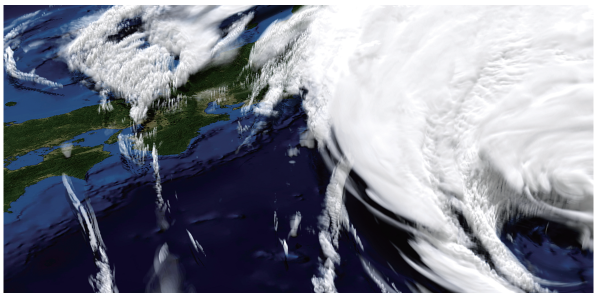
e-Science Journal
Global Scientific Information and Computing Center

Vol. 12

TSUBAME ESJ.



http://www.gsic.titech.ac.jp/TSUBAME_ESJ

Extreme Big Data with TSUBAME2 and Beyond

Hitoshi Sato* Koji Ueno** Koichi Shirahata** Hideyuki Shamoto** Satoshi Matsuoka*
* Global Scientific Information and Computing Center, Tokyo Institute of Technology
** Graduate School of Information Science and Engineering, Tokyo Institute of Technology

Emerging new commodity devices, especially such as GPU accelerators and NVM (Non-Volatile Memory) devices, etc., which are employed on modern high-end supercomputers, may drastically improve performance on the current "Big Data" processing, although the existing "Big Data" processing is mostly operated on poor and cheap infrastructures derived from the cloud-based architecture that employs commodity web-oriented servers equipped with HDDs and GigE networks by using MapReduce-based frameworks. However, the case studies of BigData-enabled software execution on large-scale environments with the modern commodity devices are underinvestigated. This article introduces recent research activities on Big Data-related software techniques, including Graph500, GPU-based MapReduce, large-scale distributed sorting, with TSUBAME2 toward future extreme-scale supercomputers and cloud data centers.

Introduction 1

"Big Data" recently attracts many attentions in various application domains such as Social Networks, Bioinformatics, Internet of Things, etc., and the demand for fast and scalable processing to petabyte- or yottabyte-scale data sets drastically increases. The existing Big Data processing is mostly operated on poor and cheap infrastructures derived from the cloud-based architecture that employs commodity web-oriented servers equipped with HDDs and GigE networks by using MapReduce-based frameworks, i.e., Hadoop. By contrast, emerging new commodity devices, especially such as GPU accelerators and NVM (Non-Volatile Memory) devices, etc. as shown in Fig.1, may drastically improve performance on Big Data processing. For example, GPUs can provide high peak performance and rich memory bandwidth for applications with specific workload patterns, while CPUs offer flexibility and generality over wide-ranging classes of applications. Also, NVMs such as Flash have positive aspects of inexpensive cost, high energy-efficiency, and huge capacity compared with conventional DRAM devices, as well as negative aspects of low throughput and latency. These new devises may bring various benefits to the architecture design of BigData-oriented extreme-scale supercomputers. Indeed, modern high-end supercomputers, such as TSUBAME2 that anticipates the architecture of future extreme-scale supercomputers and cloud data centers, employ commodity-based novel devices such as GPU accelerators, Infiniband interconnects, and flash devices, etc. and provide possible platforms for extremely fast processing to gigantic data sets; however, the case studies of BigData-enabled software execution on large-scale environments with the modern commodity devices are underinvestigated. This article introduces recent research activities on Big Data-related software techniques, including Graph500, GPU-based MapReduce, Large-scale distributed sorting, with TSUBAME2 toward future extreme-scale supercomputers and cloud data centers.

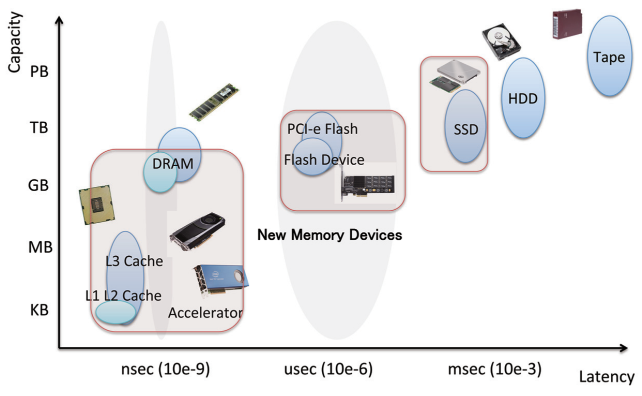


Fig. 1 Hierarchical Memory Devices

Graph and HPC 2

2.1 Graph500 as a Big Data Kernel for Supercomputers

Graph is a fundamental mathematical representation of connected objects described as vertices and edges. Various important applications, such as health care, systems biology, social networks, business intelligence, and social networks, and electric power grids, etc. as shown in Fig. 2 are modeled as graphs. Moreover, since various data source recently generates massive amounts of volumes, demands for large-scale graph processing is significantly increasing, so that graph applications are considered an important kernel for HPC applications. In fact, the Graph500 list[1], which ranks supercomputers by executing large-scale graph problems, are employed as a major metric to evaluate the ability of Big Data processing for supercomputers, instead of the Top500 list known as a list that ranks supercomputers by executing the Linpack benchmark to evaluate the ability of computation. Fig. 3 shows an overview of the instruction of the Graph500 benchmark. The current benchmark in Graph500 measures the time for performing the Breadth-first search (BFS) to a Kronecker graph[2] that models a real-word network with scale-free and small diameter properties; however, the optimal algorithm for supercomputers, especially with distributed memory, is not well investigated. Our Graph500 activities aims for clarifying various hardware- and software-related issues on large-scale graph processing on supercomputers and making the Graph500 a fair benchmark.

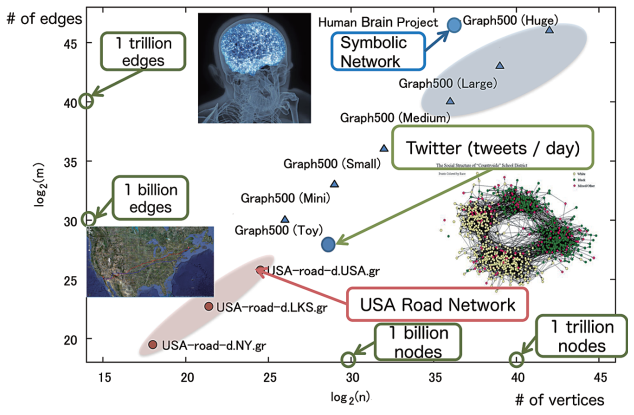


Fig. 2 Large-scale Graphs

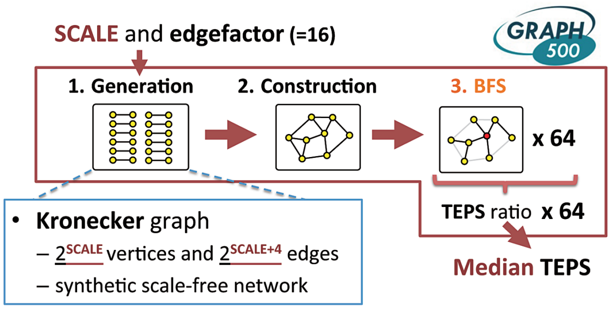

SCALE and edgefactor (=16)
GRAPH 500
1. Generation
2. Construction
3. BFS
x 64
TEPS ratio x 64
Median TEPS
Kronecker graph
– 2^SCALE vertices and 2^SCALE+4 edges
– synthetic scale-free network

Fig. 3 Overview of Graph500

2.2 Scalability Issues

In order to process BFS on supercomputers with distributed memory, we have to partition a graph to several subgraphs; however, to achieve the scalability of BFS on supercomputers with more than over thousands of compute nodes, we have to deal with several problems such as high memory consumption, large communication data transfers, and high computation costs, etc. Specifically, the scalable BFS implementation requires the following algorithms and the data structures:

- Graph data structures with low memory consumption and access costs to other vertex's edge lists on large-scale environments
- Communication algorithms to reduce the number of communications and the communication data volumes between over thousands of compute nodes
- Efficient search algorithms with reduced access costs to the graph data structures

The existing Graph500 reference code has limited scalability on thousands of compute nodes due to the naive data structure and communication algorithm. The latest advanced BFS algorithm for large-scale supercomputers with distributed memory is known as the Wave method with 2D partitioning proposed by Checconi et al.[3] However, the algorithm is not employed another important BFS algorithm, called the hybrid BFS algorithm[4], which drastically improve BFS performance by reducing inefficient edge scans proposed by Beamer et al., although the hybrid BFS algorithm itself has negligible scalability on more than thousands of nodes.

