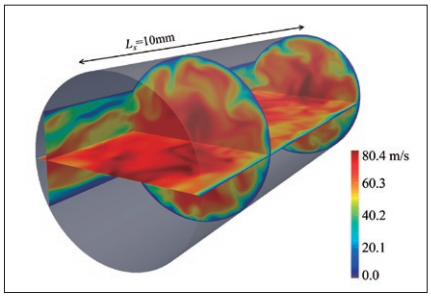
e-Science Journal
Global Scientific Information and Computing Center

Vol. 15

TSUBAME ESJ.



TTX: A Direct Numerical Simulation Code for Turbulent Reacting Flows

A High-level Framework for Parallel and Efficient AMR on GPUs

DS-CUDA: A Handy Tool to Use GPUs in a Cloud Network

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

TTX : A Direct Numerical Simulation Code for Turbulent Reacting Flows

Yuki Minamoto* Kozo Aoki** Mamoru Tanahashi*

* Department of Mechanical Engineering, Tokyo Institute of Technology **Mechanical and aerospace engineering, Tokyo Institute of Technology

Over eighty percent of world energy supply is estimated to be provided by burning fossil fuels for the next 30 years, and this imposes more and more stringent environmental regulations on the design of combustion related devices. Direct numerical simulations (DNS) have played important roles in the research of turbulent combustion. DNS data base provide key information for the development of turbulent combustion models, which are to be used in computational fluid dynamics (CFD) of various combustion devices during design and development phases. This article summarises a DNS code and implemented numerical techniques that are developed at Tokyo Institute of Technology.

1 Introduction

Over eighty percent of world energy supply is estimated to be provided by burning fossil fuels for the next 30 years [1], and this imposes more and more stringent environmental regulations on the design of combustion related devices. Direct numerical simulations (DNS) have played important roles in the research of turbulent combustion. DNS data base provide key information for the development of turbulent combustion models, which are to be used in computational fluid dynamics (CFD) of various combustion devices during design and development phases.

For decades, DNS have been used for canonical combustion problems such as statistically 1D planar propagating premixed flames and HCCI-type combustion in a periodic rectangular domain. However, with the recent advancement in high-performance computing, DNS of slightly-more complicated and computationally costly combustion configurations such as V-flame, jet flame and flames stabilized in jet-in-cross-flow and swirl flow with using complex chemical mechanisms have been performed, and such simulations will further our understanding on the physics of turbulent combustion. Since these configurations include walls that do not necessarily conform with the preferred structured mesh coordinates for combustion DNS, most of these simulations use presumed profiles for inflow/near-wall flows as boundary conditions. A portable high-order immersed boundary method suited for parallel computation is one way to improve these simulations. Also, the use of more practical and complicated hydrocarbon fuels such as methane, n-heptane and gasoline surrogates with complex chemical kinetic models is important to reveal underlying flame—turbulence interaction that is a key for the development of "green" combustor. However, computing elementary reaction rates existing in chemical mechanisms takes most of combustion DNS costs and a method to improve this computation speed is necessary to achieve such DNS. The present research implements such a boundary technique and methods to accelerate kinetic computation in a DNS code, TTX, and simulations are performed to confirm its accuracy and performance for several laminar/turbulent reacting/non-reacting flow problems.

2 Mathematical Background

2.1 Governing equations and PDE solver

The governing equations consist of fully compressible conservation equations for mass, momentum, energy and mass fractions of N-1 chemical species, where N is the number of chemical species involved in combustion. The equations are discretised by a fourth order central finite difference scheme and integrated in time by using a third order explicit Runge-Kutta scheme on Cartesian mesh. Unphysical numerical oscillations resulted from the use of finite difference scheme are removed by using either a sixth order explicit spatial filtering or a fourth order compact filter. All open computational boundaries are described based on the Navier-Stokes characteristic boundary conditions (NSCBC) formulation. The detailed description can be found in previous studies [2-6].

2.2 MTS and CODACT

One of the difficulties in conducting combustion DNS is the time integration of the governing equations with chemical reaction terms. The time stepping (Δt) for fully explicit

** Current position: Japan Atomic Energy Agency

schemes has to be smaller than the smallest chemical timescale which can become much smaller than the time stepping limited by a CFL condition. MTS (multi-timescale) method[7] is an algorithm for integrating ODEs associated with chemical reaction using a large Δt. In MTS method, each chemical species (Y_k) belongs to one integration group based on its chemical timescale (τ_k). The index (M) of an integration group which Y_k belongs to is obtained as $M = \log10\,(\Delta t / \tau_k) + 1$. Species in Mth group are integrated using a time stepping $\Delta t_M = \Delta t / 10^{(M-1)}$. The integration procedure starts with the smallest Δt_M. After convergence of Mth group, the mass fractions of the species in the group are fixed and then the ODE system continues to be integrated using Δt_{M-1}. Thus, MTS enables us to use a large Δt, which is not limited by the smallest chemical timescale, as a global time stepping for DNS.

Another approach to reducing computational cost is reduction of the reaction ODE system. CODAC (correlated dynamic adaptive chemistry)[8] is one of the methods for dynamically generating reduced kinetic mechanisms at each spatial location and each time step. In CODAC method, a phase space consisting of a few key parameters is constructed. Temperature and mass fractions of fuel, oxidizer and important radicals are selected as the key parameters. Comparing the instantaneous phase parameters at a grid point with those at another reference point (different point in time/space), correlation between the two points in the phase space is examined. Threshold values used for examining the correlation are specified by a user. If the phase parameters at the two points are correlated, the reduced mechanism used at the reference point is reused at the correlated grid point. If the phase parameters at a grid point are not correlated with any other reference point, a new reduced mechanism needs to be generated by PFA (path flux analysis)[9].

Further reduction of computational cost is achieved by using CODACT (CODAC and transport)[10]. The same correlation procedure in CODAC is applied to reducing the computation of
transport properties, i.e. viscosity, diffusivity and thermal conductivity. The phase parameters for reducing transport calculation are temperature and molar fractions of the first several abundant species. As with CODAC, transport properties at a grid point are not calculated if the phase parameters at the point are correlated with those at a different grid point.

2.3 Immersed boundary method

The present immersed boundary method (IBM) is based on well-known ghost region/reverse profile approach[11] with some modification to achieve high accuracy and portability. The important change from the conventional ghost region approach is that the ghost points are identified not only from the non-fluid points that face to the fluid region, but the region has certain non-fluid mesh points in the wall normal directions. The points width in wall normal direction is max (n_d, $n_f/2$) for uniform one-dimensional configuration, where n_d and n_f are the order of differentiation and explicit filter, respectively. Figure 1 shows an example schematic of ghost region for a $n_d = 4$ and $n_f = 6$ case. Once the ghost points are identified, a reference point is identified for each ghost point:

$$\mathbf{x}_{ref} = 2\mathbf{x}_{wall} - \mathbf{x}_{gs}, \qquad (1)$$

where the three $\mathbf{x}$ vectors denote the locations of reference point, corresponding wall and ghost point, respectively. For each physical quantity, q, solved in the set of governing equations, the reference value is obtained by interpolating the field onto the reference point. Using the reference point, the value at each ghost point is estimated depending on the wall conditions as:

$$q_{gs}(\mathbf{x}_{gs}) = -q(\mathbf{x}_{ref}) + 2q(\mathbf{x}_{wall}), \qquad (2)$$

$$q_{gs}(\mathbf{x}_{gs}) = q(\mathbf{x}_{ref}). \qquad (3)$$

Here, Eq. (2) is used for Dirichlet, and Eq. (3) is for Neumann wall boundary conditions. In the present implementation, ghost values for velocity and temperature are obtained by using Eq. (2), while species mass fractions and pressure are treated with Eq. (3). The density values on ghost points are obtained using a combination of Eqs. (2) and (3) to achieve mass conservation.

