Ultra-high resolution atmospheric global circulation model NICAM on graphics processing unit

I. Demeshko[1], **S. Matsuoka**[2], **N. Maruyama** [3], **and H. Tomita** [3]
[1]Dep. of Mathematical and Computing Science, Tokyo Institute of Technology, Tokyo, Japan
[2]Global Scientific Information and Computing Center, Tokyo Institute of Technology, Tokyo, Japan
[3]RIKEN Advanced Institute for Computational Science, Kobe, Japan

.

Abstract— *Climate simulations have significant role in analyzing changes that have occurred in the Earth, give us better understanding of the recently happened processes. Most of the existing models perform approximate results and fundamental improvements of the software models are necessary to increase accuracy. We present an efficient implementation of ultra-high resolution atmospheric global circulation model on graphics processing units(GPUs). The model based on the nonhydrostatic system with the icosahedral grid and called "NICAM". We ported computationally intensive part of the NICAM code to GPU by using CUDA FORTRAN, then validated and compared its GPU performance to that for the parallel CPU version of the code. This approach shows a good performance results together with reducing memory consumption compared to a fully GPU approaches. Our results show 5x speedup for a computationally intensive part of shallow water simulation model on a single GPU in comparison with a parallel CPU implementation (5 cores).*

Keywords: GPU computations, CUDA Fortran, climate simulations, nonhydrostatic system

1. Introduction

One of the main goals of climate simulations is a prediction of future climate changes and their impact on the Earth and society. It is essential to get a reliable results from the simulations to plan our future ecological, financial and human strategy. For this purpose we need significantly increase scales of the current climate simulations. Thus, it is important to adapt current climate applications to take advantage of the performance capabilities of novel hybrid architectures.

Performance rise of the climate and weather simulation software was based on increasing processor speed rather than increasing parallelism for last years[1]. Nowadays, to sustain such an increase towards the exascale era, we need some significant changes in the software models. GPUs are a very high performance alternative to conventional microprocessor. The massive parallelism of GPUs offers tremendous performance in many high-performance computing applications. GPUs were designed to exploit fine-grained parallelism, which gives us an ability to create weather and climate models with much finer parallelism.

In order to increase performance of the existing climate simulation, we ported and optimized a high resolution climate model to GPU. We aimed to get significant acceleration from applying heterogeneous computing with both conventional CPUs and vector-oriented GPU accelerators.

In this work we use single GPU to run a new type of ultra-high resolution atmospheric global circulation model NICAM (Nonhydrostatic ICosahedral Atmospheric Model) [2] being developed at Advanced Institute for Computational Science, RIKEN (see section II). Initial NICAM code uses 2-dimensional MPI-parallelization and shows good scalability results.

We propose to localize and port the most computation-intensive part of the climate model to GPU. For this purpose, we first study the original NICAM code and isolate the most time-consuming modules. Then, we ported those modules to GPU and evaluated the performance of our implementation.

The main difference of our strategy from some of existing GPU-based approaches [3], [4] is that we do not port to the GPU entire climate model application. Our method allows to reduce the memory to be allocated on GPU by performing on CPU some low-cost computations and porting to the GPU only the most time-consuming computations.

The results of our evaluation show that we reach an almost optimal performance for most of the ported kernels and we see an important speedup. We have got 5x speedup for a computationally intensive part of shallow water simulation model on a single GPU, in comparison with a parallel CPU implementation. Our GPU kernels give us performance, close to the maximum one, which indicate the efficient hardware utilization.

This paper is organized as follows. We describe some important background in the "NICAMmodel" section. Section III outlines our GPU NICAM implementation approach, presents step-by-step algorithm of the "mapping to GPU" process. In section IV we give some information about environment we used, describe performance results for both CPU and GPU versions of the code and present GPU performance model. We give some conclusions and outline our plans for future work in section V.