2.3 Ueno's Algorithm

Our Koji Ueno proposes a new sophisticated algorithm for scalable BFS on large-scale supercomputers. Our algorithm is based on the distributed hybrid BFS algorithm proposed by Beamer et al.[5], whereas we also apply the following new techniques to improve the scalability and the performance of BFS on over thousands of compute nodes:

- A new sparse matrix data structure based on bitmap
- An adaptive data representation of a vertex queue to reduce both communication data and memory consumption
- A data structure highly optimized for edge scans
- A memory efficient technique using shared memory between processes on the same compute node to reduce communication data volumes

2.4 Performance Evaluation

Fig. 4 shows the result on the performance of our new implementation in a Giga TEPS (Traversed Edges per second) metric on TSUBAME2.5. We also show the result of our previous implementation (September 2012) that utilizes GPUs without the hybrid BFS algorithm. By using our optimized hybrid BFS implementation, we achieve 1280 GTEPS using 1024 nodes and 2.78x times speed up compared with our previous result (September 2012).

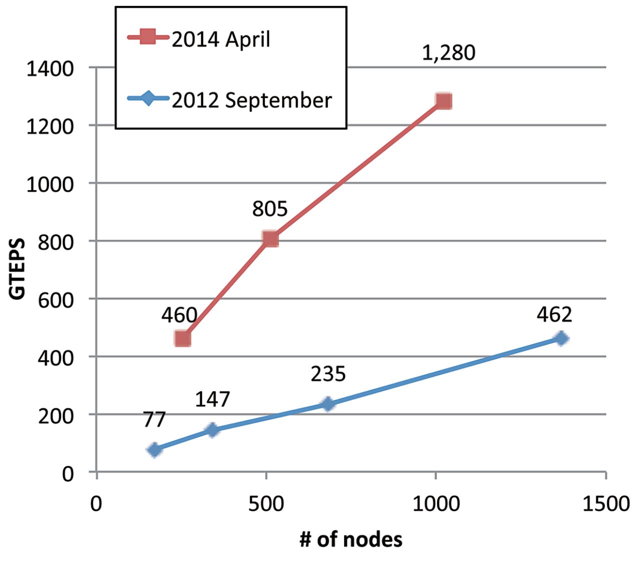


Fig. 4 Graph500 performance results

Fig. 5 shows the history of the achievements on the Graph500 performance on the TSUBAME system. The first submission for Graph500 was conducted on November 2011 and ranked 3rd in the list with 100 GTEPS. Compared with the results, the latest score is achieved about 13 times faster than the first submission. Thus, we believe that continuous software development on a given large-scale real environment makes amazing performance improvement.

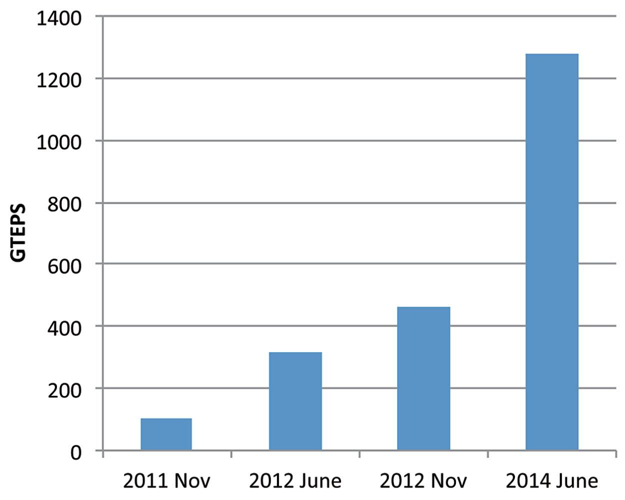


Fig. 5 Graph500 submission history on TSUBAME2

GPU-based MapReduce 3

3.1 MapReduce and Deep Memory Hierarchy Machines

MapReduce[6] is a successful programing model for efficient scalable massive data processing in clouds with large-scale commodity compute clusters, since MapReduce can achieve scalable processing on distributed systems by utilizing the locality and conceal elaborate efforts on the system, such as localized data access for petabyte-scale large data volumes, communication between thousands of nodes, and fault tolerance, etc. MapReduce may also be a good programming model for GPU accelerators for hiding massive parallelism and deep hierarchical memory. However, how much MapReduce-based applications can be accelerated on large-scale GPU-based heterogeneous clusters is an open problem.

3.2 HAMAR: Highly Accelerated Data Parallel Processing Framework for Deep Memory Hierarchy Machines

In order to investigate the above issues, we have developed a highly accelerated data parallel framework, including the MapReduce programming model, for deep memory hierarchy systems with thousands of compute nodes with GPU accelerators and NVM devices. Current version of our implementation (See Fig. 5) automatically handles memory overflows from GPUs by dynamically dividing processing data into multiple chunks and overlaps CPU-GPU data transfer and computation in Map, Reduce Shuffle phases on GPUs as much as possible. We also employ a GPU-based external sorting in our framework.

3.3 Case Study: GIM-V (Generalized Iterative Matrix-Vector multiplication)

GIM-V (Generalized Iterative Matrix-Vector multiplication) is a general expression of matrix-vector multiplication with iterative operations for MapReduce-based large-scale graph processing[7]. Here, let $M = (m_{i,j})$ be a matrix of size n x n, and $v = (v_i)$ be a vector of size n, where i, j in {1, ..., n}. By introducing the operator $\times_G$, we can define the GIM-V algorithm as follows:

$$\begin{aligned} &v' = M \times_G v \\ &\text{where } v'_i = assign(v_i, combineAll_i(\{x_j \mid j = 1..n, \\ &\qquad\qquad \text{and } x_j = combine2(m_{i,j}, v_j)\})) \end{aligned}$$

Here, the above expression is described by using three operators: *combine2*, *combineAll*, and *assign*. We iterate the above operation until satisfying a convergence condition defined by graph algorithms such as PageRank, Random Walk with Restart, and Connected Component, etc. We can describe these graph applications by defining the above three operators in our HAMAR framework.

3.4 Performance Evaluation

We have conducted large-scale experiments of PageRank algorithm based on the GIM-V model on top of the HAMAR framework on TSUBAME2.5 with 1024 nodes (12288 CPU cores, 3072 GPUs). The results (Fig. 7) in exhibit that our GPU-based implementation performs 2.81 GEdges/sec (billion edges per second, 47.7GB/sec) to a large-scale graph with 17.18 billion vertices and 274.9 billion edges, which is 2.10x faster than the multi-core CPU-based implementation even when the graph data size exceeds the capacity on multiple GPUs. As for the weak scaling performance, our GPU-based implementation also shows good scalability: 686x performance improvement by using 1024 nodes (3072 GPUs) compared with using a single node (3 GPUs).