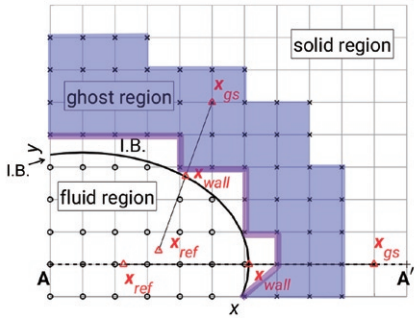


Fig. 1 Schematic of DNS domain. Blue shade indicates identified ghost region.

3 Performance and Accuracy

3.1 MTS and CODACT

A typical computational performance is shown in Fig. 2 in terms of weak scaling. The used DNS configuration is a periodic cuboid domain without using MTS, CODACT and IBM. The domain was initially filled with a reactant mixture of CH_4 - air preheated to 1000 K at 1 atm. The chemical reactions are described by using GRI-3.0 mechanism. The mixture is then ignited at the centre location by gradually imposing an energy source term having a profile based on a Gaussian distribution. Each CPU core are allotted 12^3 mesh points and simulations were continued for the first 100 time steps. Clearly, TTX has excellent parallel computation performance for a range of number of cores.

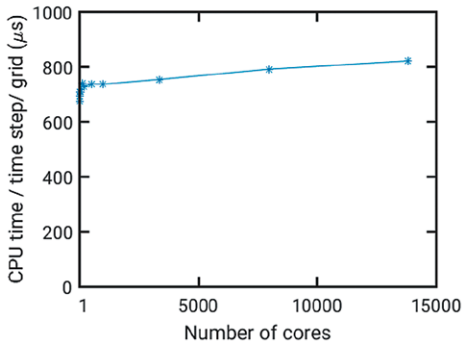


Fig. 2 Weak scaling plot of combustion DNS in a typical configuration using TTX.

A comparison between explicit time integration without CODACT (i), time integration using MTS without CODACT (ii) and time integration using MTS and CODACT (iii) is shown in Fig. 3 for a homogeneous ignition simulation of methane-air combustion. The time stepping for these cases are set to be Δt =1ns (i), 5ns (ii) and 5ns (iii). Smaller Δt for the case (i) is necessary as chemical time scales typically decreases rapidly with pressure rise. In Fig. 3, the temperature profiles show identical variation except for the explicit case (i) which starts to show a deviation after t=0.5ms. This is due to the lack of temporal resolution (Δt) since the chemical time scales are typically shortened as pressure increases with ignition proceeds. Thus, without the aid of MTS, Δt needs to be set significantly smaller values than other cases. However, the use of CODACT does not influence the results. Figure 4 shows accumulated computational times for the cases (i)—(iii). Clearly, due to significantly small Δt, the computational time for the case (i) takes longest of all. Clearly, the use of MTS and CODACT decreases computational by a factor of 7—8 times for complex chemistry. Similar trends are observed for more complex fuel combustion mechanism such as n-heptane.

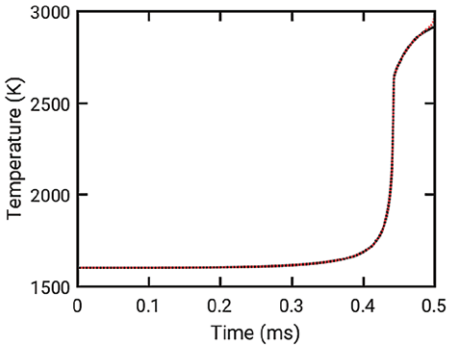


Fig. 3 Comparison of temperature profiles between the cases (i)—(iii).
Red: case (i),
black dashed: case (ii),
black solid: case (iii).

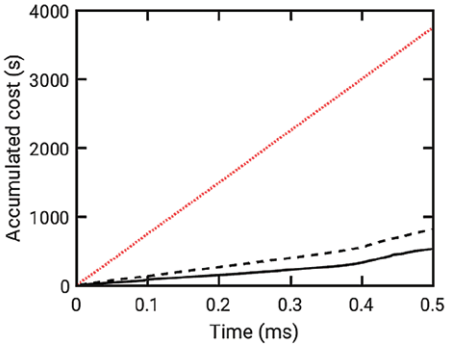


Fig. 4 Comparison of accumulated computational costs between the cases (i)—(iii).
Red: case (i), black dashed: case (ii),
black solid: case (iii).
black dashed: case (ii), black solid: case (iii).

3.2 Immersed boundary method

To access developed IBM, non-reacting laminar Taylor-Couette flows are simulated. The inner and outer walls are described based on the present IBM, and the DNS solutions at different spatial resolutions are compared with the analytical solution. The error convergence plot is shown in Fig. 5 in terms of maximum and L_2 norm of relative errors. The convergence

rate of L_2 norm decreases at a rate of between third and fourth orders, and this is comparable with the accuracy of differentiation in the interior points in the present DNS.

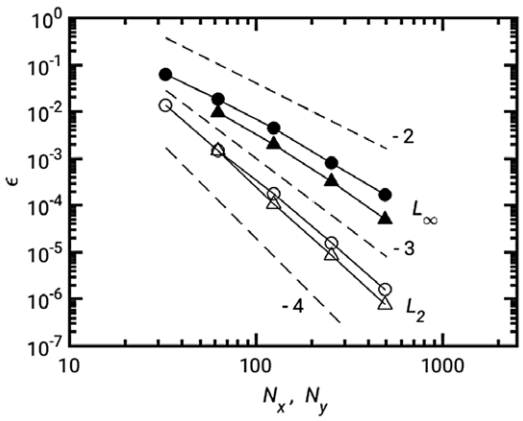


Fig. 5 Error convergence plot of the present IBM obtained from a laminar Taylor-Couette DNS.

DNS of a turbulent pipe flow is performed. The present pipe wall boundary conditions are specified as in the developed IBM, whereas previous pipe flow DNS database applied structured mesh that represent the pipe shape. The initial nominal Reynolds number Re_τ is set as 200, which is defined based on the friction velocity and radius of the pipe. The dimensions of the pipe are shown in Fig. 6 with the DNS result for the streamwise velocity field. The simulation was continued for over 40 mean-flow-through times to achieve quasi-steady state and samples are collected after 25 mean-flow-through times for statistics. Turbulent statistics are compared with previous DNS with conventional wall boundary method for mean streamwise velocity and Reynolds shear stress (not shown) in Fig. 7. The present pipe flow DNS results show excellent agreement with the previous DNS results with $Re_\tau = 180$ and 181, ensuring that the present IBM can capture the large velocity gradient and reproduce the turbulent boundary layer accurately. Also, the computational cost per time step is 1020 ms with IBM and when IBM is switched off, it decreases to 950 ms with 1296 cores at Reedbush-U system at the University of Tokyo. Therefore, the additional computational cost due to the IBM is less than 10% of the overall cost.

