e-Science Journal
Global Scientific Information and Computing Center

Vol. 5
SC'11 Special Issue

TSUBAME ESJ.

SC'11 Special Issue

Gordon Bell Prize

Special Achievements in Scalability and Time-to-Solution
Honorable Mention

Graph500 Ranking No.3

3 Technical Papers

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

SC'11 Special Issue

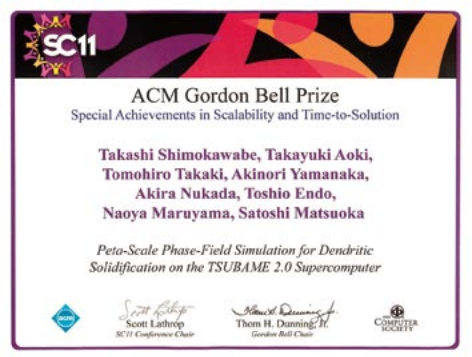
03

ACM Gordon Bell Prize:
Special Achievements in Scalability and Time-to-Solution

& SC'11 Technical Paper

Peta-scale Phase-Field Simulation for Dendritic Solidification on the TSUBAME 2.0 Supercomputer

Takashi Shimokawabe Takayuki Aoki Tomohiro Takaki Akinori Yamanaka
Akira Nukada Toshio Endo Naoya Maruyama Satoshi Matsuoka



10

ACM Gordon Bell Prize: Honorable Mention

Large scale biofluidics simulations on TSUBAME2

Massimo Bernaschi Mauro Bisson Toshio Endo
Massimiliano Fatica Satoshi Matsuoka Simone Melchionna Sauro Succi

SC11
Seattle, WA
Connecting communities through HPC

17

Graph500 Ranking No.3

Graph500 Challenge on TSUBAME 2.0

Toyotaro Suzumura　Koji Ueno

21

SC'11 Technical Paper

Physis: A High-Level Stencil Framework for Heterogeneous Supercomputers

Naoya Maruyama　Tatsuo Nomura　Kento Sato　Satoshi Matsuoka

26

SC'11 Technical Paper (Achieving a Perfect Score)

FTI: high performance Fault Tolerance Interface for hybrid systems

Leonardo Bautista-Gomez　Dimitri Komatitch　Naoya Maruyama　Seiji Tsuboi
Franck Cappello　Satoshi Matsuoka　Takeshi Nakamura

Peta-scale Phase-Field Simulation for Dendritic Solidification on the TSUBAME 2.0 Supercomputer

Takashi Shimokawabe* Takayuki Aoki** Tomohiro Takaki*** Akinori Yamanaka****
Akira Nukada** Toshio Endo** Naoya Maruyama** Satoshi Matsuoka**

* Interdisciplinary Graduate School of Science and Engineering, Tokyo Institute of Technology
** Global Scientific Information and Computing Center, Tokyo Institute of Technology
*** Graduate School of Science and Technology, Kyoto Institute of Technology **** Graduate School of Engineering, Tokyo Institute of Technology

To realize a low-carbon society, it is indispensable to develop new strong and light materials. The material microstructure controlling the mechanical property is determined in the solidification process. The phase-field model is a meso-scale physical theory describing phase transformations such as solidification. Realistic prediction of material solidification requires a millimeter-scale simulation heretofore impossible with supercomputers. The phase-field equation was discretized by using the finite-difference method, and a simulation code in CUDA was developed. A large-scale solidification simulation of an aluminum-silicon alloy was carried out with 4,000 GPUs on TSUBAME 2.0 and achieved 2.0 petaflops in single precision. The ACM Gordon Bell Award - Special Achievement in Scalability and Time-to-Solution was awarded for this achievement in 2011.

1 Introduction

The development of new strong and light materials contributes greatly to transportation systems with high fuel efficiency and realization of a "low-carbon" society. The strength of a metal strongly depends on its microstructure, which is, in turn, determined by the solidification process of the metal (see Fig.1). However, determining material properties (such as strength) requires millimeter-scale macroscopic study.

The phase-field model [1] is a physical theory describing the evolution of complicated material morphologies on the "meso-scale," namely, between the molecular scale and the macroscopic scale of materials. It is a powerful numerical tool for studying the dynamics of phase transformations such as solidification. The derived equations are partial differential equations in time and space and often discretized by FDM (finite-difference method) or FEM (finite element method). The phase-field equations include many complex nonlinear terms, and the amount of computations per mesh or element becomes large compared to normal stencil applications. Furthermore, the phase-field model assumes a narrow-thickness interface between liquid phase and solid phase, so the time integration requires high spatial resolution and short time step, and phase-field simulations remain restricted to two- or small three-dimensional computations due to their large computational cost.

In this study, we carry out a large-scale phase-field simulation for the aluminum-silicon dendritic growth during directional solidification on the GPU supercomputer TSUBAME 2.0. The simulation code is developed in CUDA, and almost all the capability of TSUBAME 2.0 is used for the simulation, which uses a domain decomposition and inter-node communications with an MPI library. Such large-scale phase-field simulation has not been done before and will have a big impact on material science.

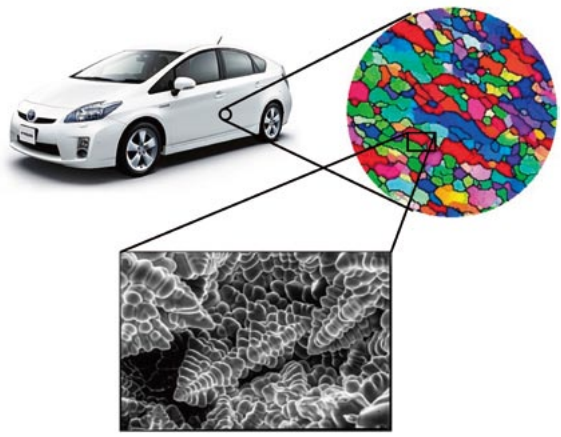


Figure 1 Images of a material microstructure

2 Phase-field model

The phase-field model is derived from non-equilibrium statistical mechanics and can describe meso-scale phenomena that occur on a mid-scale between the molecular and macro scales. The phase-field model introduces a "continuous-order parameter," i.e., a phase-field variable, to describe whether a material is solid or liquid. The phase field ϕ takes a value of 0 in the liquid phase and 1 in the solid phase. Interfaces between solid and liquid phases are treated as diffuse ones described by localized regions, where the parameter changes smoothly between the two fixed values. The interface positions are represented by $\phi = 0.5$. Thanks to this approach, the phase-field method can describe the locations of the interfaces without using a traditional interface-tracking method and carry out the same calculations throughout a computational domain.

The dendritic solidification process in a binary alloy is simulated by solving the phase-field equation and the diffusion

equation of the solute concentration. The time integration of the phase field is described by the following equation, which takes into account the anisotropy of interfacial energy:

$$\frac{\partial \phi}{\partial t} = M_\phi \left[\nabla \cdot (a^2 \nabla \phi) + \frac{\partial}{\partial x}\left(a \frac{\partial a}{\partial \phi_x} |\nabla \phi|^2 \right) + \frac{\partial}{\partial y}\left(a \frac{\partial a}{\partial \phi_y} |\nabla \phi|^2 \right) + \frac{\partial}{\partial z}\left(a \frac{\partial a}{\partial \phi_z} |\nabla \phi|^2 \right) - \Delta S \Delta T \frac{dp(\phi)}{d\phi} - W \frac{dq(\phi)}{d\phi} \right] \quad (1)$$

where a, M_ϕ, W, ΔS, and ΔT represent the gradient coefficient that describes interface anisotropy, the mobility of the phase field, the height of the potential energy barrier, the entropy of fusion and undercooling, respectively. The two functions $p(\phi)$ and $q(\phi)$ are given as $p(\phi) = \phi^3(10 - 15\phi + 6\phi^2)$ and, $q(\phi) = \phi^2(1-\phi)^2$ respectively.

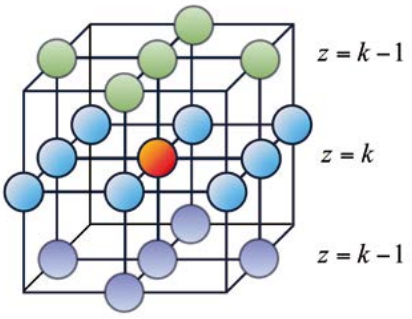
The time integration of the solute concentration is derived by the following diffusion equation:

$$\frac{\partial c}{\partial t} = \nabla \cdot [D_S \phi \nabla c_S + D_L (1-\phi) \nabla c_L] \quad (2)$$

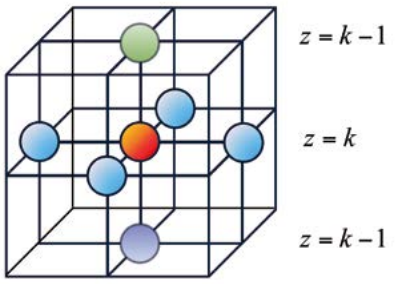
where D_S and D_L are the diffusion coefficients in the bulk solid and liquid phases, respectively. Solid concentration c_S and liquid concentration c_L satisfy $c = (1-\phi)c_L + \phi c_S$.

3 Single-GPU implementation

The time integration of the phase field and the solute concentration given by Equations (1) and (2) are 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. The simulation code is written in CUDA to run on GPUs. All variables are allocated on GPU video memory (also called global memory in CUDA), which virtually eliminates all the CPU-GPU data transfers through a PCI-Express bus during simulation runs. The spatial patterns of the neighbor points of phase field ϕ and solute concentration c that are required to solve the discretized governing equations at the center point (i, j, k) of the grid are, respectively, shown in Figures 2 (a) and (b). In one time step , 19 neighbor elements of ϕ and seven neighbor elements of c are used with the governing equations to update the phase field value and concentration value at the center point; thus, 26 elements must be read from the memory and two updated values must be written back to the memory for each point of the grid.



(a) Stencil of phase-field variable.



(b) Stencil of concentration variable.

Figure 2 Spatial access patterns of neighbor points required for time integration.

Peta-scale Phase-Field Simulation for Dendritic Solidification on the TSUBAME 2.0 Supercomputer

The computation in the phase-field model involves a large number of memory accesses. It is thus highly effective to reduce access to the global memory. In GPU computing, a key factor in performance improvement is how threads and blocks are assigned to actual calculation. The elements of the computational domain of size nx × ny × nz are calculated as follows. First, as shown in Figure 3, the given domain is divided into pieces of size 64 × 4 × 32. Our kernel function is invoked on a GPU, and each thread block (which has 64 × 4 × 1 threads) handles each piece. Each thread calculates 32 elements by marching in the z direction; a thread that corresponds to (i, j) updates grid points (i, j, k), where k varies from k0 to k0 + 31, and k0 is a multiple of 32 (i.e., k0 = 0, 32, 64, ..., nz − 32). When a thread computes a point in the k + 1-th plane, some elements to be referred to in this computation have already been accessed by that thread in the previous k-th plane computation. Such data are reused by holding them in registers, thereby reducing global memory accesses. On the other hand, on-chip shared memory is not used to store data shared by several neighbor threads; instead, the L1/L2 cache available on Fermi GPUs including M2050 is relied on.

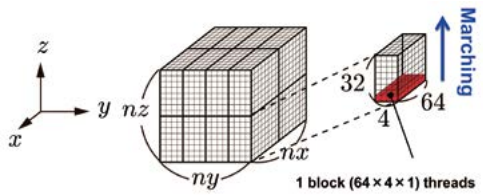


Figure 3 Layouts of CUDA threads and blocks.

4 Optimization of multi-GPU computing

We describe our strategies of multi-GPU computation. Using multiple GPUs is necessary for high-performance computing of large-size problems. In addition to the intrinsic parallel-processing capabilities of GPUs, it is necessary to do parallel computing by using a large number of distributed GPUs. We decompose the whole computational domain in both the y- and z-directions (i.e., 2D decomposition) and allocate each subdomain to one GPU. We have chosen this method because 3D decomposition, which is better in terms of reducing communication amount, tends to degrade GPU performance due to complicated memory access patterns in the y-z plane for data exchanges between the GPU and CPU. Similar to conventional multi-CPU implementations, multi-GPU implementation requires boundary-data exchanges between subdomains. Because a GPU cannot directly access the global memory of other GPUs, host CPUs are used as bridges for data exchange. For inter-node cases, this data exchange is composed of the following three steps: (1) data transfer from the GPU to the CPU by using CUDA APIs, (2) data exchange between nodes with the MPI library, and (3) data transfer back from the CPU to the GPU by using CUDA APIs.

Unlike in conventional multi-CPU computation, in multi-GPU computing, data-communication time with neighbor GPUs is not ignored in the total execution time, especially in large-scale computation, since the performance of the GPU is higher than that of the CPU. To achieve higher performance, it is important to hide communication overheads by overlapping communication with computation. In this study, we implement the following three methods for multi-GPU computing: (a) GPU-only method, (b) Hybrid-YZ method, and (c) Hybrid-Y method.

(a) GPU-only method: This basic method (which is used as a reference) computes all subdomains without applying any specific optimization. After that, the boundary regions of subdomains are exchanged between GPUs by using the above three steps.

(b) Hybrid-YZ method: By dividing each subdomain into five regions, which are two y-boundaries, two z-boundaries and an inside region, we can overlap communication for the boundary data exchange with the computation of the inside region. A similar overlapping technique has been described in previous reports [2]. In this technique, separate GPU kernels are invoked for separate regions. Instead of dividing a GPU kernel, in this study, we let CPU cores compute four y- and z-boundary regions, while the inside

region is computed by GPU. We observed that CPUs often become bottlenecks by taking longer time to do computation of all boundaries and communications than the GPU computation time.

(c) Hybrid-Y method: This method is a slightly modified method from the (b) method. It reduces CPU loads by assigning only the y-boundaries to CPUs instead of both y- and z-boundaries. The GPU can compute the z-boundaries effectively, since it accesses consecutive addresses of the global memory.

Since each node of TSUBAME 2.0 has three GPUs and 12 CPU cores (two 6-core Xeon CPUs), we assign four cores to each of the three GPUs. Each subdomain is thus cooperatively computed by a single GPU and four CPU cores. Multi cores are exploited by using OpenMP.