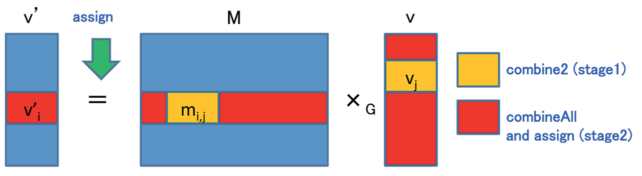


Fig. 7 Overview of GIM-V

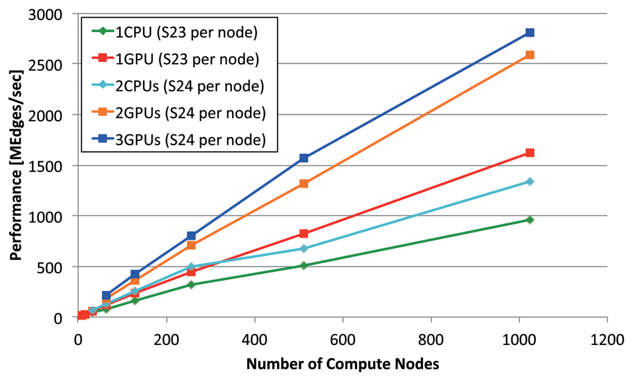


Fig. 8 GIM-V scalability on TSUBAME2

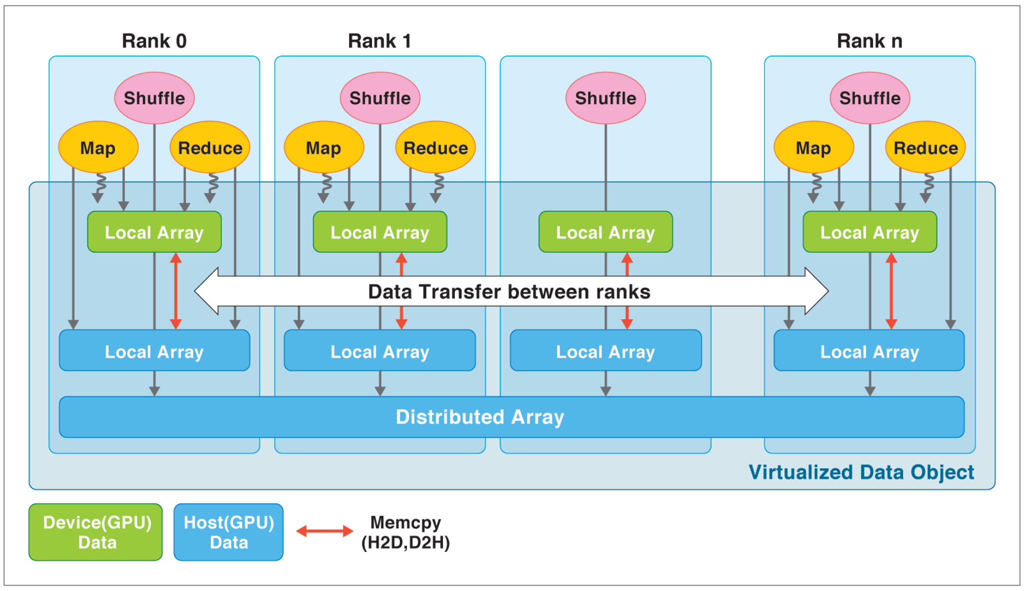


Fig. 6 Overview of HAMAR

GPU-based Distributed Sorting 4

4.1 Sorting on Distributed Memory Architectures

Sorting is also considered a key building block in many data-intensive supercomputing applications in various domains such as genomics and astrophysics, etc., in which the need for processing terabyte- and petabyte-scale data sets is drastically increasing due to the generation of science experiments and observations. In order to sort such huge data sets, many sorting algorithms for distributed memory architectures are proposed. In particular, splitter-based parallel sorting algorithms are known as fast distributed sorting algorithms, since the communication costs of the algorithms are relatively small. However, as the communication costs go down in the splitter-based parallel sorting, the computation costs dominate the overall performance.

4.2 Sorting on Distributed Memory Architectures

Most splitter-based parallel sorting algorithms consist of the following steps shown in Fig. 8. Firstly, data on each process are sorted in the local sort phase. After the sorting phase, splitters are selected and data are transferred based on the splitters. Finally, each node merges sorted chunks into a single sorted array. Although other distributed sorting algorithms, such as Merge sort, Radix sort, etc., introduce significant data transfers between processes; splitter-based parallel sorting can reduce the number of iterations for data transfers.

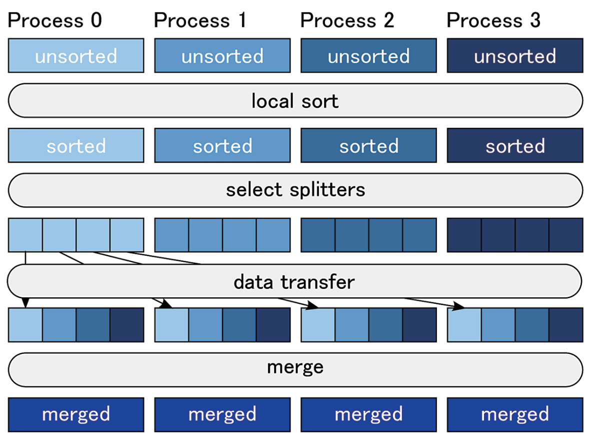


Fig. 9 Splitter-based parallel sorting algorithm

4.3 GPU-based Implementation of Splitter-based Sorting Algorithms

In order to demonstrate the acceleration of performance bottlenecks in splitter-based parallel sorting algorithms, we extend the existing algorithm, HykSort[8], by offloading the costly local sort phase. Since the GPU memory capacity is limited, we firstly break an unsorted array int chunks appropriately sized to fit the capacity of GPU memory. After creating the chunks, the chunks are sorted on GPU iteratively and the sorted chunks are merged on DRAM.

4.4 Performance Evaluation

We conduct weak-scaling experiments using up to 1024 nodes (2048 of CPU cores and GPU devices) and compare our GPU-based hyksort implementation that uses 6 threads per CPU socket with OpenMP-based hyksort implementation that uses 1 thread and 6 threads per CPU socket. Note that each process is bound to a single socket for processing 2GB of 64 integer data sets. Fig. 9 shows the results, where the x-axis denotes the number of processes and the y-axis denotes the throughput performance in Keys per second. Our GPU-based implementation achieves 0.25 TB/s when we sort 4TB of data on 1024 nodes, which is 1.40x times faster than the OpenMP-based implementation using 12 threads per CPU socket and 3.61x times faster than using 1 thread per CPU socket.

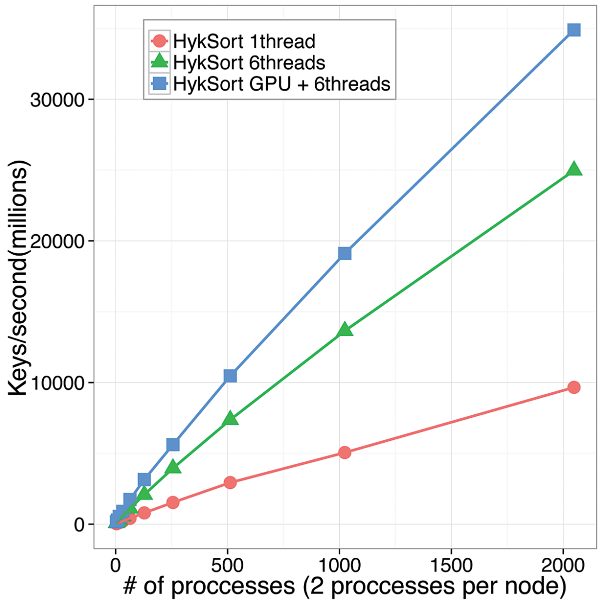


Fig. 10 Results of GPU-based hyksort

Conclusions 5

We have introduced our recent activities on "Big Data" related issues, including Graph500, GPU-based MapReduce, Large-scale distributed sorting, with TSUBAME2 and demonstrates efficient execution of the "Big Data" software implementations on a large-scale modern high-end supercomputer. These activities are also applied to the design and development of the production machines, such as TSUBAME3.0, which is slated to be deployed in the first half of 2016 as one of the first "Extreme Big Data" convergent production machine.

Acknowledgements

This research was supported in part by JST-CREST Projects "Advanced Computing and Optimization Infrastructure for Extremely Large-Scale Graphs on Post Peta-Scale Supercomputers" and "Extreme Big Data (EBD): Next Generation Big Data Infrastructure Technologies Towards Yottabyte/Year", JSPS KAKENHI Grant Number 26011503, 26540050, and TSUBAME Grand Challenge Program.

References

[1] Graph500: http://www.graph500.org/
[2] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, Zoubin Ghahramani, "Kronecker graphs: An approach to modeling networks", The Journal of Machine Learning Research, Vol11, p985-1042, 2010.
[3] Fabio Checconi, Fabrizio Petrini, Jeremiah Willcock, Andrew Lumsdaine, Anamitra Roy Choudhury, and Yogish Sabharwal, "Breaking the speed and scalability barriers for graph exploration on distributed-memory machines", Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12), Article No.13, p13:1-13:12, 2013.
[4] Scott Beamer, Krste Asanović, and David Patterson, "Direction-optimizing breadth-first search", Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12), Article No.12, p12:1-12:10, 2012.
[5] Scott Beamer, Aydin Buluc, Krste Asanović, and David Patterson, "Distributed Memory Breadth-First Search Revisited: Enabling Bottom-Up Search", Proceedings of IPDPSW'13, p1618-1627, 2013.
[6] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: simplified data processing on large clusters", Communications of the ACM, Vol. 51, Issue 1, p107-113, 2008.
[7] U. Kang, Charalampos E. Tsourakakis, and Christos Faloutsos, "PEGASUS: A Peta-Scale Graph Mining System Implementation and Observations", Proceedings of the 9th IEEE International Conference on Data Mining (ICDM '09), p229-238, 2009.
[8] Hari Sundar, Dhairya Malhotra, and George Biros, "HykSort: a new variant of hypercube quicksort on distributed memory architectures", Proceedings of the 27th international ACM conference on International conference on supercomputing, p293-302, 2013.