3.3 3D Turbulent combustion simulations

Typical DNS results of three-dimensional turbulent combustion performed by using TTX is shown in Fig. 8. With the numerical techniques described here, the turbulent combustion DNS will evolve from a current relatively small, canonical configuration to a larger scale, more complex geometry that is more relevant to practical combustion devices.

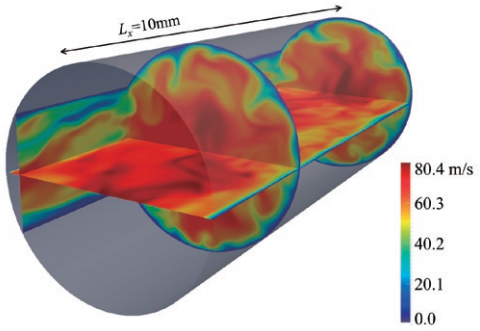


Fig. 6 Configuration and streamwise velocity field of turbulent pipe flow DNS.

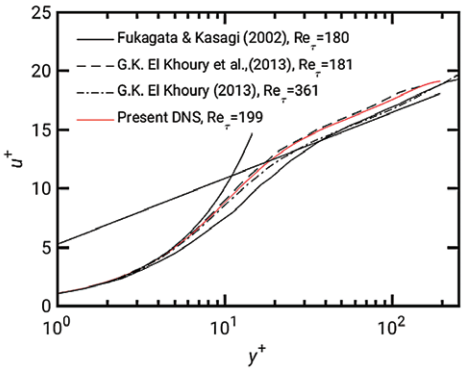


Fig. 7 Comparison of the mean streamwise velocity profile of the pipe flow DNS.

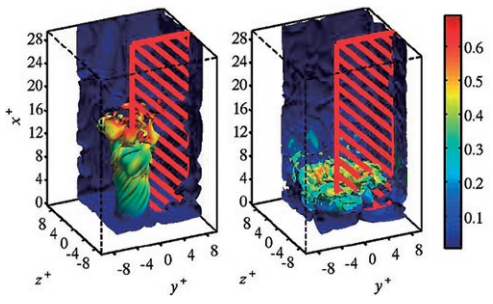


Fig. 8 Typical combustion DNS results of swirl flow stabilised turbulent flames of hydrogen-air combustion. Iso-surfaces: heat release rate. Color: normalised temperature. Left: low swirl number (0.6) case.
Right: high swirl number (1.2) case

Summary 4

A direct numerical simulation code, TTX, is developed by implementing MTS, CODACT and IBM capability for better computational speed and relatively flexible boundary configurations. The computational speed is increased by a factor of 7 to 8 when DNS are performed with both MTS and CODACT for a complex chemical mechanism, while the implemented IBM shows comparable accuracy to the accuracy of present numerical schemes.

Acknowledgements

This research was supported in part by Grant-in-Aid for Young Scientists B (16K18026) of the Japan Science and Technology Agency (JST).

References

[1] U.S. Energy Information Administration: International energy outlook 2010, DOE/EIA-0484 (2010)

[2] M. Tanahashi, M. Fujimura, and T. Miyauchi: Coherent fine-scale eddies in turbulent premixed flames, Proc. Combust. Inst., Vol. 28, pp. 529–535 (2000)

[3] Y. Minamoto, N. Fukushima, M. Tanahashi, T. Miyauchi, T. D. Dunstan, and N. Swaminathan: Effect of flow-geometry on turbulence-scalar interaction in premixed flames. Phys. Fluids, Vol. 23, 125107 (2011)

[4] M. Shimura, K. Yamawaki, N. Fukushima, Y. S. Shim, Y. Nada, M. Tanahashi, and T. Miyauchi: Flame and eddy structures in hydrogenair turbulent jet premixed flame, J. Turbulence, Vol. 13, pp. 1–17 (2010)

[5] B. Yenerdag, N. Fukushima, M. Shimura, M. Tanahashi, and T. Miyauchi: Turbulence-flame interaction and fractal characteristics of H2-air premixed flame under pressure rising condition. Proc. Combust. Inst., Vol. 35, pp.1277–1285 (2015)

[6] Sussman, P. Smereka and S. Osher: A Level Set Approach for Computing Solutions to Incompressible Two-Phase Flow, J. Comp. Phys., Vol. 114, pp.146-159 (1994)

[7] X. Gou, W. Sun, Z. Chen and Y. Ju: A dynamic multi-timescale method for combustion modeling with detailed and reduced chemical kinetic mechanisms, Combust. Flame, Vol. 157, pp. 1111-1121 (2010)

[8] W. Sun, X. Gou, H.A. El-Asrag, Z. Chen and Y. Ju: Multi-timescale and correlated dynamic adaptive chemistry modeling of ignition and flame propagation using a real jet fuel surrogate model, Combust. Flame, Vol. 162, pp. 1530-1539 (2015)

[9] W. Sun, Z. Chen, X. Gou and Y. Ju: A path flux analysis method for the reduction of detailed chemical kinetic mechanisms, Combust. Flame, Vol. 157, pp. 1298-1307 (2010)

[10] W. Sun and Y. Ju: Multi-timescale and Correlated Dynamic Adaptive Chemistry and Transport Modeling of Flames in n-Heptane/Air Mixtures, Proc. 53rd AIAA Aerospace Sciences Meeting (2015).

[11] P. Parnaudeau, E. Lamballais, D. Heitz, J. H. Silverstrini: Combination of the Immersed Boundary Method with Compact Schemes for DNS of Flows in Complex Geometry, ERCOFTAC seriese, Direct and Large-Eddy Simulation V, Vol. 9, pp. 581-590.

A High-level Framework for Parallel and Efficient AMR on GPUs

Mohamed Wahib* Naoya Maruyama* Takayuki Aoki**
* RIKEN Advanced Institute for Computational Science **Global Scientific Information and Computing Center, Tokyo Institute of Technology

Adaptive Mesh Refinement methods reduce computational requirements of problems by increasing resolution for only areas of interest. However, in practice, efficient AMR implementations are difficult considering that the mesh hierarchy management must be optimized for the underlying hardware. Architecture complexity of GPUs can render efficient AMR to be particularity challenging in GPU-accelerated supercomputers.
This article presents a compiler-based high-level framework that can automatically transform serial uniform mesh code annotated by the user into parallel adaptive mesh code optimized for GPU-accelerated supercomputers. Experimental results on three applications show that the speedups of code generated by our framework are comparable to hand-written AMR code, while achieving good and weak scaling up to 3,640 GPUs of the TSUBAME2.5 supercomputer.

Introduction 1

In many scientific and engineering simulations, Partial Differential Equations (PDEs) are solved in a uniform mesh arrangement by using finite difference schemes, referred to as iterative stencils. Typically, the resolution of the mesh is uniformly set to the highest resolution to provide accurate solutions. For meshes that require only high resolution for some portions of the mesh, an alternative method, known as Adaptive Mesh Refinement (AMR), can be used instead of the uniform mesh. The AMR method solves the problem on a relatively coarse grid, and dynamically refines it in regions requiring higher resolution. However, AMR codes tend to be far more complicated than their uniform mesh counterparts due to the software infrastructure necessary to dynamically manage the hierarchical mesh framework. Despite this complexity, it is generally believed that future applications will increasingly rely on adaptive methods to study problems at unprecedented scale and resolution.

