IPSJ SIG Technical Report

Vol.2013-ARC-207 No.6
Vol.2013-HPC-142 No.6
2013/12/16

Scalability model for multi-GPU computation of stencil applications using regular structured meshes with explicit time integration

Takashi Shimokawabe[1,a)] Takayuki Aoki[1] Naoyuki Onodera[1]

Abstract: A multiple-GPU scalability model is presented for large-scale stencil applications with explicit time integration running on regular structured meshes. The cost to exchange the boundary data of the decomposed domain is the major factor to degrade the scalability. The presented model predicts this prior to application development. In this model, GPU computation time and both GPU-CPU and inter-node communication times are considered as the components of elapsed times in applications. This computation time is evaluated by the number of floating-point operations and the amount of memory access in an application. The communication times are evaluated as a function of transferred data size. For the evaluation of this model, a diffusion equation and a lattice Boltzmann method are performed on the TSUBAME2.0 supercomputer and a Cray XK6m system at Tokyo Institute of Technology. This model can sufficiently predict the results of measured performance in these applications.

1. Introduction

In the fields of parallel computing and high-performance computing, mesh-based physical simulations are one of the important applications running on supercomputers and large-scale PC clusters. Because of extremely memory-bottlenecked computation, these simulations are difficult problems on conventional supercomputers. A graphics processing unit (GPU) has been exploited as a high-performance computing device to accelerate a wide variety of applications and becomes an active research area in parallel computing in the last several years. Although GPU had been originally designed solely for graphics purpose, since Compute Unified Device Architecture (CUDA) [7] was released by NVIDIA as a general-purpose computing framework for GPU in 2006, programming GPUs for scientific computing has been made possible without using graphics-oriented APIs. It is well known that exploiting GPU that provides both large computation power and wide memory bandwidth can successfully accelerate scientific simulations dozens time faster than a conventional central processing unit (CPU) [1], [6], [10], [11], [12], [13]. Thanks to GPUs' large computation power and wide memory bandwidth at relatively-low power consumption peta-scale supercomputers such as TSUBAME 2.0 at Tokyo Institute of Technology, Japan [3], [4] and Titan at the Oak Ridge National Laboratory are equipped with more than a thousand GPUs, which provide most part of their computing performance, along with conventional CPUs.

To fully exploit the benefits of GPUs, the whole application should be executed on GPUs with minimizing the interaction with its host CPU and memory; all computational modules run on a GPU accessing the variables allocated on its video memory. This approach allows us to fully exploit the high performance and the wide bandwidth of GPUs. This implementation also virtually eliminates all host-GPU memory transfers during simulation runs, resulting in much larger performance improvements. To achieve the optimal performance with this approach, it is essential to rewrite the CPU code, which is often written in the Fortran language in the field of the scientific numerical simulations, to the GPU code from scratch in CUDA although the implementation cost is relatively high.

Using multiple GPUs is necessary when simulating large meshes beyond the size that is possible with a single GPU. In multi-GPU computation, we decompose the whole computational domain in directions and allocate each subdomain to a single GPU. Since the inter-node communication bandwidth is unlikely to be able to catching up the performance increase of massively parallel vector-oriented GPUs, increasing the performance with peta and post-peta scale heterogeneous machines is becoming more and more difficult primarily. Good parallel efficiency when using distributed-memory machines often requires careful programming techniques for hiding communication overhead by overlapping communication with computation, especially for strong-scaling cases. In our previous works [11], [12], we introduced the overlapping techniques and observed performance improvements.

Multi-GPU computation of mesh-based applications has the potential to achieve high performance. Introducing optimizations such as the overlapping techniques allow us to achieve optimal performance. The cost of implementation of these techniques, however, is higher than that of the naive implementation. In ad-

1 Tokyo Institute of Technology, Meguro, Tokyo, 152–8550, Japan
a) shimokawabe@sim.gsic.titech.ac.jp

dition, the performance of multi-GPU computing depends on not only its implementation but also on its runtime configuration such as computational domain size and decomposition configuration. For these reasons, it is useful to predict the reachable performance by an application employing the overlapping techniques and the performance increase from the performance obtained by using the naive implementation prior to application development. Scalability model are useful for this purpose.

In this paper, we present a scalability model specialized for multi-GPU computation of stencil applications with explicit time integration running on regular structured meshes. Thanks to this specialization, the proposed model can be simple and easily applied to intended mesh-based applications. In this model, an overlapping technique to hide communication cost with computation is taken into account since this technique is one of the major optimizations to improve the performance of the multi-GPU computation. In order to verify the scalability model, we compare the predicted performance evaluated by this model with the actual measured performance results of the following three stencil applications: a diffusion equation and a lattice Boltzmann method (LBM). For the evaluation of this scalability model, the TSUBAME 2.0 supercomputer at Tokyo Institute of Technology is mainly used. To evaluate this scalability model on other distributed GPU systems, in the case of the diffusion computation, a Cray XK6m system equipped with NVIDIA Tesla K20X GPUs is also used for our evaluation. As a result, the proposed scalability model for both non-overlapping and overlapping methods can sufficiently predict the results of measured performance in these mesh-based applications running on multi-GPU systems.

2. GPU supercomputers used for our evaluation

The scalability model presented in this paper targets GPU clusters and GPU-rich supercomputers. This section describes the multi-GPU systems used for our evaluation; the TSUBAME 2.0 supercomputers and the Cray XK6m system with NVIDIA Tesla K20X GPUs are described.

2.1 TSUBAME 2.0 supercomputer

We describe the GPU-rich supercomputer TSUBAME 2.0 with 4,224 GPUs at Tokyo Institute of Technology, which is mainly used for our evaluation of this scalability model. The main part of TSUBAME 2.0 supercomputer consists of 1,408 Hewlett-Packard Proliant SL390s nodes. Each node is equipped with three Tesla M2050 GPUs, thus the system has 4,224 GPUs in total. The nodes are designed to be bandwidth rich; each GPU is attached to distinct PCI Express bus 2.0 ×16 (8GB/s). Also each node has two sockets of Intel CPU Xeon X5670 (Westmere-EP) 2.93 GHz 6-core, 54GB DDR3 main memory, and two QDR InfiniBand HCAs (8GB/s in total). All the nodes are connected to the fat-tree interconnection with 200 Tbps bi-section bandwidth. In order to achieve multi-GPU computing on TSUBAME 2.0, we use OpenMPI version 1.4.2 for inter-node communication, and CUDA version 4.1 for GPU computation.