2. NICAM model

NICAM is a Nonhydrostatic ICosahedral Atmospheric Model, used as a Global Cloud Resolving Model (GCRM). It was designed to perform "cloud resolving simulations" by directly calculating deep convection and meso-scale

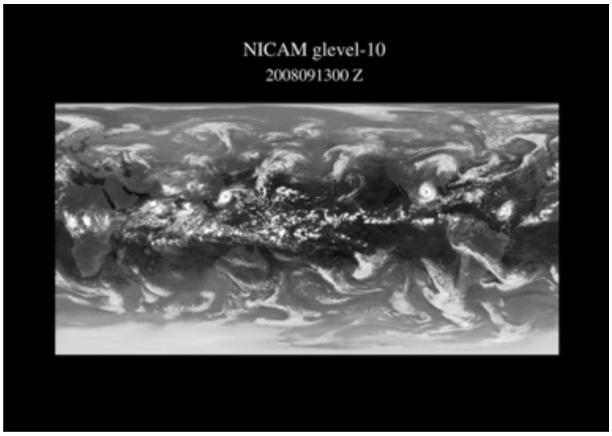

Fig. 1: Tropical cyclones SINLAKU and IKE reproduced by NICAM 7km simulation [6] (00UTC 13 Sep. 2008) (by H.L. Tanaka).

circulations, which play key roles not only in the tropical circulations but in the global circulations of the atmosphere[2]. NICAM - is a unified model in the sense that it can be used for both short term numerical predictions for weather systems such as a week, and long term simulations to obtain quasi-equilibrium climate states (see Figure 1).

The model uses fully compressible (elastic) nonhydrostatic system to obtain thermodynamically quasi-equilibrium states by long time simulations. For this purpose a nonhydrostatic numerical scheme which guarantees conservation of mass and energy was devised and implemented to the global model using the icosahedral grid configuration. The formulation and numerical scheme of NICAM along with numerical results of some test cases are thoroughly presented in [5].

The finite volume method is used for numerical discretization, so that total mass and energy over the domain is conserved; thus this model is suitable for long term climate simulation.

2.1 Numerical methods

For the horizontal discretization, icosahedral grid system on the sphere is used. Figure 2 shows an example of series of consecutive icosahedral grids.

The icosahedral grids are constructed by a recursive division of geodesic arches on the sphere. Starting from the original icosahedron, one-level finer grids are generated by bisecting the geodesic arches of the former coarser grids. We call the n-th bisection of the icosahedron glevel n (glevel: grid division level). The average grid interval of glevel 11 is about 3.5 km, for example. The total number of grid points is Ng = $10(2^n)^2 + 2$.

Numerical models with this grid system are first investigated by Sadourny et al. [7] and Williamson [8], and are recently revisited as a candidate for next-generation high-resolution global models.

In the [9] Tomita et al. describe the modifications, which was applied to the original icosahedral grids by using the

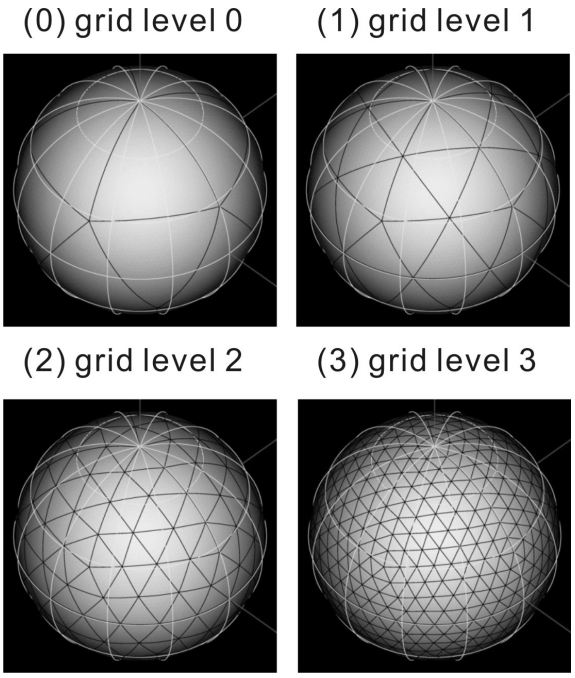

Fig. 2: Horizontal discretization scheme of NICAM model, based on icosahedral grid system on the sphere. Grid level 0 - original icosahedron, 2 grid points. The number of grid points for grid level n can be calculated by $Ng = 10(2^n)^2 + 2$

spring dynamics. With this modifications, the fractal structure of the original icosahedral grid is relaxed and more uniform grid structure with a smaller ratio of minimum to maximum grid intervals is obtained by smoothing the grid arrangement. It was found that the numerical errors can be reduced using the modified grid[10].