Implementing efficient adaptive meshes in GPU-accelerated systems is significantly hard in comparison to traditional CPU systems. More specifically, GPUs add complexity overhead for managing the mesh hierarchy and optimization of data movement. This is made evident by the relatively wide use of AMR in CPU in comparison to GPU-based systems. For example, a mature AMR framework supporting CPU, namely FLASH is reported to be in use by dozens of production applications[1]. On the contrary, a few number of individual applications adapted AMR solvers for GPU, with varying levels of optimization and scaling. To summarize the problems with GPU-based AMR, only a few frameworks enable automated AMR transformations for GPU, and their programming models require the programmer to write his own versions of the target-optimized solvers. Moreover, there can be scalability limitations caused by the overhead of the CPU-GPU communication schemes in those frameworks[2].

In this article, we present a high-level framework, called Daino[3], which auto-generates efficient and scalable structured AMR solutions to scientific applications running on GPU-accelerated systems.

Background 2

Structured AMR methods use logically rectangular meshes in the implementation of the adaptive mesh. Structured AMR utilizes a hierarchy of levels of spatial, and often temporal, mesh spacing with each level being composed of a union of logically rectangular mesh regions. One way to represent the structured AMR, namely tree-based AMR, divides the discretized domain into fixed blocks. If any cell within a block requires refinement, the whole block is refined.

In the tree-based scheme, the mesh is organized into a hierarchy of refinement levels. The mesh is usually decomposed into relatively small fixed-sized octants of mesh cells. Each octant can be recursively refined into a set of octants of fine cells. The mesh configuration is managed using a tree-based data structure that maintains explicit child-parent relationships between coarse and fine octants. Size relations between neighboring octants are typically enforced in structured AMR, which means neighboring octants can have at most one level of refinement difference (referred to as 2:1 balance). An important feature of octrees is that the traversal of an octree across its leaves corresponds to a Morton z-shaped Space Filling Curve (SFC) in the geometric domain[4]. Accordingly, sorting the blocks by their Morton ID and equally

partitioning them leads to a uniform distribution of the blocks of a mesh among different Processing Elements (PEs), while benefiting from the locality provided by SFC affinity. Fig.1 illustrates how the domain and tree are represented in AMR, and the use of SFC to divide the blocks among three PEs.

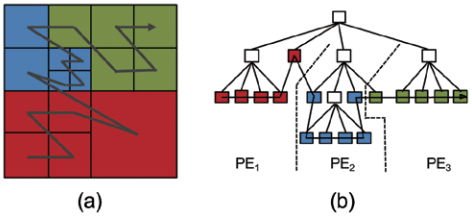


Fig. 1 Octree-based meshes. The blocks are equally partitioned using a space filling curve.
(a) Adaptive mesh
(b) Tree representation
(Note: 2D quadtree is used for illustration)

High-level Framework for Efficient AMR 3

We design a high-level programming framework that provides a highly productive programming environment for AMR. The framework is transparent and requires minimal involvement from the programmer, while generating efficient and scalable AMR code. The framework consists of a compiler and runtime components. A set of directives allows the programmer to identify stencils of a uniform mesh in an architecture- neutral way. The uniform mesh code is then translated to GPU-optimized parallel AMR code, which is then compiled to an executable. The runtime component encapsulates the AMR hierarchy and provides an interface for the mesh management operations.

3.1 Programming Model

The framework provides directives to be used with standard C (see [3] for details on directives). The programmer is required to add the directives to a serial uniform mesh code in order to identify the operations and stencil data arrays that are the target for transformation. Note that the directives are not changing the uniform mesh implementation; the programmer can still use the uniform mesh implementation if the directives are ignored by compiler. A sample example of using directives to annotate a C kernel in Daino is shown in Fig. 2.

```
#pragma dno kernel
void func(float ***a, float ***b, ..) {
#pragma dno data domName(i, j, k)
a, b;
#pragma dno timeloop
for(int t; t< TIME_MAX;t++) {
for(int i; i<NX; i++)
for(int j; i<NY; j++) {
  ... // comput. not related to a and b
  for(int k; k<NZ; k++) {
     a[i][j][k] = c*(b[i-1][j][k] +
     b[i+1][j][k] + b[i][j][k]) +
     b[i][j+1][k] + b[i][j-1][k]);
  } } } }
```

Fig. 2 Minimal example of using Daino directives

3.2 Optimizations

When an AMR code generated by Daino is executed on a GPU-accelerated cluster, the stencil and mesh adaptation kernels run on the GPU, while managing the octree data structures and load balancing is done on the CPU side. Since we pursue efficiency and scalability, code on both the CPU and GPU should be optimized. The stencil operations in the blocks are themselves optimized for the GPU architecture[5]. Other optimizations, such as data layout in memory and using user-managed cache memory, are applied on the GPU kernels responsible for adapting the mesh: error estimation, refinement (interpolation), and coarsening (extrapolation). Finally, the generated code includes optimizations to reduce the communication between nodes, i.e. boundary data exchange, and balancing the load, i.e. number of blocks per node.

3.3 Implementation

Our framework consists of a compiler and runtime components. We generate executables optimized for GPU execution by leveraging the LLVM compiler infrastructure. The compiler builds on the LLVM compiler infrastructure[6]. First, we use the front end to analyze and translate the stencil source code into GPU-optimized code in the form of LLVM Intermediate Representation (IR). Next, compiler passes are applied on the IR to add the AMR management code, which in turn make API calls to the runtime API and GPU-optimized

code generated by the Nvidia backend code generator. Finally, LLVM IR is compiled and linked with the runtime libraries to generate and executable.

The runtime includes two libraries: the first library encapsulates the AMR hierarchy management software and the second is a communication library that wraps the MPI runtime library to simplify data movement operations for the AMR driver. The stages of compilation and layout of Diano are shown in Fig. 3.

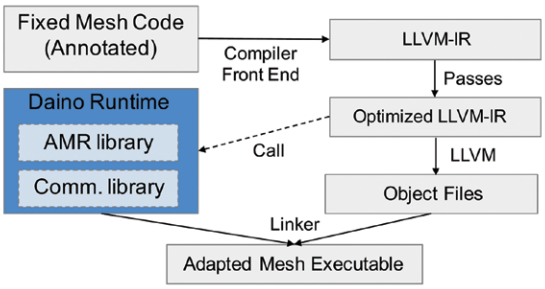


Fig. 3 The framework overview

4 Evaluation

We demonstrate the scalability of auto-generated AMR code using three production applications. We compare the speedup and scalability with hand-written AMR of all three applications using the TSUBAME2.5 supercomputer at Tokyo Institute of Technology.

4.1 Applications

Phase-field Simulation : This application simulates 3D dendritic growth during binary ally solidification[7].
Hydrodynamics Solver : This solver models a 2nd order directionally split hyperbolic schemes to solve Euler equations[8].
Shallow-water : Modelling shallow water by depth-averaging Navier-Stokes equations[9].

4.1 Results

In a weak scaling experiment, shown in Fig. 4, the run- time for uniform mesh, hand-written AMR, and auto-generated AMR are compared. The following points are important to note.

First, more than 1.7x speedup is achieved using Daino using up to 3640 GPUs of TSUBAME for the phase-field simulation. This is a considerable improvement considering that the uniform mesh implementation is a Gordon Bell prize winner for time-to-solution[7]. Second, Daino achieves good scaling that comparable to the scalability of the hand-written AMR code.