2.2 Cray XK6m system equipped with NVIDIA Tesla K20X GPUs

We provide the detail of the Cray XK6m system equipped with NVIDIA Tesla K20X GPUs, which is used for our evaluation of the proposed scalability model for the diffusion computation. Although NVIDIA Tesla X2090 GPUs are installed on the Cray XK6m system by default, all of the Tesla X2090 GPUs are replaced with NVIDIA Tesla K20X ones on the system we use.

The Cray system we use consists of 40 computational nodes. Each node has a NVIDIA Tesla K20X GPU; thus 40 GPUs are installed on the system. Also each node has a 16-core AMD Opteron 6272 processor (Interlagos) with 16 GB DDR3 main memory. A Tesla K20X GPU is connected to a node via PCI Express bus 2.0 ×16. All nodes are connected with the Cray's Gemini interconnect, the theoretical bandwidth of which reaches 9.3 GB/s. In order to perform multi-GPU computing on the Cray system, MPICH-2 optimized for the Cray XK6m system is used for inter-node communication, and CUDA version 5.0 is used for GPU computing.

3. Target of the scalability model: multi-GPU computing of mesh-based applications

This section provides the detail of multi-GPU computation for mesh-based applications for which the proposed model is intended. These mesh-based applications are performed with explicit time integration on regular structured meshes. First we describe a basic design of mash-based applications running on a single GPU. Next we describe the detail of large-scale computation over distributed GPUs.

3.1 Single GPU implementation

In order to fully exploit the benefits of GPUs, the entire part of an application should be executed on GPUs with minimizing the communication between host and device; we call this approach *full GPU computation*. Typically, the time integration of physical equations is performed as follows. In the beginning of the execution, the host CPU reads the initial data from the input files onto the host memory and executes initialization functions for these data. After that, the CPU transfers them to the video memory on a GPU board. The GPU carries out all the computational modules inside the time-step loop. When the results are obtained by GPU calculation, the minimal data are transferred to the host CPU memory, which reduces the communication between CPU and GPU. This approach makes us to fully use the computational performance of GPU.

3.2 Multi-GPU implementation

We describe a common strategy of multi-GPU computation in our applications. We exploit the distributed GPUs over InfiniBand-connected nodes using an MPI library. Similar to conventional multi-CPU computation on regular grids, the multi-GPU computation exploits domain decomposition, where the whole computational domain is decomposed to several pieces of subdomains and each subdomain is assigned to one GPU. As described in Section 3.1, GPUs carry out all the computational modules inside the time-step loop. Similar to conventional multi-CPU

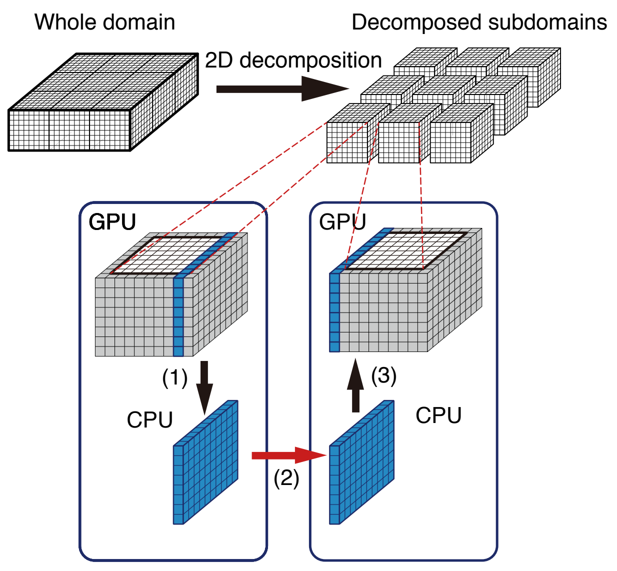

Fig. 1 Domain decomposition and the boundary data exchange between distributed GPUs for multi-GPU computation. Because a GPU cannot directly access to the global memory of other GPUs, host CPUs are used as bridges for data exchange.

implementations with MPI, our multi-GPU implementations require boundary data exchanges between subdomains. Because a GPU on a node cannot directly access the global memory of other GPUs on other nodes, host CPUs are used as bridges for data exchange. For inter-node cases, this boundary data exchange is composed of the following three steps: (1) data transfer from the GPU to the CPU by using CUDA APIs such as `cudaMemcpy`, (2) data exchange between nodes with MPI APIs such as `MPI_Isend` and `MPI_Irecv`, (3) data transfer back from the CPU to the GPU by using CUDA APIs. **Figure 1** shows that two-dimensional (2D) domain decomposition and the boundary data exchange between distributed GPUs by using above three steps. In our implementation, we usually allocate halo regions (gray elements in the figure) to be utilized to store data sent from neighbor GPUs, which data are depicted as blue elements. Although halo regions are depicted as just one element thick in this figure, the thickness depends on numerical schemes.

In naive multi-GPU implementation, each GPU computes specified subdomain and then the boundary regions of the subdomains are exchanged between GPUs by using above three-step communication. We call this implementation non-overlapping method. However, this basic method suffers from costs of three-step data transfer described above. Their impact gets larger when we use more GPUs.

