Multi-GPU Implementation of the NICAM Atmospheric Model

Irina Demeshko[1], Naoya Maruyama[2], Hirofumi Tomita[2], and Satoshi Matsuoka[1]

[1] Tokyo Institute of Technology, Tokyo, Japan
[2] RIKEN Advanced Institute for Computational Science, Kobe, Japan

Abstract. Climate simulation models are used for a variety of scientific problems and accuracy of the climate prognoses is mostly limited by the resolution of the models. Finer resolution results in more accurate prognoses but, at the same time, significantly increases computational complexity. This explains the increasing interest to the High Performance Computing (HPC), and GPU computations in particular, for the climate simulations. We present an efficient implementation of the Nonhydrostatic ICosahedral Atmospheric Model (NICAM) on the multi-GPU environment. We have obtained performance results for the number of GPUs up to 320. These results were compared with the parallel CPU version and demonstrate that our GPU implementation gives 3 times higher performance over parallel CPU version. We have also developed and validated the performance model for a full-GPU implementation of the NICAM. Results show 4.5x potential acceleration over parallel CPU version. We believe that our results are general, in that in similar applications we could achieve similar speedups, and have the ability to predict its degree over CPUs.

Keywords: GPU computations, CUDA Fortran, climate simulations, nonhydrostatic model.

1 Introduction

Climate change has significant impact on the Earth and human life as well as on the world's economic and geopolitical landscapes. The consequences of climate change such as higher temperatures, changing landscapes and different weather cataclysms potentially affect everyone. Thus, it is important to improve the understanding of the changing climate system and enable scientists to predict future climate behavior. Complex climate simulation modeling requires state-of-the-art HPC systems in order to get the "realistic" results. Particularly heterogeneous systems, which are based on using both conventional microprocessors and graphic processor units (GPUs), give us an extra performance for many kinds of scientific computations.

The main objective of this work is to increase performance of the existing climate simulation by using large-scale parallelism available on multi-GPU

I. Caragiannis et al. (Eds.): Euro-Par 2012 Workshops, LNCS 7640, pp. 175–184, 2013.

environments. In this work we have developed and optimized a multi-GPU implementation of the ultra-high resolution atmospheric global circulation model NICAM [1] (see sections 3, 4). We have also built a performance model, which allows us to estimate potential performance of full-GPU implementation for any problem configuration (see section 5).

Our method is based on the localization of the most computationally intensive part of the climate model and modifying it to be computed on GPUs. One of the main challenges which such method faces is to reduce communication overheads and, at the same time, use large-scale parallelism of the hybrid system in the most efficient way. In order to achieve an optimal performance GPU kernels were analyzed and optimized. Experiments demonstrate that we have achieved significant acceleration of the initial parallel code. We have compared maximum performance for the initial MPI-parallel code with the one for our GPU implementation. The evaluation results show that the performance of the ported module is close to maximum efficiency and we can see more then 3x speedup in case of porting only one module to GPUs. Estimation results for a full-GPU implementation of shallow water NICAM simulation show potential 4.5x speedup over CPU MPI-parallel version.

Our contributions can be summarized as follows:

- We propose an effective multi-GPU implementation for a high resolution nonhydrostatic atmospheric model, specifically optimized for the large hybrid systems.
- We have evaluated our approach experimentally on more than hundred of nodes with 320 GPUs on a TSUBAME2.0 supercomputer. Evaluation results show performance's raise and significant acceleration of the original MPI-parallel code.
- We have developed performance model for a full-GPU NICAM implementation, which produces an estimation of potential performance for any problem configuration. It was shown that performance model prediction matches to the observed results for a large scale NICAM code.

2 Related Work

Increasing interest to the HPC among developer of climate simulation software can be explained by the extreme necessity to increase computational performance in purpose to improve the accuracy of simulations.