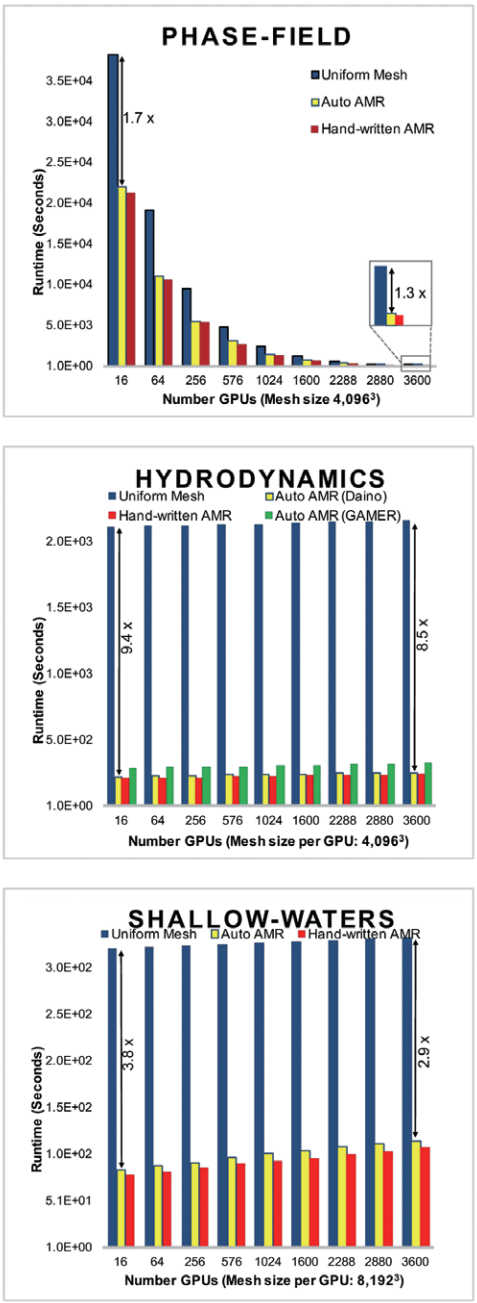


Fig. 4 Weak scaling of uniform mesh, hand-written and automated AMR

Fig. 5 shows a strong scaling comparison for hand-written and auto-generated AMR against uniform mesh implementation. The code generated by Daino achieves speedups and scalability comparable to hand-written implementations. However, when using more GPUs, reduction in speedup starts to occur as the management of AMR starts to dominate the simulation runtime.

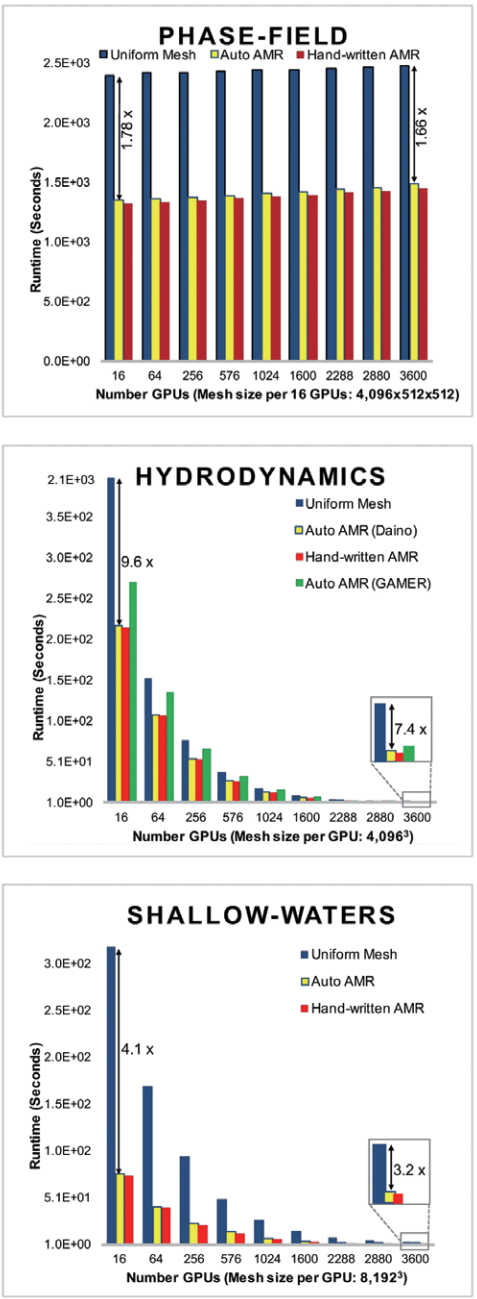


Fig. 5 Strong scaling of uniform mesh, hand-written and automated AMR

Summary 5

Producing efficient AMR code is a challenge, especially for GPUs. We introduce a framework for producing efficient and distributed AMR code for GPU-accelerated systems. We demonstrated the efficacy and scalability of three applications using the full TSUBAME supercomputer. To the authors knowledge, this is the first study to scale auto-generated AMR code to O(1,000s) of GPUs. However, there are still problems to be solved, for example, handling of customized error functions and boundary conditions.

Acknowledgements

This project is partially supported by JST, CREST through its research program: "Highly Productive, High Performance Application Frameworks for Post Petascale Computing.". This research is also partly supported by KAKENHI, Grant-in-Aid for Scientific Research (S) 26220002 from the Ministry of Education, Culture, Sports, Science and Technology (MEXT) of Japan and "Joint Usage/Research Center for Interdisciplinary Large-scale Information Infrastructures (JHPCN)" and "High Performance Computing Infrastructure (HPCI)" in Japan.

References

[1] A. Dubey, K. Antypas, M. K. Ganapathy, L. B. Reid, K. Riley, D. Sheeler, A. Siegel, and K. Weide, "Extensible Component-based Architecture for FLASH, a Massively Parallel, Multiphysics Simulation Code," Parallel Comp., vol. 35, no. 10-11, pp. 512–522, (2009)

[2] M. Wahib, N. Maruyama, Data-centric GPU-based Adaptive Mesh Refinement, IA^3 2015, Workshop on Irregular Applications: Architectures and Algorithms, co-located with SC'15, Austin, TX (2015)

[3] M. Wahib, N. Maruyama, T. Aoki, Daino: A High-level Framework for Parallel and Efficient AMR on GPUs, SC'16, ACM/IEEE Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (2016)

[4] H. Tropf and H. Herzog, "Multidimensional Range Search in Dynamically Balanced Trees," Angewandte Informatik, 1981 (1981)

[5] N. Maruyama and T. Aoki, Optimizing Stencil

Computations for NVIDIA Kepler GPUs, 1st International Workshop on High-Performance Stencil Computations HiStencils'14 (2014)

[6] http://www.llvm.org

[7] T. Shimokawabe, T. Aoki, T. Takaki, T. Endo, A. Yamanaka, N. Maruyama, A. Nukada, and S. Matsuoka, "Peta-scale Phase-field Simulation for Dendritic Solidification on the TSUBAME 2.0 Super- computer," ser. SC '11 (2011)

[8] H.-Y. Schive, U.-H. Zhang, and T. Chiueh, "Directionally Unsplit Hydrodynamic Schemes with Hybrid MPI/OpenMP/GPU Parallelization in AMR," Int. J. High Perform. Comput. Appl., vol. 26, no. 4, pp. 367–377 (2012)

[9] M. Sætra, A. Brodtkorb, K. Lie, "Efficient GPU-implementation of Adaptive Mesh Refinement for the Shallow-Water Equations," J. Sci. Comput., vol. 63, no. 1, pp. 23–48 (2015)

DS-CUDA : A Handy Tool to Use GPUs in a Cloud Network

Tetsu Narumi*
* Faculty of Informatics and Engineering, University of Electro-Communications

Cloud services have recently added GPU support for gaming and HPC purposes.
Our DS-CUDA offloads only the calculation on GPUs to a cloud system using CUDA.
The merit of DS-CUDA is that users can freely modify their applications on the client side,
as well as supporting tablet sensors and fault tolerance of networks.
We report four topics on DS-CUDA: 1) Overview of DS-CUDA, 2) 880 fold acceleration of replica-exchange molecular dynamics simulation using 1,024 GPUs on the TSUBAME 2.5 supercomputer,
3) Overhead of fault tolerant function using overclocked GPUs,
and 4) Energy efficiency using a tablet and GPU notebook.

