Study on Parallel Computing

Guo-Liang Chen[1] (陈国良), Guang-Zhong Sun[1] (孙广中), Yun-Quan Zhang[2] (张云泉), and Ze-Yao Mo[3] (莫则尧)

[1] *Anhui Province-MOST Key Co-Lab of High Performance Computing and Its Applications*
Department of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, P.R. China

[2] *Laboratory of Parallel Computing, Institute of Software, Chinese Academy of Sciences, Beijing 100080, P.R. China*

[3] *Laboratory of Computational Physics, Institute of Applied Physics and Computational Mathematics*
Beijing 100088, P.R. China

E-mail: {glchen,gzsun}@ustc.edu.cn; zyq@mail.rdcps.ac.cn; zeyao_mo@mail.iapcm.ac.cn

Received April 18, 2006; revised June 14, 2006.

Abstract In this paper, we present a general survey on parallel computing. The main contents include parallel computer system which is the hardware platform of parallel computing, parallel algorithm which is the theoretical base of parallel computing, parallel programming which is the software support of parallel computing. After that, we also introduce some parallel applications and enabling technologies. We argue that parallel computing research should form an integrated methodology of "architecture — algorithm — programming — application". Only in this way, parallel computing research becomes continuous development and more realistic.

Keywords parallel computing, parallel architecture, parallel programming, parallel algorithm, parallel application

1 Introduction

For a given application problem, first, the computational scientists have to convert it into a numerical computation problem, second the computer scientists have to design a parallel algorithm and then use some programming languages to execute that parallel algorithm on a concrete parallel computer, finally, the application experts run software to solve the given problem. The above whole process can be called parallel computing. From this, parallel computing deals with the following subject areas: the parallel computer system which is the hardware platform of parallel computing, the parallel algorithm which is a theoretical base of parallel computing, the parallel programming which is the software support of parallel computing, the parallel application which is the motive force of development in parallel computing. Obviously, parallel computing involves computational mathematicians, computer scientists, software engineers and applied domain experts. The main feature of parallel computing is related to applied domain closely.

It is here necessary to distinguish the following several professional terms in order to discuss conveniently.

Numerical computation method which focus on researching the mathematic principle of solving application problem dealing with the computation method convergence, stability and precision etc; **parallel algorithm** which is a concrete method and procedure to solve a given problem on a parallel computer. Usually, we can design a numerical parallel algorithm according to the corresponding numerical computation method; **parallel program** which is a segment code of implementing the parallel algorithm using some parallel programming languages. It can be run directly on a parallel computer; **parallel software** which is a set of parallel programs to solve a given application problem. Software should be reusable across different platforms. For this, we should use a reusable technicality to encapsulate some algorithms and technique in a reusable fashion.

When the parallel computing was in its infancy, we began to study on parallel computing. On the basis of parallel algorithm that we studied before, we extended little by little the scope of researching to study simultaneously parallel algorithm, parallel architecture, parallel programming and parallel application. Over the past 20 years, we have established a complete subject system of "Theory — Design — Implementation — Application" for parallel algorithm and formed an integrated methodology of "Parallel Computer — Parallel Algorithm — Parallel Programming" for parallel computing. We have cultivated many students in parallel algorithm and parallel computing areas for our country and published separately a series of parallel algorithms and parallel computing monographs and textbooks in Chinese[1−7].

In the following, the hardware platform of parallel computing (parallel computer systems) is given in Section 2, theoretical base of parallel computing (parallel algorithm) in Section 3, software support of parallel computing (parallel programming) in Section 4, parallel applications and enabling technologies in Section 5, and a brief conclusion in the last.

2 Hardware Platform of Parallel Computing: Parallel Computer Systems

With the rapid development of the past several decades, there are various parallel computer architectures. Based on the parallel computers' produce cost and usage, there are two series of parallel computers: common parallel computers and supercomputers[5].

Survey

Supported by the National Natural Science Foundation of China under Grant No.60533020.

2.1 Common Parallel Computer Architecture

Common Parallel Computers are normally cheap and convenient to get. So they are wildly used in many areas which need relatively low computing capability. There are three types of common parallel computers based on the architecture. The first one is the so called desktop multiprocessors, which is a new kind of personal computer with 2 or 4 CPUs. It can achieve higher performance than normal PC. The second type is SMP (Symmetric Multiprocessor) servers, which can easily be bought from computer markets. It has several symmetrical processors on a single main board. Usually an SMP server may contain 8, 16 or even 32 processors. Its architecture is shared memory and tightly coupled. The last one is COW (Cluster of Workstation) or PC cluster, it is a distributed memory, loosely coupled architecture, as the development of commercial networks and commercial processors, this architecture becomes more and more popular[7].

2.2 Supercomputer Architecture

Compared with the common parallel computers, the supercomputers have high performance but corresponding high cost and long producing time. The main architecture of supercomputers includes MPP, PVP and DSM and so on. MPP (Massively Parallel Processor) is usually made up of a large number of commodity processors which are connected by a customized high bandwidth and low latency communication network. The processors have physically distributed memory and they are synchronized through blocking message-passing operations. PVP (Parallel Vector Processor) is another kind of architecture of the supercomputers. It is made up of a few of specific vector processors, which are connected by a customized high bandwidth and low latency communication network. So PVP can achieve extraordinary performance for some specific applications. On the other hand, the PVP supercomputers are much more expensive than those based on MPP. DSM (Distributed Shared Memory) has physically distributed, but system hardware and software support a single address space to application users. In DSM, DIR (Cache directory) is used to support distributed coherent caches. Some commercial DSM machines with large amount of processors are often called constellation.