A Global Atmosphere Simulation on a GPU Supercomputer using OpenACC: Dry Dynamical Core Tests

Hisashi Yashiro* Akira Naruse** Ryuji Yoshida* Hirofumi Tomita*

* RIKEN Advanced Institute for Computational Science ** NVIDIA

In many cases, weather and climate simulations are memory-bound. Therefore, use of a graphics processing unit (GPU) is expected to be helpful; however, rewriting the source code for execution using a GPU is time-consuming, as such applications are typically large and complex. Here, we describe the application of OpenACC to the dynamical core package of a global high-resolution atmosphere model application and report the successful execution of the dynamical core without re-writing any specific kernel subroutines for GPU execution. The performance and scalability was evaluated using the TSUBAME2.5 supercomputer. The results showed that the kernels generated by OpenACC achieved good performance, which was appropriate to the memory performance of GPU, as well as weak scalability. A large-scale simulation was carried out using 2560 GPUs, which achieved 60 TFLOPS.

Introduction 1

Weather and climate simulations play important roles in weather forecasting and climate prediction, and also in researches such as those on typhoon genesis and feedback mechanisms of climate systems, and so on. Global-scale atmospheric phenomena are interacted with small-scale motions. High-resolution simulations are one of the approaches for reducing uncertainties in numerical simulations. However, to perform a simulation with high resolution, significant computational resources are required. Recent trends in supercomputers are for the increased use of multi-core processors, and in particular, graphics processing units (GPUs). Heterogeneous architectures are becoming more common, and it is expected that weather and climate simulations will benefit from such approaches. Applications for atmospheric simulations require a large byte per floating-point operation (FLOP) ratio, or a B/F ratio; however, the performance of such simulations is typically limited by the memory throughput. The use of GPUs for weather and climate simulations is therefore interesting because of the large memory bandwidth.

Weather and climate simulations typically contain 10,000–100,000 lines of source code; therefore, the cost of rewriting these applications using languages, such as CUDA or CUDA-Fortran, is large. Furthermore, weather and climate models are typically developed via interdisciplinary cooperation, so they contain different modules written by people from different research fields. This makes maintaining the source code and supporting the architecture particularly challenging.

Recently, a new programming model called OpenACC has appeared. OpenACC is directive-based and enables the straightforward use of GPUs. Placing directives in the existing source code enables data transmission between the CPU and GPU, and calculations can be performed on a GPU. In this study, we applied OpenACC to an existing meteorological code that is large and complicated, and evaluated its performance on a parallel GPU supercomputer. We also evaluated the portability, readability, and maintainability of the source code.

NICAM and the dynamical core 2

Nonhydrostatic ICosahedral Atmospheric Model (NICAM)[1][2] is a weather and climate application for high-resolution global simulations of the atmosphere using massively parallel machines. NICAM was developed by the Japan Agency for Marine–Earth Science and Technology (JAMSTEC), the University of Tokyo, and the RIKEN Advanced institute of Computational Science (AICS). The source code was mostly written using Fortran90. NICAM employs a fully compressible non-hydrostatic dynamics, where the finite volume method (FVM) is used for discretization into the icosahedral grid configuration. The icosahedral grid system covers the sphere quasi-homogeneously. A grid point method, such as FVM, has the advantage of reducing data transfer between the computational nodes over a spectral transform method, which requires global communication between nodes, and is one of the bottlenecks in a massively parallel machine.

The components of weather and climate models can be divided into two categories. The first are components to solve the fluid dynamics of the atmosphere (referred to as the dynamics). The dynamics of NICA consists of the kernel of stencil operators, including divergence, gradient, Laplacian, and tracer advection with a non-negative flux limiter. The dynamics require a large B/F ratio, as well as frequent communication during the calculations. The second category is referred to as the physics, which contains the cloud microphysics, atmospheric radiation transfer, the sub-grid scale boundary layer turbulence, and so on. These components do not require such a large B/F ratio compared with the dynamics, and typically do not require communication, because they have no reference of neighbor grid information in the horizontal direction.

To obtain effective performances of weather and climate applications using GPUs, all of the dynamics and physics components for execution using a GPU should be optimized. These applications exhibit a "flat profile" (i.e., no clear hotspots of the calculation exist). Because of this, we carried out GPU optimization of the entire dynamics part of the NICAM. In this study, we used the dynamical core package of NICAM, named NICAM-DC. NICAM-DC is distributed under the BSD 2-clause license (http://scale.acis.riken.jp/nicamdc/), and is not simply a kernel program, but is a stand-alone application that includes file input/output (I/O) routines. A performance evaluation of the data throughput from the GPU to the hard disk drive (HDD) is important, and was implemented using a number of popular test cases for the dynamical core.

Implementation of OpenACC 3

We optimized NICAM-DC using OpenACC employing the following policies.

- Full separation of the memory allocation stage and computation stage. Arrays such as the metric term, which should be executed on the GPU, were prepared in the setup stage. Memory allocation of the working array was excluded from the main loop
- Arrays reside in GPU memory if they are not updated during the computation: Before start of time integration, non-updated arrays were transferred in the setup stage using the "present_or_copyin" clause.
- Asynchronous execution of loop kernels. The structure of loops was refactored, and the timing of the communication was arranged to execute using the "async" clause as much as possible.
- Optimization of halo exchange. For the point-to-point exchange in the halo grid, data were packed in the sending node and unpacked in the receiving node. We conducted these packing/unpacking actions on the GPU configuration and minimized the data size for transfer between the host and device.
- Calculation of a singular point. There are special loops for two singular grids: north-pole and south-pole points. These loops incur less computational cost, so they were not executed on the GPU. If the CUDA programming model is used, offloading loops and non-offloading loops should be separated. OpenACC enables more flexible treatment regarding whether each loop uses the GPU in a subroutine. This improves the readability of the source code and facilitates maintenance.
- Output of simulation results. Many of the 2D and 3D variables are output during the simulation for analysis. File I/O incurs a significant cost for execution on a GPU because of the data transfer from the device. We calculated diagnostic variables, converted the vertical coordinates, and stored them on the GPU. Only at the time of writing data to files were these data transferred from the GPU.

The order of the array dimensions used in NICAM was (*ij,k,l*), which represented the horizontal grid, vertical grid, and the unit that was divided for process parallelization (=region). In many cases, the number of horizontal grid points is the largest. For execution using GPUs, we did not have to change the dimension ordering. In NICAM-DC, only a few arrays were changed to "Array of Structure (AoS)" for optimization to the scalar machine. These arrays were reverted to "Structure of Array (SoA)" for calculation using the GPUs. Approximately 2,000 lines of code were modified or inserted to implement OpenACC, which accounted for 5% of the total lines of source code in the NICAM-DC.

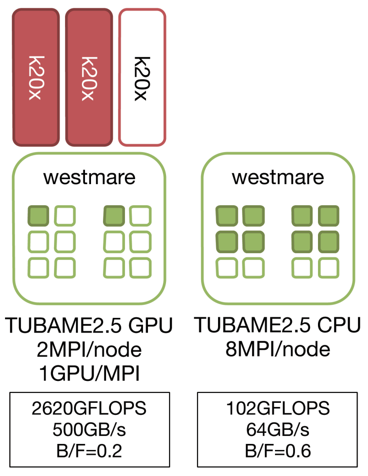


Fig. 1 Usage of the CPU and GPU in the computing node of TSUBAME2.5. Data shown on the left represent the CPU–GPU calculation, and those on the right represent the CPU-only calculation.

Performance evaluation using TUBAME2.5 4

4.1 Single node performance

We evaluated NICAM-DC on the TSUBAME2.5 supercomputer. We implemented two architecture settings to achieve a node-to-node comparison between the CPU–GPU and CPU-only calculations, as shown in Figure 1. NICAM-DC is a memory-bound application, so the total throughput of memory should be the most important parameter. Each node of TSUBAME2.5 had three GPUs; however, we only used two of them due to the limitations of process division of NICAM. The number of floating operations and the amount of memory transfer were measured in advance. At the time of the performance tests, we only measured the elapsed time of each part of the application to avoid perturbing the performance counters. We also used monitor tools for power consumption of each node and each GPU. The problem size was as follows. For the control run, the horizontal mesh size was 56 km, and there were 160 vertical layers corresponding to 26 million grids points. The total number of floating point operations in the main loop was 420 GFLOPS, and 2.3 TB/step of memory transfer was required for each message passing interface (MPI) process. For the CPU-only calculation, we used an AoS version of the code, and increased the number of MPI processes 4-fold. A total of 2 problem sizes were used: one with 5 nodes and 10 MPI processes for the CPU–PGU calculation, and one with 5 nodes and 40 MPI processes for the CPU-only calculation. The performance test was based on a baroclinic instability test case for the global atmospheric model,[3] and only small steps were executed. The file output of some of the variables for analysis was included in the test.