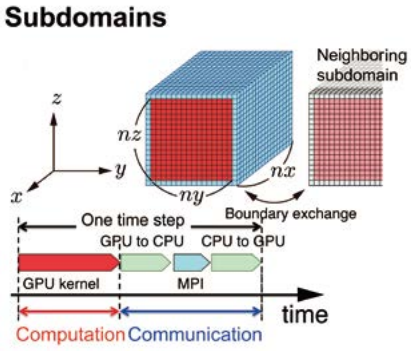


Figure 4 Schematic diagram of the GPU-only method.

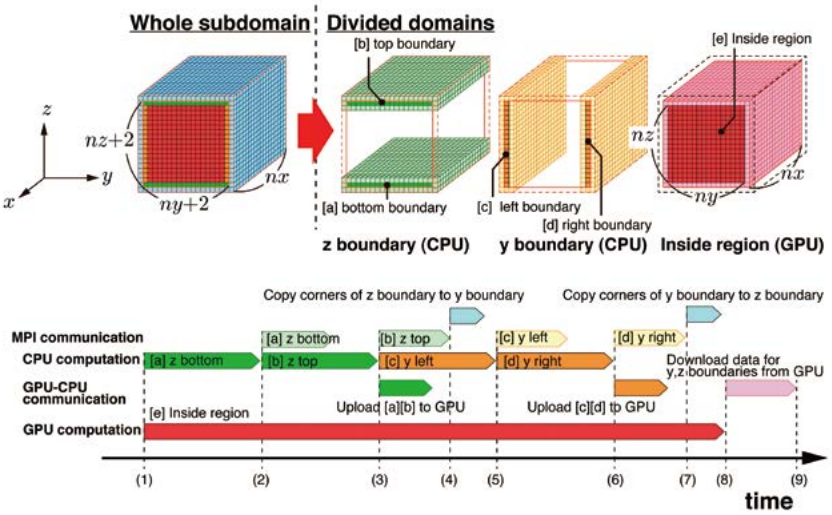


Figure 5 Schematic diagram of the Hybrid-YZ method.

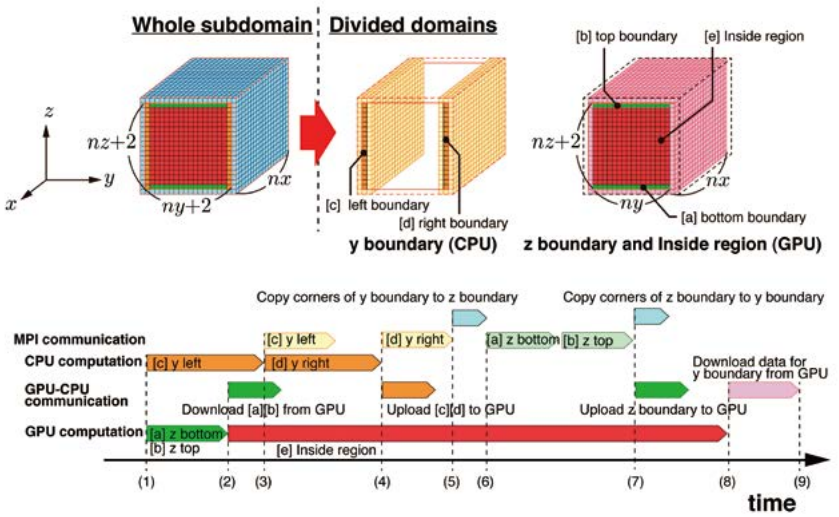


Figure 6 Schematic diagram of the Hybrid-Y method.

Peta-scale Phase-Field Simulation for Dendritic Solidification on the TSUBAME 2.0 Supercomputer

5 Performance of multi-GPU computing

We show the performance of (a) the GPU-only method, (b) the Hybrid-YZ method, and (c) the Hybrid-Y method, all using multiple M2050 GPUs of the GPU-rich supercomputer TSUBAME 2.0 at the Global Scientific Information and Computing Center of the Tokyo Institute of Technology. In this study, we simulate solidification of aluminum–silicon (Al–Si) alloy.

An image of the solidification process of the alloy observed at the world's largest synchrotron radiation facility, SPring- 8, by Hideyuki Yasuda, a professor at Osaka University, and his research group, is shown in Figure 7 (a) . Results of the phase-field simulation using a 4096 × 128 × 4096 mesh are shown in Figure 7 (b) . Although the alloy systems in the images differ, the growth process obtained by the simulation agrees well with the observed one.

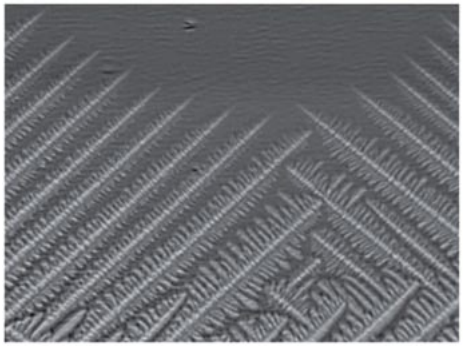


Figure 7(a) Solidification process of an alloy observed at SPring-8 (courtesy of Professor Yasuda).

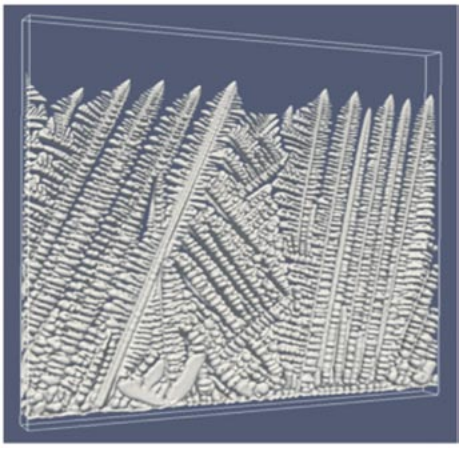


Figure 7(b) Solidification growth simulated by the phase-field model using GPUs.

To measure the performance of the GPU computation, we count the number of floating-point operations in the C/C++-based phase-field simulation code by running it on a CPU with a performance counter provided by PAPI (Performance API). The obtained count and GPU elapsed time are used for evaluating the performance of the GPU computing. Figure 8 shows the results on strong scaling, which show variation of the performance with the number of GPUs for a fixed total mesh size. This figure compares the strong-scaling characteristics of the three proposed methods i.e., (a), (b), and (c). We perform simulations in single precision for three different mesh sizes: 512^3, 1024^3, and 2048^3. It is clear from the figure that the overlapping methods, (b) Hhybrid-YZ method and (c) Hhybrid-Y method, both work effectively in hiding communication overhead as expected, resulting in an improvement of overall performance when a small number of GPUs are used. When the number of GPUs is larger, the volume that each GPU handles becomes smaller, and the percentage of the boundary region computed by the CPU in the whole computational domain increases. Eventually, the GPU computation is no longer able to hide the communication cost. In the case of the Hybrid-YZ method, method (b), we observed that the CPUs often become bottlenecks by taking a longer time to do their operations than the GPU computation time especially when the number of GPUs is larger. Since the Hybrid-Y method alleviates this bottleneck, it achieves a significantly improved performance compared with that of the GPU-only method.

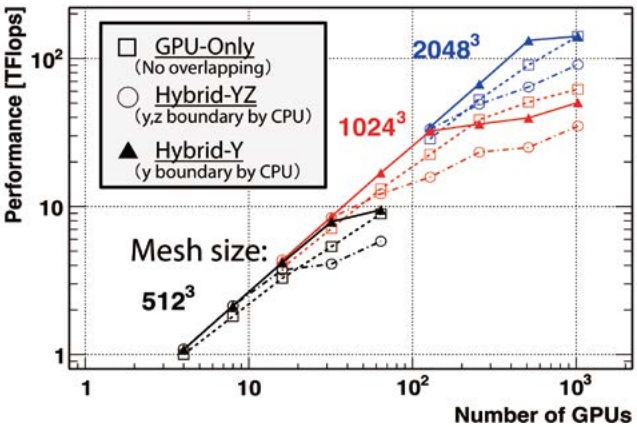


Figure 8 Results for strong scaling of multi-GPU computation in single precision.

Next, we show the weak scaling results, which show how the performance of the three methods varies with the number of GPUs for a fixed mesh size per GPU. Each GPU handles a domain of 4096 × 160 × 128 for single precision. In demonstrating weak

scalability, the Hybrid-Y method has achieved extremely high performance in single precision (Figure 9), namely, reaching 2.000 petaflops, specifically 1.975 petaflops by the GPU and 0.025 petaflops by CPU, for a 4096 × 6480 × 13000 mesh using 4,000 GPUs along with 16,000 CPU cores. This is the first peta-scale result as a real stencil application we know to date.

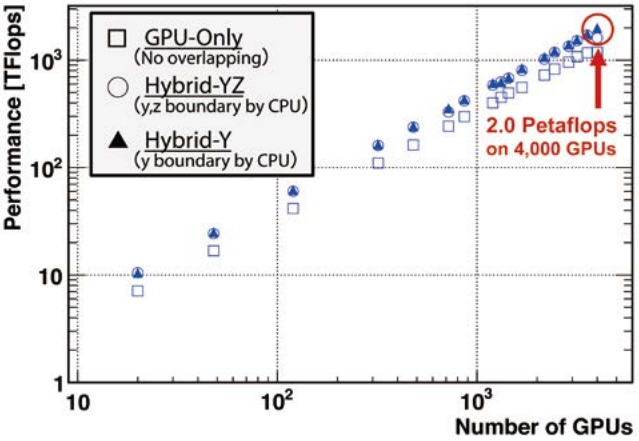


Figure 9 Results on weak scaling of multi-GPU computation in single precision.

Figure 10 shows that electric power consumption of TSUBAME 2.0 when the simulation achieved 2 petaflops, namely, 44.5% of the efficiency to the peak performance, using 4,000 GPUs along with 16,000 CPU cores. The 2-petaflops simulation consumed electric power of 1.36 MW for all computational nodes and networks on TSUBAME 2.0. The simulation thus achieved 1468 Mflops/W. These results show that the simulation results were obtained by small electric power consumption.

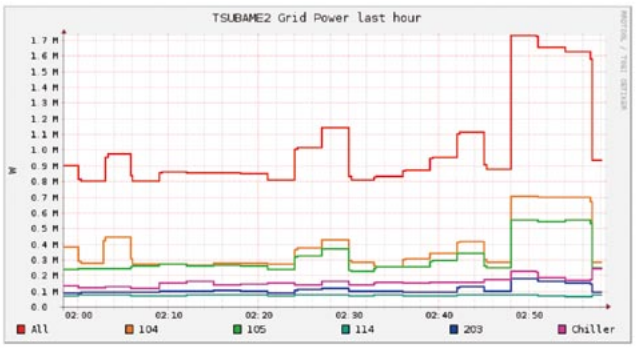


Figure 10 Electrical power consumption of the 2-petaflops phase-field simulation running on TSUBAME 2.0.

Figure 11 demonstrates the dendritic growth during binary-alloy solidification using a mesh size of 4096 × 1024 × 4096 on TSUBAME 2.0. As an initial condition, 32 nuclei are put on the z = 0 plane. This simulation will allow us to clarify the mechanisms of competitive growth between parallel dendrites. Evaluating these mechanisms is important to design and control the mechanical properties of solidified products.

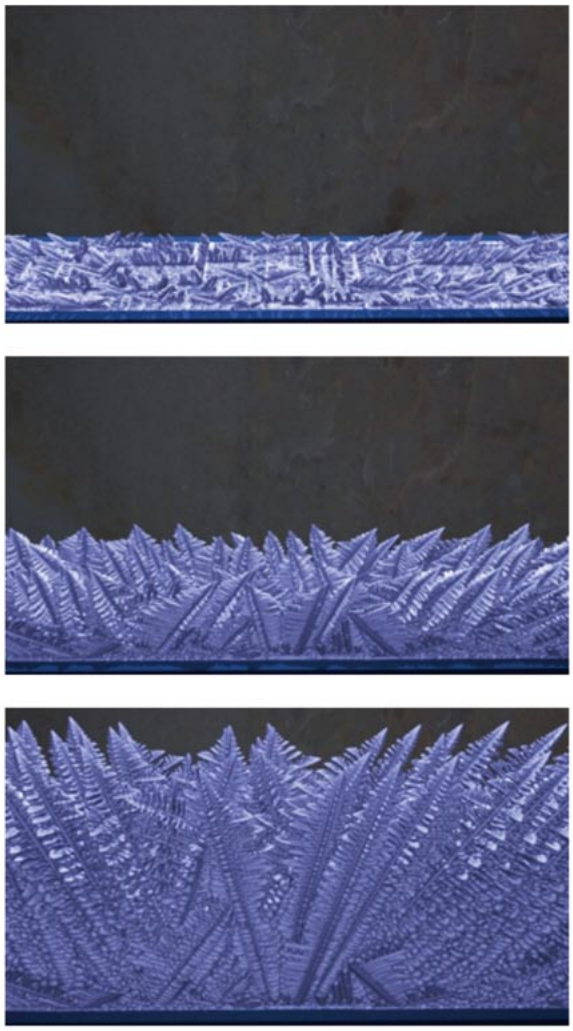


Figure 11 Large-scale simulation of dendritic solidification of aluminum–silicon alloy.

Peta-scale Phase-Field Simulation for Dendritic Solidification on the TSUBAME 2.0 Supercomputer

Summary

A large-scale GPU simulation of the dendritic growth during binary-alloy solidification was carried out on the basis of a phase-field model. An extremely high performance of 2.0 petaflops in single precision was achieved on 4,000 GPUs of TSUBAME 2.0, in spite of a stencil application, which is hard to extract high performance. The computation reached 44.5% of the peak performance and simultaneously high efficiency from the viewpoint of electrical power consumption. It means that we have the computational result on TSUBAME 2.0 with smaller energy compared to conventional supercomputing. It is concluded that a GPU supercomputer is available for various practical stencil applications.