China has made the world-noted achievements on the development of supercomputers. Besides the United States and Japan, China has become the third country which is able to independently develop and produce high-performance computers. Now, several supercomputer series such as Dawning, Galaxy, Shenwei, Lenovo have made remarkable progresses. A supercomputer from Lenovo ranked the 14th place among the top 500 supercomputers in Nov. 2003 and Dawning4000A ranked the 10th in June 2004.

3 Theoretical Base of Parallel Computing: Parallel Algorithms

Generally speaking, algorithm is a method and steps to solve a given problem. Parallel algorithm is designed to execute in parallel on parallel computers. Parallel algorithms can be classified in different ways[6]: for example, based on the executing fashion, there are synchronous algorithms and asynchronous algorithms. Based on the executing operations, we can classify parallel algorithms in numerical algorithms and non-numerical algorithms. Numerical algorithms mainly solve the problems based on algebraic operation, such as matrix multiplication, computing the result of polynomial, solving linear equations, etc. Non-numerical algorithms mainly solve the problems based on comparison relation, which include sorting, selecting, searching, matching and so on. There are many distinguished achievements on parallel algorithms during the past several decades[8].

3.1 Parallel Computational Models

Usually, a computational model consists of several measurable parameters that reflect the computational characteristics of target architecture. Based on the computational behavior defined by the model (such as synchronous vs. asynchronous, etc.), usually a cost function can be used to analyze algorithm complexity. Algorithm designer and program developer can use such a model to design algorithms, to calculate cost function and to assess the running time of parallel algorithm.

Based on the historical development of parallel computational models, we think they can be classified into three generations. The first generation is the shared memory parallel computational model, started from 1978. The second generation is the distributed memory parallel computational model, started from 1984. The third generation is the hierarchical memory parallel computational model mainly started from 1993.

First Generation Parallel Computation Model. PRAM (Parallel Random Access Memory) was first built up by Fortune, Wyllie[9] and Goldschlager[10] independently. The PRAM model consists of a collection of RAM processors, and they load/store data through a common central memory. Concurrent access to the common memory is allowed and only takes one unit of time to be finished. The RAM processors can execute instruction concurrently in unit time and in lock-step with automatic and free synchronization. Through hiding the architectural details of communication, memory hierarchy and process synchronization, the PRAM model can simplify the design and analysis of parallel algorithms, but sometimes may lead to misleading analysis results. Thus, many researches have been incorporated various features of current parallel machines into PRAM model to enhance its reflectivity. These features and corresponding models are asynchrony PRAM[11], queuing shared memory[12] and latency PRAM[13] etc.

Second Generation Parallel Computation Model.

There are several models that assuming a distributed memory paradigm and processor communicates through message passing, we will introduce BSP-like and LogP-like models as follows.

The BSP (Bulk Synchronous Parallel) model was proposed by L. Valiant in 1990[14]. The parameters of this model include p (the number of processors), g (unary packet transmission time) and L (Synchronous periodicity). The algorithm executed on BSP model consists of a sequence of supersteps with periodicity L. In each superstep, each processor can perform the combination of local computation on local available data and message passing. After each period of L unit times, a global check is performed to ensure all processors have finished one superstep, otherwise, another superstep allocated for the unfinished superstep. The BSP model posits bandwidth limitation on the algorithm through limiting the maximum messages that can be sent/received in each superstep. The synchronization is charged at most L unit times. The advantages of BSP model are its separated consideration on computation and communication, easy programming and correctness guarantees of algorithm using superstep. But the length of superstep is affected by h-relations ($h = L/g$).

The development of LogP model was based on the observation of the convergence of parallel computer architecture in the 90s. It was proposed by D. Culler *et al.* in 1993[15]. LogP consists of four parameters, as the four characters in its name: L (latency of message passing), o (overhead of processor involved in message preparation and processing), g (maximum time gap between successive messages, its reciprocal is essentially the bandwidth of the communication) and P (the number of processors). The LogP model gives detailed descriptions of the bottleneck of parallel computation and hides the topology of interconnection network, but it cannot be used to analyze algorithm easily.

Third Generation Parallel Computation Model. Since the speed gap between processor and memory systems becomes larger and larger, the cost on memory access becomes non-ignorable. The following distributed memory models which consider incorporating the memory hierarchy into the analysis of parallel algorithms.

Two early models, HMM (Hierarchical Memory Model) and Hierarchical Memory Model with Block Transfer (BT), were proposed by Alok Aggarwal *et al.*[16,17] The UMH (Uniform Memory Hierarchy) model[18] has different assumption on memory location at each level, the memory access cost for each level is the same, i.e., the cost function in UMH is the function of level number, not data address. Parallel versions of these models are easy to be built through replicating the serial model p times. P-UMH model was proposed in [18] and P-HMM and P-BT models were proposed in [19]. The LogP-HMM model[20] extends an existing parameterized network model (LogP) with HMM model characterizing each processor. It captures both network communication costs and the effects of multilevel memory, such as local cache and I/O.

DRAM (h, k) (Distributed Random Access Memory)[21,22] is a computational model with considerations on instruction/thread level parallelism (k ways) and memory hierarchy (h levels) based on memory access complexity concept[23]. In Memory LogP[24] model, the communication cost is divided into memory communication cost and network communication cost. All the above models ignores TLB (Translation Look-aside Buffer) and disk cost.