3.3 Overlapping method for multi-GPU computation

Since GPU computation is often dozens time faster than conventional CPU computation in mesh-based applications, hiding communication by computation is necessary to reach optimal performance especially in multi-GPU computation. Since each element of a variable can be computed independently with each other in a time step, we can compute the boundary regions of subdomains separately from the rest parts. In an overlapping method, by dividing each subdomain into several boundaries and an inside

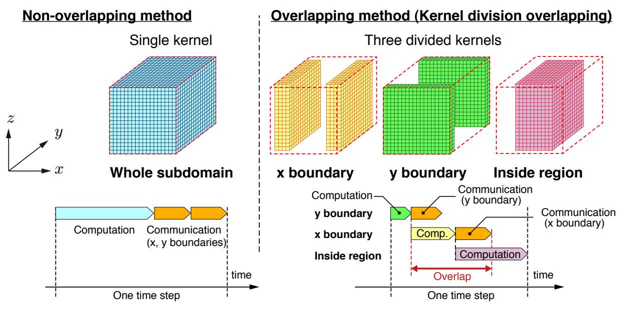

Fig. 2 Non-overlapping method and overlapping method by the kernel division. Figure adapted from Shimokawabe et al [12].

region, we can overlap communication for the boundary data exchange with the computation of the inside region. Our previous work [11], [12] reported overlapping methods contribute to improving overall application performance.

In this paper, we exploit an overlapping method described as follows. A single GPU kernel is divided into several kernels for the boundaries and the inside region. GPU computation for the inside region is performed in parallel with the boundary data exchange, resulting in hiding communication costs. All calculations for a subdomain are executed on a GPU. **Figure 2** shows the basic concept of the kernel division overlapping in 2D decomposition where a whole computational domain is decomposed in both x- and y-directions. This overlapping method is introduced in our previous work [11], [12].

4. Improved Roofline model: a performance prediction model for single-GPU computation

Our final goal is to present a scalability model for mesh-based applications running over distributed GPUs. First, however, we shall describe a performance model for single-GPU computation. In our previous paper [11], we proposed and used a performance model for single-GPU computation named *Improved Roofline model*. Since prediction of reachable performance for multi-GPU computation by the proposed scalability model is based on performance of single-GPU computation, we can use the attainable performance evaluated by this model as the value of single-GPU performance instead of an actual measured value of single-GPU performance. In this section, we review the Improved Roofline model.

The Improved Roofline model is a model that predict attainable performance of an application executed on a device. In this model, this attainable performance P is evaluated as a following equation:

$$P = \frac{F}{F/F_{peak} + B/B_{peak} + \alpha} = \frac{F/B}{F/B + F_{peak}/B_{peak} + \alpha F_{peak}/B} F_{peak}, \quad (1)$$

where F is the number of floating-point operations in the application, B is the amount of memory access in bytes in the application, F_{peak} is the peak performance of floating-point operation of the device and B_{peak} is the peak memory bandwidth of the device. F_{peak} and B_{peak} are device-dependent constant values. Here, F/F_{peak}, B/B_{peak} and α represent the times taken by floating-point operations, memory access operations, and other

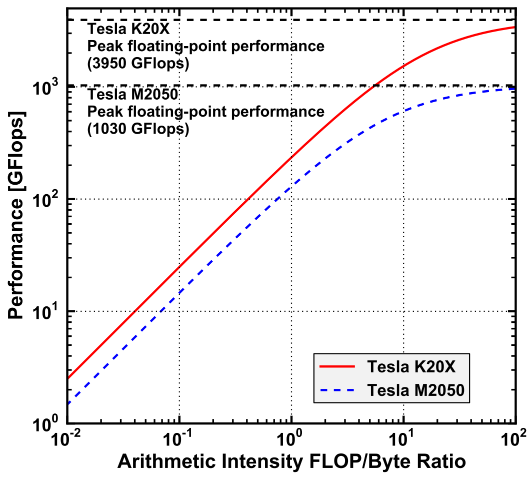

Fig. 3 Improved Roofline model for a NVIDIA Tesla M2050 and a NVIDIA Tesla K20X.

operations except both floating-point and memory access operations, respectively. The term F/B is called arithmetic intensity, which is the ratio of floating-point operations per byte of memory accessed. Hereafter α is zero because of simplification. **Figure 3** shows the Improved Roofline model for a NVIDIA Tesla M2050 and a NVIDIA Tesla K20X in single precision. The former has a theoretical floating-ponit performance of 1030 GFlops and a theoretical memory bandwidth of 148 GB/s. The latter has a theoretical floating-ponit performance of 3950 GFlops and a theoretical memory bandwidth of 250 GB/s. These theoretical values are used as F_{peak} and B_{peak}.

Since this model does not depend on the characteristics of GPU architecture, it can also be applied to conventional CPU computation as well as GPU one. Although this model is similar to the Roofline model [8], [14], it can evaluate the estimated attainable performance slightly accurately in comparison with the Roofline model since the equation we used is based on the times taken by computation and memory operation.

5. Scalability model for multi-GPU computation of mesh-based applications

This section describes the scalability model proposed in this paper. This model can be applied to the mesh-based applications described as follows.

- The mesh-based applications must be performed with explicit time integration. This model assumes that computation and communication take specific and same amounts of times each timestep. The scalability of applications using implicit time integration, utilizing, e.g., Poisson's equation, cannot be predicted by this model since the amount of computation change every timestep.
- The applications must run on regular structured meshes. The model cannot predict the scalability of the multi-GPU applications running on unstructured meshes since the estimation of the communication costs depends on shapes of the unstructured meshes.
- The applications must be run over distributed GPUs and each GPU computes the same size of subdomain; the computation is load-balanced over GPUs. This model also assumes that each subdomain transfers the same amount of data.
- The applications can utilize the overlapping technique to hide communication cost with computation as an optimization. Introducing the overlapping technique to mesh-based applications contributes to improving performance and reaching the optimal performance.

First, the scalability model for mesh-based applications using the naive implementation, i.e., the non-overlapping version, is described. Based on this, the model for the overlapping version is described.

