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

M. Sato (ed.), *Advanced Software Technologies for Post-Peta Scale Computing*,
https://doi.org/10.1007/978-981-13-1924-2_15

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

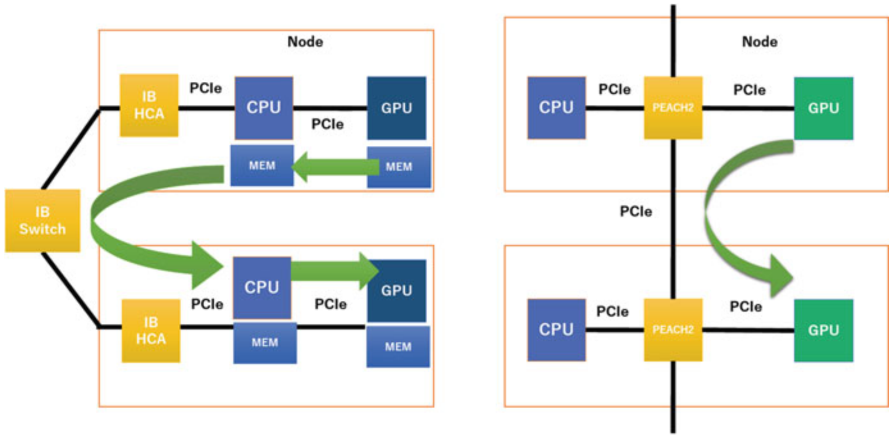


Fig. 15.1 TCA implementation by PCIe bus (left, ordinary method; right, TCA by PCIe)

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.

[1]Before we started this research, we had made another PCIe base communication. Then it is named as the second version.

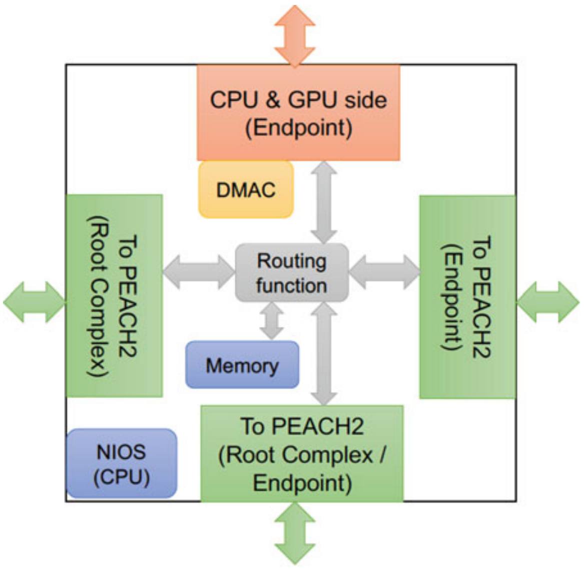


Fig. 15.2 Block diagram of PEACH2 chip

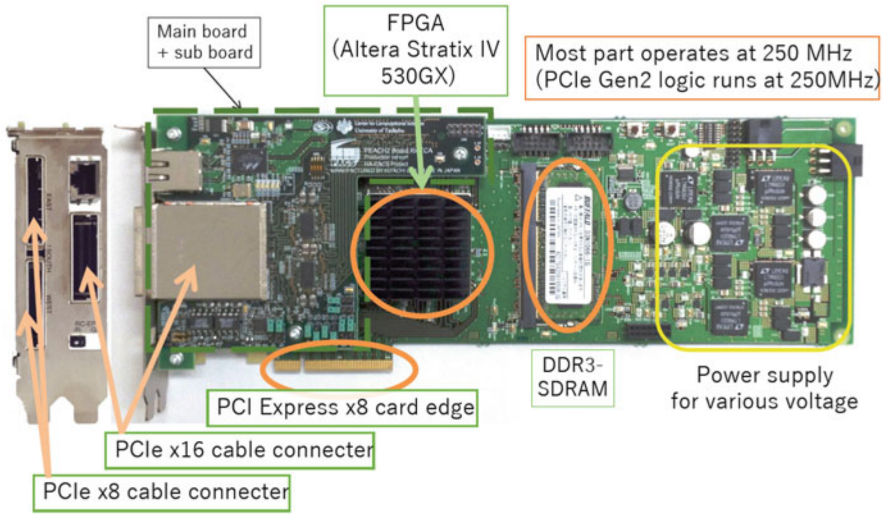


Fig. 15.3 PEACH2 board

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).