Introduction 1

Several services and tools have been released to reduce the cost of preparing high-performance GPUs (Graphics Processing Units) on a client device. Amazon EC2 [1] provides a cloud server with 16 GPUs for GPGPU (General-Purpose computing on GPUs) applications. Nvidia GRID [2] enables us to run high-end GUI applications on a low-performance client device by utilizing GPU-enabled calculation and rendering on the cloud server and transferring the result via a low-latency network.

In this report, we consider an interactive molecular dynamics simulation [3], which calculates on GPUs with CUDA (Compute Unified Device Architecture) and visualizes at the same time. Seamless methods to use high-end GPUs on a cloud device to achieve interactive simulations and visualizations would be required from mobile devices like tablets which supports many sensors.

DS-CUDA [4, 5] is a tool to virtualize cloud-side GPUs as if they are attached to a client side. It supports fault tolerance of GPUs and networks. In this report, the overview of DS-CUDA and their recent results are presented.

Role sharing of a client device and cloud servers 2

Doing everything on one computer often results in higher cost or lower energy-efficiency compared to role sharing of client- and cloud-side computers. Fig. 1 shows three ways of role sharing between a client device and cloud servers: A) Most of the calculation and rendering are performed on a cloud server, B) Only rendering is performed on a cloud server, and C) Only CUDA calculation is performed on a cloud server.

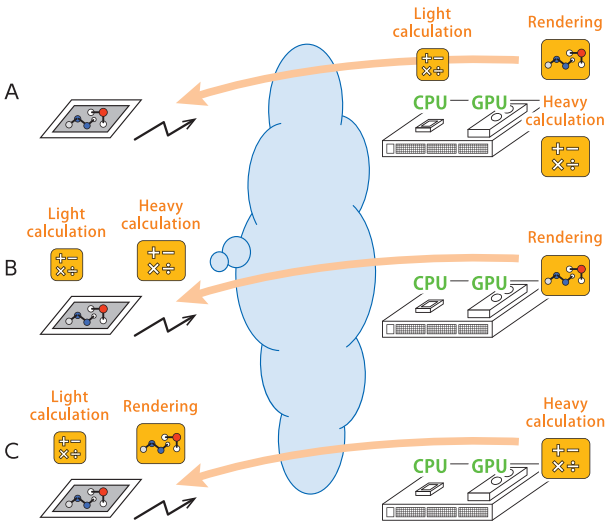


Fig. 1 Role sharing of a client device and cloud servers

In the case of A), the client works as a zero-client computer, which means that the keyboard input or touch information of the client device is transferred to the cloud server, and the rendered video or sound is transferred from the cloud server. However, cloud service like Nvidia GRID does

not support the recompiling and execution of user's own applications on the cloud server. They only provide popular ISV applications. Amazon EC2 supports the compilation and execution of any application since users can customize the development environment on the virtual machine, while the video transfer via a network might become the bottleneck. Special mechanism will be needed to support tablet sensors like accelerometers on a client device for both services.

In the case of B), a tool can hook rendering APIs like OpenGL on a client device and transfer them to a cloud server [6]. However, CUDA is not supported, which is a big disadvantage for HPC applications.

In the case of C), rendering and light processing including sensors are performed on a client device. Users can compile their applications on a familiar environment without changing their source codes. DS-CUDA or rCUDA [7] is suitable for such situations. Users can enjoy the benefit of high-end GPUs on a cloud server by hooking CUDA APIs in their applications.

Comparison against other tools 3

Table 1 compares the cloud tools mentioned in the previous section. DS-CUDA and rCUDA in method C) can easily support specific functions of applications since most parts run on the client side. Only the information called from CUDA APIs are transferred via a network, and attackers cannot see readable information like files on the cloud server, which should increase security. However, they currently do not support interoperability functions of OpenGL and Direct3D, which is necessary to render the result of CUDA calculation without moving data from the GPU memory.

The recompilation of CUDA code is not needed in the case of rCUDA since the change of the path of dynamic link library is enough, while DS-CUDA needs recompilation. rCUDA requires a CUDA environment on the client side, while DS-CUDA supports client devices without CUDA when specially prepared [8, 9]. Special environment like CUDA is not needed on a client side for Nvidia GRID or Amazon EC2, but users have to prepare the same software-development environment on the cloud server.

The merit of DS-CUDA is that it supports fault tolerant mechanism for the cases when GPUs caused calculation errors or a network is unstable. The reliability of GPUs are usually lower than those of CPUs even when they are specially designed for GPGPU [10]. The fault tolerance should be useful for reliable simulations, especially for unstable network environments in cloud computing.

In the method of A), implementing fault tolerance is a little difficult. Users can customize their applications to support fault tolerance if Amazon EC2 is used. However, the cost of such modification is usually very high. In the method of C), users have potentially no cost for fault tolerance since the tool can virtually provide reliable CUDA APIs to them.

Tool	Nvidia GRID	Amazon EC2	rCUDA	DS-CUDA
Video transfer	Fast	Slow	Not needed	Not needed
Client specific function like accelerometer	Not supported	Not supported	Work	Work
Interoperability function between CUDA and OpenGL/Direct3D	Work	Work	Not supported	Not supported
Recompilation of CUDA code	Not assumed	Not needed	Not needed	Required
Required software for a client side	Nvidia GRID	None	CUDA	Ruby, (CUDA)
Fault tolerant mechanism	No	No	No	Yes

Table 1 Comparison of tools for using cloud GPUs

Overview of DS-CUDA 4

Fig. 2 and 3 show the procedure of compilation and the block diagram of DS-CUDA system, respectively.