Acknowledgements

The peta-scale simulation was executed as a TSUBAME Grand Challenge Program 2011 spring and we express our special thanks to have a chance to use the whole TSUBAME resources. This research was supported in part by Grant-in-Aid for Scientific Research (B) 23360046 from The Ministry of Education, Culture, Sports, Science and Technology (MEXT), two CREST projects, "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" from Japan Science and Technology Agency (JST), and JSPS Global COE program "Computationism as a Foundation for the Sciences" from Japan Society for the Promotion of Science (JSPS).

References

[1] R. Kobayashi: Modeling and numerical simulations of dendritic crystal growth. Physica D, Nonlinear Phenomena, 63(3-4), 410 - 423 (1993)

[2] Takayuki Aoki, Satoi Ogawa, Akinori Yamanaka: GPU Computing for Dendritic Solidification based on Phase-Field Model, TSUBAME e-Science Journal, Vol.1, pp.5-8, Global Scientific Information and Computing Center, Tokyo Tech (2010 September)

[3] A. Yamanaka, T. Aoki, S. Ogawa, and T. Takaki: GPU-accelerated phase-field simulation of dendritic solidification in a binary alloy. Journal of Crystal Growth, 318(1):40 - 45 (2011). The 16th International Conference on Crystal Growth (ICCG16)/ The 14th International Conference on Vapor Growth and Epitaxy (ICVGE14)

[4] T. Shimokawabe, T. Aoki, T. Takaki, A. Yamanaka, A. Nukada, T. Endo, N. Maruyama, and 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.

Large scale biofluidics simulations on TSUBAME2

Massimo Bernaschi* Mauro Bisson* Toshio Endo**
Massimiliano Fatica*** Satoshi Matsuoka** Simone Melchionna* Sauro Succi*

* CNR-IAC, Istituto Applicazioni Calcolo, Consiglio Nazionale delle Ricerche, Rome, Italy
** Global Scientific Information and Computing Center, Tokyo Institute of Technology
*** Nvidia Corp. Santa Clara, CA, USA

Multi-scale simulations of real-life biofluidic problems entail simulating suspensions composed by hundreds of millions of bodies interacting with each other and with a surrounding fluid in complex geometries. One such example is the simulation of blood flow through the human coronary arteries, with spatial resolution comparable with the size of red blood cells, and physiological levels of hematocrit (the red blood cell volume fraction).
We developed a methodology such that hemodynamic simulations exhibit excellent scalability on the TSUBAME2 installation, achieving an aggregate sustained performance of hundreds of Teraflops. The result demonstrates the capability of predicting the evolution of biofluidic phenomena of clinical significance, by using a suitable combination of novel mathematical models, computational algorithms, hardware technology and code tuning.

1 Introduction

Transport phenomena are ubiquitous in living systems, underlying muscle contraction, digestion, the nourishment of cells in the body, blood circulation, to name but a few. Blood is a reference biofluid, being the carrier of those biological components that fuel the most basic physiological functions, such as metabolism, immune response and tissue repair.

Building a detailed, realistic representation of blood and the vasculature represents a formidable challenge since the computational model must combine the motion of the fluid within an irregular geometry, subject to unsteady changes in flow and pressure driven by the heartbeat, as coupled to the dynamics of red and white blood cells and other suspended bodies of biological relevance.

Large-scale hemodynamic simulations have made substantial progress in recent years[1–3], but until now the coupling of fluid dynamics with the motion of blood cells and other suspended bodies in vessels with realistic shapes and sizes, has remained beyond reach. Owing to non-local correlations carried by the flow pressure, the global geometry plays a significant role on local circulation patterns, most notably on the shear stress at arterial walls. Wall shear stress is a recognized trigger for the complex biomechanical events that can lead to atherosclerotic pathologies. Accurate and reliable hemodynamic simulations of the wall shear stress may provide a non-invasive tool for the prediction of the progression of cardiovascular diseases.

We illustrate here the first multiscale simulation of cardiovascular flows in human coronary arteries reconstructed from computed tomography angiography. The coronary arteries form the network that supply blood to the heart muscle and span the entire heart extension. Spatial resolution extends from 5 cm down to 10 μ m, where a red blood cell has a diameter of about 8 μ m. The simulations involve up to a billion fluid nodes, embedded in a bounding space of about a three hundred billion voxels, with 10-450 million suspended bodies, as shown in Fig. 1. They are performed with the (MUlti PHYsics/multiscale) MUPHY code, which couples Lattice Boltzmann method for the fluid flow and a specialized version of Molecular Dynamics for the suspended bodies[4,5]. The simulation achieves an aggregate performance in excess of 600 Teraflops, with a parallel efficiency of more than 90 percent on 4000-GPU of the TSUBAME2 system.

Our work presents a number of unique features, both at the level of high-performance computing technology and in terms of physical/computational modeling. The extreme complication of the irregular geometries demands that the workload be evenly distributed across the pool of as many as 4000 GPU of the TSUBAME2 supercomputer. The resulting domain-partitioning problem, even at the mere level of the fluid computation, poses a formidable challenge. On top of this, the application adds the further constraint of keeping a good workload balance also for the Molecular Dynamics (MD) component of the multiscale methodology.

Complex and large-scale geometries, such as the one considered here, are rare in the literature. By leveraging large-scale parallel architectures, this work demonstrates the feasibility of cardiovascular simulations of unprecedented size that do not rely on any geometrical regularity of the computational domain.

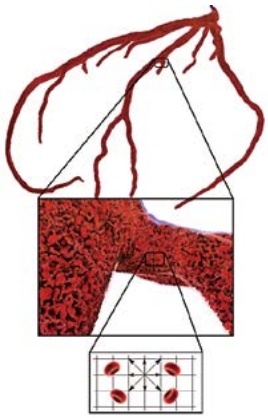


Figure 1 Geometry of the simulated coronary arteries, with the underlying level of red blood cells embedded in the Lattice Boltzmann mesh.

Large scale biofluidics simulations on TSUBAME2

2 Multiscale biofluidics

Living matter is typically composed of two main components: a generic liquid, given by an aqueous solution, plasma, cytosol, etc., and suspended bodies, such as bioparticles, cells, proteins, DNA, *etc.*, that move within the embedding solvent. We have developed the MUPHY[4,5] software to simulate such generic conditions.

As compared to the previous versions of MUPHY[4,5], the current simulation framework handles the numerical solution of the motion of a generic fluid and suspended bodies. Within this vision, MUPHY represents a major effort to design a computational infrastructure for biofluidics research based on the use of a library of solvents and solutes. A non-exhaustive list of functionalities includes: selection of Newtonian versus non-Newtonian rheological response; selection of different kernels for the collision, to reproduce from molecular hydrodynamics to stochastic frictional dynamics; inclusion of stochastic fluctuations; selection of solutes as disconnected particles or forming polymer and molecules; suspended bodies with polymorphic hydrodynamic shapes; selection of different body-body forces; scale-adaptive body-fluid coupling mechanisms; handling of irregular confining media and tissues. Within this range of options, biofluids are modeled in multiple ways and the multiscale/multiphysics behavior of biological or physiological systems can be studied accordingly.

MUPHY leverages two computational engines. The first one handles the motion of the generic fluid within the hydro-kinetic formulation embodied by the Lattice Boltzmann (LB) method: collision-driven mechanisms reproduce the dynamics of fluids in the continuum (in contrast to the direct macroscopic description of fluid motion via the Navier-Stokes equations). The second engine handles the motion of Lagrangian bodies by using a technique that shares several technical aspects with Molecular Dynamics (MD). However, the nature of the suspended bodies is non- conventional and requires a substantial extension of the basic MD technique. Finally, the coupling between fluid and moving particles takes place via specifically designed kernels based on kinetic modeling, again significantly distinct from stresslet, boundary-integral and other methods based on macroscopic hydrodynamics. The end result of the computational environment is a fully time-explicit simulation technique that offers strategic advantages for the study of biofluids under realistic conditions, in particular: *i)* employing geometries with irregular boundaries, such as in the case of the blood vasculature; *ii)* describing non-trivial rheology, as emerging from the underlying particle dynamics; *iii)* enabling scale-specific hydrodynamic interactions at sub-mesh spacing resolution; *iv)* avoiding the detailed representation of the fluid-body dividing interface.

The last two points are crucial to boost the performances of the simulations and to guarantee numerical stability of the dual method, in particular as related to the stiff forces exerted on the fluid by the immersed bodies. The framework employs either a single or a multiple timestep algorithm to handle stiff forces. It should be stressed that the LB method is largely tolerant towards the presence of rapidly varying forces and allows simulating dense suspensions from creeping flow conditions up to Reynolds number in the order of 1000. This strategic asset allows covering a wide range of physical phenomena in real-world physiological conditions, enabling to reproduce highly non-local rheological response that cannot be assimilated to a continuum governed by constitutive relations.

2-1 Lattice Boltzmann/Molecular Dynamics

The LB method[10] is based on the evolution of the singlet distribution representing the probability of finding, at mesh location x and at time t, a "fluid particle" traveling with a given discrete speed. "Fluid particles" represent the collective motion of a group of physical particles (often referred to as populations). The hydrodynamic fluid-body coupling is based on specific roto-translational kernels that represent either rigid entities moving in the fluid as impenetrable bodies, soft vesicles, or a combination thereof.

The fluid-body hydrodynamic interaction is constructed according to the transfer function centered on the i^{th} particle and having spherical or ellipsoidal symmetry and compact support, with a hydrodynamic shape that can be smaller than the mesh spacing[6]. The fluid-particle coupling requires the computation of convolutions over the mesh points {x} and for each configuration of the N suspended bodies.

While, in principle, the calculation of hydrodynamic interactions grows cubically with the system size, the concurrent evolution of fluid and bodies provides a strategic algorithmic advantage. In fact, the LB method features a computational cost which scales linearly with the number of mesh points M, that is, O(*M*)=O(*N/c*), for a fixed solute concentration $c = N/M$. By employing the link-cell method to compute the direct forces among bodies, the particle dynamics also shows O(*N*) complexity. Thanks to the fact that the solvent-mediated particle-particle interactions are localized and explicit, the LB-MD coupling scales linearly with the number of mesh elements and suspended bodies. However, hydrodynamic coupling represents the most time-consuming component of the methodology due to the large overhead arising from the O(*100*) number of mesh points enclosed in the particle support and scattered access to memory.

3 MUPHY

The MUPHY (MUlti PHYsics/multiscale) code was originally written in Fortran 90 by using MPI for the parallelization[4]. For a flexible and efficient handling of complex geometries, MUPHY makes use of an indirect addressing scheme that not only limits the memory requirements to the bare minimum, but also facilitates the achievement of a good load balancing and the selection of the most suitable communication pattern depending on the platform in use. MUPHY was originally developed for the IBM BlueGene architecture[7], a system characterized by thousands of relatively slow PowerPC processors connected by a high performance custom network. On BlueGene/P we achieved an excellent scalability up to the largest configuration available to us (294,912 cores)[11]. The obtained total performance was, in that case, in the range of tens of TeraFlops, with limited room for further improvements since our code cannot exploit the SIMD-like operations of the PowerPC architecture (those operations require stride-one access whereas both the LB and the MD components of MUPHY have a "scattered" data access pattern). In the meantime, the steadily increasing computing power of modern GPUs motivated us to develop a version of MUPHY targeted to clusters of GPUs[7].

A multi-GPU code resorts to parallelism at two levels: intra- and inter- GPU. Unlike the case of codes developed for clusters of traditional multi-core systems that can be implemented either in a hybrid (e.g., OpenMP+MPI) or in a simple distributed memory layout (counting on the avail- ability of highly efficient MPI libraries for shared memory systems), for a multi-GPU platform, a hybrid paradigm is the only choice. This is not the only difference that needs to be taken into account: on a traditional multi-core platform the parallelism is limited to a few threads, at most tens on high-end systems. On a GPU hundreds of threads are required to keep the hardware busy, so that a much finer-grain parallelization is required. Finally, albeit the combination of latest generation Nvidia GPUs and CUDA drivers offers the chance to copy data from/to the global (i.e., main) memory of GPU to/from the global memory of another GPU sitting on the same system, GPUs can not, in general, exchange data with each other without using the CPU.

At first glance, the passage through the CPU introduces an overhead in the communication among the GPUs but, by using the CUDA concepts of *stream* and asynchronous memory copies, it is possible to overlap data transfers between GPU and CPU memory with the execution of kernels (functions in the CUDA jargon) on the GPU. Moreover, also the execution of functions running on the CPU (like MPI primitives) may be concurrent with the execution of kernels on the GPU. As a result, the CPU should be seen as a MPI co-processor of the GPU.

3-1 Domain decomposition

The geometry used in our simulations is highly irregular, as shown in Figure 1, and partitioning in subdomains handled by the available computing resources represents a major challenge in itself. In a previous attempt to solve this issue, we used the third-party software PT-SCOTCH, the parallel version of the SCOTCH graph/mesh partitioning tool[12], in order to distribute the computational load in an even manner. The graph-based procedure utilizes a graph bisection algorithm and it is completely unaware of the geometry of the computational domain. We then realized that the lack of geometrical information degrades the quality of the partitioning as the number of partitions increases, in which case the subdomains reduce to highly irregular shapes with large contact areas each other that increase the communication overhead. An optimal solution was found by combining the graph-based partitioning with a flooding-based approach (also known as graph-growing method) according to the following procedure: the mesh is first partitioned, by using PT-SCOTCH, in a fixed number of subdomains (256). Then each subdomain is further divided by using a flooding algorithm.

In MUPHY, the communication pattern is set up at run time. Each task determines the neighboring tasks owning mesh points that need to be accessed during the simulation, regarding the non-local streaming of populations in the LB algorithm, the migration of particles and calculation of inter-domain forces for the particle dynamics. During this pre-processing step, we employ mainly MPI collective communication primitives whereas, in the remainder of the execution, most communications are point-to-point and make use of the following scheme: the receive operations are always posted in advance by using corresponding non-blocking MPI primitives, then the send operations are carried out. Finally, each task waits for the completion of its receive operations, by using the MPI wait primitives.