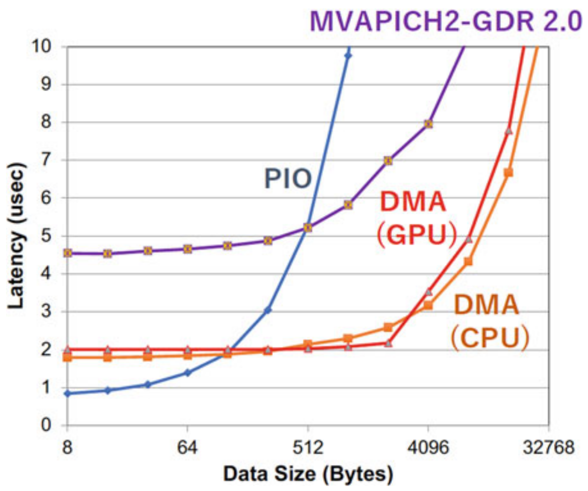


Fig. 15.4 PEACH2 latency

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 μs 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.

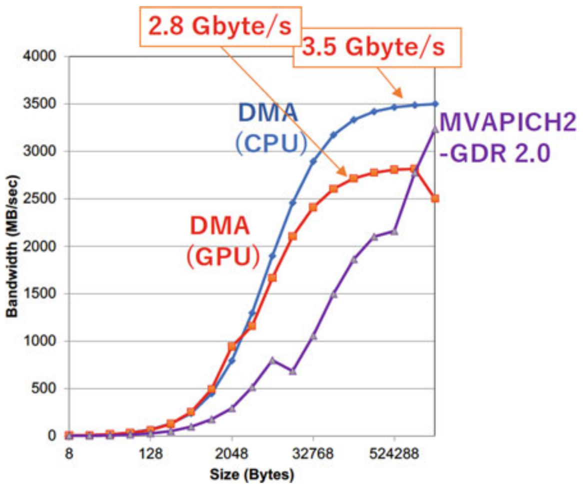


Fig. 15.5 PEACH2 bandwidth

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.

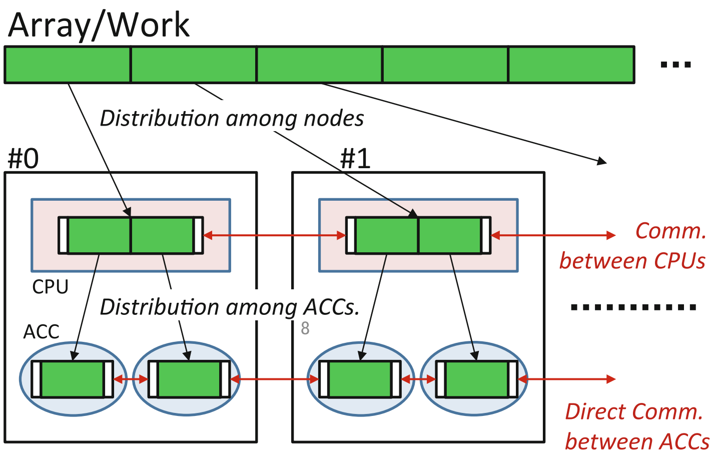


Fig. 15.6 Execution model of XACC for data distribution, off-loading, and communication

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.

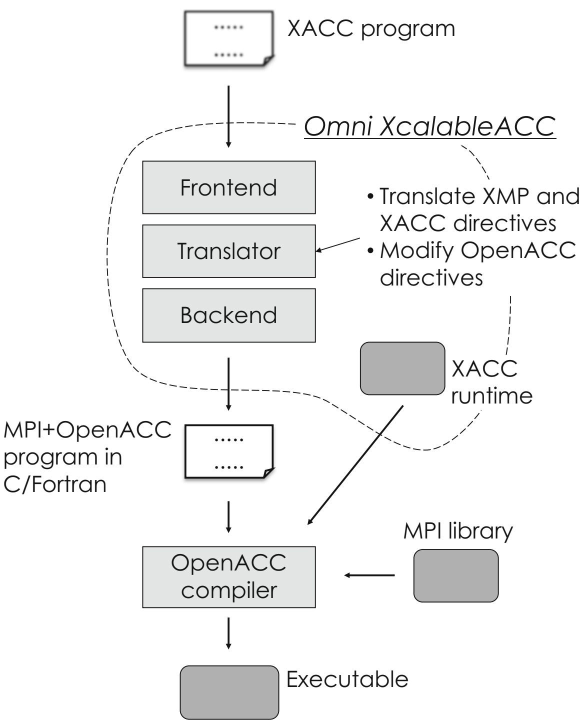
- Data/work mapping onto multiple accelerators
  – Data and works are distributed among nodes by an XMP directive and further distributed among accelerators within each node by the additional `layout` clause of the `declare` and `loop` directives (lines 11 and 18 in Fig. 15.7).
  – The `on_device` clause 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 XcalableACC Compiler