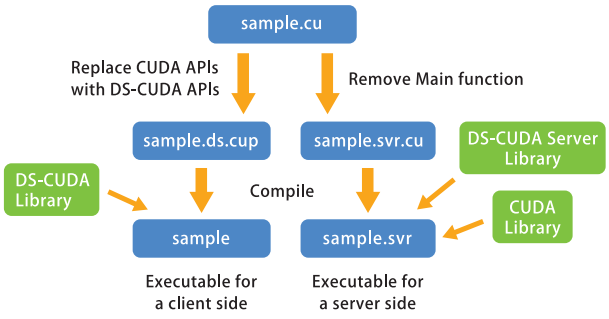


Fig. 2 Compilation with DS-CUDA compiler

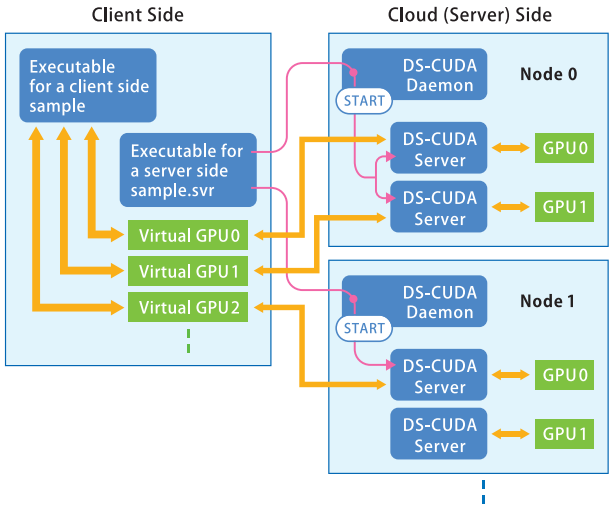


Fig. 3 Block diagram of a DS-CUDA system

First, a CUDA source code is compiled with DS-CUDA compiler, which is actually a Ruby script, and two executables are generated for both the client and cloud (or server) sides (Fig. 2). On the client side, CUDA APIs are replaced with DS-CUDA APIs, and compiled with the DS-CUDA library to generate the executable. On the server side, the main function is removed from the source code, and compiled using the nvcc compiler with the DS-CUDA server library, which includes the body of a server as well as the CUDA library to generate a server executable.

Before running the user application, a DS-CUDA daemon should be ready for each node (Fig. 3). When the client executable is started, the server executable is transferred from the client side to the server side and started via the DS-CUDA daemon. Every time the client executable accesses the virtual GPU in the client device, its data is transferred to the server side via a network through the DS-CUDA server, and the server accesses the physical GPU in the node.

When the function of redundant calculations is activated for fault tolerance, each CUDA API is automatically copied to two identical APIs and executed in two servers. Then the results from cudaMemcpy(DeviceToHost) APIs are compared. If the results do not match, previous APIs are automatically re-executed. When the migration function is activated, data in GPU memory which is allocated by cudaMalloc() API are periodically backed up to the client side. If unrecoverable error occurs on GPUs or network between client and servers is disconnected, backed-up data are transferred to a new GPU to continue the calculation.

Using many GPUs from a single thread 5

CUDA provides APIs to use multiple GPUs in a node even from a single thread program. However, parallel programming like MPI is needed to use many GPUs in a cluster by controlling them node by node. Beginners in GPGPU might feel difficulty in programming since a distributed memory paradigm must be considered. Programming of GPUs with DS-CUDA is simple since the client sees all the GPUs in the server side as if they are attached to the client. In the following, result of using 1,024 GPUs from a single thread program is explained as an extreme case [11].

In this case, replica-exchange Molecular Dynamics (MD) simulation with 14,336 replicas was executed with 1,024 GPUs in the TSUBAME 2.5 supercomputer. Replica-exchange MD simulation can search energy minimum state efficiently by exchanging temperature between parallel MD simulations with different temperatures every constant steps (100 in this result). One replica holds 256 or 2,048 Argon particles. Since the amount and frequency of the communication between different replicas are small, a single thread is enough to control all the GPUs in the cluster.

Fig. 4 shows the strong scaling of the relative calculation speed. 880 and 340 times acceleration is achieved with 1,024 GPUs for 256 and 2,048 particles, respectively.

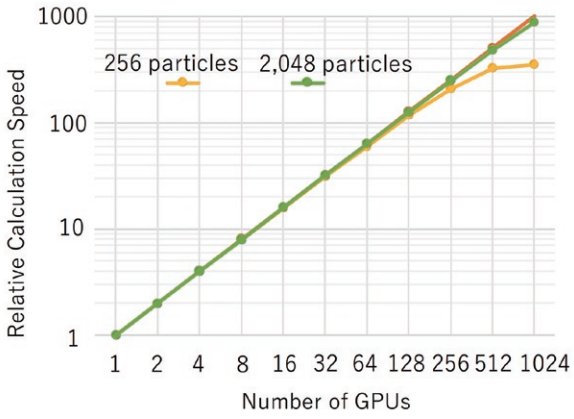


Fig. 4 Strong scaling of performance against the number of GPUs

Fig. 5 shows the time ratio of calculation and communication with 2,048 particles for 256, 1,024, and 4,096 GPUs. We made a performance model which fits up to 1,024 GPUs and applied it to 4,096 GPUs.

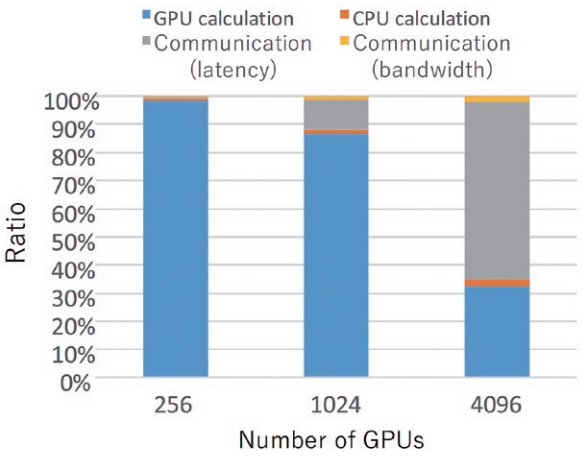


Fig. 5 Ratio of calculation and communication

In the performance model, the total calculation time is divided into three parts: GPU calculation, CPU calculation, and communication. The communication is further divided into two parts: Latency-oriented communication time and bandwidth-oriented communication time. The average amount of the communication is only 1.9 kbytes, which is relatively small. Therefore, the latency-oriented communication time is 57.5 msec for 1,024 GPUs, while the bandwidth-oriented one is only 6.7 msec. In total, GPU calculation time is still dominant, which is 471 msec. However for 4,096 GPUs, GPU calculation time would be reduced to 118 msec, while latency-oriented communication time be increased to 230 msec. The bottleneck would be caused by the latency-oriented communication for this case. To avoid the bottleneck, one idea is to use multi thread programming like OpenMP on the client side.

Overhead to support fault tolerance 6

Users have only to set environment variables to activate the fault tolerant function, but the overhead in calculation time exists. Backing-up GPU memory to the client side every few steps, known as checkpointing, causes performance degregation, especially when the frequency of checkpoinging is not optimal.

Fig. 6 shows the relative calculation time when the period of checkpointing was changed [12]. The same application in the previous section was used for this experiment. Over-clocked GPUs were used to artificially generate calculation errors, and we chose data when the period is close to 900 seconds per one error, which is shown as crosses in Fig. 6. A special DS-CUDA server, which is designed to generate an error every 900 seconds, is also used, and its data was plotted as circles in Fig. 6.

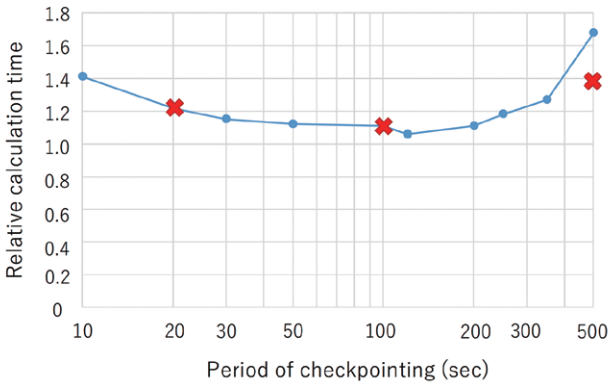


Fig. 6 Overhead of calculation with fault tolerant function

On the one hand, the calculation time becomes larger when the period of checkpointing is small, since the operation of backing-up GPU memory takes more time. On the other hand, the calculation time also becomes larger when the period of checkpointing is large, since the re-execution of previous CUDA APIs takes time as the possibility of errors occurring becomes larger. In this configuration, the optimal checkpointing interval was 100 seconds, and the overhead was roughly 10% . Non-over-clocked GPUs are more reliable [10], and the overhead would be much less than this result.