For the Lattice Boltzmann component, only the evaluation of global quantities (*e.g.*, the momentum along the x, y, z directions) is carried out by using MPI collective (reduction) primitives. Also for the Molecular Dynamics component most of the communications are point-to-point but, here, the irregularity of the geometry and, as a consequence, of the corresponding domain decomposition results in other issues, as detailed out in the next subsection.

Large scale biofluidics simulations on TSUBAME2

3-2 Parallel Molecular Dynamics

In most parallel Molecular Dynamics applications the geometry of the spatial domain is regular and simple Cartesian decompositions are applied such that each task has (approximately) the same number of particles. In an irregular domain, this strategy would produce two distinct domain decompositions: one for the LB (as described above) and one for the particle dynamics. As a consequence, the same subdomain might belong to two or more processors for the LB and for the MD components and consequently the interaction between particles and fluid would become highly non-local with a complex and expensive communication pat- tern. We then decided to resort to a domain decomposition strategy where the MD parallel domains and the LB parallel domains coincide. In this way, each computational task performs both the LB and MD calculations and the inter- actions of the particles with the fluid are quasi-local. The underlying LB mesh serves the purpose of identifying particles that belong to the domain via a test of membership: a particle with position R belongs to the domain if the vector of nearest integers coincides with a mesh point of the domain. Since an even number of bodies is expected to populate the domains, a pretty good load balancing of the MD parts is granted by the mesh partitioning.

We have developed a novel parallelization strategy suitable for domains with irregular geometry. Among others, a major issue regards the identification of particles that reside next to the subdomain frontiers and that interact with intra- and interdomain particles. Our solution[8] relies on the notion of cells, cubes with side greater or equal to the interaction cutoff, that tile the whole irregular domain handled by a computational task (see Fig.2). This representation allows the processors to perform an efficient search of both interdomain and intradomain pairs of particles and to reduce data transfers by exchanging a limited superset of both the particles actually involved in the evaluation of forces among interdomain pairs and the particles moving across domains.

The final component of our multiscale application deals with the fluid-particle coupling. Each suspended particle experiences hydrodynamic forces and torques arising from the fluid macroscopic velocity and vorticity, smeared over a domain made of 4×4×4 mesh points. Analogously, mesh points experience a momentum transfer arising from the surrounding particles. These non-local operations require multiple communication steps such that each processor owning a given particle, exchanges hydrodynamic quantities with the surrounding domains.

Particle-fluid coupling is a non-local operation involving the interaction between particles and mesh nodes inside the frontier cells of neighboring domains. For this reason, the coupling is done by exploiting the cell tiling, analogously to the computation of inter-domain forces. In order to compute the forces/torques acting from the fluid to the particles, each processor exchanges particles inside the frontier cells with its neighbors. Finally, forces and torques associated to the external particles are exchanged back with neighboring processors and, on the receiving side, the external contributions are distributed to the frontier particles. This approach is particularly efficient as compared to other strategies. For example, by exchanging the mesh points located inside frontier cells in place of particles, one could lower the number of exchanged data to a single communication step. However, such alternative would impose a large communication overhead since all mesh nodes near the interfaces among domains should be exchanged.

Finally, regarding the computation of the momentum transfers exerted from the particles to the fluid, it is carried out by exchanging the frontier particles as in the particle-on-fluid case but without a second data exchange.

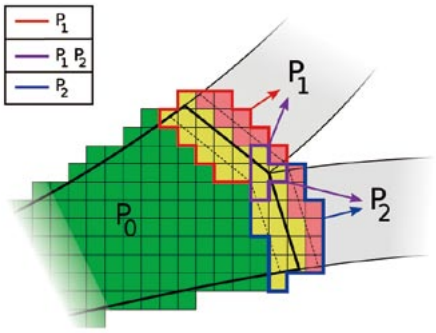


Figure 2 Tiling of an irregular domain in external, frontier and internal cells, and selective exchange with neighboring domains.

4 Results

In our benchmarks, we measure the total runtime for the simulation together with the breakdown between computation and communication. All simulations include about 1 billion lattice sites for the fluid, within a bounding box having a total of almost 300 billion nodes, and 450 million Red Blood Cells (RBC). We choose the fluid-body coupling parameters such that a single RBC carries a hydrodynamic shape with linear size of 4 μ m and 8 μ m for the smallest and largest principal directions of the globule, respectively, and corresponding to a hematocrit level of 58% .

Most computations are performed in single floating point precision with only few reduction operations carried out in double precision. Each GPU thread is responsible for the update, in the LB phase, of a number of mesh nodes that depends on the total number of available GPU (the thread configuration is fixed on each GPU). For instance, with 512 GPU, each thread is in charge of 8 mesh nodes. For the MD phase, interactions among the particles are processed on a per-particle basis. In this case, the grid of threads is directly mapped onto the arrays of particles. Threads are assigned to particles according to a global id and the search for interacting pairs proceeds for each thread in an independent fashion. Each thread scans the cell neighbors and, for each interacting pair, it computes the contribution to the total force. The obtained results provide a fundamental check of the reliability of the code up to physiological levels of hematocrit.

Fig.3 (Upper panel) shows the elapsed time per simulation-step, as well as the breakdown for the LB and MD components separately. A few comments are in order. At first, the performance of the Lattice Boltzmann component of MUPHY on a single GPU is in line with other, highly tuned, CUDA LB kernels. Secondly, the elapsed time decreases significantly with the number of cores, with a speed-up of 12.5 between the 256 and 4000 GPU configurations, corresponding to a parallel efficiency around 80% . The calculation of the MD direct forces features an efficiency as high as 95% . The LB component performs in an optimal way, with a work share that stays always below 4% .

The results show that the combination of asynchronous communication and overlap between communication and computation, significantly alleviates the lack of support for the direct exchange of data among GPUs.

Fig.3 (Lower panel) shows the total parallel efficiency referred to either 256 GPUs, if the available memory allows simulating the hematocrit level, or 512 GPUs for higher levels of hematocrit. The efficiency features superlinear scaling for a number of GPUs up to 1024, whereas at the largest number of GPUs available the efficiency slightly lowers to $\simeq$ 80% .
The excellent scalability should be ascribed to the optimal load balancing obtained with our hybrid graph partitioning/ flooding scheme for the multi-branched arteries. Although the distribution of red blood cells over a test case of 1200 domains shows a broad distribution of values, the execution time for both the aggregate MD and the LB component is compact. The result is a direct consequence of the minimal extension of contact regions among domains that optimizes the share between computations and communications.

Fig.4 shows the distribution of the time spent in communication for a run employing 4000 GPUs. On average, most of the tasks (~3000 out of 4000) spend about 50% of the total time in communication. This result confirms that the asynchronous communication scheme in use works at sustained rate, and the CPU actually plays the role of MPI co-processor of the GPU, where most of the computations take place.

For 1334 nodes of the TSUBAME2 (4000 MPI tasks), we estimate a (weighted) average performance of slightly less than 600 TeraFlops (see Fig. 5 for the corresponding breakdown). Just to convey the flavor of the practical impact of this application, the above performance corresponds to simulating a complete heartbeat at microsecond resolution and fully inclusive of red blood cells, in 48 hours time on the full TSUBAME2 system.

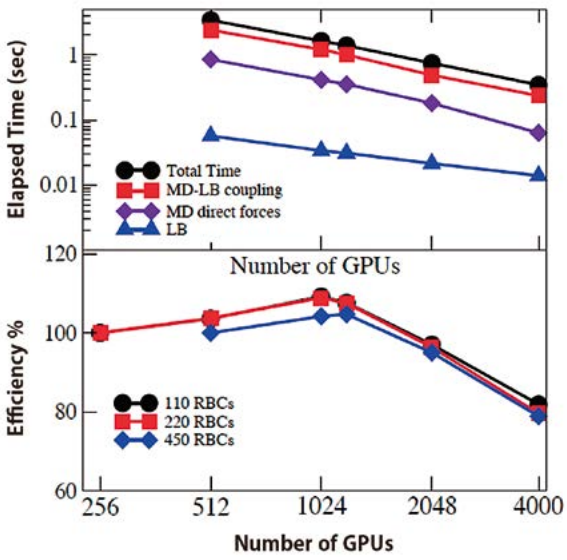


Figure 3 Upper Panel: Elapsed time per timestep.
Lower Panel: Parallel efficiency.

Large scale biofluidics simulations on TSUBAME2

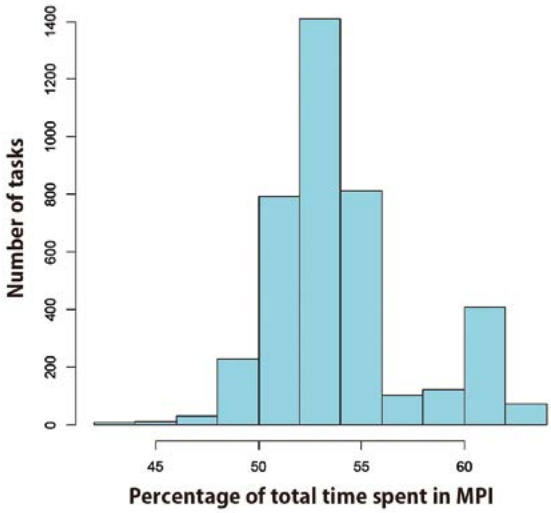


Figure 4 Distribution of the percentage of total time spent in MPI running with 4000 tasks.

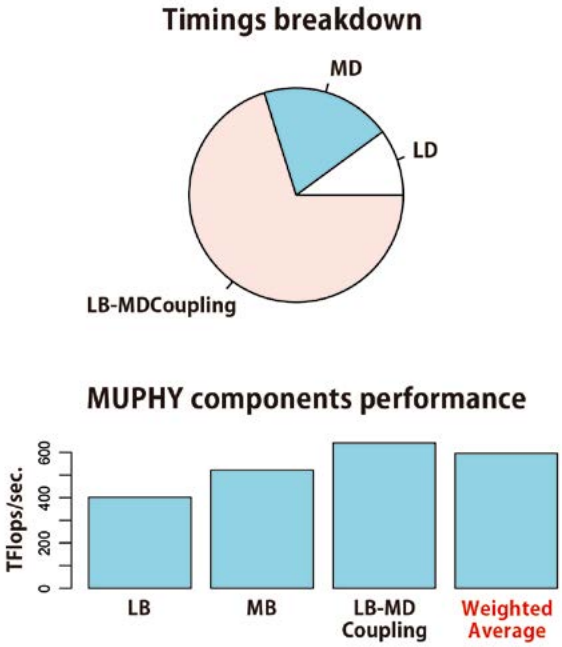


Figure 5 Breakdown of computational components and relative performances.

Summary 5

In the context of computational biofluidics, we have performed the first large-scale simulation of the entire heart-sized coronary system on a world-class cluster of GPUs. The hemodynamic system consists of a realistic representation of the complex human arterial geometry at the spatial resolution of red-blood cells. The computational environment involves one-billion fluid nodes, embedded in a bounding space of one trillion voxels and coupled with the concurrent motion of hundreds of millions of red-blood cells. We achieved a close to 600 Teraflops performance on the 4000 GPU configuration of the TSUBAME2 supercomputer, with a parallel efficiency in excess of 90 percent, performing about 2000 billion lattice updates per second concurrently with the simulation of the dynamics of up to 450 million red blood cells.

The above accomplishment results from the development of several unique features, in terms of both high-performance technology and physical/computational modeling. We are not aware of any previous implementation dealing with non-idealized geometries. The present work represents a major progress in the predictive capabilities of computer simulations for real-life biofluidic research, with special, yet not exclusive, focus on cardiovascular clinical practice.

Acknowledgements

This research was carried out as a TSUBAME Grand Challenge Program.

We thank E. Kaxiras, C.L. Feldman, A.U. Coskun, F.J. Rybicki, A.G. Whitmore, G. Amati, F. Pozzati and F. Schifano for numerous discussions.

References

[1] D.A. Vorp, D.A. Steinman, C.R. Ethier, Comput. Sci. Eng., pp. 51 (2001).

[2] A. Quarteroni, A. Veneziani, P. Zunino, SIAM J. Num. Analysis, 39, 1488 (2002).

[3] L. Grinberg, T. Anor, E. Cheever, et al., Phil. Trans. Royal Soc. A, 367 1896 2371 (2009).

[4] M. Bernaschi, S. Melchionna, S. Succi et al., Comp.Phys. Comm., 180, 1495, (2009).

[5] S. Melchionna, M. Bernaschi, S. Succi et al, Comp.Phys. Comm., 181, 462, (2010).

[6] S. Melchionna, Macromol. Theory Sim., DOI: 10.1002/mats.201100012 (2011).

[7] M. Bernaschi, M. Fatica, S. Melchionna, S. Succi and E. Kaxiras, Concurrency and Computation: Practice and Experience, DOI: 10.1002/cpe.1466 (2009).
[8] M. Bisson, M.Bernaschi, S.Melchionna, Commun. Comput. Phys., 10, 1071 (2011).
[9] S. Melchionna, J. Comput. Phys. 230, 3966 (2011).
[10] S. Succi, The Lattice Boltzmann Equation for Fluid Dynamics and Beyond, Oxford University Press, USA (2001).
[11] A. Peters et al., Proceedings of Supercomputing 2010, New Orleans, 2010.
[12] http://www.labri.fr/perso/pelegrin/scotch

Graph500 Challenge on TSUBAME 2.0

Toyotaro Suzumura*/ Koji Ueno***

*Graduate School of Information Science and Engineering, Tokyo Institute of Technology
**IBM Research – Tokyo

Graph500 is a new benchmark that ranks supercomputers by executing a large-scale graph search problem. Our early study reveals that the provided reference implementations by the Graph500 benchmark are not scalable in a large-scale distributed environment. For the Graph500 benchmark on TSUBAME 2.0, we ran our highly scalable BFS method that divides adjacent matrix - representing large-scale graph - with 2D partitioning and distributes the portion to all the processors. In contrast to traditional 1D partitioning, this can greatly reduce the communication exchange. With the optimization method, we successfully solved BFS (Breadth First Search) for large-scale graph with 236 (68.7 billion) vertices and 240 (1.1 trillion) edges for 10.955 seconds with 1366 nodes and 16392 CPU cores. This record corresponds to 100.366 GE/s, which was the 3rd-ranked score in the latest ranking announced in SC2011.