Figure 2 shows the results of the performance test for one node. The upper panel shows that the CPU–GPU calculation was completed in approximately 1/8th of the elapsed time of the CPU-only calculation. This result is approximately proportional to the difference in memory transfer performance. We achieved a peak memory transfer of ~50 % in both of the architecture settings, which demonstrates that OpenACC, which has a directive-based programming style, can generate the code with a sufficient level of performance. From the point of view of the number of floating-point operations, the peak ratio was worse in GPU calculation (see the middle panel of Fig. 2). Because of the large B/F ratio, the computational resources of the GPUs could not be effectively utilized by the application. To more effectively use the GPUs, in the future, we plan to make greater use of mixed precision, and further consideration of the trade-off between numerical accuracy and computational efficiency is warranted. The lower panel in Figure 2 shows the power consumption in FLOPS/W; the difference was approximately inversely proportional to the elapsed time, so the total energy consumption was similar.

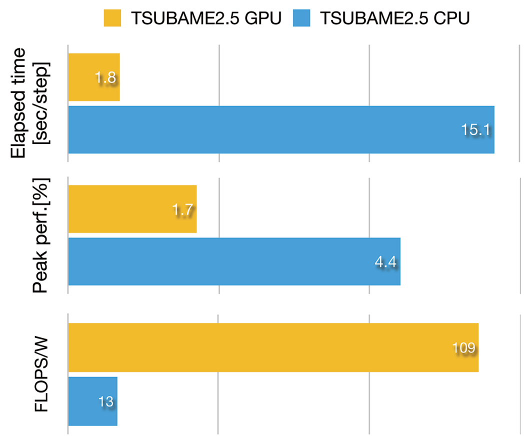


Fig. 2 The performance of NICAM-DC on TSUBAME2.5. The elapsed time per step in the main loop (upper panel), the peak performance ratio (middle panel), and the power consumption (lower panel) for both the CPU–GPU and CPU-only calculations.

4.2. Scalability

Weak scaling tests were performed by decreasing the horizontal mesh size from 56 km to 3.5 km, while increasing the number of nodes accordingly. The number of vertical layers and steps remained unchanged. The largest simulation used 1280 nodes (2560 GPUs) and achieved 46.5 TFLOPS. The data showed good scalability for both the CPU–GPU and CPU-only calculations. Only 30 % of the elapsed time of the 1280-node test increased compared to the 5-node test. The main cause of the increase in the elapsed time was communication between nodes in the CPU-only calculation, whereas file I/O was dominant in the CPU–GPU calculation. In the CPU-only calculation, there were more MPI processes per node. This led to greater communications congestion than with the CPU–GPU calculation. Data transfer from device to host for file I/O was the limiting factor for scalability with the GPU–CPU calculation. To reduce the data transfer due to file I/O, the precision of output data should be reduced. Data compression on the GPU will also be favorable. In this test, output variables were called every 15 min. Typically, data output is less frequent than this in climate and weather simulations. We changed the interval for data output to every 12 h, and the performance increased from 46.5 TFLOPS to 60 TFLOPS for the 1280-node case.

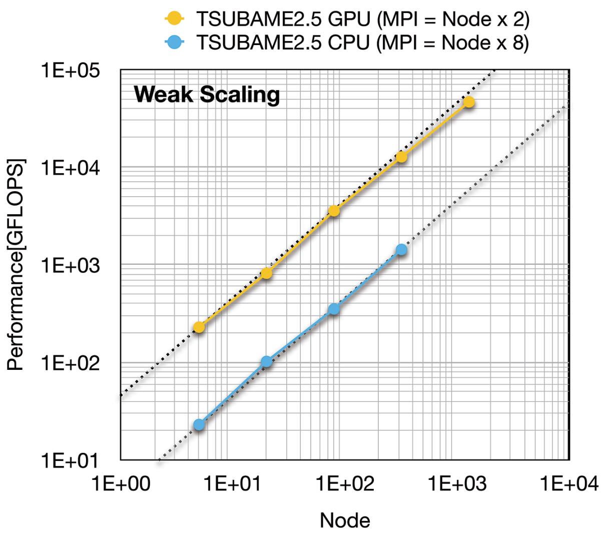


Fig. 3 The weak scaling performance of NICAM-DC on TSUBAME2.5.

Figure 4 shows the results of strong scaling tests. Here, a 56 km mesh size was used, and the number of nodes increased from 5 to 1280. The number of horizontal grids points in each process was changed from 16,900 to 100. We observed saturation in the performance of the CPU–GPU calculation as a function of the number of the nodes. This was related to the decrease in the horizontal grid size per process. When we used a larger number of nodes, the number of horizontal grid points was not sufficient, considering the number of threads on the GPU. The data transfer between the nodes decreased as the number of horizontal grids per process decreased; however, the latency of MPI communication did not decrease, and the ratio of the communication time to the time for computation increased. Reducing the frequency of communication will therefore be more effective than reducing the size of the data in these communications. Support for 'pinned memory' is also expected to be useful in reducing data transfer latency between the host and the device. The performance of strong scaling is more critical for climate simulations, which typically consider 10 to 100 years (10^6–10^7 steps). A drastic improvement in this aspect of performance is therefore required in the future.

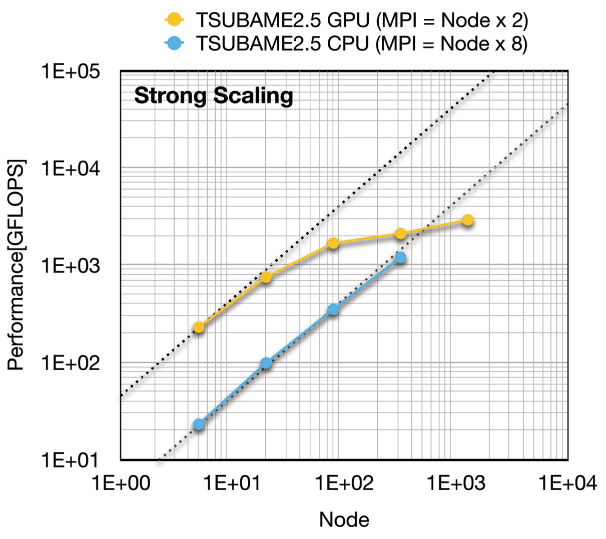


Fig. 4 Strong scaling performance of NICAM-DC on the TSUBAME2.5.

Summary 5

In this study, we implemented GPU-based calculations of the dynamical core of a global high-resolution atmosphere model using OpenACC. We obtained a performance level that was appropriate for the memory transfer performance of GPU. Only 5 % of the lines of source code were modified, demonstrating good portability using the approach described.

Weak scaling tests exhibited good results with thousands of GPUs. However, we believe that further improvements are required to achieve effective strong scaling. Our results demonstrate the effectiveness of OpenACC in enabling large and complex applications to be executed on GPUs.

In the future, we plan to expand the application of OpenACC to physics calculations, and will perform simulations with the full NICAM package. Further improvements to the dynamical core will be implemented using mixed precision. Some algorithms in the dynamical core should be modified to decrease the memory transfer and to reduce the frequency of communications.

Acknowledgements

The largest-node simulation was executed as a TSUBAME Grand Challenge Program, spring 2014. We express our special thanks for the use of these resources. This research was supported in part by the G8 Research Councils Initiative for Multilateral Research Funding, “Icosahedral-grid Models for Exascale Earth System Simulations (ICOMEX)”.

References

[1] Satoh, M., T. Matsuno, H. Tomita, H. Miura, T. Nasuno and S. Iga (2008) : Nonhydrostatic Icosahedral Atmospheric Model (NICAM) for global cloud resolving simulations. J. Comp. Phys., the special issue on Predicting Weather, Climate and Extreme events, 227, 3486-3514.

[2] Tomita, H. and Satoh, M. (2004) : A new dynamical framework of nonhydrostatic global model using the icosahedral grid. Fluid Dyn. Res., 34, 357-400.

[3] Jablonowski C, Williamson DL. A baroclinic instabilitiy test case for atmospheric model dynamical cores. Quart. J. Roy. Meteor. Soc. 2006; 132(621C):2943–2975.