5.1 Non-overlapping method

In multi-GPU computation, the elapsed time in an application is mainly consist of GPU computation time and communication times including the GPU-CPU communication time and the internode communication time. The proposed model evaluates the attainable performance by using these components of the elapsed time and the amount of computation. In the non-overlapping method, the performance of the multi-GPU computation is evaluated as the following expression:

$$P_m = \frac{VR}{V/P + \sum_j \sum_m s_j/B_m(s_j)}, \tag{2}$$

where V is the number of floating-point operations required for updating a time step on all grid points within a subdomain and P is the performance achieved by the single-GPU computation of the application. As a single-GPU performance P, we can use either an estimated value or an actual measured value. The estimated value of performance can be calculated easily from the arithmetic intensity of the application by using the Improved Roofline model. When single-GPU code is already developed and able to run on a single GPU, using an actual measured value of performance of the application allows us to predict the performance of multi-GPU computation more accurately than using the estimated value. The first term in the denominator, i.e., V/P, represents the time taken by computation. Here, R is the number of subdomains, which is equal to the number of GPUs. The term VR represents the number of floating-point operations required for updating a time step within the whole computational domain. B_m is bandwidth of any path m between two neighbor GPUs and s_j is the amount of transferred data between these GPUs in byte for the data j. Since effective bandwidth usually depends on the amount of transferred data, B_m is function of s_j. We determine B_m by using a bandwidth model described later in this section. We estimate the communication time elapsed in the application using the sum of $s_j/B_m(s_j)$ over j and m.

In this paper, since we use distributed GPU systems for the evaluation of this scalability model, the second term in the denominator can reduce to a following equation:

$$\sum_j \sum_m s_j/B_m(s_j) = \sum_j s_j \left[2g/B_{\mathrm{IB}}(s_j) + 2/B_{\mathrm{PCIe}}(s_j)\right], \tag{3}$$

where B_{IB} is bandwidth between two neighbor nodes (i.e., the InfiniBand bandwidth in the case of TSUBAME 2.0), B_{PCIe} is bandwidth between GPU and CPU (i.e., the bandwidth of the PCI Express bus) and g is the number of GPU on each node. Since bidirectional communication by g GPUs on each node share InfiniBand path, $B_{\mathrm{IB}}/2g$ is used as the actual bandwidth of inter-node communication. Since three-step communication described above includes both GPU to CPU communication and CPU to GPU communication, GPU-CPU communication through PCI Express bus takes $2/B_{\mathrm{PCIe}}$ for one time step instead of $1/B_{\mathrm{PCIe}}$.

5.2 Overlapping method

By using the overlapping method described in Section 3.3, the communication costs such as GPU-CPU communication and inter-node communication are hidden by the GPU computation. In order to hide the communication for boundary data transfer, the GPU computation for boundary regions must be performed first. This means that this computation can not be executed in parallel with the communication. However, since the thickness of the boundary regions is usually much thinner than that of the inside region, the computational time of the boundary regions is sufficiently shorter than that of the inside region. Thus we ignore this computational time and assume all computation may be performed in parallel with the communication for simplicity. By using the overlapping method, the longest of computational time and communication time is observed as an actual elapsed time for computing one time step. The scalability model for the overlapping method is described as follows:

$$P_m = \frac{VR}{\max\left(V/P, \sum_j \sum_m s_j/B_m(s_j)\right)}. \tag{4}$$

Assuming a strong scaling case, while V is in inverse proportion to R, s_j is typically in inverse proportion to $R^{1/d}$, where d is the dimension of decomposition. Thus, when a small number of GPU are used, $V/P > \sum_j \sum_m s_j/B_m(s_j)$ is satisfied; we obtain $P_m = RP$. When the number of GPUs used is larger, we obtain $P_m = VR/(\sum_j \sum_m s_j/B_m(s_j))$, which is typically in inverse proportion to $R^{1/d}$ and becomes smaller.

5.3 Inter-node and GPU-CPU bandwidth

We evaluate the inter-node communication bandwidth B_{IB} and the GPU-CPU communication bandwidth B_{PCIe} used in Equation (3) as follows. Generally, latency can have a negative effect on actual measured communication speed. For simplicity, we assume that the time taken by communication is composed of the actual elapsed time for data transfer and latency. Based on this, bandwidth $B(s)$ that is a function of an amount of transferred data is written as follows:

$$B(s) = \frac{s}{s/B_0 + t_0} = \frac{s}{s + B_0 t_0} B_0, \tag{5}$$

where B_0 is device-specific peak bandwidth and t_0 is device-specific latency.

For each device, we measure its bandwidth varying the amount of transferred data. Then, by fitting these measured values of the bandwidth with Equation (5), we determine B_0 and t_0 for a specific device. The transfer speed of the inter-node communication with InfiniBand is often improved by using page-aligned memory for its buffer, which is allocated by `valloc` instead of `malloc`. Thus we evaluate the bandwidth of the InfiniBand inter-node connection using aligned memory and non-aligned memory

Table 1 Parameters used for bandwidth models on TSUBAME 2.0.

InfiniBand (non-aligned memory)	B_0	3.67 GByte/sec
	t_0	6.07 μsec
InfiniBand (aligned memory)	B_0	5.80 GByte/sec
	t_0	7.47 μsec
PCI Express bus	B_0	4.29 GByte/sec
	t_0	16.9 μsec

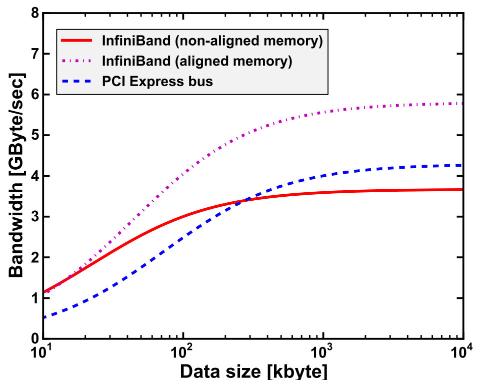

Fig. 4 Bandwidth models for the inter-node communication and the GPU-CPU communication on TSUBAME 2.0. We evaluate the bandwidth of the InfiniBand inter-node connection using aligned memory and non-aligned memory.

Table 2 Parameters used for bandwidth models on the Cray XK6m system equipped with NVIDIA Tesla K20X GPUs.