1 Introduction

Large-scale graph analysis is a hot topic for various fields of study, such as social networks, micro-blogs, cyber security, protein-protein interactions, and the connectivity of the Web. The numbers of vertices in the analyzed graph networks have grown from billions to tens of billions and the edges have grown from tens of billions to hundreds of billions. Since 1994, the best known de facto ranking of the world's fastest computers is TOP500, which is based on a high performance Linpack benchmark for linear equations. As an alternative to Linpack, Graph500 [1] was recently developed. We con-ducted a thorough study of the algorithms of the reference implementations and their performance in an earlier paper [2]. Based on that work, we implemented a scalable and high-performance implementation of an optimized Graph500 benchmark for large distributed environments. In this paper, we give an overview of the Graph500 benchmark in Section 2 and its basic parallel algorithm called level-synchronized BFS in Section 3. Our proposed scalable BFS method is described in Section 4 and the performance evaluation is shown in Section 5, and then give a conclusion in Section 6.

2 GRAPH500 Benchmark

In this section, we give an overview of the Graph500 benchmark [1]. In contrast to the computation-intensive benchmark used by TOP500, Graph500 is a data-intensive benchmark.

It does breadth-first searches in undirected large graphs generated by a scalable data generator based on a Kronecker graph [16]. The benchmark has two kernels: Kernel 1 constructs an undirected graph from the graph generator in a format usable by Kernel 2. The first kernel transforms the edge tuples (pairs of start and end vertices) to efficient data structures with sparse formats, such as CSR (Compressed Sparse Row) or CSC (Compressed Sparse Column). Then Kernel 2 does a breadth-first search of the graph from a randomly chosen source vertex in the graph.

The benchmark uses the elapsed times for both kernels, but the rankings for Graph500 are determined by how large the problem is and by the throughput in TEPS (Traversed Edges Per Second). This means that the ranking results basically depend on the time used by the second kernel.

After both kernels have finished, there is a validation phase to check if the result is correct. When the amount of data is extremely large, it becomes difficult to show that the resulting breadth-first tree matches the reference result. Therefore the validation phase uses 5 validation rules. For example, the first rule is that the BFS graph is a tree and does not contain any cycles.

There are six problem classes: toy, mini, small, medium, large, and huge. Each problem solves a different size graph defined by a Scale parameter, which is the base 2 logarithm of the number of vertices. For example, the level Scale 26 for toy means 226 and corresponds to 1010 bytes occupying 17 GB of memory. The six Scale values are 26, 29, 32, 36, 39, and 42 for the six classes. The largest problem, huge (Scale 42), needs to handle around 1.1 PB of memory. As of this writing, Scale 38 is the largest that has been solved by a top-ranked supercomputer.

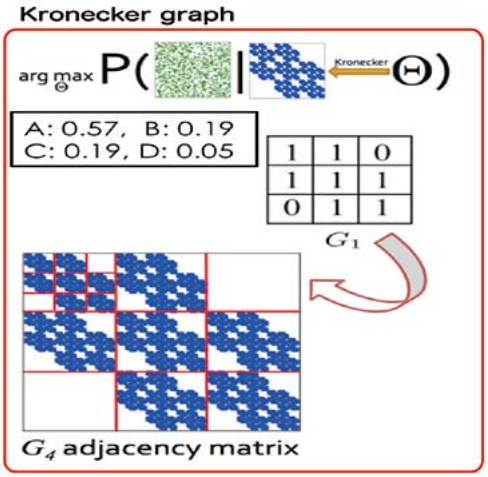


Figure 1 Kronecker Graph [4]

3 Level-Synchronized BFS

All of the MPI reference implementation algorithms of the Graph500 benchmark use a "level-synchronized breadth-first search", which means that all of the vertices at a given level of the BFS tree will be processed (potentially in parallel) before any vertices from a lower level in the tree are processed.

Algorithm I: Level-synchronized BFS

```
1   for all vertex v in parallel do
2   | pred[v]← -1;

3   pred[r] ← 0
4   Enqueue(CQ r)
5   While CQ != Empty do
6   |   NQ ← empty
7   |   for all u in NQ in parallel do
8   |   |   u ← Dequeue(CQ)
9   |   |   for each v adjacent to u in parallel do
10  |   |   |   if pred[v] = -1 then
11  |   |   |   |   pred[v] ← u;
12  |   |   |   |   Enqueue(NQ, v)
13  |   swap(CQ, NQ);
```

Algorithm I is the abstract pseudocode for the algorithm that implements level-synchronized BFS. Each MPI process has two queues, CQ and NQ, and two arrays, PRED for a predecessor array and VISITED to track whether or not each vertex has been visited.

At any given time, CQ (Current Queue) is the set of vertices that must be visited at the current level. At level 1, CQ will contain the neighbors of r, so at level 2, it will contain their pending neighbors (the neighboring vertices that have not been visited at levels 0 or 1). The algorithm also maintains NQ (Next Queue), containing the vertices that should be visited at the next level. After visiting all of the nodes at each level, the queues CQ and NQ are swapped at line 16.

VISITED is a bitmap that represents each vertex with one bit. Each bit of VISITED is 1 if the corresponding vertex has been already visited and 0 if not. PRED has a predecessor vertex for each vertex. If an unvisited vertex v is found at line 12, the vertex u is the predecessor vertex of the vertex v at line 14. When we complete BFS, PRED forms a BFS tree, the output of kernel2 in the Graph500 benchmark.

At each level, the set of all vertices v is the NQ nominee, now called "NQ-N". NQ-N has all the adjacent vertices of the vertices in CQ that would be potentially stored in NQ. NQ-N is obtained from line 9 to line 11 in the algorithm.

The Graph500 benchmark provides 4 different reference implementations based on this level-synchronized BFS method. Basically all the reference implementations has the scalability issue and are mostly saturated with 32 nodes on TSUBAME 2.0. Please refer our work [2] on their details and algorithms and their performance analysis on TSUBMAE 2.0.

4 U-BFS: Our Scalable BFS Method

To solve the scalability issue of reference implementations, our scalable BFS method named U-BFS is based upon the technique proposed in [3]. Their approach is based on the level-synchronized BFS and 2D (two dimensions) partitioning technique. This is scalable technique to reduce the communication cost unlike the 1D partitioning including vertical partitioning and horizontal partitioning of reference implementations. Our proposed method also optimizes this 2D partitioning technique and also other some optimization techniques. Therefore we give a brief overview of the 2D partitioning technique here.

Assume that we have P processors in total, P=R*C processors are logically deployed in two dimensional mesh of R (processor row) * C (processor column). An adjacent matrix is divided as shown in Figure 2 and the processor (i, j) is responsible for handling the C blocks from $A_{i,j}^{(1)} \sim A_{i,j}^{(C)}$. Vertices are divided into R*C blocks and the processor (i, j) handles the k th block where k is computed by (j-1)*R + i.

Each level of level-synchronized BFS method with 2D partitioning is performed by 2 phases called "expand" and "fold".

In the expand phase, every processor copies its CQ to all the other processors that exists in the same column just as the vertical 1D partitioning. In the fold phase, each row of processors integrates the compute results after the expand phase. Therefore, this is equivalent to a method of combining two types of 1D partitioning. If C is 1, this corresponds to the vertical 1D partitioning and if R is 1, it corresponds to the horizontal 1D partitioning.

In the fold phase, it firstly searches all the adjacent vertices against each vertex, v of CQ obtained by the expand phase, and then sends a tuple of (v, u) - where u is one of the found adjacent vertices – to the corresponding processor where v is located.

The advantage of 2D partitioning is to reduce the number of processors that needs communication among them. The two types of 1D partitioning requires all-to-all communication. However, the 2D partitioning can reduce the number of communication processors and most importantly becomes highly scalable in large computing environments since the expand phase only requires the communication among the nodes in the same column and the fold phase only requires the communication among the processors in the same row.

Performance Evaluation 5

We conducted the Graph500 benchmark on TSUBAME 2.0. Here is the evaluation environment and performance result.

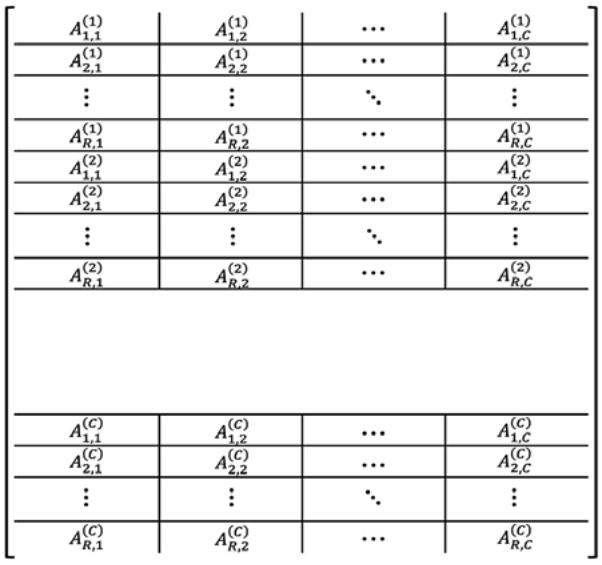


Figure 2 2D Partitioning Based BFS [3]

5.1 Evaluation Environment

Each TSUBAME 2.0 node has two Intel Westmere EP 2.93 GHz processors (Xeon X5670, 256-KB L2 cache, 12-MB L3) and 50 GB of local memory. As the software environment we used gcc 4.3.4 (OpenMP 2.5), MVAPICH2 version 1.6 and we used at maximum 1366 nodes. TSUBAME 2.0 is also characterized as a supercomputer with heterogeneous processors including tremendous amount of GPU devices, but we do not use the environment. One node of TSUBAME 2.0 has physical 12 CPU cores and virtually 24 cores with SMT (Simultaneous Multithreading). Our implementation treats 24 cores for one single node and the same number of processors are allocated to each MPI processes. Each computing node is connected to two QDR Infiniband network links, so the communication bandwidth for the node is about 80 times larger than a fast LAN (1 Gbps). Not only the link speed at the end-point nodes, but the network topology of the entire system heavily affects the performance such an I/O intensive application as Graph500. TSUBAME 2.0 uses a full-bisection fat-tree topology, which accommodates applications that need more bandwidth than provided by such topologies as a torus or mesh.

5.2 Performance Result

We compare U-BFS with the latest version (2.1.4) of reference implementations. Figure 3 compares our optimized implementation, U-BFS with reference implementations. This experiment is conducted in a weak-scaling fashion, so the problem size for one node is SCALE 26. The horizontal axis is the number of nodes and the vertical axis is TEPS (GE/s).

U-BFS and two reference implementations, replicated-csr and replicated-csc (called R-CSR and R-CSC hereafter) use 2 MPI processes for 1 node. The reference implementation, simple (called SIM hereafter), uses 16 MPI processes for one node since the implementation has not been implemented with the multithreading parallelism. As shown in the graph, there exists some result that cannot be measured due to the errors such as validation error, segmentation fault, and memory error with reference implementations. Figure 3 shows that U-BFS outperforms R-CSC and SIM. With smaller number of nodes less than 32 nodes, R-CSR shows better performance, but our method shows performance advantage with more than 32 nodes. For instance, our optimized method is 2.8 times faster than R-CSR with 128 nodes and SCALE 26 for one node (All the problem size is SCALE 33).

The scalability of our approach is shown in Figure 4. As shown in the figure, the throughput in TEPS is linearly increasing with larger number of nodes and finally achieves 99.0 GE per second (TEPS) with 1024 nodes and SCALE 36. We successfully

solved BFS (Breadth First Search) for large-scale graph with 236 (68.7 billion) vertices and 240 (1.1 trillion) edges for 10.955 seconds with 1366 nodes and 16392 CPU cores. This record corresponds to 100.366 GE/s, which was the 3rd-ranked score in the latest ranking announced in SC2011.

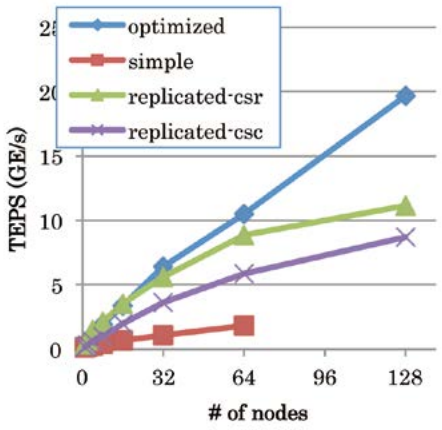


Figure 3 Performance Comparison with Reference Implementations

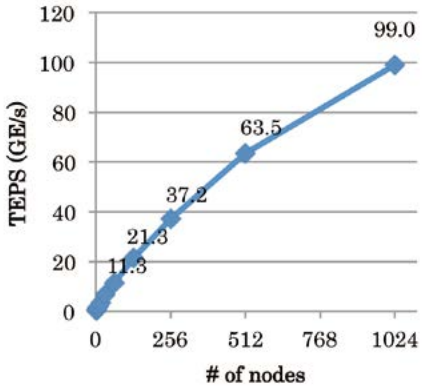


Figure 4 Performance of Our Optimized Implementation with Scale 26 per 1 node

Concluding Remarks 6

In this paper we described our challenge and our basic algorithm for the Graph500 benchmark on TSUBAME 2.0. Our proposed approach based on 2D partitioning greatly outperforms the benchmark reference implementations and also shows great scalability with up to 1366 nodes. Overall our score was 3rd score announced in 2011/11. Our achievement could not be done with only 2D partitioning but also more optimization techniques. This is the first challenge for Graph500, but this continuous challenge and research towards more optimization and scalable algorithms waits us.

Acknowledgements

This research was supported in part by TSUBAME Grand Challenge Program and a CREST project "Highly Productive, High Performance Application Frameworks for Post Petascale Computing" from Japan Science and Technology Agency (JST).

References

[1] Graph500 : http://www.graph500.org/.