There are several weather/climate simulation models, which were recently accelerated with GPUs, such as: WRF[2], COSMO[3], NIM[4], HILRAM[5], ASUCA [6], GEOS-5[7] and GRAPES[8]. ASUCA, GEOS-5 and GRAPES are full-GPU approaches, WRF and COSMO(1) are based on porting only "Physics" module to the GPU, and COSMO(2), NIM and HILARM are poring to GPU only "Dynamics" module. Performance results are different for all listed simulation models and depends on numerical scheme as well as on the GPU-implementation strategy. Full-GPU implementation of ASUCA shows 26.3x speedup for a double precision and 80x speedup for a single precision cases[6]. By using GPU

co-processor technology GEOS-5 have demonstrated a potential speedup of 15-40 times over conventional processor cores[7]. Recent results of porting the GRAPES model to the GPU system show that acceleration was not as efficient as it was supposed and implementation algorithm needs significant optimizations. For the models, based on porting only one module (Physics or Dynamics) to the GPU we also observe quite sufficient acceleration, but the performance results are lower then for a full-GPU ones. It can be explained by higher communication overheads and smaller commutation intensity on the GPU. WRF GPU implementation shows 17x speedup over 1 CPU[2]; COSMO simulation on 1 GPU is 2x faster then on 6 CPUs; HILARM single GPU implementation shows 10x speedup over 1 CPU[5].

Our multi-GPU implementation of the NICAM is based on porting shallow water computations module (2-dimension "Dynamics" module) to GPUs and shows potential speedup of 4.5 times over CPU MPI-parallel implementation.

In our previous work [9] we have implemented the module, which performs the NICAM 's main shallow water computations, on a single GPU. Due to the relatively small GPU memory, the results, we obtained, were limited only to small-scale problems (grid level up to 8). Multi-GPU implementation, described in this paper, solves the memory problem by distributing grid elements between nodes. We have modified our single GPU implementation into multi-GPU one. For this purpose we had to study the way to organize MPI-CUDA coordination efficiently.

3 The NICAM Model

NICAM is a Nonhydrostatic ICosahedral Atmospheric Model, which is designed to perform "cloud resolving simulations" by directly calculating deep convection and meso-scale circulations[1], [10]. Detailed description of the NICAM numerical scheme can be found in [1], [11], [12]. For the horizontal discretization NICAM model uses icosahedral grid system on the sphere. Grid dimension is defined by the grid division level n (gl n), and total number of grid point can be calculated by the formula: $Ng = 10(2^n)^2 + 2$

Initial NICAM code is based on FLAT-MPI parallel programming model. NICAM uses 2D decomposition, where initial grid is divided by regions. This regions are distributing between MPI processes. Total number of rectangles for the region level n can be calculated by the formula:

$$Nr = 10(4^n) \tag{1}$$

In this work we have investigated 2-dimensional (Shallow water) case of NICAM model [13], which plan to implement for the 3- dimensional case in a future work.

4 Implementation in CUDA Fortran

The CUDA architecture [14] enables hybrid computing, where both host (CPU) and device (GPU) can be used to accelerate different types of computations.

Current CUDA-enabled GPUs can contain from tens to hundreds of processor cores which are capable of running tens of thousands of threads concurrently. This gives us an opportunity to get significant acceleration of the code, but, at the same time, data transfers between host and device through the PCI bus tend to be the bottleneck in data movement. Therefore it is feasible to use GPUs only for computationally intensive code. The Portland Group (PGI) offers us an opportunity to explicitly program for GPUs using CUDA Fortran. NICAM is originally written in Fortran, thus the PGI CUDA Fortran syntax allows us to develop GPU kernels using a familiar coding environment.

Simplified flow-graph of the GPU NICAM implementation is presented on the Figure 1. It is shown that in the beginning of the computations we execute some initial modules like: MPI initialization, reading from the input file, checking the system, calculation of the grid geometry and some others. Main computations of the model are performing in the cycle by the time steps (nl).