InfiniBand	B_0	9.87 GByte/sec
	t_0	3.75 μsec
PCI Express bus	B_0	6.34 GByte/sec
	t_0	41.5 μsec

respectively. **Table 1** shows B_0 and t_0 used for our evaluation on TSUBAME 2.0. **Figure 4** shows the bandwidth models evaluated by Equation (5) for the GPU-CPU communication and the InfiniBand communication with aligned memory and with non-aligned memory on TSUBAME 2.0. **Table 2** shows B_0 and t_0 used for our evaluation on the Cray XK6m system equipped with NVIDIA Tesla K20X GPUs and **Figure 5** shows the bandwidth models for this system evaluated by Equation (5). Note that since InfiniBand communication without explicit use of aligned memory on the Cray XK6m system has achieved almost the same performance as that with the aligned memory, the aligned memory is not used explicitly for the InfiniBand communication for computing on the Cray system.

6. Evaluation of scalability model and Discussion

In this section, we present the evaluations of the scalability model for a diffusion equation and a LBM.

First, by using the proposed scalability model, we evaluate three-dimensional diffusion equation running on two different multi-GPU systems; the TSUBAME 2.0 supercomputer and the Cray XK6m system equipped with NVIDIA Tesla K20X GPUs are used for this evaluation. The proposed scalability model requires a single-GPU performance P, which can be estimated by using the Improved Roofline model. We have chosen 2D decomposition for these applications since 3D decomposition, which looks better to reduce communication amount, tends to degrade

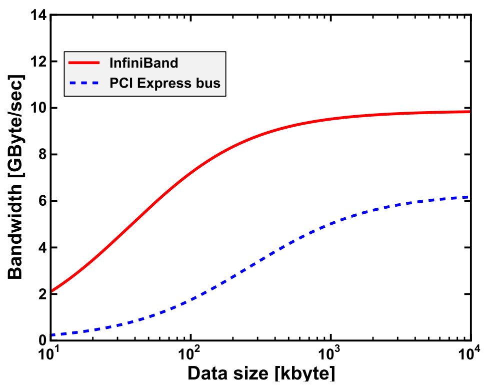

Fig. 5 Bandwidth models for the inter-node communication and the GPU-CPU communication on the Cray XK6m system equipped with NVIDIA Tesla K20X GPUs.

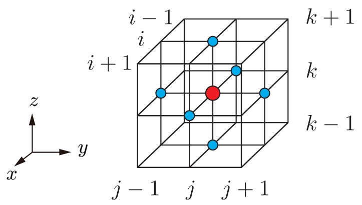

Fig. 6 Spatial access pattern of neighbor points for the diffusion equation.

GPU performance due to complicated memory access patterns for data exchanges between GPU and CPU.

For evaluation of the proposed scalability model for a real mesh-based application, we use three-dimensional LBM running on multiple GPUs of TSUBAME 2.0. Similar to the diffusion computation, 2D decomposition is chosen for this application.

6.1 Diffusion equation

6.1.1 Governing equation and implementation

The diffusion equation is one of the important partial differential equations, which provides the foundation for solving several physical equations such as the Navier-Stokes equations. Solving this equation on a mesh is a typical stencil computation.

The three-dimensional diffusion equation is described as follows:

$$\frac{\partial f(\boldsymbol{r},t)}{\partial t} = \kappa \nabla^2 f(\boldsymbol{r},t), \tag{6}$$

where $f(\boldsymbol{r},t)$ is a physical quantity at location $\boldsymbol{r}$ and time t and κ is the diffusion coefficient for the physical quantity f. The time integration of f given by Equation (6) is carried out by the second-order finite-difference scheme for space and the first-order forward Euler-type finite-difference method for time on a three-dimensional regular computational grid. In the discretized equation we use, 13 floating-point operations are needed to update each point of f; F in Equation (1) is 13 for this computation. **Figure 6** shows that the spatial patterns of the neighbor points of f that are required to solve the discretized governing equations for the center point of the grid.

6.1.2 Evaluation of scalability model on the TSUBAME 2.0 supercomputer

In order to evaluate the scalability model for diffusion equation, we perform simulations in single precision on TSUBAME 2.0 for three different mesh sizes: 512^3, 1024^3, and 2048^3 varying the number of GPUs used for the calculations; these are strong scaling measurements. Since diffusion equation requires three stencil access in each direction, a one element thick mesh is transferred for all directions. Since 2D decomposition is chosen for this computation, the size of data that each subdomain transfers to others is $4\frac{N^2}{\sqrt{R}}$ byte for a given computational domain N^3. We use three GPUs per each TSUBAME node for these calculations, i.e., $g = 3$ in above equations. In this application, we use aligned memory for the buffer of the InfiniBand inter-node connection.

In order to evaluate the diffusion computation by using the proposed scalability model, the single-GPU performance is estimated by using the Improved Roofline model. Since an arithmetic intensity F/B of this computation is 0.41, we obtain an estimated performance of 56.8 GFlops by using Equation (1) for a Tesla M2050. **Figure 7** shows the predicted scalability obtained by the proposed scalability model with using the above estimated performance and the measured performance of diffusion computation over distributed GPUs. We shows both results of the overlapping method and the non-overlapping method. The overlapping method successfully improves the application performance compared with the non-overlapping method since the communication cost is hidden. As shown in this figure, the proposed scalability model for both methods can sufficiently predict the results of measured performance.

The computational time for a subdomain on a GPU is in proportion to the volume of this subdomain, while the communication time is approximately in proportion to that of the boundary region of this subdomain. When the number of GPUs in use is larger, the volume that each GPU handles becomes smaller, resulting in an increase of the percentage of the boundary region in the whole computational domain. Thus, when a larger number of GPUs is exploited, the communication time is relatively longer than the computation time and eventually the communication cost can not be hidden perfectly by the computation. In that case, the communication time characterizes the elapsed time, which slows down the performance increasing. When the number of GPUs used is relatively small, the communication is overlapped with the computation, which allows us to achieve optimal performance resulting in the good scaling. As shown in this figure, the scalability model for the overlapping method represents these characteristics.

