(a[*][j]) on_device(d)
19     for (int j = 0; j < 100; j++){
20         a[i][j] = ...
21     }
22 }
23
24 ...

```

Fig. 15.7 Example code of XACC

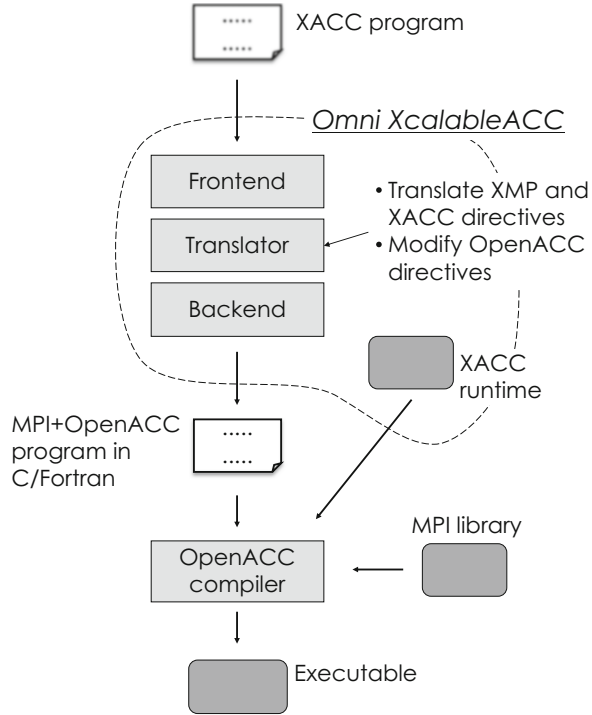
- XMP’s communication directives, such as `reflect`, `bcast`, and `gmove`, act on data that reside in the device memory when the `acc` clause is specified in them (line 14 in Fig. 15.7).
- Data in device memory can be also declared as *coarray*, which can be remotely accessed by other nodes.
- Data/work mapping onto multiple accelerators
 - Data and works are distributed among nodes by an XMP directive and further distributed among accelerators within each node by the additional `layout` clause of the `declare` and `loop` directives (lines 11 and 18 in Fig. 15.7).
 - The `on_device` clause can be put on some OpenACC directives (e.g., `declare` and `data`) to explicitly specify their target device (lines 12 and 18 in Fig. 15.7).

15.3.3 Omni XscalableACC Compiler

Omni XscalableACC is a compiler of XACC based on the Omni compiler infrastructure [14], which is being developed by RIKEN R-CCS and University of Tsukuba.

Omni XACC accepts an XACC source program and translates it into an MPI+OpenACC program, which is then compiled and linked with the XACC runtimes by the backend OpenACC compiler, such as PGI’s, to generate an exe-

Fig. 15.8 Omni XACC architecture



cutable (Fig. 15.8). Note that Omni has already supported OpenACC and therefore can work as the backend compiler for itself.

Omni XACC supports TCA-based direct communication between accelerators as well as that based on MVAPICH2-GDR [16], which is an implementation of MPI that takes advantage of the GPUDirect RDMA (GDR) technology [12]. In addition, Omni XACC is also able to concurrently utilize a standard interconnect between CPUs, such as Infiniband, and a dedicated direct interconnect between accelerators to make the most of the interconnect throughput of the system [13].

15.3.4 Case Study: Lattice QCD Mini-application

15.3.4.1 Implementation

We evaluate performance and productivity of XACC through an implementation of a Lattice Quantum Chromo-Dynamics (QCD) mini-application which is one of the most important applications in the HPC field. Figure 15.9 shows the declarations of distributed arrays on the accelerator memory. Note that these arrays have shadow regions for halo exchange. Figure 15.10 shows how to exchange halo regions between adjacent nodes. $WD()$ in line 5 is the Wilson-Dirac operator [23], which is the main kernel of this mini-application. Since $WD()$ requires halo exchange,

```

1  Gluon_t U[4][NT][NZ][NY][NX];
2  Quark_t X[NT][NZ][NY][NX];
3  #pragma xmp template t[NT][NZ]
4  #pragma xmp nodes n[PT][PZ]
5  #pragma xmp distribute t[block][block] onto n
6  #pragma xmp align [*][i][j][*][*] with t[i][j] shadow[0][1][1][0][0] :: U
7  #pragma xmp align [i][j][*][*] with t[i][j] shadow[1][1][0][0] :: X
8  #pragma acc enter data copyin(U, X)

```

Fig. 15.9 Declaration of distributed arrays

```

1  #pragma xmp reflect_init(U) width(0,/periodic/1:0, ...) orthogonal
2  #pragma xmp reflect_init(X, ...) width(/periodic/1, ...) orthogonal
3  :
4  #pragma xmp reflect_do(U, X) acc
5  WD(..., U, X);

```

Fig. 15.10 Halo exchange and calling Wilson-Dirac operator

the **reflect_do** directive performs halo exchange based on information set by the **reflect_init** directive.