3.2 Design and Implementation of Parallel Algorithms

Policy. There are three general policies to design a parallel algorithm. One is parallelizing a sequential algorithm. In this way, the designer needs to detect and exploit any inherent parallelism in an existing sequential algorithm. Another is designing a new parallel algorithm without regard to the related sequential algorithms. In terms of the description of a given problem, we redesign or invent a new parallel algorithm. The last design policy is borrowing other well-known algorithms to solve the given problem. This is necessary to find relationship between to be solved problem and well-know problem. Then a similar algorithm that solves a given problem using a well-know algorithm can be presented[4].

Method. There are several of tricks and methods in parallel algorithm design. The most frequently used methods are *partitioning*, *divide and conquer*, *pipelining* and so on. *Partitioning* is a simple but quite important method. It breaks up the given problem into several non-overlapping sub-problems of almost equal sizes and solves these sub-problems concurrently. *Divide and conquer* first divides the problem into several sub-problems, then solves recursively the sub-problems, at last merges solutions of sub-problems into a solution for original problem. *Pipelining* is also a useful method in parallel algorithm. It breaks an algorithm into a sequence of segments in which the output of each segment is the input of its successor. In this way, all segments must produce results at the same rate. For numerical computation problem, the iterative methods are often used in design of parallel algorithms[4,6].

Implementation. When we want to implement parallel algorithms on parallel computers, some problems such as task decomposition, assignment, mapping, scheduling may be encountered.

Only decomposing a big problem into several sub-problems properly can we implement effectively. There are two types of decomposition: domain decomposition and functional decomposition. The former is also called data decomposition and the latter is also called computation decomposition. The decomposition should make the computing tasks balanced and reduce the communication between the related tasks.

After the decomposition step, we need to assign the sub-problems on proper processors and decide their execution sequence. This is a scheduling problem. It plays an important role in optimizing the algorithm performance of implementation. There are three types of scheduling: static scheduling or compile time scheduling, it decides all the tasks' executing processor and sequence before they operate; dynamic scheduling does not know the information of the tasks, so it can only decide a task's executing processor and sequence when it available; hybrid scheduling is a combination of the above two scheduling, it schedules parts of tasks at the compile time, the rest tasks are scheduled dynamically at the running time.

Scheduling can also be divided as optimal scheduling and sub-optimal scheduling or heuristic scheduling. The optimal scheduling algorithm can get an optimal scheduling result in a polynomial time. Unfortunately, most optimal scheduling algorithms can only be used on very simple and particular circumstances. Therefore, many works have been done on designing efficient heuristic scheduling algorithms for various scheduling models. Although these algorithms cannot get the optimal scheduling result, they can get relatively good result in a low computing complexion[25−28].

3.3 Performance Evaluation of Parallel Algorithm

There are various methods and metrics that are used to measure the performance of a certain parallel algorithm. No single method or metric is preferred over another since each of them reflects certain properties of the parallel algorithm.

Speedup. It is defined as the ratio of the execution time on a single processor to that on a parallel computer. Amdahl's Law[29] states that if f is the fraction of a sequential calculation, then the maximum speedup that can be achieved is $1/f$. Since 1967, this law has been used as a negative argument against massively parallel processing. In fact, it is not necessary to fix workload and let algorithm run on different numbers of processors as Amdahl said. In large scientific computation application, in order to increase accuracy, we have to increase workload. Correspondingly, we have to increase the number of processors to keep the execution time unchanged. In 1987, Gustafson gave such a formulation which is often referred to as the Gustafson's Law[30] and has been widely refereed to as a "scaled speedup measure". Since 1988, Gustafson's Law has been used to justify massively parallel processing system.

Efficiency. The efficiency of a parallel algorithm describes the fraction of the time that is usefully employed by the processors for a given computation. It is defined as s_p/p, where s_p is speedup and p is the number of processors. Usually efficiency is between 0 and 1.

Scalability. A computer system (hardware, software, algorithms etc.) is scalable, if it can scale up (increase its resources) to accommodate performance and functionality demand and scale down (decrease its resources) to reduce cost.

There are different dimensions of scalability. Scalability in machine size indicates how well the performance will improve with additional processors. This scalability measures the maximum number of processors a system can accommodate. Scalability in problem size shows how well the system can handle larger problems with larger data size and workload. And technology scalability describes how well the performance improvement with the changed technology. Iso-efficiency[31], Iso-speed[32] and average latency[33] are three metrics proposed for the study of scalability.

4 Software Support of Parallel Computing: Parallel Programming

With the rapid development of parallel computing in the past several decades, there are various parallel programming models, languages and benchmarks. They construct the software support of parallel computing[34].

4.1 Parallel Programming Models

Parallel programming model is an abstraction between hardware and software for programmers. There are several parallel programming models in common use: shared variable, message passing, and data parallel, etc.[35]

Shared Variable. The shared variable programming is the native model for PVP, SMP and DSM machines. In this model, tasks share a common address space, where tasks exchange data through reading and writing shared variables. Various mechanisms such as locks/semaphores may be used to control access to the shared variable. An advantage of this model from the programmer's point of view is that the notion of data "ownership" is lacking, so there is no need to specify explicitly the communication of data between tasks. Program development can often be simplified, but the portability is problematic.

Message Passing. The message passing programming is the native model for MPP, COW machines. This model consists of a set of tasks that use their own local memory during computation. Tasks exchange data through communications by sending and receiving messages. Data transfer usually requires cooperative operations to be performed by each process. The special synchronous operations (barrier, event etc.) are used. The programmer is responsible for determining all parallelism.