We find the apparent slower performance of both overlapping and non-overlapping methods on the mesh size of 2048^3 than the estimated scalability by the proposed model when we use more than 300 GPUs. We consider the major reason comes from slow MPI communication because we observe the fluctuations in MPI communication speed occur in our application running on TSUBAME 2.0. This slow MPI communication was also observed in our previous simulations reported in [12].

6.1.3 Evaluation of scalability model on the Cray system

In order to evaluate the proposed scalability model on other distributed GPU systems, we perform the diffusion computations on a Cray XK6m system equipped with NVIDIA Tesla K20X

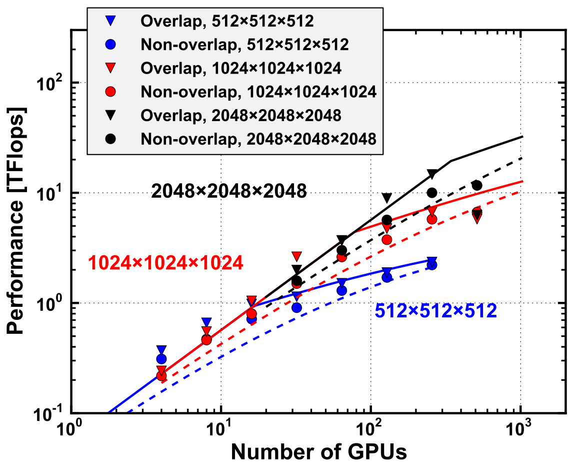

Fig. 7 Measured performance and predicted performance of diffusion computation over multiple GPUs. As the performance of the single-GPU computation, the estimated value by the Improved Roofline model is used. The solid and dashed lines shows predicted performance with the overlapping and non-overlapping methods, respectively. The results of actual measured performance with the overlapping and non-overlapping methods are depicted as triangle and circle points, respectively. Three different colors represent the different mesh sizes, i.e., 512^3 (blue), 1024^3 (red) and 2048^3 (black).

GPUs and compare these performance results with the scalability evaluated by the proposed model. We use the same code as that used for TSUBAME 2.0. Due to the limitation in the number of GPUs on the Cray system, we perform simulations in single precision for two different mesh sizes: 512^3 and 1024^3. Unlike the computations on TSUBAME 2.0, we use one GPU per each Cray node for these calculations.

Figure 8 shows the predicted scalability of the diffusion computation on the Cray system obtained by the proposed scalability model with using the estimated single-GPU performance by the Improved Roofline model. Due to an arithmetic intensity $F/B = 0.41$, we obtain an estimated performance of 99.0 GFlops by using Equation (1) for a Tesla K20X. The measured performance data of diffusion computation over the distributed GPUs of the Cray system are also depicted. We shows both results of the overlapping method and the non-overlapping method. Similar to the case of TSUBAME 2.0, the scalability model can successfully predict performance and represent the characteristics of the scalability of both overlapping and non-overlapping methods.

6.2 Lattice Boltzmann method

6.2.1 Governing equation and implementation

The lattice Boltzmann method (LBM) [2], [5], [9] is based on the Boltzmann equation and an alternative method to Navier-Stokes based methods for computation fluid dynamics. LBM is exploited in many areas of fluid dynamics. We choose this method for the evaluation as an example of real applications, which have more complicated computations than diffusion computation. **Figure 9** shows a snapshot of computational results of large-scale wind simulations in widespread urban area performed by multi-GPU computation of LBM.

In order to evaluate the proposed scalability model for LBM, we develop GPU implementation of LBM based on the literature [15]. We provide a brief explanation of LBM we use. A

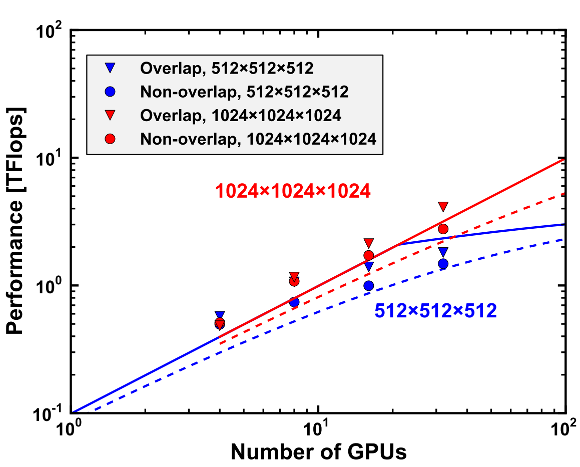

Fig. 8 The same as Fig. 7, but obtained on the Cray system instead of TSUBAME 2.0. The estimated performance is used as a single-GPU performance.

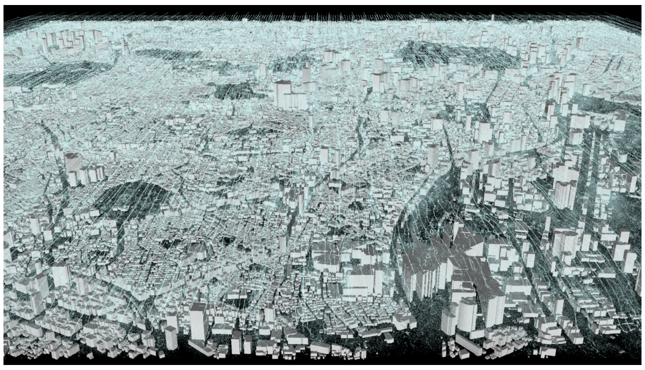

Fig. 9 A snapshot of computational results of large-scale wind simulations in widespread urban area performed by multi-GPU computation of LBM.