The governing equations of the global model are a newly developed nonhydrostatic schemes that guarantees conservation of mass and energy. The finite volume method is used for the flux form equations. The Arakawa-A type grid is used where all the variables are allocated at the vertices of triangles. The shape of the control volume is either hexagon or pentagon.

Further details about numerical scheme, used in NICAM are described in [2], [9], [10].

2.2 MPI parallelization

In this paper we present our single GPU implementation of NICAM model, but we plan to use current NICAM MPI-parallelization method to create a multi-GPU implementation for a following work.

Initial NICAM code use 2-d domain decomposition with FLAT-MPI parallel programming model (see Figure 3, Figure 4).

MPI parallelization strategy is based on discretization grid by regions, which then managed by the different MPI processes.

Region discretization algorithm is shown at the Figure 3. First, we create region level 0 by connecting two neighboring icosahedral triangles. In this case we have

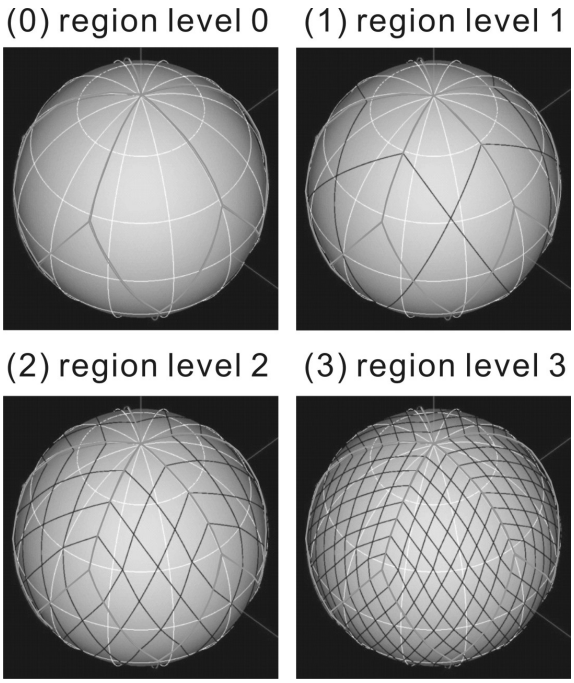

Fig. 3: NICAM region decomposition, used for FLAT MPI parallelization scheme. Region level 0 has 10 rectangles. The number of rectangles for the region level n can be calculated by $Nr = 10(4^n)$

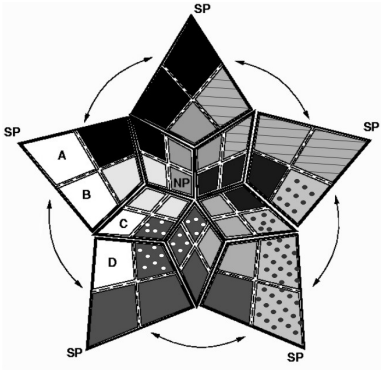

Fig. 4: MPI distribution strategy: region level 1, 40 regions and 10 MPI processes. Each process manage 4 regions with the same color

only 10 rectangles for the level 0 region. To increase region level to 1, we divide each of rectangles into 4 sub-rectangles by connecting the diagonal mid-points (revel-1). Continuing this process recursively we can get region level-n.

MPI distribution strategy is presented on Figure 4. One process manages rectangle regions with the same color. Figure 4 shows an example case for the region level 1, 40 regions and 10 MPI processes. Each process manages 4 regions with the same color.

Assuming one process manages one rectangle region, increasing r-level computational intensity on 1 process is reducing.