Data Parallel. The data parallel programming is the native model for SIMD and VP machines. It originates from vector programming. Under the data parallel model, most of the parallel work focuses on performing operations on a data set. The data set is typically organized into a common structure, such as an array or cube. A set of tasks work collectively on the same data structure, however, each task works on a different partition of the same data structure[35]. Users do not have to specify communication operations.

Unified Model. Unified parallel programming model is suitable for various shared memory or distributed memory. It is a multi-layer model. The core support layer includes GOOMPI[36] (Generic Object Oriented MPI) and PMT (Parallel Multi-Thread). The core application layer centers on smart parallel and distributed abstract data structures and implementation of a highly reusable basic parallel algorithm library. At the high-level framework layer, it provides extensible parallel algorithmic skeletons and supports the research and design of new parallel algorithms. This model has advantage to support intuitively mapping from algorithm to program.

4.2 Parallel Programming Languages and Parallelization Methodologies

Designing and developing parallel programs has ever been a very manual process. Usually, there are three ways to develop parallel programs, i.e., auto-parallelization compilers, library extension and new parallel programming languages.

Auto-Parallelization Compilers. A parallelizing compiler generally works in two different ways: 1) fully automatic: the compiler analyzes the source code and identifies opportunities for parallelism. Loops (do, for) are the most frequent target for automatic parallelization; 2) programmer directed: using compiler directives or possibly compiler flags, the programmer explicitly tells the compiler how to parallelize the code and distribute the data. But researchers finally found out that automatic parallelization compiler is not effective. The way of adding compiler directives as HPF[37] became an acceptable way for some applications with regular data access (i.e., data parallel model).

Library Extensions of Sequential Programming Languages. Providing a user callable parallel programming library embedded in a sequential programming languages is an easy and popular way to realize parallel programming. MPI and OpenMP are two such popular library extensions. MPI[38] is a message passing library standard based on the consensus of the MPI Forum. Its goal is to establish a portable, efficient, and flexible standard for message passing programs. MPI is now the "de facto" industry standard for message passing. MPICH[39] and LAM/MPI[40] are two popular and freely available MPI packages. OpenMP is an Application Program Interface (API) that may be used to explicitly direct multi-threaded, shared memory parallelism. It comprises of three primary API components: compiler directives, runtime library routines and environment variables.

New Parallel Programming Languages. Parallel programming paradigms, over the past decade, have focused on how to exploit more computational power of contemporary parallel machines. Ease of use and code development productivity have been a secondary goal. Partitioned Global Address Space (PGAS) programming model aimed at leveraging the ease of programming of the shared memory paradigm, while enabling the exploitation of data locality. Co-array Fortran (CAF)[41] and UPC[42] are two emerging languages for single-program, multiple-data global address space programming. There are also several other new parallel languages developed for HPCS project.

4.3 Parallel Benchmarks

There are several freely available and popular benchmarks that are commonly used by the HPC community. The current trend on benchmark design is to develop application-oriented benchmarks. Such benchmarks, as MM5[43], WRF[44], will play more and more important roles in the future.

NPB. NPB (NAS Parallel Benchmarks)[45] is a small set of programs designed to help evaluate the performance of supercomputers. They were developed by the NASA Ames Research Center. NPB has been widely used because it mirrors real-world parallel scientific applications better than most other available benchmarks. The NPB suite of programs provides a comprehensive view of the various performance characteristics of HPC systems. The suite consists of five kernels that mimic the computational core of numeric methods used by CFD (Computational Fluid Dynamics) applications and three CFD pseudo-applications. NPB results are provided in MOPS (Millions of Operations Per Second). For each kernel, the NAS benchmarks specify five classes of increasing workloads, called W, A, B, C and D.

Linpack. Linpack[46] is widely used today for characterizing HPC systems. This benchmark uses a set of functions to measure the time taken to solve the double precision (64 bits) system of linear equations using Gaussian elimination method. Its results are expressed as MFLOPS (Millions of FLoating-point Operations Per Second), and used by TOP500 and China TOP100[47] as one rank metric. One of the most popular packages of Linpack benchmarking is High-Performance Linpack (HPL)[48].

HPCC. The current HPCC (HPC Challenge) benchmark measures the performance of several elements of a machine. It consists of a set of 23 measurements in eight groups. It supplements and extends Linpack, exercises critical features of an HPC machine, calculation speed, memory access, MPI communication and application kernels[49].

5 Parallel Applications and Enabling Technologies

This section surveys the various applications in numerical simulations for parallel computing especially highlighting the advanced technologies to enable the parallel computing into realities. Note that only some snapshots of typical applications are discussed here, we encourage the reader to explore the citations to get more detail.

5.1 Computational Fluid Dynamics (CFD)

The most traditional application is the field of computational fluid dynamics[50]. The motion of a fluid is governed by the well-known Navier-Stokes equations. On a series of discrete zones called by mesh covering the computational domain, these equations are solved by the computational methods of finite difference, finite volume and finite elements[51]. Methods for mesh generation[52], solvers for the sparse nonlinear or linear systems of equations[53,54], and programming techniques for the managements of software complexities are the most basically enabling techniques for the success of these computational methods. With respect to the requirements of higher resolution and higher fidelity for numerical simulations, terascale parallel computing is an essential path.