High-productivity Framework on GPU-rich Supercomputers for Weather Prediction Code

Takashi Shimokawabe* Takayuki Aoki* Naoyuki Onodera*
* Global Scientific Information and Computing Center, Tokyo Institute of Technology

Numerical weather prediction is one of the major applications in high-performance computing and is accelerated on GPU supercomputers. Skillful programming techniques are required for obtaining good parallel efficiency on GPU supercomputers. The Japan Meteorological Agency is developing a next-generation high-resolution meso-scale weather prediction code ASUCA. Our framework-based weather prediction code ASUCA has achieved good scalability with hiding complicated implementation and optimizations required for distributed GPUs, contributing to increasing the maintainability.

Introduction 1

Recently, exploiting accelerators, including GPUs, along with conventional CPUs on supercomputers has emerged as an effective way to achieve high performance with relatively-low power consumption [1,2,3]. It is well known that the advantages of GPU in both computation power and wide memory bandwidth allow various scientific simulations. In the field of numerical weather prediction, a computationally expensive physics module of the WRF model was accelerated by using a GPU [6]. By using large-scale of GPUs, the weather prediction code ASUCA was accelerated by using multiple GPUs on the TSUBAME supercomputer in our previous research [1,2,5]. WRF model was also accelerated by using NVIDIA Kepler GPUs of the Cray XE6 ``Blue Waters'' at NCSA at the University of Illinois [7].

Although various applications are accelerated by GPUs, programming on different types of devices by using low level platform-specific programming languages such as CUDA that is specific to NVIDIA GPUs forces the programmer to learn multiple distinctive programming models especially to achieve high performance as expected. To solve this problem and improve programmer productivity, various types of high-level programming models were proposed [8].

In this research, in order to implement ASUCA on GPU-rich supercomputers effectively with high portability, we propose a high-productivity framework for multi-GPU computation of mesh-based applications. The proposed framework can be used in the user code developed in the C++ language. The framework itself is written in the C++ language with CUDA. The framework provides C++ classes that support the programmer to write stencil functions that update a grid point, execute these functions and describe efficient GPU-GPU communication. By using these classes, the programmer can write user code just in the C++ language and develop program code optimized for multiple GPU systems including GPU-rich supercomputers without introducing complicated optimizations. Since the programmer can write the stencil functions without depending on platform-specific programming languages, the framework is possible to translate these user-written functions to several platforms; the proposed framework currently generates CPU code and GPU code.

This article reports that the programming model and our implementation strategies of the proposed framework, and the performance results of the dynamical core and a portion of physics processes in the framework-based ASUCA. We also show the performance evaluation of ASUCA running on TSUBAME. Please see the references [4,5] for more details.

Weather prediction code ASUCA and its GPU acceleration 2

ASUCA (Asuca is a System based on a Unified Concept for Atmosphere) is a next-generation high resolution mesoscale atmospheric model being developed by the Japan Meteorological Agency (JMA). ASUCA is going to succeed the Japan Meteorological Agency Non-Hydrostatic Model (JMA-NHM) as an operational non-hydrostatic regional model at the JMA. In the ASUCA, a generalized coordinate and flux-form non-hydrostatic balanced equations are used for the dynamical core. The time integration is carried out by a fractional step method with the horizontally explicit and vertically implicit (HE-VI) scheme. One time step consists of short time sub-steps and a long time step. The horizontal propagation of sound waves and the gravity waves with implicit treatment for the vertical propagation are computed in the short time step with the third-order Runge-Kutta scheme. The long time step is used for the advection of the momentum, the density, the potential temperature and the water substances, the Coriolis force, the diffusion and other effects by physical processes with the third-order Runge-Kutta method. The physical processes that are equivalent to or more enhanced than those employed in the JMA-NHM are implemented in the current ASUCA.

In our previous research, we have developed the full GPU version of ASUCA. All variables are allocated on GPU memory and all computational modules inside the time-step loops are carried out by GPU. Since ASUCA is being developed in Fortran language at the JMA, the GPU code has to be developed from scratch in CUDA. Before implementing the ASUCA on GPU, we re-wrote the Fortran ASUCA code to C/C++ language because we changed the element order of the 3-dimensional array to improve the memory access performance of the GPU computing. In 2011, we achieved 145 TFlops for the domain of 14368 x 14284 x 48 in single precision using 3,990 GPUs of GPU-rich supercomputer TSUBAME.

3 GPU-computing Framework

In our previous porting of ASUCA, in order to archive high performance, we changed element order of arrays to an appropriate one that was suitable for GPUs and introduced optimization for GPU architectures. In the large-scale GPU computation, we introduced optimizations such as overlapping technique to hide communication overhead by computation. Through implementing ASUCA on GPU, multi-GPU computation of mesh-based applications, including weather prediction codes, has the potential to achieve high performance. However, it requires relatively-high cost of implementation. To apply these complicated optimizations to various mesh-based applications including ASUCA easily, 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. The proposed framework is designed to provide highly-productive programming environment for stencil applications with explicit time integration running on regular structured grids, including the weather prediction codes. The framework updates the physical variables defined on grid points and stored in arrays in user programs. The framework is intended to execute user programs on NVIDIA's GPUs; the C/C++ language and CUDA are used for the implementation of CPU code and GPU code, respectively. The framework also supports multi-GPU computation.

Our major design goals of the framework are described as follows.

- To perform stencil computations on grids, the programmer only defines C++ functions that update a grid point, which is applied to entire grids by the framework. Our framework automatically translates these functions and generates both GPU and CPU code. The framework allows us to write the user code just in the C++ language and we can develop program code optimized for GPU computing without introducing complicated optimizations.
- The user code with the framework should be written in a standard language without using the non-standard programming model and language extension, especially considering the cooperation with external existing libraries.
- The framework should provide unified interfaces for both inter-node and intra-node communications while each of these communications is performed using the most appropriate method.

3.1 Writing Stencil Functions

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:

```
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. Similarly, classes for writing 1D and 2D access patterns are provided.

`ArrayIndex3D` holds the size of each dimension of a grid (n_x, n_y, n_z) and index parameters (i, j, k). `ArrayIndex3D` can be used for an array `f` that has elements. When `idx` is an object of `ArrayIndex3D`, `f[idx.ix()]` will return an element on the (i, j, k) point of the grid. `ArrayIndex3D` has C++ template member functions that provide indices of points around the (i, j, k) point of the grid; `idx.ix<1,0,0>()` and `idx.ix<-1,-2,0>()` will, for example, return indices of (i+1, j, k) and (i-1, j-2, k) points, respectively.

The function parameter of stencil functions must begin with `ArrayIndex3D`, which represents the coordinate of the point where this function is applied. This is followed by any number of additional parameters, including scalar values and pointers of arrays.

3.2 Run Stencil Functions on Grids

In order 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 as follows:

```
Loop3D loop3d(nx+2*mgnx, mgnx, mgnx,
              ny+2*mgny, mgny, mgny,
              nz+2*mgnz, mgnz, mgnz);
loop3d.run(Diffusion3d(), ce, cw, cn, cs,
           ct, cb, cc, f, fn);
```

`Loop3D` is initialized with parameters that specify a 3D rectangular range where stencil functions are applied. The parameters of `Loop3D::run` must begin with a stencil function defined as a functor, followed by any number of additional parameters that are provided to this functor. We use C++ type inference and call an appropriate functor at `Loop3D::run`. The programmer can define stencil functions as both host and device (i.e., GPU) functions using the qualifiers `__host__` and `__device__` provided by CUDA. `Loop3D` executes stencil functions on grids sequentially for CPU while it executes the stencil functions in parallel for GPU using CUDA's global kernel functions. `Loop3D` determines whether a pointer given by `Loop3D::run` as a parameter points to host memory or device memory, and call appropriate internal functions within `Loop3D`.

3.3 GPU-GPU Communication

In the multi-GPU computation of mesh-based applications, the domain decomposition is often used for these parallelization. The fundamental structure of this framework is based on this strategy. Figure 1 shows the domain decomposition of computational grid. 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 the `BoundaryExchange` class to write this communication. The `BoundaryExchange` class utilizes appropriate GPU-GPU communication as follows. For intra-node parallelization, we utilize OpenMP and GPUs directly access data stored on device memory of other GPU on the same node. When two GPUs within a same node support GPUDirect peer-to-peer access, communication between these two GPUs no longer needs to be staged through the host and is therefore faster. Figure 2 illustrates intra-node GPU-GPU communication based on peer-to-peer access. On the other hand, inter-node GPU-GPU communication is performed by using the MPI library. Figure 3 illustrates this communication. Since GPUs cannot directly access data stored on device memory of other GPUs on other nodes, the host CPUs are used as bridges to exchange boundary data between neighbor GPUs.

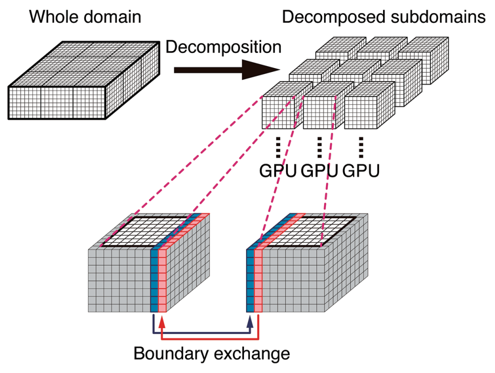


Fig. 1 Multi-GPU computing of mesh-based computation

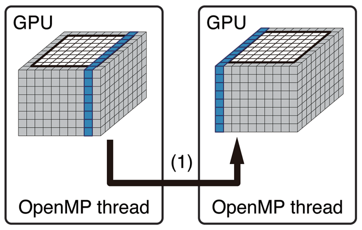


Fig. 2 Intra-node GPU-GPU communication by the OpenMP threads. This process is composed of (1) GPUDirect peer-to-peer access.

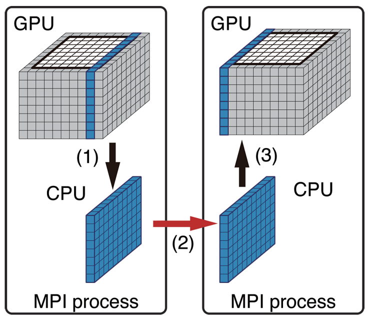


Fig. 3 Inter-node GPU-GPU communication by MPI. This process is composed of (1) memory copy from GPU to host, (2) data exchange by MPI communications and (3) memory copy from host to GPU.

`BoundaryExchange` is typically used as follows:

```
BoundaryExchange *exchange = domain.exchange();
exchange->append(array1);
exchange->append(array2);
exchange->append(array3);
exchange->transfer();
```

`BoundaryExchange` is initialized by `domain`, which is a `Domain` object, and holds the connection relation with neighbor subdomains and the size of data exchanged with them. When `BoundaryExchange::transfer` is called, boundary regions of arrays specified by `BoundaryExchange::append` are exchanged.

3.4 Overlapping Method

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 contributes to performance improvement.

This framework provides kernel-division overlapping method reported in our previous work[1,2]. This method exploits data independency within a single variable. Since each element of a variable can be computed independently for one calculation, computations for the boundary regions can be executed separately from other calculations for the rest of the domain. Figure 4 illustrates the flow of the overlapping method. This method consists of the following. First, the values in the inside region are computed, while simultaneously the boundary exchange between GPUs is executed. When this boundary exchange is completed, the computations for the four boundaries are executed.

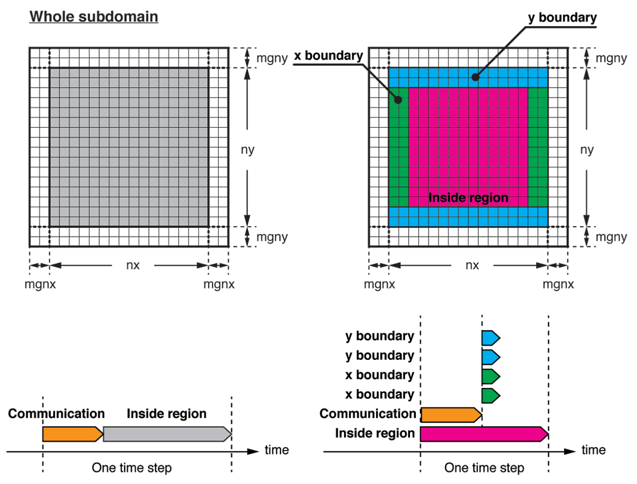


Fig. 4 Scheme of the overlapping method based on kernel division.

In order to apply the kernel-division overlapping method to the user program, the framework provides the `CompCommBinder` class, which is used to execute diffusion computations along with boundary exchange as follows:

```
BoundaryExchange *exchange = domain.exchange();
exchange->append(f);
CompCommBinder<Loop3D> ccbinder(exchange);
ccbinder.set_post_func(&loop,
    create_funcholder<Loop3D>(Diffusion3d(),
        ce, cw, cn, cs ,ct, cb, cc, f, fn));
ccbinder.set_use_overlapping();
ccbinder.run();
```

`CompCommBinder` is initialized with a `BoundaryExchange` object. To apply the overlapping method to a user-written function, by using `CompCommBinder::set_post_func`, the programmer specifies a loop range and the stencil function with additional parameters that are provided to this function. `CompCommBinder::run` divides the specific loop range into several loop ranges that correspond to boundaries and the inside region, and executes this stencil function on these loop ranges in several CUDA streams. Holding the parameters provided to the stencil function in `CompCommBinder` enables multiple executions of the stencil function.

Performance Evaluation of framework-based ASUCA 4

In this section, we present the strong and weak scaling results obtained by the weather prediction code ASUCA on the TSUBAME 2.5 supercomputer. The TSUBAME 2.5 supercomputer is equipped with 4224 NVIDIA Tesla K20X GPUs.

Figure 5 demonstrates the real case of the ASUCA operation with both the real initial and the boundary data used for the current weather forecast at the JMA. This simulation was performed with a 5,376 × 4,800 × 57 mesh with horizontal mesh resolution of 500 meters using 672 GPUs of the TSUBAME 2.5 in single precision.

Figure 6 shows the performance of ASUCA running on multiple GPUs in single precision and compares the strong scalability of the non-overlapping version and the overlapping version of ASUCA. In this graph, Flat-MPI version, in which each MPI process handles a single GPU, is also shown as reference. Since two of three GPUs on each TSUBAME node can utilize GPUDirect peer-to-peer access, we use these two GPUs per each node for these calculations. As shown in this figure, we observe that the overlapping method works effectively to hide communication cost for both mesh sizes we expected, resulting in performance improvement. In the results using a 3,072 × 2,560 × 60 mesh on 512 GPUs, the overlapping method achieves 18.9 TFlops.

Figure 7 shows the weak scaling results of the framework-based ASUCA code running on multiple GPUs. We measure the performance of ASUCA using both the overlapping method and non-overlapping methods in single precision. Bigger domain has better performance and we choose that each GPU handles the domain of 768 × 128 × 60. Unlike the strong scaling study, we use three GPUs per each node of TSUBAME 2.5 in all cases of calculations in order to maximize attainable performance by using TSUBAME 2.5. As shown in the graph, we achieve an extremely high performance of 209.6 TFlops using 4,108 GPUs with the overlapping method in single precision.

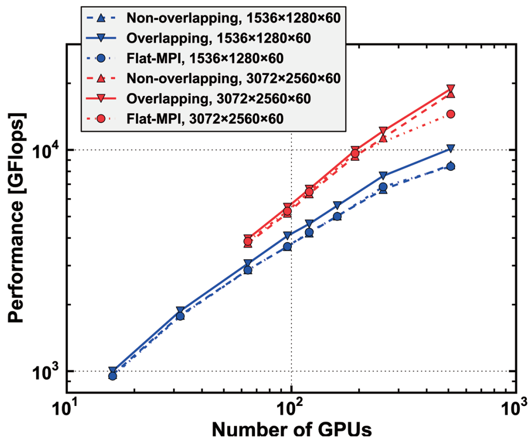


Fig. 6 Strong scaling results of ASUCA.

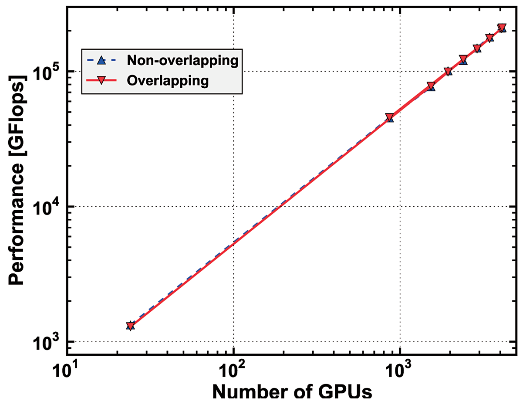


Fig. 7 Weak scaling results of ASUCA.

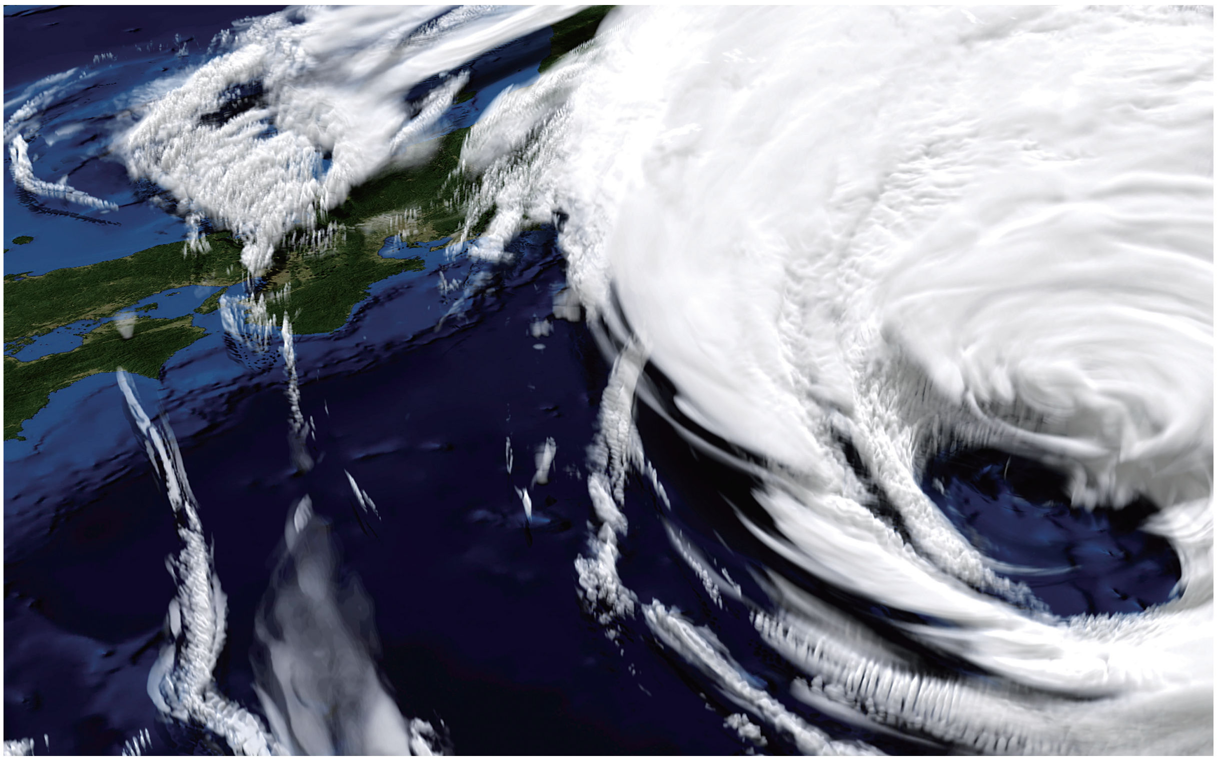


Fig. 5 ASUCA real operation to describe a typhoon with 5,376 × 4,800 × 57 mesh using 672 GPUs of the TSUBAME 2.5.

Summary 5

This article has presented the programming model and implementation of our framework that is developed for multi-GPU computation of stencil applications, and evaluation of the weather prediction code ASUCA based on the proposed framework running on a supercomputer equipped with multiple GPUs. The design of framework focuses on the portability of both framework and user code and cooperation with the existing codes. Our code can effectively utilize intra-node GPU peer-to-peer direct accesses with optimizations to hide communication overhead by overlapping of computation and communication. With our proposed framework, we have conducted performance studies using thousands of GPUs on the TSUBAME 2.5 supercomputer at Tokyo Institute of Technology. The performance evaluation has successfully demonstrated that strong and weak scalabilities are improved by the overlapping method provided by the framework.

Acknowledgements

This research was supported in part by KAKENHI, Grant-in-Aid for Young Scientists (B) 25870223, Grant-in-Aid for Scientific Research (B) 23360046 and Grant-in-Aid for Scientific Research (S) 26220002 from the Ministry of Education, Culture, Sports, Science and Technology (MEXT) of Japan, in part by the Japan Science and Technology Agency (JST) Core Research of Evolutional Science and Technology (CREST) research program ``Highly Productive, High Performance Application Frameworks for Post Petascale Computing'', and in part by "Joint Usage/Research Center for Interdisciplinary Large-scale Information Infrastructures" and "High Performance Computing Infrastructure" in Japan. The authors would like to thank Dr. Chiashi Muroi, Dr. Junichi Ishida, Dr. Kohei Kawano and Dr. Tabito Hara at the Japan Meteorological Agency the for providing the original ASUCA code, the real initial and boundary data.

References

[1] Takashi Shimokawabe, Takayuki Aoki, Chiashi Muroi Junichi Ishida, Kohei Kawano, Toshio Endo, Akira Nukada, Naoya Maruyama, and Satoshi Matsuoka, "An 80-Fold Speedup, 15.0 TFlops, Full GPU Acceleration of Non-Hydrostatic Weather Model ASUCA Production Code," in Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC'10, New Orleans, LA, USA, Nov 2010.

[2] Takashi Shimokawabe, Takayuki Aoki, Junichi Ishida, Kohei Kawano, and Chiashi Muroi, "145 TFlops Performance on 3990 GPUs of TSUBAME 2.0 Supercomputer for an Operational Weather Prediction," Procedia Computer Science, Volume 4, Proceedings of the International Conference on Computational Science, ICCS 2011, 2011, Pages 1535-1544.

[3] T. Shimokawabe, T. Aoki, T. Takaki, A. Yamanaka, A. Nukada, T. Endo, N., Maruyama, S. Matsuoka: Peta-scale Phase-Field Simulation for Dendritic Solidification on the TSUBAME 2.0 Supercomputer, in Proceedings of the 2011 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC'11, IEEE Computer Society, Seattle, WA, USA, Nov. 2011.

[4] Takashi Shimokawabe, Takayuki Aoki and Naoyuki Onodera, "A High-productivity Framework for Multi-GPU computation of Mesh-based applications," First International Workshop on High-Performance Stencil Computations (HiStencils), Vienna, Austria, Jan 2014

[5] Takashi Shimokawabe, Takayuki Aoki and Naoyuki Onodera "High-productivity Framework on GPU-rich Supercomputers for Operational Weather Prediction Code ASUCA," in Proceedings of the 2014 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC'14, New Orleans, LA, USA, Nov 2014. (to appear)

[6] J. C. Linford, J. Michalakes, M. Vachharajani, and A. Sandu, "Multi-core acceleration of chemical kinetics for simulation and prediction," in Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC'09. New York, NY, USA: ACM, 2009, pp. 1–11.

[7] P. Johnsen, M. Straka, M. Shapiro, A. Norton, and T. Galarneau, "Petascale WRF simulation of hurricane sandy deployment of NCSA's Cray XE6 Blue Waters," in Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC'13. New York, NY, USA: ACM, 2013, pp. 63:1–63:7.

[8] N. Maruyama, T. Nomura, K. Sato, and S. Matsuoka, "Physis: 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. New York, NY, USA: ACM, 2011, pp. 11:1–11:12.

● **TSUBAME e-Science Journal vol.12**

Published 9/22/2014 by GSIC, Tokyo Institute of Technology ©
ISSN 2185-6028
Design & Layout: Kick and Punch
Editor: TSUBAME e-Science Journal - Editorial room
Takayuki AOKI, Toshio WATANABE,
Atsushi SASAKI, Eri Nakagawa
Address: 2-12-1-E2-6 O-okayama, Meguro-ku, Tokyo 152-8550
Tel: +81-3-5734-2085 Fax: +81-3-5734-3198
E-mail: tsubame_j@sim.gsic.titech.ac.jp
URL: http://www.gsic.titech.ac.jp/

TSUBAME

vol. 12

International Research Collaboration

The high performance of supercomputer TSUBAME has been extended to the international arena. We promote international research collaborations using TSUBAME between researchers of Tokyo Institute of Technology and overseas research institutions as well as research groups worldwide.

Recent research collaborations using TSUBAME

1. Simulation of Tsunamis Generated by Earthquakes using Parallel Computing Technique
2. Numerical Simulation of Energy Conversion with MHD Plasma-fluid Flow
3. GPU computing for Computational Fluid Dynamics

Application Guidance

Candidates to initiate research collaborations are expected to conclude MOU (Memorandum of Understanding) with the partner organizations/ departments. Committee reviews the “Agreement for Collaboration” for joint research to ensure that the proposed research meet academic qualifications and contributions to international society. Overseas users must observe rules and regulations on using TSUBAME. User fees are paid by Tokyo Tech’s researcher as part of research collaboration. The results of joint research are expected to be released for academic publication.

Inquiry

Please see the following website for more details.
http://www.gsic.titech.ac.jp/en/InternationalCollaboration

GSIC
Global Scientific Information and Computing Center