15.3.4.2 Performance Evaluation

We evaluate the performance of the Lattice QCD mini-application in XACC on HA-PACS/TCA. The communication mechanism between GPUs of Omni XACC is based on “hybrid” communication via TCA having low latency and Infiniband having high bandwidth, which allows communication among sub-clusters of HA-PACS/TCA. For a comparison purpose, we also evaluate it in the combination of CUDA and MPI (CUDA+MPI) and the combination of OpenACC and MPI (OpenACC+MPI). We assign a single process with a single compute node, and we use up to 64 compute nodes. The problem size is $(NT, NZ, NY, NX) = (16, 16, 16, 16)$ in Fig. 15.9, and we measure performance in strong scaling.

Figure 15.11 shows performance results of the implementations, where the performance in XACC is the best at the high degree of parallelism. The performance of XACC is up to 9% better than that of CUDA+MPI and up to 18% better than that of OpenACC+MPI.

15.3.4.3 Productivity Evaluation

We evaluate the productivity of each the implementation using Delta Source Lines of Code (DSLOC), which is one of evaluation criterions for productivity [19]. DSLOC is a value to count the amount of changes (add, delete, and modify) required to implement a parallel Lattice QCD code from a sequential Lattice QCD code. When DSLOC is small, the programming costs and the possibility of program bugs will be small as well. Table 15.1 shows DSLOC in each implementation, where XACC is

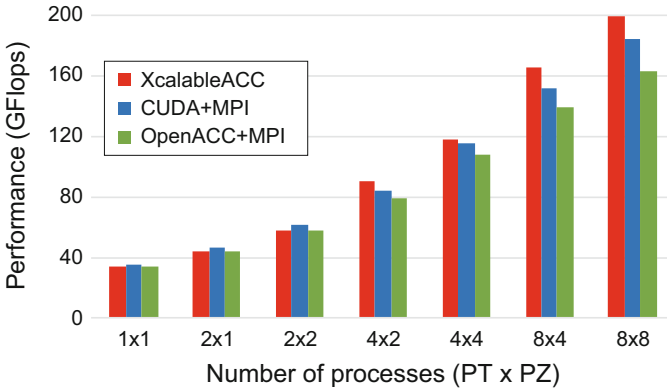


Fig. 15.11 Performance results of lattice QCD mini-application

Table 15.1 Delta source code of lines in each implementation

	XcalableACC	CUDA+MPI	OpenACC+MPI
Total	86	767	219
Add	80	348	173
Delete	0	73	0
Modify	6	346	46

the smallest. DSLOC of XACC is 89% less than that of CUDA+MPI and 61% less than that of OpenACC+MPI.

15.3.5 Summary

We proposed XcalableACC that is a directive-based language extension for accelerated clusters and developed a compiler for it. It is basically an integration of XcalableMP and OpenACC and has advanced features of direct communication between accelerators and data/work mapping onto multiple accelerators. The case study for a Lattice QCD mini-application showed that XACC would be useful in both performance and productivity to program accelerated clusters.

15.4 Applying Accelerator in Switch for Astrophysics

15.4.1 Introduction

Simulations of gravitating collisionless particles, say N -body simulations, are a fundamental tool in astrophysics. We have developed a gravitational octree code

on GPU that adopts a block time step. Parallelization of the code is a mandatory procedure to run N -body simulations with a large number of N -body particles that cannot be stored in the memory of single GPU. Warren and Salmon proposed an algorithm named Locally Essential Tree (LET) for the parallel tree code. Adopting the LET reduces the communication between processes by paying an additional cost to generate subtracted tree structure for all other processes. Accelerator in Switch (AiS) is a framework to accelerate pre-/post-processes of communications and provide better parallel efficiency. We have implemented LET generator on PEACH3, which is a switching hub with Altera's FPGA (Field Programmable Gate Array) board, as a test bed for AiS in actual simulations. The LET generator on PEACH3 is always faster than that on GPU and achieves 4.5 times acceleration. Performance optimization on PEACH3 such as adopting lower accuracy of floating point operations than single precision would provide further acceleration.