Computation Method of CFD. Once parallel computing is applied to the CFD, the traditional computational methods can be directly translated with little efforts provided that efficient parallel algorithms are designed and are implemented for their enabling techniques. Firstly, the mesh generated in advance should be partitioned into subdomains distributed to each processor and suitable information should be maintained for processor's boundaries, or the mesh is generated in parallel synchronizing the numerical simulation. Nevertheless, workloads should be balanced for each processor not only in the beginning but also in the proceeding of parallel simulation[55]. Secondly, robust and scalable solvers should be designed towards solving the sparse system of equations with thousands of millions of unknowns arising from the implicit discretization of partial differential equations governing fluids. Thirdly, further software techniques should be considered for data structure complexities and programming for parallel implementations on thousands of processors and parallel visualization on terascale data set.

Partitioning Methods. The data dependencies among zones of a mesh and the workload of each zone can usually be accurately depicted by an undirected graph where each node represents a zone and each edge represents two zones connected depending on each other. The multilevel strategy is most efficient to partition a mesh for hundreds or thousands of processors[56]. It can often partition a graph into subgraphs with a small number of cut-edges and uniformed workloads for each subgraph. However, this method always introduces larger communications for load redistribution especially in the case of parallel adaptive computing where mesh is dynamically refined or coarsened towards capturing the physical interests locally[57]. Partitioning methods based on space filling curves[58] are good substitutions for less distribution overheads. Therefore, one can use the graph partitioning method in the beginning of a parallel simulation for good load distribution, but should use a cheaper partitioning method such as multilevel averaging weighting method[59,60] or space filling curves method[61] to dynamically adjust the workloads during the proceeding of parallel simulation.

Parallel Solver. After the mesh is partitioned, the well-known BSP model can be used to describe the data flow of parallel implementations of the computational methods of CFD[62]. Therefore, parallel programming techniques mentioned in previous section should be taken into consideration towards running the code more efficiently. Global synchronizations should be suppressed especially while using hundreds or thousands of processors.

Jacobi-free Newton method[63] is often the most robust and efficient parallel solver for the nonlinear sparse system of partial differential equations provided that a suitable solution is given approximately before iterations. However, the approximate solution resulted from the last time step are usually efficient for the unsteady fluids. Preconditioned Krylov subspace iterative solver[54] is the most suitable solver for the linear system resulted from the Newton linearization though its global reductions for inner products of vector are really challenges for hundreds or thousands of processors. Suitable decision of preconditioner is very important for the convergence of Krylov iterations. Usually, BILU[54] is efficient for small numbers of unknowns and processors. Multigrid methods and their variations[64] are essential for larger numbers of unknowns and processors especially for the radiation driven hydrodynamics[65]. Hypre is the state-of-the-art library of high performance preconditioners[66].

Reusable Software. The reuse of parallel software is also very important for the development of CFD in a long term. PETSc[67], UG[68] and SAMRAI[69] are good examples to show the software frameworks for the simplifications of the parallel programming and codes development on single level mesh, adaptive unstructured mesh, and adaptive structured mesh respectively.

5.2 Particle Transportation

The field of particle transport is another type of application requiring parallel computing[70]. Different from the field of fluid dynamics, it concerns with the discretization in seven dimensions such as the time, space and velocity, etc. So, it requires more challenging computations than CFD. Both the Monte Carlo method and the discrete coordinate method are efficient for solution of such equations. The parallel implementation for the Monte Carlo simulation is trivial. However, the parallel implementations for the discrete coordinate methods are not easy because neighboring zones depend on each other along the direction of particle flux swept. In the case of deformed structured grid or unstructured grid, this directed data dependencies will challenge the parallel algorithms. In fact, such dependencies can be accurately depicted by the DAG (Directed Acyclic Graph). So, the parallel algorithm based on the DAG is also suitable for the parallel implementations of discrete coordinate method. Many works address such problems[71,72], work in [73] generalizes these ideas to be suitable for more general cases.

5.3 Other Applications

Environment and Energy. The numerical simulations in many other fields are also typically based on the discrete mesh. These fields include environment and energy, ocean modeling, earthquakes, cosmological structure formation, radiation astrophysics, electromagnetics, and so on. They require the capabilities of tens of TFLOPS for parallel computers. The book[74] surveys the important developments in these fields. The essential enabling techniques for these fields are also similar as that for the CFD and particle transport.

Chemistry. The computational chemistry has a long history and spans a broad range of computational methods. NWChem[75] is the well-known parallel software including many of these methods such as Hartree-Fock or self-consistent field, density functional theory, molecular dynamics, perturbation theory, coupled theory, and so on. However, the enabling techniques for these methods are different from that mentioned above because they should support irregular data references globally. The parallel computing model of BSP or DAG mentioned above is no longer useful. Fortunately, Global Array (GA) programming model[76] supports such data references well. NWChem is a successful application using GA.

Biology. The discipline of biology is one of the state-of-art applications requiring parallel computing. Many of the new enabling techniques are underdeveloped. Part of these work uses molecular dynamics method having the similar communicational characteristics as stated in NWChem. We do not discuss these applications here because, currently, such applications are typically the data intensive applications which have no special requirements on communications with respect to the traditional applications.

With the rapid developments of parallel computers, parallel computing becomes more and more successful in the numerical simulations using thousands of processors. The excellent results on applications and enabling techniques are showed in the proceedings of supercomputing conference in each year[77].

6 Conclusions

This paper provides a broad briefly look at the state of parallel computing in hardware platform, theoretical base, software support and some parallel applications. We argue that parallel computing research should form an integrated methodology of "architecture — algorithm — programming — application". Only in this way, parallel computing research becomes continuous development and more realistic. In the following, we briefly anticipate to the future of parallel computing.