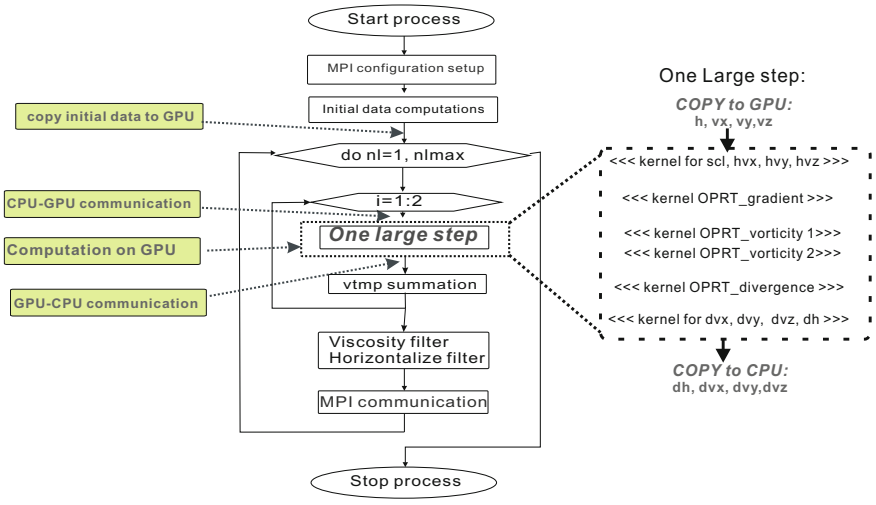

Fig. 1. GPU implementation scheme

In purpose to analyze runtime behavior of the given code we have used the Scalasca performance toolset [15]. According to the profiling results "One large step" is the most time consuming module of the code. It takes more then 50% of the whole time to compute this module. Therefore, in purpose to accelerate computations, we have decided to port this module to GPUs. We describe our GPU implementation algorithm below.

Before starting main cycle computations we send initial data arrays to GPUs (see Figure 1). This data are constant during entire computational process and we keep them in the GPU global memory until the end of the computations. The data, which are variable and necessary for the "One large step" module computations, are copying to GPUs in the beginning of the "One large step" module.

Then, after we finish computations on GPUs, we copy output data arrays to CPU. "One large step" module consist of 3 main subroutines: OPRT_gradient, OPRT_vorticity and OPRT_divergence. In purpose to reduce communication overheads we have also ported initial data and output data computation modules to GPUs and thus, keep all temporary data in the GPU global memory. Initial CPU code for this module was slightly modified in purpose to reduce the amount of memory to be allocated on GPU. We have created one kernel for OPRT_gradient and one for OPRT_divergence. For OPRT_vorticity module it was necessary to create 2 kernels, because we need to synchronize data within GPU in the middle of the vorticity module computations. Each modified module based on one nested loop or several loops calculation and each loop iteration computes 1 element of 2-dimensional array. Each thread of our GPU kernels calculates 1 element of the array.

CUDA programming model requires the programmer to organize parallel kernels into a grid blocks, which divided into thread blocks with at most 512 threads each. The NVIDIA GPU architecture executes threads of a block in SIMT (single instruction, multiple thread) groups of 32 called warps. NICAM model uses 2-dimensional grid, which size depends on the of grid level and region level sizes. We have used a block configuration of 256 threads, one thread per element. Our block size is a multiple of 32 which fits with the warp size and, therefore, allows us to achieve maximum efficiency. We have performed several tests to find the best way of organizing GPU-CPU communications and found that using pinned memory with PGI CUDA Fortran assignment gives as the best results and reduces communication time for about 3x time over a non-optimized version. By this means CPU-GPU communication bandwidth reached 5.6 GB/sec, which is close to the maximum. GPU kernels are located inside of each MPI process (see Figure 1) Therefore in the evaluation experiments we used the same number of MPI processes and GPUs.

It is shown in section 6 that CPU-GPU communications are quite significant and became a bottleneck. In purpose to increase performance we need to reduce communication time to computation time ratio. For this reason we plan to increase computational intensity on GPUs by porting the rest modules of the main Shallow Water computations to graphic processor units. In purpose to estimate potential performance for a full-GPU implementation of the NICAM we have built a performance model, described in the next section.

5 Performance Model

Simplifyed flow-graph for the full-GPU implementation of the NICAM is shown on the Figure 2 a). We assume that we compute "One large step", "Viscosity filter" and "Horizontalize filter" modules on GPUs. For this reason, we send h, vx, vy, vz arrays to GPU Global memory before starting GPU computations and copy output arrays dh, dvx, dvy, dvz to CPU after finishing "Horizontalize filter" computations. The size of arrays we copy to GPU and from GPU to CPU are the same as it was for a case of porting only "One Large step" module, and, therefore, the communication time ($COMM\_time$) should be the same

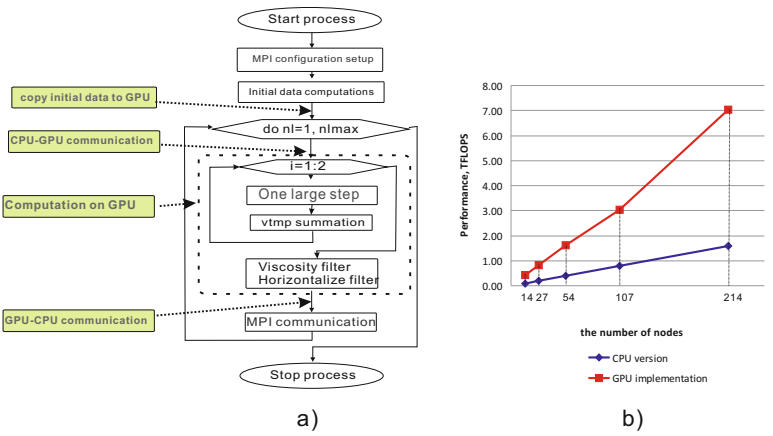

Fig. 2. Performance model a)flow-graph for a full-GPU NICAM implementation b) performance estimation results