15.4.2 Development of Gravitational Octree Code Accelerated by Block Time Step

Simulations of gravitating collisionless particles, say N -body simulations, are a fundamental tool in astrophysics. In order to perform N -body simulations in realistic elapsed time with a large number of N -body particles that are sufficient to resolve astrophysical phenomena, the tree method [2] is frequently employed to accelerate simulations through reducing the amount of force calculations. In most astrophysical phenomena, the mass density and dynamical timescale are not uniform and have difference by more than an order of magnitude. Therefore, block time stepping (sometimes called hierarchical time stepping) is more effective to accelerate N -body simulations than the shared time stepping [1, 8]. We have developed a Gravitational Oct-Tree code Accelerated by Hierarchical Time step Controlling, named GOTHIC, which adopts both the tree method and the block time step [10]. The code is optimized for GPUs and adopts adaptive optimizations by monitoring the execution time of each function on-the-fly and minimizes the time-to-solution by balancing the measured time of multiple functions.

The decrease in the number of steps having long execution time is attributed to the acceleration by the block time step. In the case of the block time step, execution time in some steps is smaller than shared time step, because the number of activated N -body particles is reduced by order of magnitude. Figure 15.12 shows the execution time of tree traverse on NVIDIA Tesla K20X with CUDA 7.5. Out of the first 201 steps, the number of steps having execution time above 1 s is 26 owing to the reduction of force calculations on slowly moving N -body particles; this is the main reason for the acceleration by the block time step. The achieved mean execution time per step is 0.21 s, and the contributions from the steps with long execution time, which is $1.2 \text{ s} \times 26/201 = 0.16 \text{ s}$, are dominant.

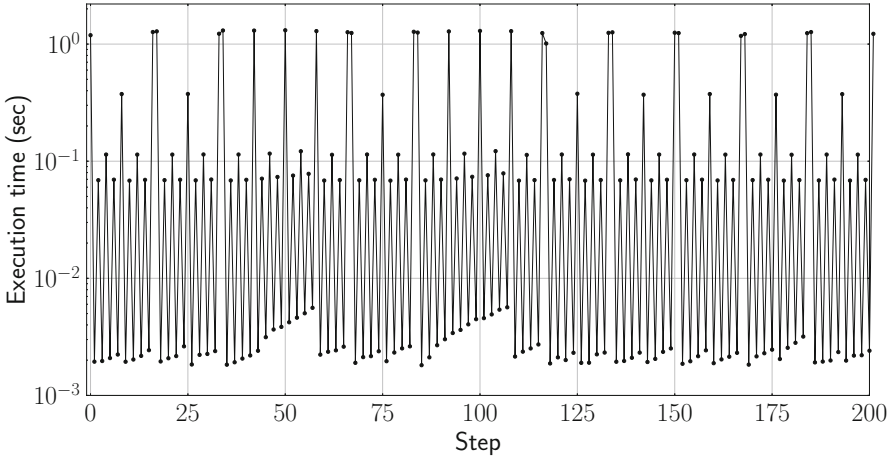


Fig. 15.12 Execution time of gravitational force calculation on NVIDIA Tesla K20X as a function of the time step. The particle distribution is a model reproducing the Andromeda galaxy with $2^{22} = 4,194,304$ particles generated by MAGI [11]

15.4.3 Parallelization of the Code and Barrier for Scalability

Parallelization of the code is a mandatory procedure to run N -body simulations with a large number of N -body particles that cannot be stored in the memory of single GPU ($N \sim 10$ M or $N \sim 30$ M are the upper limit for GOTHIC on NVIDIA Tesla K20X or NVIDIA Tesla P100, respectively). Warren and Salmon proposed an algorithm named Locally Essential Tree (hereafter, LET) to reduce the amount of the communication among processes for the parallel tree code [22]. When one applies the domain decomposition to the tree code, particle distribution in other domains is necessary to calculate gravitational force. However, a subtracted tree provides sufficient data to calculate gravitational force properly, since the detailed information in the distant regions is not required for the tree method. The LET contains the data that is necessary to calculate the gravitational force on every N -body particle in a local domain pulled by particles in other domains. Adopting the LET reduces the communication between processes by paying an additional cost to generate subtracted tree structure for all other processes. The difficulty in achieving the scalability comes from the collision with aspects of block time step and computational cost for LET. The parallel efficiency decreases when the execution time of LET-related operations exceeds or is comparable to that of gravitational force calculations. As shown by Fig. 15.12, the execution time of force calculations has various ranges: ~ 1.2 s, ~ 0.4 s, ~ 0.1 s, and $\sim 2 \times 10^{-3}$ s. Let us consider a case of the LET-related operation takes ~ 0.5 s, for example. The operation becomes the dominant procedure in most time steps except for the steps with execution times above 1 s, and therefore the parallel efficiency would become worse. Since the execution time of the shared time step is corresponding to the

longest execution time of the block time step, a condition to achieve good parallel efficiency for the block time step is more severe than that for the shared time step.