Parallel Computer. Various parallel computer architectures will be coexisted. However, the future computational hardware platform will consist of multiple parallel systems with different performance, size and OS etc. The main difficulty of the heterogeneous hardware environments will increase complexity for both system software and application software[78]. Since the high performance parallel computer has thousands of processors. The issues of fault tolerance[79] and energy-consume[80] must be considered for the usage of the real systems. The development of modern processor technologies and many applications on the many-core architecture are also studied by many researchers[81,82].

Parallel Algorithm. The research on parallel algorithms will be emphasized more realistic and focused on solving problems from the applied domain. The theoretical work on parallel algorithm will be complemented by extensive experimentation which can guide us how to build parallel computer, how to make parallel algorithm more efficient in practice, how to model parallel computer more accurately and how to express parallel algorithm more intuitive using parallel programming languages. Commodity PC with multiple processors is more and more common on the market. We can expect the use of parallel algorithm to increase dramatically[8].

Parallel Programming. The parallel programmer requires to provide a unified parallel programming model, language and tools that can be suitable for various distributed and shared memory parallel architectures, can easily express the complex algorithm, can hide implementation details, can support fast and intuitive mapping from parallel algorithm to parallel program, can help programmers to debug and correct errors to optimize performance of application software etc.[83]

Parallel Software. It is very important to make application software reusable portable and scalable during developing software. Software should be reusable across different platforms. Software should be scalable as the number of processors increased providing significant performance maintaining reasonable efficiency. We should say that good commercial software is rare at the high end. The scientific community is generally conservative, community acceptance is essential to the success of software. Software has always followed hardware in the past years. However, this relationship will be reversed moving toward software-centric in parallel computing[74].

New Applications. Many socially relevant applications will increasingly share the market space with traditional computation applications. These new applications deal with massive data processing. Data, rather than computation, will be transformational elements. These data will be of varying format, type, quality and overwhelming to currently available I/O systems[74].

Non-Traditional Computation Modes. Neuro-computing, nature inspired computing, molecule computing and quantum computing etc. are with inherent and massive parallelism. It will be expected to solve some intractable problems using traditional computation model.

In summary, there are some serious problems in the parallel computing. Firstly, the development of high performance hardware is faster than software. It is very difficult for researchers of parallel algorithms and software to use thousands upon thousands of processors in the parallel computer. Secondly, high performance application is still weak. Most of the applications of high performance computing are in low level. High-end parallel software and killer application are still rare. At last, the education of high performance computing in China is not widespread. Many graduates of some universities know little about parallel computing. With the development of high performance computing in China, the gap between the need of professional and the lack of current education will be much greater. So right now, it is very necessary to provide the course of parallel computing for students with major of computation science in more universities.

Acknowledgments We would like to thank the anonymous referees for their valuable suggestions and comments to improve the presentation of this paper.

References

[1] Chen G. Parallel Algorithm of Sorting and Selection. University of Science and Technology of China Press, 1990.

[2] Chen G, Chen L. Computational Theory and Parallel Algorithms of VLSI. Univ. Science and Technology of China Press, 1991.

[3] Tang C *et al.* Parallel Graph Algorithm. University of Science and Technology of China Press, 1991.

[4] Chen G. Parallel Computing — Architecture, Algorithm, Programming. 2nd Edition, Higher Education Press, 2003.

[5] Chen G, Wu J *et al.* Parallel Computer Architectures. Higher Education Press, 2002.

[6] Chen G. Design and Analysis of Parallel Algorithms. 2nd Edition, Higher Education Press, 2002.

[7] Chen G, An H *et al.* Parallel Algorithms Practice. Higher Education Press, 2003.

[8] Blelloch G E, Maggs B M. Parallel algorithms. *ACM Computing Surveys,* 1996, 28(1): 51–54.

[9] Fortune S, Wyllie J C. Parallelism in random access machines. In *Conference Record of the 10th Annual ACM Symp. Theory of Computing*, San Diego, California, 1978, pp.114–118.

[10] Goldschlager L M. A universial interconnection pattern for parallel computers. *J. the ACM*, 1982, 29(4): 1073–1086.

[11] Cole R, Zajicek O. APRAM: Incorporating asynchrony into the PRAM model. In *Proc. 1st Annual ACM Symp. Parallel Algorithms and Architectures*, Santa Fe, New Mexico, 1989, pp.158–168.

[12] Gibbons P, Matias Y, Ramachandran V. The QRQW PRAM: Accounting for contention in parallel algorithms. In *Proc. the SPAA'94*, Cape May, New Jersey, 1994, pp.638–648.

[13] Aggarwal A, Chandra A, Snir M. On communication latencies in PRAM computations. In *Proc. SPAA'89*, Santa Fe, New Mexico, 1989, pp.11–21.

[14] Valiant L. A bridging model for parallel computation. *Communications of the ACM,* 1990, 33: 103–111.

[15] Culler D, Karp R, Patterson D *et al.* LogP: Towards a realistic model of parallel computation. In *Proc. ASPLOS IV*, New York, 1993, pp.1–12.

[16] Aggarwal A, ALpern B, Chandra A, Snir M. A model for hierarchical memory. In *Proc. the 19th Annual ACM Symp. Theory of Computing*, Chicago, Illinois, USA, 1987, pp.305–314.

[17] Aggarwal A, ALpern B, Chandra A, Snir M. Hierarchical memory with block transfer. In *Proc. of the 28th Annual IEEE Symp. Foundations of Computer Science*, Los Angeles, CA, 1987, pp.204–216.

[18] Alpern B, Carter L, Feig E, Selker T. The uniform memory hierarchy model of computation. *Algorithmica*, 1993.