The results we have got for the described multi-GPU implementation show that the "One large step" - is a memory bound module. Experimental results show, that its performance on GPU without communications tends to 14 TFLOPS (we will call it $OLSPerfomanse$ below). Also it was estimated that the average CPU-GPU communication bandwidth ($COMM\_Bandwidth$) is about 5.6 GB/sec . The estimation of the potential performance of the full-GPU implementation is based on the number of floating point operations for entire shallow mode computation cycle ($FLOP\_N$), obtained $OLSPerformanse$ and $COMM\_Bandwidth$, and was calculated by the formula (2):

$$Performance = FLOP\_N/(COMM\_time + time\_on\_GPU), \tag{2}$$

where $COMM\_time = transfered\_memory/COMM\_Bandwidth$,
$transfered\_memory = 64 * ((2 + (grid\_level - region_level)^2)^2), Bytes$,
$time\_on\_GPU = FLOP\_N/OLSPerforman?e$

Size of the transferred memory depends on the grid dimension and, therefore, a function of the grid level and region level parameters. The formulas above estimate performance for any configuration of the problem size. Results are presented on the Figure 2 b). It is shown that the performance of the multi-GPU implementation potentially achieve 7 TFLOPS on 217 nodes, which is 4.5x higher than performance for the parallel CPU implementation. The performance model results were compared to the experimental ones (see section 6). We have observed that the model predictions match to the experimental results, which indicates that proposed model is valid and efficient.

6 Experimental Evaluation

In this work we have presented results of multi-GPU implementation of NICAM code on TSUBAME 2.0 supercomputer.

6.1 Environment

TSUBAME 2.0 consist of 1408 compute nodes of two Intel Xeon Westmere-EP 2.9 GHz CPUs and three NVIDIA M2050 GPUs with 52GB and 3GB of system and GPU memory, running SUSE Linux Enterprise Server 11 SP1. Each node has 2 sockets, 12 cores/node. We have used PGI CUDA Fortran compiler for the GPU code and PGI mpich2 compiler for the code on CPUs.

6.2 Results of the Multi-GPU Implementation

We have compared maximum performance, which is possible to get from each node, for the original MPI-parallel implementation of the code with the one for our multi-GPU implementation in our experiments. We have presented results for the grid levels 9, 10 and 11, which correspond to 2621440, 10485762 and 41943042 number of elements accordingly.

Due to the fact that we have 2 sockets per node, 6 cores per each socket and 3 GPUs per node, we have used 12 CPU cores per node for the CPU implementation and 3 CPU+GPU hybrid cores per node for the GPU version of the code. Initial NICAM code has a limitation in the number of MPI processes, which should be a division of the number of regions. The number of regions can be calculated by the formula (1) and only next numbers of processes are available: 1, 4, 5, 8, 10, 16, 20, 32, 40, 64, 80 and so on. According to the statements above we have used following configurations: we have compared 14 nodes with 160 CPU cores for the CPU implementation versus 40 CPU+GPU hybrid cores for the multi-GPU one; 27 nodes with 320 CPU cores versus 80 CPU+GPU hybrid cores; 54 nodes with 640 CPU cores versus 160 CPU+GPU hybrid cores, and 107 nodes with 1280 CPU cores versus 320 CPU+GPU hybrid cores. The results are presented on Figures 3, 4.

We present performance breakdown results for a GPU implementation in the Figure 3 a). It is shown that about 30-40% of the overall time of the GPU implementation is spent for CPU-GPU communications. This ratio is quite significant, which signify that communication overheads are the bottleneck for the described implementation. It was shown in the previous section that we achieve higher performance by increasing computational intensity on GPU. Therefore, we suppose that we will get much better results when we switch from the 2D case to the 3D one, because computation intensity on GPU for a 3D case is significantly higher than for a 2D implementation. We compare overall computation time for GPU implementation with the one for initial MPI-parallel version on the Figure 3 b) It is shown that our multi-GPU implementation is up to 3 times faster then the initial parallel CPU code. At the same time, we can observe that for smaller grid sizes (e.g. gl 09) acceleration is slightly lower and we have smaller execution time

difference between parallel CPU and multi-GPU implementations, which can be explained by lower computation intensity per node. Thus, the GPU implementation is more efficient for grid levels higher than 9, which is the lowest grid level required by the NICAM model to produce relatively "realistic" results.