15.4.4 Accelerator in Switch

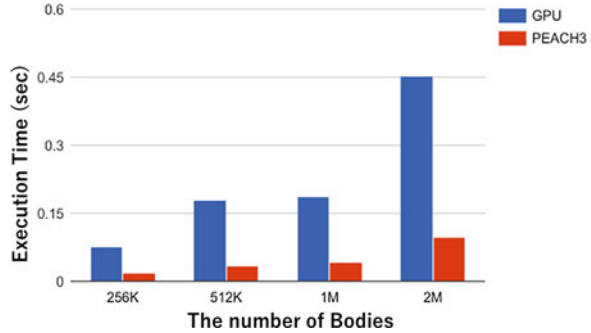
Accelerator in Switch (AiS) is a framework to accelerate pre-/post-processes of communications and provide better parallel efficiency [21]. Communications among multiple processes are sometimes tightly coupled with related computations. In the case of the LET, the communications require subtraction of local tree just before sending data to other processes. Moreover, some of the recent switching hubs for high-performance networks are equipped with high-end FPGAs [6, 9, 26]. Such FPGAs in switching hubs now become a candidate for an accelerator device for pre-/post-processes of communications among multiple processes by exploiting redundant logic elements in FPGAs. We adopt PEACH3, which is a PCIe gen3 switch developed for tightly coupled accelerators, as a test bed for AiS in actual simulations. PEACH3 is implemented on Stratix V GX EP5SGXA7N3F45C2, an FPGA board by Altera, and possesses 622K logic elements with 512MB DDR3 SDRAM.

15.4.5 Development of LET Generator in PEACH3

The LET generator is one of the attractive applications suitable for AiS. LET construction is implemented as tree traverse of a single imaginary particle representing the particle distributions in a distant domain and decimating tree nodes which are not required for gravitational force calculation by the target process. GPU is not a very suitable candidate for accelerator device for the LET generator, because its high performance mainly comes from massive parallelization utilizing its many-core architecture. FPGA can handle the LET generation and makes GPUs concentrate on gravitational force calculation by releasing them from the burden on unsuitable tasks.

We have implemented LET generator on PEACH3 using Quartus II ver.16.1 Standard Edition. The LET module is redesigned for AiS and has an ability to handle large data set based on the implementation by [20]. Figure 15.13 shows the execution time of LET generators on GPU (NVIDIA Tesla K40 with CUDA 6.5) and PEACH3, including communication between two GPUs. The LET generator on PEACH3 is always faster than that on GPU and achieves 4.5 times acceleration as shown in Fig. 15.13. The low latency communication of PEACH3 is also responsible for the acceleration. The observed acceleration of LET generator and communication help GOTHIC to achieve a good parallel efficiency. Performance optimization on PEACH3 such as adopting lower accuracy of floating point operations than single precision would provide further acceleration. Since the LET generator does not

Fig. 15.13 Execution time of LET generator including communication as a function of the number of N -body particles. Blue and red bars show the results for LET generator on GPU and PEACH3, respectively



require accurate floating point operations and recent GPUs provide only limited half-precision floating point operations such as fused multiplication and addition of small matrices, the acceleration by lower precision operations is another potentiality for FPGA-specific performance optimization.