[19] Vitter J, Shriver E. Algorithms for parallel memory II: Hierarchical multilevel memories. *Technical Reports*, CS–1993–02, Department of Computer Science, Duke University, 1993.

[20] Li Z, Mills P H, Reif J H. Models and resource metrics for parallel and distributed computation. In *the 28th Int. Conf. System Sciences* (*HICSS'95*), Hawaii, USA, 1995, pp.51–61.

[21] Zhang Y. Performance optimizations on parallel numerical software package and study on memory complexity [Dissertation]. Institute of Software, CAS, 2000.

[22] Zhang Y. DRAM(h): A parallel computation model for high performance numerical computing. *Chinese Journal of Computers*, 2003, 12(26): 1660–1670.

[23] Zhang Y, Sun J, Tang Z, Chi X. Memory complexity in high performance computing. In *Proc. the 3rd Int. Conf. High Performance Computing in Asia-Pacific Region*, Singapore, 1998, pp.142–151.

[24] Cameron K, Sun X H. Quantifying locality effect in data access delay: Memory log P. In *Proc. the 2003 IEEE Int. Parallel and Distributed Processing Symp.*, Nice, France, 2003, pp.212–219.

[25] Gerasoulis A, Yang T. On the granularity and clustering of directed acyclic task graphs. *IEEE Trans. Parallel and Distributed Systems*, 1993, 4(6): 686–701.

[26] Shirazi B A, Hurson A, Kavi K. Scheduling and Load Balancing in Parallel and Distributed Systems. IEEE Computer Science Press, 1995.

[27] Kwok Y, Ahmed I. Dynamic critical-path scheduling: An effective technique for allocating task graph to multiprocessors. *IEEE Trans. Parallel and Distributed Systems*, 1996, 7: 506–521.

[28] Topcuoglu H, Hariri S, Min-You W. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel and Distributed Systems*, 2002, 13(3): 260–274.

[29] Amdahl G M. Validity of the single-processor approach to achieving large scale computing capabilities. In *AFIPS Conference Proc.*, Atlantic City, New Jersey, 1967, pp.483–485.

[30] Gustafson J L. Revaluating Amdahl's law. *Communications of the ACM*, 1987, 31: 532–533.

[31] Grama A Y, Gupta A, Kumar V. Isoefficiency: Measuring the scalability of parallel algorithms and architectures. *IEEE Parallel and Distributed Technology*, 1993: 1(3), 12–21.

[32] Sun X, Rover D. Scalability of parallel algorithm-machine combinations. *IEEE Trans. Parallel and Distributed System*, 1994, 5(6): 599–613.

[33] Zhang X, Yan Y, He K. Latency metric: An experimental method for measuring and evaluating parallel program and architecture scalability. *Journal of Parallel and Distributed Computing*, 1994, 22(3): 392–410.

[34] Quinn M J. Parallel Programming in C with MPI and OpenMP. McGraw Hill, 2004.

[35] http://www.llnl.gov/computing/tutorials/parallel_comp/

[36] Yao Z, Zheng Q, Chen G. GOOMPI: A generic object oriented message passing interface. In *Proc. NPC*, 2004, pp.261–271.

[37] http://www.vcpc.univie.ac.at/information/mirror/HPFF/.

[38] http://www-unix.mcs.anl.gov/mpi/.

[39] http://www-unix.mcs.anl.gov/mpi/mpich/.

[40] http://www.lam-mpi.org/.

[41] http://www.co-array.org/.

[42] http://upc.lbl.gov/.

[43] http://www.mmm.ucar.edu/mm5/.

[44] http://www.wrf-model.org/.

[45] http://www.nas.nasa.gov/Software/NPB/.

[46] http://www.netlib.org/linpack/.

[47] http://www.samss.org.cn.

[48] http://www.netlib.org/benchmark/hpl/.

[49] http://icl.cs.utk.edu/hpcc/.

[50] CFD, http://www.cfd-online.com/.

[51] Ferziger J H, Peric M. Computational Methods for Fluid Dynamics. Springer-Verlag, 1999.

[52] Thompson J F, Soni B K, Weaherill N P (eds.). Handbook of Grid Generation. CRC Press, Boca Raton, FL, 1999.

[53] Rheinboldt W C. Methods for Solving Systems of Nonlinear Equations. Second Edition, SIAM, Philadelphia, 1998.

[54] Saad Y. Iterative Methods for Sparse Linear Systems. Second Edition, SIAM, Philadelphia, 2003.

[55] Teresco J D. Hierarchical partitioning and dynamic load balancing for scientific computation. In *PARA'04 State-of-the-Art in Scientific Computing*, Copenhagen, Denmark, 2004.

[56] Schloegel K, Karypis G, Kumar V. Graph Partitioning for High Performance Scientific Simulations. Chapter 18, Sourcebook of Parallel Computing, Dongarra J, Foster I, Fox G *et al.* (eds.), New York: Morgan Kaufmann Publishers, 2003.

[57] Meiron D, Deiterding R. Load balancing strategies for parallel SAMR algorithms. SURF 2005 technical report, Available at http://scdrm.caltech.edu/publications/cit-asci-tr, 2005.

[58] Sagan H. Space-Filling Curves. New York: Springer-Verlag, 1994.

[59] Mo Z, Zhang J, Cai Q. Dynamic load balancing for short-range parallel molecular dynamics simulations. *Int. J. Computer Math.*, 2002, 79(2): 165–177.

[60] Mo Z, Zhang B. Multilevel averaging weight method for dynamic load imbalance problems. *Int. J. Computer Math.*, 2001, 76(4): 463–477.