Omni XcalableACC 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,

```
Gluon_t U[4][NT][NZ][NY][NX];
Quark_t X[NT][NZ][NY][NX];
#pragma xmp template t[NT][NZ]
#pragma xmp nodes n[PT][PZ]
#pragma xmp distribute t[block][block] onto n
#pragma xmp align [*][i][j][*][*] with t[i][j] shadow[0][1][1][0][0] :: U
#pragma xmp align [i][j][*][*] with t[i][j] shadow[1][1][0][0] :: X
#pragma acc enter data copyin(U, X)
```

Fig. 15.9 Declaration of distributed arrays

```
#pragma xmp reflect_init(U) width(0,/periodic/1:0, ...) orthogonal
#pragma xmp reflect_init(X, ...) width(/periodic/1, ...) orthogonal
  :
#pragma xmp reflect_do(U, X) acc
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

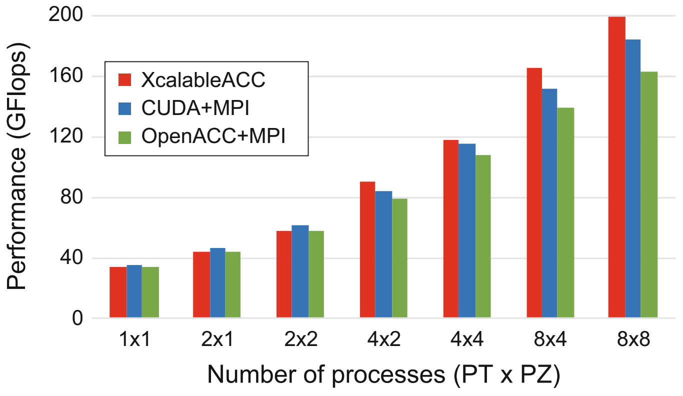


Fig. 15.11 Performance results of lattice QCD mini-application

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 s $\times 26/201 = 0.16$ s, are dominant.

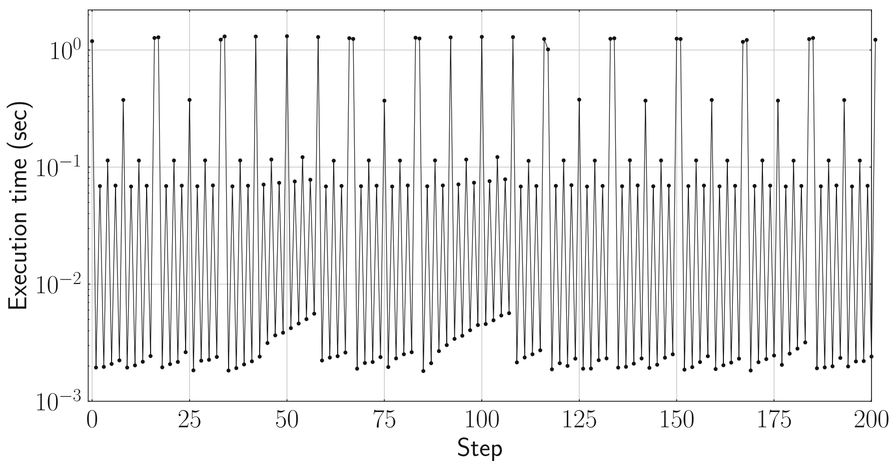


Fig. 15.12 Execution time of gravitational force calculation on NVIDIA Tesla K20X as a function of the time step. The particle distribution is a model reproducing the Andromeda galaxy with $2^{22} = 4{,}194{,}304$ particles generated by `MAGI` [11]

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 ($N \sim 10\,\mathrm{M}$ or $N \sim 30\,\mathrm{M}$ 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 ~2×10^{-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

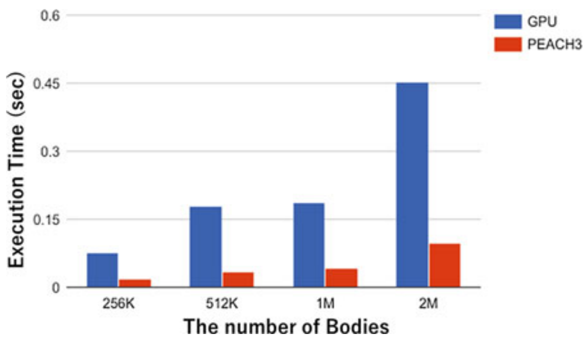


Fig. 15.13 Execution time of LET generator including communication as a function of the number of N-body particles. Blue and red bars show the results for LET generator on GPU and PEACH3, respectively

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 Sympsium 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 cgpop 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)