Combining tablet and GPU notebook 7

Fig. 7 shows a block diagram of a DS-CUDA system, which composed of a non-GPU tablet on the client side, a GPU notebook on the cloud side, and a WiFi router between them. Table 2 shows the energy efficiency (Gflops/Watt) and rendering speed (Frame/sec) of an MD simulation and visualization using 5,832 particles [13, 14]. Power consumption of both of the tablet and the notebook are summed for the last row.

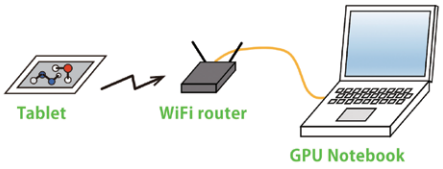


Fig. 7 Using a tablet and notebook with DS-CUDA

System	Calculation speed (Gflops)	Power Consumption (Watt)	Energy efficiency (Gflops/Watt)	Rendering speed (Frame/sec)
GPU Notebook	1,655	129	12.8	60.2
Tablet	4.4	17.5	0.25	0.16
Tablet + GPU Notebook	701	78	9.0	19.6

Table 2 Calculation speed and energy efficiency

The DS-CUDA system achieves 9.0 Gflops/Watt, which is good enough compared with Green500 [15], though the number is a little lower than that of a Notebook of 12.8 Gflops/Watt. Note that it should not be compared with Green500 numbers precisely, since it was calculated by a single-precision flops count and complicated function and division is counted as several operations.

Though the tablet itself is slow both in calculation and rendering, combining a tablet and a notebook with DS-CUDA achieved several tens of acceleration in calculation speed, energy efficiency, and rendering speed. Especially, rendering speed of 20 Frame/sec is smooth enough for interactive simulation.

Although not shown in this table, a SHIELD tablet, which supports native CUDA, was found to achieve lower numbers in calculation speed, energy efficiency, and rendering speed compared with the DS-CUDA system.

Concluding remarks 8

The effectiveness of the idea of DS-CUDA to use multiple GPUs was discussed as a cloud and visualization tool.

DS-CUDA enables the use of many GPUs via a network from a usual client computer. Acceleration of calculation and visualization can be achieved while utilizing touch and accelerometer sensors on a tablet client device. Moreover, a fault tolerant capability can be easily added.

Although mobile GPUs might have better energy performance and high-end GPUs might have better absolute performance, combining a client with cloud computers can achieve balanced performance in calculation speed, rendering speed, and power efficiency.

Recent advancement of tablets makes them an ideal tool for interactive simulation and visualization. Cloud tools like DS-CUDA would be an interesting approach to realize such requirements.

Acknowledgements

This research was supported in part by the Japan Science and Technology Agency's Core Research of Evolutional Science and Technology research program:"Highly Productive, High Performance Application Frameworks for Post Petascale Computing." TSUBAME 2.5 supercomputer in Tokyo Institute of Technology was used in part for this research. Author thanks Atsushi Kawai in K&F Computing Research, Minoru Oikawa in Chiba University, Edgar Josafat Martinez-Noriega in University of Electro-Communications, and Kentaro Nomura and Kenji Yasuoka in Keio University for the development and evaluation of DS-CUDA.

References

[1] Amazon EC2: https://aws.amazon.com/jp/ec2/instance-types/p2/ (Accessed in Jan. 2017)

[2] Nvidia GRID: http://www.nvidia.com/object/grid-technology.html (Accessed in Jan. 2017)

[3] N. Luehr, A. G. Jin, and T. J. Mart´ınez, "Ab Initio Interactive Molecular Dynamics on Graphical Processing Units (GPUs)," Journal of chemical theory and computation, Vol. 11, No. 10, pp. 4536–4544, (2015)

[4] A. Kawai, K. Yasuoka, K. Yoshikawa, and T. Narumi,

"Distributed-Shared CUDA: Virtualization of large-scale GPU systems for programmability and reliability," Future Computing 2012, Nice, France, (2012)

[5] DS-CUDA: http://narumi.cs.uec.ac.jp/dscuda/ (Accessed in Jan. 2017)

[6] S. Shi, C.H. Hsu, "A Survey of Interactive Remote Rendering Systems," ACM Computing Surveys (CSUR), Vol. 47, No. 4, Article 57, pp. 1-29, (2015)

[7] J. Duato, A. J. Pena, F. Silla, R. Mayo, and E. S. Quintana-Orti, "Performance of CUDA virtualized remote GPUs in high performance clusters," International Conference on Parallel Processing, IEEE, pp. 365–374, (2011)

[8] T. Shimada, "PluginGPU Box : Virtual CUDA environment for Windows PC," Graduation thesis in Department of Communication Engineering and Informatics, University of Electro-Communications, (2014)

[9] E. J. Martinez-Noriega, "Running CUDA on Android Through GPU Virtualization," GPU Technology Conference, San Jose, P4160, (2014)

[10] I. S. Haque, V. S. Pande, "Hard Data on Soft Errors: A Large-Scale Assessment of Real-World Error Rate in GPGPU," 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, pp.691-696, (2010)

[11] M. Oikawa, K. Nomura, K. Yasuoka, T. Narumi, " Parallel Molecular Dynamics Simulation with Replica-exchange Method Using 1,024 GPUs," IPSJ Transactions on Advanced Computing Systems, Vol. 7, No. 4, pp. 1-14, (2014)

[12] T. Narumi, M. Oikawa, E. J. Martinez-Noriega, K. Yasuoka, "DS-CUDA: GPU Virtualization Middleware to Support Migration Functionality," IPSJ SIG Technical Report, 2016-HPC-153(17), pp.1-6, (2016)

[13] E. J. Martinez-Noriega, "High Performance Computing on Mobile Devices through Distributed Shared CUDA," GPU Technology Conference, San Jose, S5290, (2015)

[14] E. Martinez-Noriega, S. Yazaki, T. Narumi, "Performance Evaluation of Remote CUDA Offloading on Mobile Devices for Energy Efficient Systems," submitted

[15] Green500 List: https://www.top500.org/green500/ (Accessed in Jan. 2017)

TSUBAME e-Science Journal vol.15

Published 3/8/2017 by GSIC, Tokyo Institute of Technology ©
ISSN 2185-6028
Design & Layout: Kick and Punch
Editor: TSUBAME e-Science Journal - Editorial room
Takayuki AOKI, Toshio WATANABE,
Atsushi SASAKI, Yuki ITAKURA
Address: 2-12-1-E2-6 O-okayama, Meguro-ku, Tokyo 152-8550
Tel: +81-3-5734-2085 Fax: +81-3-5734-3198
E-mail: tsubame_j@sim.gsic.titech.ac.jp
URL: http://www.gsic.titech.ac.jp/

TSUBAME

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