[61] Cao X, Mo Z. A new scalable parallel method for molecular dynamics based on Cell-Block data structure. In *Proc. ISPA2004*, Hong Kong, Cao J, Yang L T, Lau F (eds.), *Lecture Notes in Computer Science*, 2004, 3358: 757–764.

[62] Bisseling R H. Parallel Scientific Computation: A Structured Approach Using BSP and MPI. Oxford University Press, 2004.

[63] Knoll D A, Keyes D E. Jacobian-free NewtonKrylov methods: A survey of approaches and applications. *Journal of Computational Physics* (*JCP*), 2004, 193: 357–397.

[64] Trottenberg U, Osterlee C W, Schuller A. Multigrid. Academic Press, 2001.

[65] Mo Z, Shen L, Wittum G. Parallel adaptive multigrid algorithm for 2-D 3-T diffusion equations. *Int. J. Computer Math.*, 2004, 81(3): 361–374.

[66] Falgout R D, Jones J E, Yang U M. The Design and Implementation of Hypre, a Library of Parallel High Performance Preconditioners. Chapter in Numerical Solution of Partial Differential Equations on Parallel Computers, Bruaset A M, Bjørstad P, Tveito A (eds.), Springer-Verlag, to appear. Also available as LLNL Technical Report UCRL-JRNL-205459, 2004.

[67] Balay S, Groppy W D, McInnes L C *et al.* PETSc 2.0 Users Manual. Technical Report ANL-95/11, Argonne National Laboratory, Argonne, IL, Mar 2000.

[68] Bastian P, Birken K *et al.* UG—A flexible software toolbox for solving partial differential equations. *Computation and Visualization in Science*, 1997, 1: 27–40.

[69] Wissink A M, Hornung R D, Kohn S R *et al.* Large scale parallel structured AMR calculations using the SAMRAI framework. In *Proc. High-Performance Computing and Networking Conf.* (*SC'2001*), Denver, 2001, pp.22–28.

[70] Lewis E E, Miller W F. Computational Methods of Neutron Transport. John Wiley & Sons Publisher, 1984.

[71] Mo Z, Fu L, Parallel flux sweep algorithm for neutron transport on unstructured grid. *J. Supercomputing*, 2004, 30(1): 5–17.

[72] Plimpton S, Hendrickson B, Burns S *et al.* Parallel algorithms for radiation transport on unstructured grids. In *Proc. SuperComputing*'2000, Dallas, Nov. 4–10, 2000, pp.25–31.

[73] Mo Z, Zhang A, Cao X. Towards a parallel framework of grid-based numerical algorithms on DAGs. In *Proc. 18th Int. Symp. Parallel and Distributed Computing* (*IPDPS'06*), Greece, April 25–29, 2006, pp.416–424.

[74] Dongarra J, Foster I, Fox G *et al.* (eds.). Sourcebook of Parallel Computing. Morgan Kaufmann Publishers, New York, 2003.

[75] Bernholdt D E. Parallel computational chemistry: An overview of NWChem. Chapter 7 of Sourcebook of Parallel Computing, Dongarra J, Foster I, Fox G *et al.* (eds.), New York: Morgan Kaufmann Publishers, 2003.

[76] Nieplocha J, Ju J, Krishnan M K *et al.* The global arrays user's manual. Pacific Northwest National Laboratory Technical Report No.13130, October 1, 2002.

[77] http::/www.supercomputing.org/.

[78] Jordan H F, Alaghband G, Jordan H E. Fundamentals of Parallel Computing. Prentice Hall. 2003.

[79] Chakravorty S, Kale L V. A fault tolerant protocol for massively parallel systems. In *Proc. 18th International Parallel and Distributed Processing Symposium* (*IPDPS*), Santa Fe, New Mexico, 2004, pp.212–219.

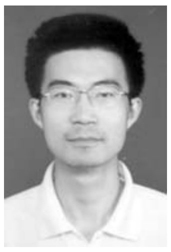
[80] Stou Q F. Algorithms minimizing peak energy on mesh-connected systems. In *Proc. 18th ACM Symp. Parallelism in Algorithms and Architectures* (*SPAA*), Cambridge, MA, USA, 2006, pp.331–334.

[81] Shan J, Chen Y, Diao Q *et al.* Parallel information extraction on shared memory multi-processor system. In *Proc. Int. Conf. Parallel Processing* (*ICPP*), Columbus, Ohio, USA, 2006, pp.215–224.

[82] So B, Ghuloum A, Wu Y. Optimizing data parallel operations on many-core platforms. *First Workshop on Software Tools for Multi-Core Systems* (*STMCS*), Manhattan, NY, 2006, pp.66–70.

[83] Mattson T G, Sanders B A, Massingill B L. Patterns for Parallel Programming. Prentice Hall. 2005.

Guo-Liang Chen is a professor and academician of the Chinese Academy of Sciences. He works with Dept. Computer Sci. & Tech., University of Science and Technology of China. His major research areas include parallel computing theory and algorithms.



Guang-Zhong Sun is a lecturer in the Dept. Computer Sci. & Tech., University of Science and Technology of China (USTC). His research interests include parallel algorithms and scheduling theory.

Yun-Quan Zhang is an associate professor and vice director of the Lab. of Parallel Computing, Institute of Software, CAS. His research interests include performance evaluation, parallel software design and parallel computational model.

Ze-Yao Mo is a professor. He has been doing researches on parallel algorithms and parallel application software for larger scale scientific and engineering numerical simulations.