[2] Toyotaro Suzumura, Koji Ueno, Hitoshi Sato, Katsuki Fujisawa and Satoshi Matsuoka, "Performance Evaluation of Graph500 on Large-Scale Distributed Environment", IEEE IISWC 2011 (IEEE International Symposium on Workload Characterization), 2011/11, Austin, TX, US

[3] Andy Yoo, et al, A Scalable Distributed Parallel Breadth-First Search Algorithm on BlueGene/L. SC 2005.

[4] J. Leskovec, D. Chakrabarti, J. Kleinberg, and C. Faloutsos, "Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication," in Conf. on Principles and Practice of Knowledge Discovery in Databases, 2005.

Physis: A High-Level Stencil Framework for Heterogeneous Supercomputers

Naoya Maruyama* Tatsuo Nomura** Kento Sato*** Satoshi Matsuoka*
*Global Scientific Information and Computing Center, Tokyo Institute of Technology **Google, Inc.
***Graduate School of Information Science and Engineering, Tokyo Institute of Technology

We propose a compiler-based programming framework that automatically translates user-written structured grid code into scalable parallel implementation code for GPU-equipped clusters such as TSUBAME2.0.
Our framework automatically translates user-written stencil functions to GPU execution code as well as message passing parallel code for inter-node parallelism. It also includes several optimizations for better scalability with a large number of GPUs, such as compute and communication overlapping. We present an overview of our framework and report performance results using TSUBAME2.0, which demonstrate good scalability up to 256 GPUs.

1 Introduction

Heterogeneous computing with both conventional CPUs and vector-oriented GPU accelerators is becoming common because of superior performance as well as power efficiency. The peak performance of latest NVIDIA GPUs can be as high as 515 GFLOPS per chip, which is faster than the latest CPUs by several factors, allowing significant performance boost in compute-bound applications such as N-body problems. GPU's memory bandwidth is also much greater than conventional CPU memory, reaching over 100 GB/s in properly aligned memory accesses, making it possible to achieve significant speedups in memory-bound applications such as computational fluid dynamics.[2]

Programming such heterogeneous systems, however, is a notoriously difficult task. The reason is two-fold. First, most of the existing programming models for such systems only provide low-level platform-specific abstractions. The lack of high-level unified programming models forces the programmer to learn multiple distinctive models for parallel computing, e.g., message passing for distributed memory machines and GPU-centric models for accelerators, often resulting in ad hoc hybrid programming models. Since parallel programming even with a single model is known to be difficult and error prone, exploiting the potential performance advantage with hybrid models is thus a highly difficult task. Second, in the current heterogeneous architecture, data movements often involve complex performance considerations such as locality optimizations for keeping data close to processor cores and overlapping of communications and computations. While these techniques have long been well known and studied on parallel platforms, realizing them on complex heterogeneous systems further increases the programmer burden. As a result, further scaling performance with a large number of GPUs remains to be challenging even for highly skilled experts.

To solve the problem and improve programmer productivity, we envision a high-level programming model that provides a uniform application programming interface for heterogeneous systems. While low-level interfaces are indeed essential when the maximum programming flexibility is required, such a case should not be common but exceptional, and simplifying programming even with limited flexibilities and small performance cost should be highly important to allow the adoption of heterogeneous systems for a wider range of application programmers.

This article presents our high-level programming framework called Physis that is specialized to stencil computations with regular multidimensional Cartesian grid.[1] In stencil computations, each grid point is repeatedly updated by only using neighbor points, exhibiting regular spatial locality. Such a computation pattern, called " structured grids ", frequently appears in numerical simulation codes for solving partial differential equations. The performance of stencil applications is often determined by memory system performance since the typical byte-per-flop ratio in such code is higher than the ratio of today's processor and memory systems, including GPUs. Therefore, optimizing data movements is the most important to improve performance of such applications. Typical such optimizations for the GPU include latency hiding by scheduling a large number of concurrent threads, data alignment to allow coalesced memory accesses, and locality optimizations by thread blocking.[3] In addition to these optimizations, more coding effort is required to scale well with a large number of GPUs, such as communication and computation overlapping. GPU performance scalability is especially important for applications using a large amount of data, since a single GPU is equipped only with a few giga bytes of memory.

In the Physis framework, we design its programming model such that architecture neutrality can be realized on various parallel platforms, with a particular focus on GPU-based heterogeneous supercomputers. It provides portable and declarative constructs for describing stencil computations, such as

creating multidimensional grids, data copying to and from them, and applying stencils over them. Global view memory model and implicit parallelism are adopted to realize high productivity as well as architecture neutrality. The declarative programming interface at the same time allows for static compilation techniques to automatically parallelize stencil computations over distributed memory environments with optimizations such as compute-communication overlapping. While it is beyond the scope of this paper, the framework is designed to allow for further advanced software techniques to be applied transparently for the application programmer, such as model-based and experimental performance tuning, resiliency through error checking and scalable and fast checkpointing.

We describe an implementation of the framework based on the standard C language. We introduce a small set of custom data types and intrinsics for stencil computations into C as an embedded DSL.[4] Those custom extensions are translated to platform native code, such as CUDA for GPU and MPI for message passing. Programs written in the Physis DSL can also be automatically translated to parallel code using MPI with the overlapping optimization for better scalability.

To evaluate our framework, we implement several stencil applications in the Physis DSL and evaluate its performance using the TSUBAME2.0 supercomputer at Tokyo Tech, which is the fifth fastest machine at the Nov 2011 list of Top500. We present results of performance studies using up to 256 NVIDIA Fermi GPUs, and demonstrate that our framework can achieve performance comparable to hand-written versions with good strong and weak scalability. For more complete presentation, refer to our SC11 paper.[1]

2 High-Level Framework For Stencil Computations

We design a high-level programming framework that provides a highly productive programming environment for stencil computations. The framework consists of a domain-specific language and platform-specific runtimes. The DSL allows for declarative and flexible descriptions of stencils in an architecture-neutral way, which is then translated to architecture-specific code by source-to-source translators. The framework runtime encapsulates architecture-specific data management tasks and provides a uniform interface of virtual shared memory for multidimensional grids. The rest of this section discusses our major design goals of the framework.

2-1 Design Goals

Automatic parallelization: We design the Physis DSL amenable to compiler-based automatic parallelization on distributed-memory parallel environments. Although automatic parallelization has been an active research topic for the past decades, it has not been widely successful in practice for general-purpose languages, especially on distributed memory environments, since effectively exploiting data localities available in applications is a complex and difficult task. In contrast, our framework is limited to a small set of domain-specific computations, but by doing so we eliminate the difficulties of automatic parallelization in conventional general-purpose languages, and realize implicit parallelism on a variety of parallel platforms.

Embedded DSL rather than external DSL: Inventing a completely new language for a given problem domain, i.e., an external DSL approach, potentially allows for a maximally optimized language design. In practice, however, being dissimilar to existing familiar languages may hinder adoption by a wide body of application programmers. We design our DSL as a small set of extensions on existing general-purpose languages, i.e., an embedded DSL. We choose C as the base language in our current design and implementation since it is one of the most commonly used languages in high performance computing.

Declarative and expressive programming model: In order to improve productivity, we maximize programming abstraction by adopting a declarative programming model that allows for less manual programming than imperative models. For example, in the Physis DSL the programmer expresses how each grid element is computed, but it is determined by the framework how the whole grid is processed with the user-specified computation; the

overall computation may be performed sequentially or in parallel depending on target environments of the framework. Having too much abstraction, however, can be too restricted to implement real-world scientific simulations. For example, some stencils may be applied only to a part of a whole grid, such as boundary regions. Although we attempt to keep the language extensions minimal, we adopt additional domain-specific constructs if they can further improve productivity of programmers and performance of final implementation codes.

3 Programming Model

The Physis DSL extends the standard C with several new data types and intrinsics for stencil computations. The user is required to use the extensions to express stencil-based applications, which are then translated to actual implementation code by the Physis translator.

Physis supports multidimensional Cartesian grids of floating-point values (either float or double). To represent multidimensional grids, we introduce several new data types named based on its dimensionality and element type, e.g., PSGrid3DFloat for 3-D grids of float values and PSGrid2DDouble for 2-D grids of double values. The type does not expose its internal structure, but rather works as an opaque handle to actual implementation, which may differ depending on translation targets.

Since many of the Physis intrinsics are overloaded with respect to the grid types, below we simply use PSGrid to specify different grid types when not ambiguous.

Grids of type PSGridFloat3D can be created and destructed with intrinsics PSGridFloat3DNew and PSGridFree, as defined as follows:

```
PSGrid3DFloat PSGrid3DFloatNew(
  size_t dimx, size_t dimy,
  size_t dimz,
  enum PS_GRID_ATTRIBUTE attr)
void PSGridFree(PSGrid g)
```

They can be accessed both in bulk and point-wise ways using the following intrinsics:

```
void PSGridCopyin(PSGrid g,
                  const void *src)
void PSGridCopyout(PSGrid g, void *dst)
PSGridGet(PSGrid g, size_t i,
          size_t j, size_t k)
void PSGridSet(PSGrid g,
  size_t i, size_t j, size_t k, T v)
void PSGridEmit(PSGrid g, T v)
```

The set of size_t parameters specify the indices of a point within the given grid, so the number of index parameters depend on the dimensionality of the grid (e.g., three for 3-D grids). The return type of PSGridGet and the v parameter of PSGridSet and PSGridEmit have the same type as the element type of the grid, which is either float or double.

PSGridGet returns the value of the specified point, while PSGridSet writes a new value to the specified point. PSGridEmit performs similarly to PSGridSet, but does not accept the index parameters, and is solely used in stencil functions.

3-1 Example

Fig.1 shows a function of 9-point stencil on 2-D grids. Such a function can be applied to grids by using two declarative intrinsics: PSStencilMap and PSStencilRun. Fig. 2 illustrates how these intrinsics can be used to invoke the diffusion stencil of Fig.1 on 2-D grids.

```
void diffusion(const int x, const int y,
 PSGrid2DFloat g1, PSGrid2DFloat g2, float t) {
 float v=
  PSGridGet(g1,x,y)+PSGridGet(g1,x+1,y)
  +PSGridGet(g1,x-1,y)+PSGridGet(g1,x,y+1)
  +PSGridGet(g1,x,y-1)+PSGridGet(g1,x+1,y+1)
  +PSGridGet(g1,x+1,y-1)+PSGridGet(g1,x-1,y+1)
  +PSGridGet(g1, x-1, y-1);
 PSGridEmit(g2, v / 9.0 * t);
}
```

Figure 1 Example 9-point stencil

PSStencilMap creates an object of PSStencil, which encapsulates a given stencil function with its parameters bound to actual arguments. It is analogous to closures in functional

programming. PSStencilRun executes PSStencil objects in a batch v. Each stencil function may be executed in parallel, exhibiting implicit parallelism.

```
PSGrid2DFloat g1 = PSGrid2DFloatNew(NX, NY);
PSGrid2DFloat g2 = PSGrid2DFloatNew(NX, NY);
// initial_data is a pointer to input data
PSGridCopyin(g1, initial_data);
PSDomain2D d = PSDomain2DNew(0, NX, 0, NY);
PSStencilRun(
   PSStencilMap(diffusion, d, g1, g2, 0.5),
   PSStencilMap(diffusion, d, g2, g1, 0.5));
```

Figure 2 Example code to apply stencils to grids

4 Experimental Evaluation

To evaluate our proposed framework, we have implemented a prototype for parallel machines using the ROSE compiler framework.[5] It consists of a source-to-source translator and runtime components for each target platform. As translation targets, we currently generate C for CPU execution and CUDA for GPU execution. In addition, for platforms involving multiple distributed compute resources, we generate message-passing parallel code using MPI.

As performance benchmark programs, we implemented two stencil codes in Physis:

- Diffusion: 3-D 7-point stencil.
- Seismic: 3-D seismic wave simulation with 27 stencil functions.

Diffusion is relatively small scale, consisting of only one stencil function, whereas the seismic code consists of 27 stencils with staggered grids. Among the 27 stencils, six are for computing the 2-D surfaces of the 3-D domain, which are implemented with PSDomain objects. All benchmarks use single-precision floating-point data.

We use the TSUBAME2.0 supercomputer at Tokyo Institute of Technology, which consists of 1408 compute nodes. Each node has two Intel Xeon Westmere-EP 2.9GHz CPUs and three NVIDIA M2050 GPUs with 52GB and 3GB of system and GPU memory, running SUSE Linux Enterprise Server 11 SP1. The compute nodes are interconnected by dual QDR Infiniband networks with a full bisection-bandwidth fat-tree topology network. We use CUDA v3.2 for GPU code and gcc/g++ v4.1.2 for CPU code.

4-1 Weak Scaling Evaluation

Fig. 3 shows the results of weak scaling evaluation with the diffusion code. The red and blue lines are the cases where each GPU is assigned a subdomain of 256x128x128 and 512x256x256, respectively. In both cases, the 3-D domain is decomposed only over y and z dimensions. As expected, the larger problem size allowed for better performance and scaling, which is almost linear scaling up to 256 GPUs, although even the smaller case achieved 11 times speedup with 256 GPUs compared to the 16-GPU case.

Fig.4 shows the results of weak scaling evaluations with the seismic code, where each GPU computes a subdomain of 256^3 region. Unlike the diffusion case, the problem domain is decomposed over x and y dimensions; in other words, the domain is expanded in the x-y plane with the problem size of each GPU fixed. The decomposition implies that boundary exchanges involve non-unit stride data accesses, thus resulting lower scalability than the diffusion code. The performance of seismic benchmark exhibits significant drop at 64 GPUs and relatively low scalability afterward, which remains to be a subject of more detailed performance analysis.

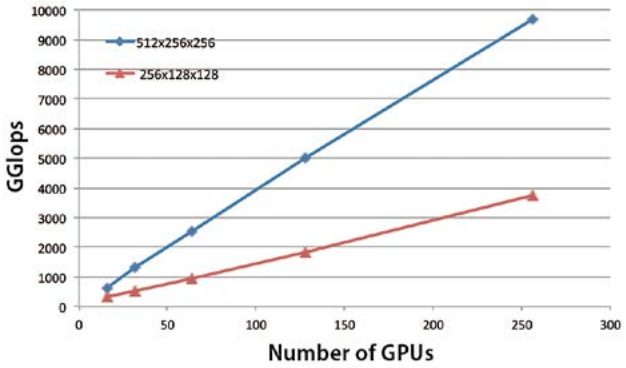


Figure 3 Weak scaling performance of Diffusion benchmark with up to 256 GPUs.