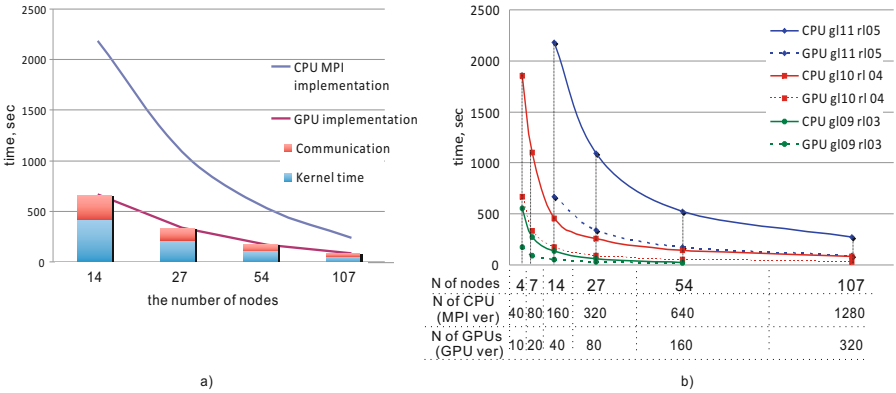

Fig. 3. Average running time for MPI-parallel CPU version versus time multi-GPU implementation of NICAM as a function of the number of nodes

Performance as a function of the number of nodes for a multi-GPU implementation and initial NICAM version is shown on the figure Figure 4a). The results are similar to the ones on the Figure 3 b) and performance for a GPU implementation is up to 3 times higher than for a MPI-parallel version.

Figure 4 b) compares performance we obtained for the proposed GPU implementation with the theoretical one, estimated by the performance model described at the section 5. According to the formulas from the section 5 and due

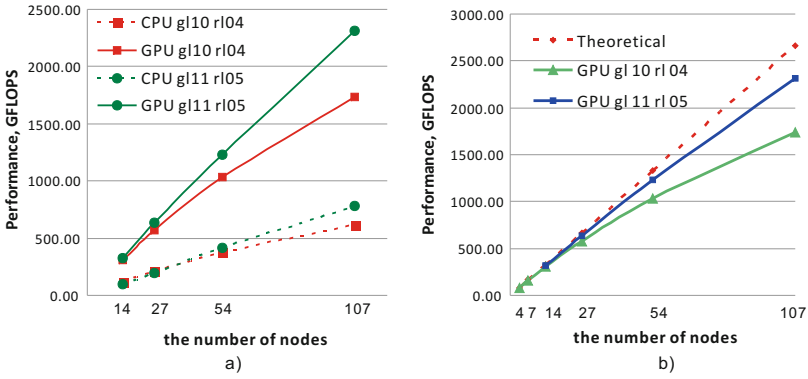

Fig. 4. Performance as a function of the number of nodes. a) Parallel MPI version versus multi-GPU implementation. b) Theoretical and experimental performances

to the fact, that increasing the grid level by 1 we 4x time increase computational intensity and transferred data size, theoretical performance is the same for any configuration of the problem. From the graph we can see, that the higher grid dimension the closer experimental performance to the theoretical one. This indicates that our theoretical model better fit to the real-size problem, due to we have higher computational intensity for a fine-grained tasks and less overheads. It can be observed that for the grid level 11 up to 54 nodes the experimental performance close to the theoretical one, which validates the proposed performance model. A little difference of the experimental performance for the 107 nodes with the theoretical one can be explained by the reduced computation intensity per node, which reduces efficiency of the GPU implementation. We have also performed some additional experiments to compare our performance model estimations with the experimental results. It was observed, that the model is well matching tho the experiments for such a large-scale model as NICAM. So, we believe that we can use the model for the purpose to estimate reliable performance results for a reasonable sized problem configuration.

7 Conclusion

Climate changes becoming a critical topic in everyday life and, therefore, an interest to climate simulation models is rapidly increasing. The climate models are tools that demand high performance computations to achieve reliable accuracy of the results. Heterogeneous systems are emerging as attractive computing platforms for HPC applications. Large-scale parallelism available on multi-node GPU environment gives us an opportunity to get better performance for many kinds of scientific computations.