References

1. Aarseth, S.J.: Dynamical evolution of clusters of galaxies, I. *Mon. Not. R. Astron. Soc.* **126**, 223 (1963). <https://doi.org/10.1093/mnras/126.3.223>
2. Barnes, J., Hut, P.: A hierarchical $O(N \log N)$ force-calculation algorithm. *Nature* **324**, 446–449 (1986). <https://doi.org/10.1038/324446a0>
3. Cunningham, D., et al.: GPU programming in a high level language: compiling X10 to CUDA. In: Proceedings of the 2011 ACM SIGPLAN X10 workshop (X10 '11), New York (2011)
4. Edwards, H.C., Trott, C.R.: Kokkos: enabling performance portability across manycore architectures. In: Proceedings of the 2013 extreme scaling workshop (XSW 2013), pp. 18–24, Aug 2013
5. Garland, M., Kudlur, M., Zheng, Y.: Designing a unified programming model for heterogeneous machines. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '12, pp. 67:1–67:11, Los Alamitos (2012)
6. Hanawa, T., Kodama, Y., Boku, T., Sato, M.: Interconnect for tightly coupled accelerators architecture. In: IEEE 21st Annual Symposium on High-Performance Interconnects (HOT Interconnects 21) (2013)
7. Hornung, R.D., Keasler, J.A.: The RAJA portability layer: overview and status. Technical Report LLNLTR-661403, LLNL (2014)
8. McMillan, S.L.W.: The vectorization of small- N integrators. In: Hut, P., McMillan, S.L.W. (eds.) *The Use of Supercomputers in Stellar Dynamics*. Lecture Notes in Physics, vol. 267, p. 156. Springer, Berlin (1986). <https://doi.org/10.1007/BFb0116406>
9. Mellanox Fabric Collective Accelerator. <http://www.mellanox.com/>
10. Miki, Y., Umemura, M.: GOTHIC: gravitational oct-tree code accelerated by hierarchical time step controlling. *New Astron.* **52**, 65–81 (2017). <https://doi.org/10.1016/j.newast.2016.10.007>
11. Miki, Y., Umemura, M.: MAGI: many-component galaxy initializer. *Mon. Not. R. Astron. Soc.* **475**, 2269–2281 (2018). <https://doi.org/10.1093/mnras/stx3327>
12. NVIDIA Corporation: NVIDIA GPUDirect (2014). <https://developer.nvidia.com/gpudirect>

13. Odajima, T., et al.: Hybrid communication with TCA and infiniband on a parallel programming language XcalableACC for GPU clusters. In: Proceedings of the 2015 IEEE International Conference on Cluster Computing, pp. 627–634, Sept 2015
14. Omni Compiler Project: Omni compiler project (2018). <http://omni-compiler.org/>
15. OpenACC-Standard.org: The OpenACC application programming interface version 2.0 (2013). http://www.openacc.org/sites/default/files/OpenACC.2.0a_1.pdf
16. Potluri, S., Hamidouche, K., Venkatesh, A., Bureddy, D., Panda, D.K.: Efficient inter-node MPI communication using GPUDirect RDMA for infiniband clusters with NVIDIA GPUs. In: Proceedings of the International Conference on Parallel Processing, pp. 80–89 (2013)
17. RIKEN AICS and University of Tsukuba: XcalableACC language specification version 1.0 (2017). <http://xcalablemp.org/download/XACC/xacc-spec-1.0.pdf>
18. Sidelnik, A., et al.: Performance portability with the Chapel language. In: Proceedings of the IEEE 26th International Parallel and Distributed Processing Symposium, pp. 582–594 (2012)
19. Stone, A.I., et al.: Evaluating coarray fortran with the cpop miniapp. In: Proceedings of the Fifth Conference on Partitioned Global Address Space Programming Models (PGAS), Oct 2011.
20. Tsuruta, C., Miki, Y., Kuhara, T., Amano, H., Umemura, M.: Off-loading LET generation to PEACH2: a switching hub for high performance GPU clusters. In: ACM SIGARCH Computer Architecture News – HEART15, vol. 43, pp. 3–8. ACM, New York (2016). <http://doi.acm.org/10.1145/2927964.2927966>
21. Tsuruta, C., Kaneda, K., Nishikawa, N., Amano, H.: Accelerator-in-switch: a framework for tightly coupled switching hub and an accelerator with FPGA. In: 27th International Conference on Field Programmable Logic & Application (FPL2017) (2017)
22. Warren, M.S., Salmon, J.K.: Astrophysical N-body simulations using hierarchical tree data structures. In: Proceedings of the 1992 ACM/IEEE Conference on Supercomputing, pp. 570–576. IEEE Computer Society Press (1992)
23. Wilson, K.G.: Confinement of quarks. *Phys. Rev. D* **10**, 2445–2459 (1974)
24. XcalableMP Specification Working Group: XcalableMP specification version 1.2 (2013). <http://www.xcalablemp.org/download/spec/xmp-spec-1.2.pdf>
25. Zenker, E., Worpitz, B., Widera, R., Huebl, A., Juckeland, G., Knpfer, A., Nagel, W.E., Bussmann, M.: Alpaka – an abstraction library for parallel Kernel acceleration. In: Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 631–640, May 2016
26. Zilberman, N., Audzevich, Y., Kalogeridou, G., Bojan, N.M., Zhang, J., Moore, A.W.: NetFPGA – rapid prototyping of high bandwidth devices in open source. In: 25th International Conference on Field Programmable Logic and Applications (FPL) (2015)