4-2 Strong Scaling Evaluation

Fig. 5 shows the strong scaling performance of the diffusion stencil with the problem size fixed at 512 x 512 x 4096. We evaluated 1-D, 2-D, and 3-D decompositions using up to 128 GPUs. In the 1-D decomposition, we uniformly decomposed the z-direction by the number of GPUs. In 2-D, we also decomposed the y-direction by two GPUs and again uniformly decomposed the z-direction with the rest of GPUs. Similarly, in the 3-D decomposition, we decomposed the x-direction into two GPUs in addition to the uniform y- and z- direction decompositions. As expected, the 1-D and 2-D cases performed better with a smaller number of GPUs, but as the number increases, the 3-D version outperformed the other two versions. While it is well known that 3-D decomposition often performs better in large-scale settings, our contribution is to allow application scientists to transparently enjoy such better performance and scalability without investing significant amount of efforts.

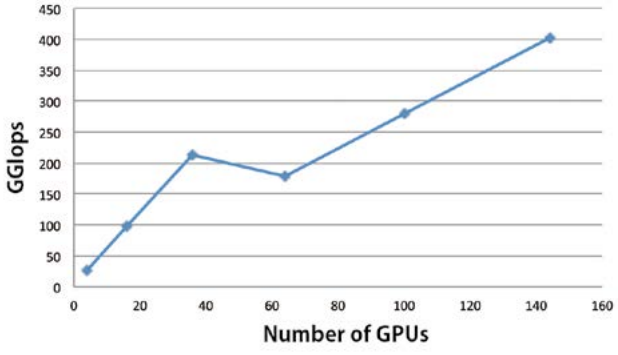


Figure 4 Weak scaling performance of Seismic benchmark with up to 144 GPUs. The problem size of each GPU is fixed at 256x256x256

Related Work 5

Recent developments of GPU accelerators for scientific computing have enabled low-cost power-efficient approaches to increase compute performances. However, one of the side effects of this trend is the significant decrease of programmer productivity due to the complexities involved in programming heterogeneous architecture. Although little work has been done for large-scale heterogeneous supercomputers, several recent projects attempted to solve the problem.

Mint is a high-level directive-based framework for stencil computations.[6] It allows for regular loop-based stencil programs to be annotated with its custom directives so that stencil loops can be executed on GPUs. Ypnos is a Haskell-based DSL for stencil computations that is designed so that compiler-based automatic parallelization is possible.[7] Both of them share common objectives with ours, such as automatic parallelization, but so far they are limited to single-GPU platforms, whereas our primary focus is to realize scalable multi-GPU implementations.

Listz is a DSL for unstructured mesh-based simulations.[8] As in our framework, actual implementations of the mesh interface are hidden from the programmer, which allows Listz to perform aggressive domain-specific optimizations. The implementation of the Listz is based on Scala's extensive language features, which facilitate developments of DSLs. Listz could be use to implement structured-grid stencil applications; however, since it targets unstructured meshes, it may not be able to fully exploit the optimization opportunities of structured data.

Conclusion 6

In order to improve programmer productivity on large-scale heterogeneous GPU clusters, we have designed and implemented the Physis framework that supports portable programming of stencil computations with structured grids. The C-based DSL represents a high-level declarative programming model for stencil computations. The DSL translator and runtime together realize an efficient implementation of the programming model with optimizations such as automatic overlapping of computations and communications. This paper presented our current framework implementation and evaluations of its productivity and performance. We have shown that our framework successfully generates scalable code for up to 256 GPUs. We plan to extend the presented framework for generating further optimized code and to evaluate its effectiveness using a wider variety of stencil applications.

Acknowledgements

This project was partially supported by JST, CREST through its research program: ``Highly Productive, High Performance Application Frameworks for Post Petascale Computing.'' We also thank MEXT Grant-in-Aid for Young Scientists (22700047), JST-ANR FP3C, and NVIDIA CUDA Center of Excellence.

References

[1] 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 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11), Seattle, WA, USA (2011)

[2] T.Shimokawabe, T.Aoki, C.Muroi, J.Ishida, K.Kawano, T.Endo, A.Nukada, N.Maruyama, S.Matsuoka, "An80-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 (2010)

[3] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, and W. M. W. Hwu, "Optimization principles and application performance evaluation of a multi- threaded GPU using CUDA," In Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming, PPoPP '08, zpages 73–82, (2008)

[4] M. Fowler. Domain-Specific Languages. Addison- Wesley Professional (2010)

[5] M. Schordan and D. Quinlan, "A source-to-source architecture for user-defined optimizations," In Modular Programming Languages, volume 2789 of Lecture Notes in Computer Science, pages 214–223. Springer Berlin / Heidelberg (2003)

[6] D. Unat, X. Cai, and S. B. Baden. "Mint: realizing CUDA performance in 3D stencil methods with annotated c," In Proceedings of the International Conference on Supercomputing (ICS'11), ICS '11, pages 214–224 (2011)

[7] D. A. Orchard, M. Bolingbroke, and A. Mycroft. "Ypnos: declarative, parallel structured grid programming," In DAMP '10: Proceedings of the 5th ACM SIGPLAN workshop on Declarative aspects of multicore programming, pages 15–24 (2010)

[8] [H. Chafi, Z. DeVito, A. Moors, T. Rompf, A. K. Su- jeeth, P. Hanrahan, M. Odersky, and K. Olukotun, "Language virtualization for heterogeneous parallel computing," In Proceedings of the ACM international conference on Object oriented programming systems languages and applications, OOPSLA '10, pages 835–847 (2010)

FTI: high performance Fault Tolerance Interface for hybrid systems

Leonardo Bautista-Gomez* Dimitri Komatitch** Naoya Maruyama* Seiji Tsuboi***
Franck Cappello † Satoshi Matsuoka ‡ Takeshi Nakamura***

* Tokyo Institute of Technology, INRIA ** Observatoire Midi-Pyrénées, University of Toulouse
*** JAMSTEC † INRIA, University of Illinois ‡ Tokyo Institute of Technology

Large scientific applications deployed on current petascale systems expend a significant amount of their execution time dumping checkpoint files to remote storage. New fault tolerant techniques will be critical to efficiently exploit post-petascale systems. In this work, we propose a low-overhead high-frequency multi-level checkpoint technique in which we integrate a highly-reliable topology-aware Reed-Solomon encoding in a three-level checkpoint scheme. We efficiently hide the encoding time using one Fault-Tolerance dedicated thread per node. We implement our technique in the Fault Tolerance Interface FTI. We evaluate the correctness of our performance model and conduct a study of the reliability of our library. To demonstrate the performance of FTI, we present a case study of the Mw9.0 Tohoku Japan earthquake simulation with SPECFEM3D on TSUBAME2.0. We demonstrate a checkpoint overhead as low as 8% on sustained 0.1 petaflops runs (1152 GPUs) while checkpointing at high frequency.

1 Introduction

In high performance computing (HPC), systems are built from highly reliable components. However, the overall failure rate of supercomputers increases with component count. Nowadays, petascale machines have a mean time between failures (MTBF) measured in hours or days[14] and fault tolerance (FT) is a well-known issue. Long running large applications rely on FT techniques to successfully finish their long executions. Checkpoint/Restart (CR) is a popular technique in which the applications save their state in stable storage, frequently a parallel file system (PFS); upon a failure, the application restarts from the last saved checkpoint. CR is a relatively inexpensive technique in comparison with the process-replication scheme that imposes over 100 % of overhead.

However, when a large application is checkpointed, tens of thousands of processes will each write several GBs of data and the total checkpoint size will be in the order of several tens of TBs. Since the I/O bandwidth of supercomputers does not increase at the same speed as computational capabilities, large checkpoints can lead to an I/O bottleneck, which causes up to 25% of overhead in current petascale systems[12]. Post-petascale systems will have a significantly larger number of components and an important amount of memory. This will have an impact on the system' s reliability. With a shorter MTBF, those systems may require a higher checkpoint frequency and at the same time they will have significantly larger amounts of data to save.

Although the overall failure rate of future post-petascale systems is a common factor to study when designing FT-techniques, another important point to take into account is the pattern of the failures. Indeed, when moving from 90nm to 16nm technology, the soft error rate (SER) is likely to increase significantly, as shown in a recent study from Intel [13, 6]. A recent study by Dong et al. explains how this provides an opportunity for local/global hybrid checkpoint using new technologies such as phase change memories (PCM) [2]. Moreover, some hard failures can be tolerated using solid-state-drives (SSD) [8] and cross-node redundancy schemes, such as checkpoint replication or XOR encoding [9] which allows to leverage multi-level checkpointing, as proposed by Moody et al. [1]. Furthermore, Cheng et al. demonstrated that more complex erasure codes such as Reed-Solomon (RS) encoding can be used to further increase the percentage of hard failures tolerated without stressing the PFS [3]. Our work goes in the same direction as these three studies and partially leverages some of those results.

1-1 Contributions

We propose a model of a highly reliable erasure code scheme based on our topology-aware RS encoding published in previous work [4]. We extend our previous research by studying not only the scalability of the encoding algorithm but also the impact of the checkpoint size per node and the group size, on encoding and decoding performance. We evaluate and prove the accuracy of our performance model and show that our topology-aware RS encoding scheme is several orders of magnitude more reliable than XOR encoding. We apply our FT-dedicated thread scheme presented in previous work [5] in order to further decrease the checkpoint overhead and we integrate it for the first time in a multi-level checkpoint technique that we implement in our FTI library. Our evaluation shows that by using FT-dedicated threads in the nodes we can efficiently hide the encoding time, making its overhead negligible in comparison with a simple local write checkpoint. We extend our evaluation with a functional test in a real case study by simulating the March 11th Mw9.0 Tohoku, Japan earthquake and we present synthetic seismograms for the Hirono seismic recoding station in Fukushima prefecture. We perform a large scalability and overhead evaluation of our library with SPECFEM3D and we show that FTI can successfully scale to more than 1000 GPUs and reach over 100 TFlops while checkpointing at high frequency and causing only about 8 % overhead in comparison with a not checkpointed execution.

EVALUATION 2

In this section we evaluate the performance, scalability and efficiency of our FTI library using SPECFEM3D. All our experiments were done on TSUBAME2.0 with the configuration given in table 1.

2-1 Simulating the March 11th Mw9.0 Tohoku Japan earthquake

In an effort to extend our evaluation with a functional test of FTI in a real case simulation with a production level application, we decided to simulate the devastating Mw9.0 Tohoku Japan earthquake that struck the northeast part of the island on March 11th, 2011. The simulation is done with SPECFEM3D on TSUBAME2.0 using an input model that describes the source fault.

SPECFEM3D is used by more than 300 research groups in the world for a large number of applications, for example to model the propagation of seismic waves resulting from earthquakes, seismic acquisition experiments carried out in the oil industry, or laboratory experiments with ultrasounds in crystals. This application won the Gordon Bell SuperComputing award for Best Performance [11] for a calculation of seismograms in the whole 3D Earth down to periods of approximately 5 seconds, carried out at 5 teraflops (sustained) on 1944 processors using 14.6 billion degrees of freedom stored in 2.5 terabytes of memory on the Earth Simulator, the fastest computer in the world at that time (2002).

For the source model, we apply waveform inversion [19] to obtain slip distribution in the source fault at the 2011 Tohoku, Japan earthquake in the same manner as Nakamura et al[22]. We use broadband seismograms of IRIS GSN and IFREE OHP seismic stations with epicentral distance between 30 and 100 degrees. The broadband original data are integrated into ground displacement and band-pass filtered in the frequency band 0.002-1 Hz. We use the velocity structure model of the earth IASP91 [18] to calculate the wavefield near the source and stations. We assume that the strike of the fault plane is 201 degree and the dip angle is 9 degree, based on the Global Centroid Moment Tensor model of the earthquake source. The length of a subfault is 20 km along strike. The assumed fault length is 440 km in total, consistent with the aftershock distribution.

The nonnegative least-squares method [21] is employed for constraining the rake angle in the waveform inversion. The results of the inversion show the bilateral rupture to the northeast and the southwest with two main asperities along the fault; maximum slip is of around 40 m with the reverse fault mechanism approximately 100 km northeast of the epicenter and another large slip with reverse fault mechanism at 100 km southeast of the epicenter. The total amount of released seismic moment corresponds to moment magnitude Mw = 9.1. We calculate synthetic seismograms with this source propagation model for a realistic 3D Earth model using the spectral-element method [20, 23]. As we can observe in figure 1, the synthetic seismogram for the Hirono seismic recording station located in the Fukushima prefecture, the East component shows about 2m static displacement to the East, which seems to be consistent with the observed crustal deformation caused by this earthquake. Figures 1 also shows the other two components at the same station.