In this paper we have presented results of the multi-GPU implementation of the Nonhydrostatic ICosahedral Atmospheric Model on a big heterogeneous system. We have ported the most time-consuming module of the initial code to GPUs by using PGI CUDA Fortran. We have demonstrated that our multi-GPU implementation gives 3 times higher performance comparing with the maximum performance of the MPI-parallel version. It was also shown, that performance of the proposed GPU implementation is almost optimal for "reasonable-sized" problems. We have developed a performance model for a full-GPU implementation. Estimation results show 7 TFLOPS potential performance on 217 nodes for a multi-GPU implementation, which is 4.5x times higher than the performance of the parallel CPU version. The model was validated and shows a good matching to the experimental results. The results above prove that proposed method is highly efficient and we can recommend it for many kinds of climate simulations.

We intend to apply described GPU implementation of NICAM shallow water code for a tree-dimensional NICAM climate simulation model, which would give us higher performance results due to increasing computational intensity per node. Also we plan to investigate performance of the NICAM code with OpenACC and compare it with the one, we obtained with PGI CUDA Fortran.

References

1. Satoh, M., Matsuno, T., Tomita, H., Miura, H., Nasuno, T., Iga, S.: Nonhydrostatic Icosahedral Atmospheric Model (NICAM) for global cloud resolving simulations. Journal of Computational Physics, The Special Issue on Predicting Weather, Climate and Extreme events 227, 3486–3514 (2007)
2. Michalakes, J., Vachharajani: GPU acceleration of numerical weather prediction. In: IPDPS, pp. 1–7. IEEE (2008)
3. COSMO-model (2012), http://www.cosmo-model.org/
4. Govett, M.W., Middlecoff, J., Henderson, T.: Running the NIM next-generation weather model on GPUs. In: Proc. 10th IEEE/ACM Int. Conf. Cluster, Cloud and Grid Computing, CCGrid, pp. 792–796 (2010)
5. Vu, V.T., Cats, G., Wolters, L.: GPU acceleration of the dynamics routine in the HIRLAM weather forecast model. In: HPCS 2010, pp. 31–38. IEEE (2010)
6. Shimokawabe, T., Aoki, T., Muroi, C., Ishida, J., Kawano, K., Endo, T., Nukada, A., Maruyama, N., Matsuoka, S.: An 80-Fold Speedup, 15.0 TFlops, Full GPU Acceleration of Non-Hydrostatic Weather Model ASUCA Production Code. In: The 2010 ACM/IEEE Conference on Supercomputing, SC 2010, pp. 1–11 (2010)
7. Putnam, W: Graphics Processing Unit (GPU) Acceleration of the Goddard Earth Observing System Atmospheric Model. NASA Technical Report, Goddard Space Flight Center (2011), http://ntrs.nasa.gov/search.jsp?R=20120009084/
8. Wang, Z., Xu, X., Xiong, N., Yang, L.T., Zhao, W.: GPU Acceleration for GRAPES Meteorological Model. In: 2011 IEEE 13th International Conference on High Performance Computing and Communications, HPCC, pp. 365–372 (2011)
9. Demeshko, I., Matsuoka, S., Maruyama, N., Tomita, H.: Ultra-high resolution atmospheric global circulation model NICAM on graphics processing unit. In: Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications (2012)
10. Tomita, H., Satoh, M.: A new dynamical framework of nonhydrostatic global model using the icosahedral grid. Fluid Dynamics Research 34, 357–400 (2004)
11. Tomita, H., Tsugawa, M., Satoh, M., Goto, K.: Shallow water model on a modified icosahedral geodesic grid by using spring dynamics. J. Comp. Phys. 174, 579–613 (2001)
12. Satoh, M., Tomita, H., Miura, H., Iga, S., Nasuno, T.: Development of a global cloud resolving model – a multi-scale structure of tropical convections. J. Earth Simulator 3, 11–19 (2005)
13. Toro, E.F.: Shock-Capturing Methods for Free-Surface Shallow Flows, 309 pages. Wiley and Sons Ltd. (2001)
14. NVIDIA Corporation. NVIDIA CUDA Programming Guide, version 4.1 (2012), http://www.nvidia.com/cuda
15. The Scalasca performance toolset (2012), http://www.scalasca.org/