discrete-velocity Boltzmann equation is described as follows:

$$f_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) = f_i(\boldsymbol{x}, t) - \frac{1}{\tau}(f_i(\boldsymbol{x}, t) - f_i^{\mathrm{eq}}(\boldsymbol{x}, t)) + F_i(\boldsymbol{x}, t), \quad (7)$$

where f_i is the density distribution function with discrete velocity c_i along the i direction. Here, f^{eq} is the equilibrium distribution function and τ is relaxation time due to fluid particle collisions. The standard *D3Q19* LBM, which model exploits 19 discrete velocities in three dimensions, is used for our computation; $\boldsymbol{c}_0$ to $\boldsymbol{c}_{18}$ are defined in this model. **Figure 10** shows that the distribution of these discrete velocities for each point. The equilibrium distribution function for incompressible flow are written as follows:

$$f_i^{\mathrm{eq}} = w_i\rho\left(1 + \frac{3\boldsymbol{c}_i\cdot\boldsymbol{u}}{c^2} + \frac{9(\boldsymbol{c}_i\cdot\boldsymbol{u})^2}{2c^4} - \frac{3u^2}{2c^2}\right), \quad (8)$$

where ρ is density, w_i is weighting factor, $\boldsymbol{u} = \frac{1}{\rho}\sum_i \boldsymbol{c}_i f_i$ is velocity, and $c = \Delta x/\Delta t = 1$ in lattice units (i.e., $\Delta t = \Delta x = 1$). In our implementation, computation of above equation is performed on a three-dimensional regular computational grid and needs 476 floating-point operations to update each point of the grid in every time step.

6.2.2 Evaluation of scalability model

In order to evaluate the scalability model for LBM, we perform simulations in single precision for three different mesh sizes: $192 \times 512 \times 512$, $192 \times 2048 \times 2048$, and $192 \times 4096 \times 4096$

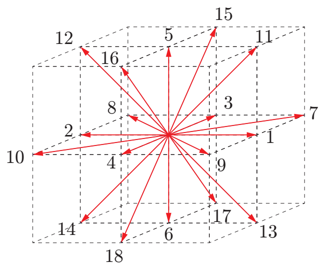

Fig. 10 Discrete velocities used in the D3Q19 LBM.

varying the number of GPUs used for the calculations; these are strong scaling measurements. Unlike the diffusion computation, the x dimension is relatively small since we perform large-scale wind simulations in widespread urban area by using LBM. We use TSUBAME 2.0 for this evaluation. In our LBM implementation, 13 single-precision floating-point variables are transferred at each point of the boundary region of subdomains in one timestep. We use three GPUs per each TSUBAME node for these calculations. Unlike the diffusion calculation, we use non-aligned memory for the buffer of the InfiniBand inter-node connection in this application.

We predict the scalability of LBM computation over multiple GPUs by using an estimated performance as single-GPU performance P. We use the Improved Roofline model to obtain the estimated performance. Since an arithmetic intensity F/B of this computation is 1.83, we obtain an estimated performance of 214.5 GFlops by using Equation (1) for a Tesla M2050; this performance is used for evaluating the scalability. Note that our implementation of LBM computation has achieved 198.0 GFlops on a Tesla M2050, which corresponds to 450.6 MLUPS since 476 floating-point operations are needed for updating each point of the grid in every time step. The implementation of LBM computation is more complicated than that of diffusion computation. The complexity of LBM computation is likely to suppress the effect of cache more than that of diffusion computation. Thus the estimation in the Improved Roofline model fits the LBM computation and the attainable performance of this computation is evaluated more accurately compared with the case of the diffusion computation.

Figure 11 shows the scalability obtained by the proposed scalability model with the estimated performance of the single-GPU computation of LBM. We show both results of the overlapping method and the non-overlapping method. As shown in this figure, the proposed scalability model successfully predicts the scalability of multi-GPU computation and its characteristics. The proposed scalability model predicts that the overlapping method can improve the application performance compared with the non-overlapping method, which improvement is actually observed in measured data depicted in this figure. Especially in the result of $192 \times 512 \times 512$, the scalability model can clearly represent the characteristics of the slowing down of the performance improvements in the overlapping method.

7. Conclusion

In this paper, we have presented a scalability model for mesh-based applications running on distributed GPUs systems. Multi-GPU computation is often more affected by communication than multi-CPU computation. In order to reduce performance degradation by communication, multi-GPU computation employs the overlapping techniques to hide communication cost as optimization. Since the cost of implementation of these overlapping techniques is often higher than the naive implementation, it is useful to predict the scalabilities of applications and their characteristics prior to application development.

The proposed scalability model is intended to multi-GPU computations of stencil applications with explicit time integration running on regular structured meshes. This model takes into account GPU computation time and the communication time as the main components of the elapsed time in applications. Since the overlapping method is often used as optimization in multi-GPU computation, we present the overlapping version of the model as well as the non-overlapping version. In order to predict performance accurately, the bandwidth values used in the scalability model are evaluated as a function of an amount of the transferred data. The proposed scalability model for multi-GPU computation is evaluated based on the estimated performance of the single-GPU computation. The Improved Roofline model is used for this estimation in this paper.

As evaluation of the scalability model, we compare the predicted scalability with measured performance in diffusion computation and LBM. The diffusion computation is a fundamental stencil-computation and is performed on two different multi-GPU systems: the TSUBAME 2.0 supercomputer and the Cray XK6m system equipped with NVIDIA Tesla K20X GPUs. As a real mesh-based application, LBM is performed on TSUBAME 2.0 and this result is used for the evaluation of the proposed scalability model. As results, the proposed scalability model for both overlapping and non-overlapping methods can sufficiently predict the results of measured performance and represent the char-

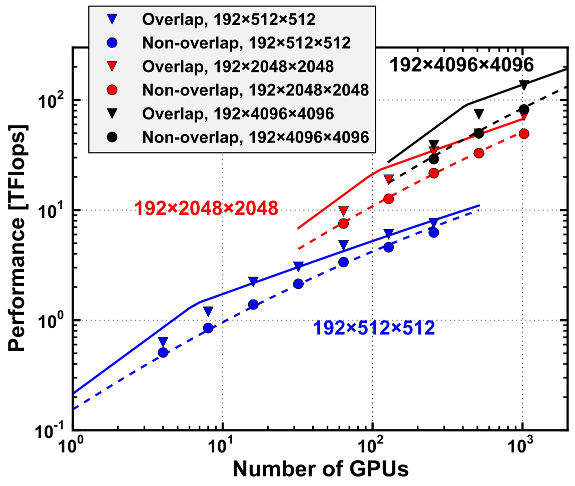

Fig. 11 Measured performance and predicted performance of LBM computation over multiple GPUs. As the performance of the single-GPU computation, the estimated value by the Improved Roofline model is used. The solid and dashed lines shows predicted performance with the overlapping and non-overlapping methods, respectively. The results of actual measured performance with the overlapping and non-overlapping methods are depicted as triangle and circle points, respectively. Three different colors represent the different mesh sizes, i.e., $192 \times 512 \times 512$ (blue), $192 \times 2048 \times 2048$ (red) and $192 \times 4096 \times 4096$ (black).

acteristics of scalabilities in two applications on several different systems using multiple GPUs. It shows our model is helpful in selecting suitable methods and optimization in terms of reachable performance and implementation cost.