Nodes	1408 High BW Compute Nodes
CPU	2 Intel Westmere-EP 2.93GHz 12Cores/node
Mem	55.8GB or 103GB (Total: 80.55TB)
GPU	NVIDIA M2050 515GFlops, 3GPUs/node (Total: 4224 NVIDIA Fermi GPUs)
SSD	60GB x 2 = 120GB (55.8GB node) Write speed : 360MB/s (RAID0) 120GB x 2 = 240GB (103GB node) (Total : 173.88TB)
Network	Dual rail QDR IB (4GB/s x 2)
File system	5 DDN DFA10000 units (3 Lustre and 2 GPFS) with 600 2TB HDDs each Measured Lustre write troughput (10GB/s)
OS	Suse Linux Enterprise + Windows HPC

Table 1 : TSUBME 2.0 Architecture

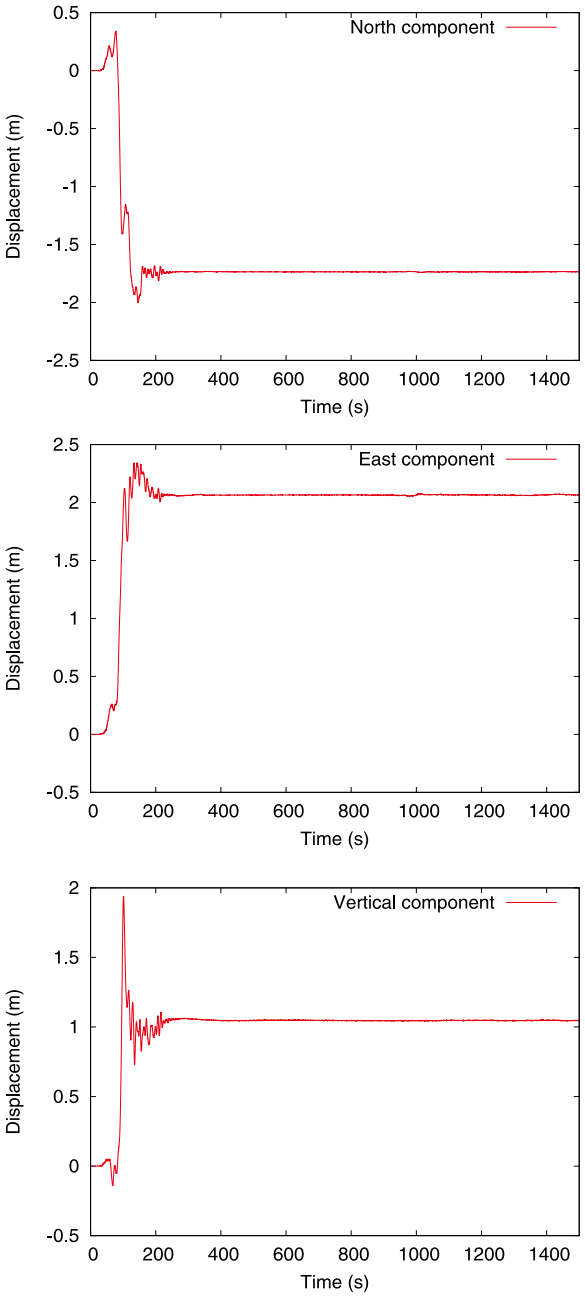


Figure 1 Synthetic seismograms for the Hirono station

2-2 FTI scalability study with SPECFEM3D

In order to demonstrate the efficiency and scalability of FTI we decided to evaluate it at large scale with SPECFEM3D. Recently, SPECFEM3D was ported to GPU clusters using CUDA [15, 16], so it can be used in hybrid systems such as TSUBAME2.0. It is important to notice that such seismic simulations do not need double precision and perform their runs in single precision [15]. Also, we want to highlight that SPECFEM3D is a memory-bound application, as any finite difference or finite element code; this is intrinsically related to the fact that in such numerical methods few operations are performed per grid point, and thus the cost comes mostly from memory accesses [15, 16].

First, we start with a strong scalability test in which we evaluate the performance of SPECFEM3D in three cases: no checkpointing, checkpointing with FTI (L2) and checkpointing on Lustre [7]. Since the problem size is fixed, the memory used (and therefore the checkpoint size) per GPU decreases when the number of GPUs increases. In this experiment, for the FTI tests all the checkpoints were done with the L2 of FTI, thus we do not take advantage of the multi-level scheme of FTI at this point. Since checkpoint size decreases, we also decrease the checkpoint interval in order to decrease the recovery cost in case of failure. All the checkpoints are done at the application level and we checkpoint only the strictly necessary data in order to restart the execution; this corresponds to about 20% of the memory used by the application.

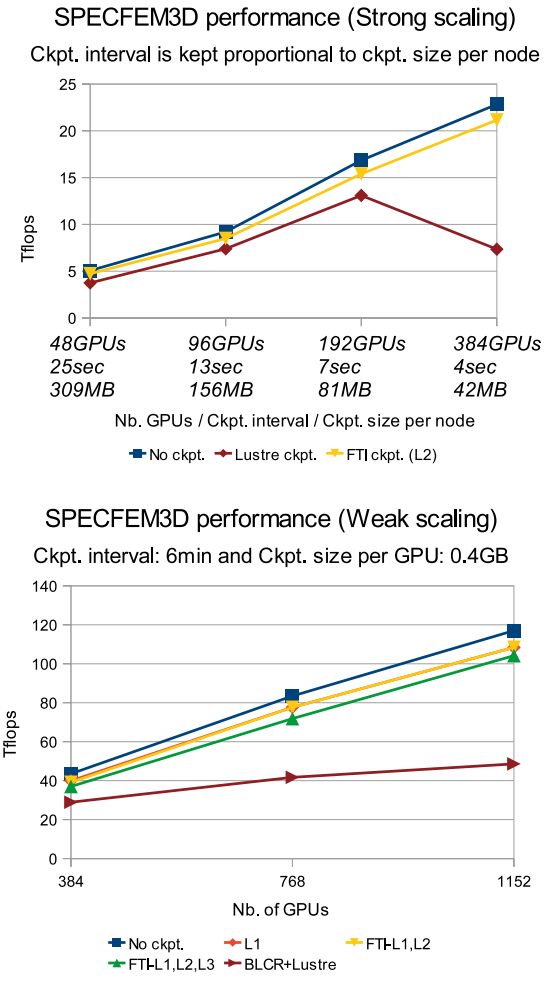


Figure 2 FTI evaluation on TSUBAME2.0

As we can see in figure 7a, SPECFEM3D strong-scales well from 5 TFlops on 48GPUs to almost 23 TFlops on 384GPUs without checkpointing. FTI follows closely this progression by causing only about 4 % of overhead for 384GPUs. In contrast, checkpointing to Lustre becomes prohibitively costly at high frequencies. Performance was measured using PAPI to measure the floating point operations [17] and each dot in the figure is the average of 5 runs.

To evaluate the overhead of FTI at large scale, we stressed even more our library by weak-scaling to more than 1000 GPUs. In this second experiment, we populate the GPUs memory with 2.1GBs of data, out of 2.6GBs available for the user (12.5 % is used for ECC in Fermi GPUs) and we keep the checkpoint interval fixed to 6 minutes, which is the Young's optimal checkpoint interval for a MTBF of 12 hours and a L1 checkpoint of 2 seconds. Then, we run SPECFEM3D for several configurations: The first one is without checkpoint (No ckpt.); the second one is checkpointing to the local SSDs without any encoding (L1); the third one is using FTI, thus in addition to the local checkpoint, every 2 checkpoints FTI will use the RS encoding proposed in our model (FTI-L1,L2); the fourth one is similar to the previous one but in addition every 6 checkpoints the latest checkpoint files are flushed to Lustre (FTI-L1,L2,L3); and finally checkpointing with BLCR on Lustre (BLCR+Lustre). Although there are some ongoing works [10] to make it possible, BLCR cannot currently checkpoint GPU-accelerated systems. Hence, we emulate it by writing 2.1GBs of data per process (therefore per GPU) to Lustre in the same way BLCR would do it. It is important to highlight that BLCR, as any other kernel-level checkpoint, will save the complete memory of every process, creating a 5 times larger checkpoint.

In figure 7b, we can see that SPECFEM3D has an almost perfect weak scaling, from 43TFlops to 117TFlops on 1152GPUs for the No ckpt. test. Also, in the figure the L1 results are actually hidden by the FTI-L1,L2 results. Indeed, both scenarios achieve almost identical results causing about 8% overhead in comparison with the No ckpt. case. This means, that the RS encoding done at L2 is completely hidden thanks to the FT-dedicated threads. The L1 checkpoints, capable of tolerating transient failures, are done between two L2 checkpoints while the FT-threads are still encoding the previous, more reliable, checkpoint. The FTI-L1,L2,L3 scheme adds an extra 3 % overhead due to Lustre writing performance. Finally, the BLCR+Lustre scheme imposes an always larger and prohibitive overhead as the size of the problem increases. For each run we let the application run between 30 and 40 minutes and every point in the figure is the average of 3 runs.

At this point, we have achieved over 100TFlops of sustained performance with a production-level scientific application such as SPECFEM3D, on an hybrid supercomputer such as TSUBAME2.0 and checkpointing with our library FTI every 6 minutes (high frequency checkpointing).

CONCLUSIONS 3

In this work we have proposed a highly reliable technique based on a topology-aware RS encoding. Also, we have exploited some characteristics of GPU computing through which many GPU applications are capable of spawning one extra FT-dedicated thread per node in order to improve checkpoint performance. We have integrated both techniques in a multi-level checkpoint model that we have implemented in our FTI library and we have conducted an exhaustive study of correctness of our performance model and the reliability of our library.

Moreover, we have conducted for the first time a large scale evaluation of such a multi-level technique with a production level scientific application on an hybrid platform. Our evaluation with SPECFEM3D on TSUBAME2.0 shows that FTI imposes only 8% of checkpoint overhead while running at over 0.1 petaflops and checkpointing every 6 minutes.

Acknowledgements

This work was supported in part by the JSPS, the ANR/JST FP3C project, and by the INRIA-Illinois Joint Laboratory for Petascale Computing.

References

[1] A. Moody, G. Bronevetsky, K. Mohror, B. R. de Supinski, Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System. In ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, New Orleans, 2010

[2] X. Dong, N. Muralimanohar, N. Jouppi, R. Kaufmann, Y. Xie. Leveraging 3D PCRAM Technologies to Reduce Checkpoint Overhead for Future Exascale Systems. In ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, Portland, 2009.

[3] Z. Cheng, J. Dongarra, A scalable Checkpoint Encoding Algorithm for Diskless Checkpointing. Proceedings of the 11th IEEE High Assurance Systems Engineering Symposium, HASE 2008, Nanjing, China, December, 2008.

[4] L. Bautista-Gomez, N. Maruyama, A. Nukada, F. Cappello, S. Matsuoka, "Low-overhead diskless checkpoint for hybrid computing systems", International Conference on High

Performance Computing, Goa, India, December 2010.

[5] L. Bautista-Gomez, N. Maruyama, F. Cappello, S. Matsuoka, "Distributed Diskless Checkpoint for large scale systems", IEEE/ACM International Symposium on Cluster, Cloud and Grid computing (CCGrid2010), Melbourne, Australia, May 2010.

[6] B. Schroeder, E. Pinheiro, W. Weber. DRAM errors in the wild: A Large-Scale Field Study. In Proceedings of the 11th international joint conference on Measurement and modeling of computer systems (SIGMETRICS), ACM, New York, NY, USA, 2009.

[7] S. Microsystems. Lustre file system, October 2008[8] A. Moody, G. Bronevetsky, Scalable I/O Systems via Node-Local Storage: Approaching 1 TB/sec File I/O. DOE technical report, 2009

[9] W. D. Gropp, R. Ross, and N. Miller. Providing efficient I/O redundancy in MPI environments. Lecture Notes in Computer Science, 3241:7786, September 2004.

[10] A. Nukada, S. Matsuoka, NVCR : A Transparent Checkpoint-Restart Library for NVIDIA CUDA in Proceedings at the International Heterogeneity in Computing Workshop, Alaska, 2011. (To appear)

[11] D. Komatitsch, S. Tsuboi, C. Ji and J. Tromp, A 14.6 billion degrees of freedom, 5 teraflops, 2.5 terabyte earthquake simulation on the Earth Simulator, Proceedings of the ACM / IEEE Supercomputing SC'2003 conference, November 2003.

[12] G. Grider, J. Loncaric, and D. Limpart, Roadrunner System Management Report, Los Alamos National Laboratory, Tech. Rep. LA-UR-07-7405, 2007.

[13] S. Y. Borkar, Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation, IEEE Micro, vol. 25, no. 6, pp. 10-16, 2005.

[14] D. Reed, High-End Computing : The Challenge of Scale, Director's Colloquium, LANL, May 2004.

[15] D. Komatitsch, D. Michéa, G. Erlebacher, Porting a high-order finite-element earthquake modeling application to NVIDIA graphics cards using CUDA, Journal of Parallel and Distributed Computing, vol. 69(5), p. 451-460, doi: 10.1016/j.jpdc.2009.01.006, 2009.

[16] D. Komatitsch, G. Erlebacher, D. Göddeke, D. Michéa, High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster, Journal of Computational Physics, vol. 229(20), p. 7692-7714, doi: 10.1016/j.jcp.2010.06.024, 2010.

[17] http://icl.cs.utk.edu/papi/

[18] B. Kennet, E. Engdahl, Traveltimes for global earthquake location and phase identification. Geophys. J. Int., 105, 429-465, 1991.

[19] M. Kikuchi, H. Kanamori, Note on Teleseismic Body-Wave Inversion Program, 2003. http://www.eri.u-tokyo.ac.jp/ETAL/KIKUCHI/

[20] D. Komatitsch, J. Ritsema, J. Tromp, The spectral-element method, Beowulf computing, and global seismology, Science 298, 1737-1742, 2002.

[21] C. Lawson, R. Hanson, Solving Least Squares Problems, Prentice-Hall, New Jersey, 340 pp, 1974.

[22] T. Nakamura, S. Tsuboi, Y. Kaneda, Y. Yamanaka, Rupture process of the 2008 Wenchuan, China earthquake inferred from teleseismic waveform inversion and forward modeling of broadband seismic waves, Tectonophysics, vol. 491, 72-84, 2010.

[23] S. Tsuboi, D. Komatitsch, C. Ji, J. Tromp, Broadband modelling of the 2002 Denali fault earthquake on the Earth Simulator, Phys. Earth Planet. Inter. 139, 305-312, 2003.

● **TSUBAME e-Science Journal No.5**

Published 02/28/2012 by GSIC, Tokyo Institute of Technology ©
ISSN 2185-6028
Design & Layout: Kick and Punch
Editor: TSUBAME e-Science Journal - Editorial room
Takayuki AOKI, Thirapong PIPATPONGSA,
Toshio WATANABE, Atsushi SASAKI, Eri Nakagawa
Address: 2-12-1-E2-1 O-okayama, Meguro-ku, Tokyo 152-8550
Tel: +81-3-5734-2087 Fax: +81-3-5734-3198
E-mail: tsubame_j@sim.gsic.titech.ac.jp
URL: http://www.gsic.titech.ac.jp/

TSUBAME

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