Acknowledgments 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 Young Scientists (B) 25870226 from the Ministry of Education, Culture, Sports, Science and Technology (MEXT) of Japan, and in part by the Japan Science and Technology Agency (JST) Core Research of Evolutional Science and Technology (CREST) research programs "ULP-HPC: Ultra Low-Power, High-Performance Computing via Modeling and Optimization of Next Generation HPC Technologies" and "Highly Productive, High Performance Application Frameworks for Post Petascale Computing". The authors thank the Global Scientific Information and Computing Center, the Tokyo Institute of Technology for use of the resources of the TSUBAME2.0 supercomputer and the Cray XK6m system.

References

[1] Aoki, T., Ogawa, S. and Yamanaka, A.: Multiple-GPU Scalability of Phase-Field Simulation for Dendritic Solidification, *Progress in Nuclear Science and Technology*, Vol. 2, pp. 639–642 (2011).

[2] Chen, H., Chen, S. and Matthaeus, W. H.: Recovery of the Navier-Stokes equations using a lattice-gas Boltzmann method, *Phys. Rev. A*, Vol. 45, pp. R5339–R5342 (online), DOI: 10.1103/PhysRevA.45.R5339 (1992).

[3] Endo, T., Nukada, A., Matsuoka, S. and Maruyama, N.: Linpack Evaluation on a Supercomputer with Heterogeneous Accelerators, *Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS'10)*, Atlanta, GA, USA, IEEE (2010).

[4] Global Scientific Information and Computing Center, Tokyo Institute of Technology: TSUBAME hardware software specifications, `http://www.gsic.titech.ac.jp/sites/default/files/TSUBAME_SPECIFICATIONS_en_0.pdf` (2011).

[5] McNamara, G. R. and Zanetti, G.: Use of the Boltzmann Equation to Simulate Lattice-Gas Automata, *Phys. Rev. Lett.*, Vol. 61, pp. 2332–2335 (online), DOI: 10.1103/PhysRevLett.61.2332 (1988).

[6] Nukada, A. and Matsuoka, S.: Auto-tuning 3-D FFT library for CUDA GPUs, *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, New York, NY, USA, ACM, pp. 1–10 (online), DOI: http://doi.acm.org/10.1145/1654059.1654090 (2009).

[7] NVIDIA: CUDA C Programming Guide 4.0, `http://developer.download.nvidia.com/compute/cuda/4_0/toolkit/docs/CUDA_C_Programming_Guide.pdf` (2011).

[8] Patterson, D. A. and Hennessy, J. L.: *Computer Organization and Design, Fourth Edition: The Hardware/Software Interface (The Morgan Kaufmann Series in Computer Architecture and Design)*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 4th edition (2008).

[9] Qian, Y. H., D'Humières, D. and Lallemand, P.: Lattice BGK Models for Navier-Stokes Equation, *EPL (Europhysics Letters)*, Vol. 17, No. 6, pp. 479–484 (online), available from ⟨http://stacks.iop.org/0295-5075/17/i=6/a=001⟩ (1992).

[10] Shimokawabe, T., Aoki, T., Ishida, J., Kawano, K. and Muroi, C.: 145 TFlops Performance on 3990 GPUs of TSUBAME 2.0 Supercomputer for an Operational Weather Prediction, *Procedia Computer Science*, Vol. 4, pp. 1535 – 1544 (online), DOI: DOI: 10.1016/j.procs.2011.04.166 (2011). Proceedings of the International Conference on Computational Science, ICCS 2011.

[11] Shimokawabe, T., Aoki, T., Muroi, C., Ishida, J., Kawano, K., Endo, T., Nukada, A., Maruyama, N. and Matsuoka, S.: An 80-Fold Speedup, 15.0 TFlops Full GPU Acceleration of Non-Hydrostatic Weather Model ASUCA Production Code, *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, New Orleans, LA, USA, IEEE Computer Society, pp. 1–11 (online), DOI: http://dx.doi.org/10.1109/SC.2010.9 (2010).

[12] Shimokawabe, T., Aoki, T., Takaki, T., Yamanaka, A., Nukada, A., Endo, T., Maruyama, N. and Matsuoka, S.: Peta-scale Phase-Field Simulation for Dendritic Solidification on the TSUBAME 2.0 Supercomputer, *Proceedings of the 2011 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, Seattle, WA, USA, ACM, pp. 1–11 (2011).

[13] Wang, X. and Aoki, T.: Multi-GPU Performance of Incompressible Flow Computation by Lattice Boltzmann Method on GPU Cluster, *Parallel Computing*, Vol. 37, No. 9, pp. 521–535 (2011).

[14] Williams, S., Waterman, A. and Patterson, D.: Roofline: an insightful visual performance model for multicore architectures, *Commun. ACM*, Vol. 52, No. 4, pp. 65–76 (online), DOI: http://doi.acm.org/10.1145/1498765.1498785 (2009).

[15] Yu, H., Girimaji, S. S. and Luo, L.-S.: DNS and LES of decaying isotropic turbulence with and without frame rotation using lattice Boltzmann method, *J. Comput. Phys.*, Vol. 209, No. 2, pp. 599–616 (online), DOI: 10.1016/j.jcp.2005.03.022 (2005).