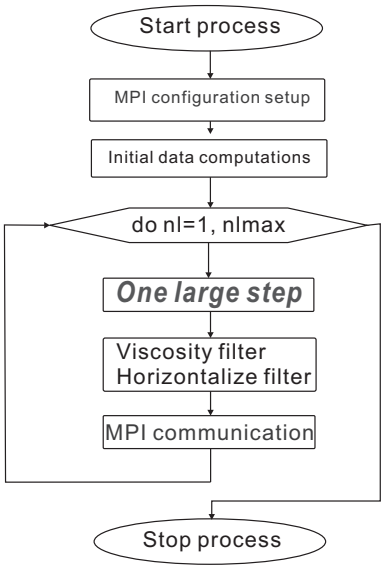

Fig. 5: Simplified flow-graph of the original NICAM Shallow Water model code.

3. GPU implementation

Our GPU implementation approach is based on porting the most computationally intensive part of the initial NICAM code to GPU. In this work we investigate 2-dimensional (Shallow water) case of NICAM model [11], which plan to implement for the 3- dimensional case in future work.

First, we selected a computationally intensive module of NICAM, then modified this Fortran model to run on NVIDIA GPU using PGI CUDA Fortran[12], [13], validated and compared its GPU performance to that of the original MPI parallel module.

In purpose to analyze runtime behavior of the given code we used the Scalasca performance toolset [14]. We have got profiling results for different configurations of the grid and region level numbers.

In the Figure 5 we present a simplified flow-graph of the original NICAM Shallow Water model code. The main computations performs in the cycle by the time steps (nl). Before starting the main cycle computations we need to setup MPI configuration and compute the initial data. At the end of each time step we collect the data from each MPI process and go to the next time iteration.

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 decided to port this module to GPU.

In the Figure 6 we present our GPU implementation approach scheme.

After we computed an initial data on CPU we send the ones, necessary for "One large step" computations, to GPU and store them in GPU global memory. Some of this initial data are constant and by porting them once in the beginning of the NICAM computations we reduce CPU-GPU communicational overheads.

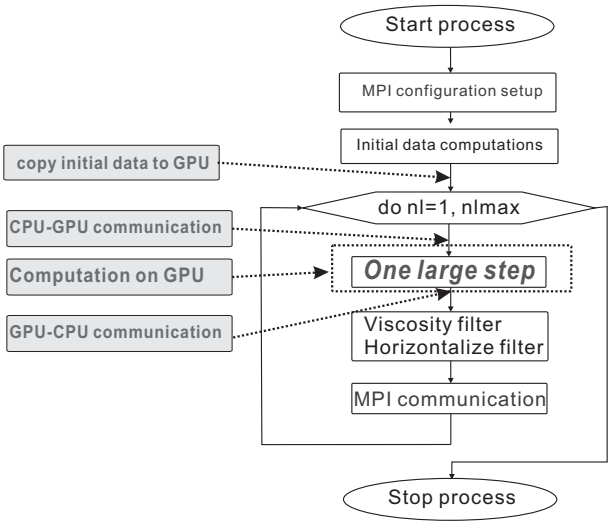

Fig. 6: NICAM GPU implementation approach. Porting the most computationally intensive module "One large step" to the GPU

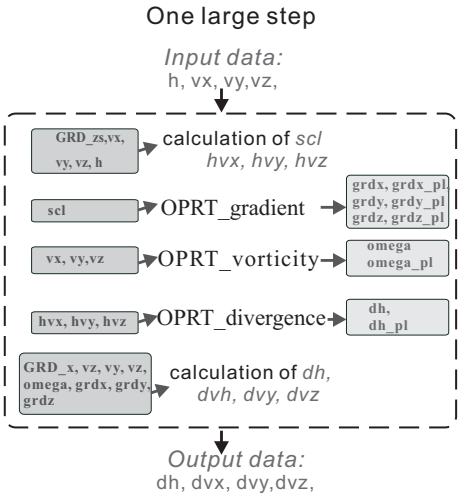

Fig. 7: "One large step" module. Left side blocks - input data, right side blocks - output data.

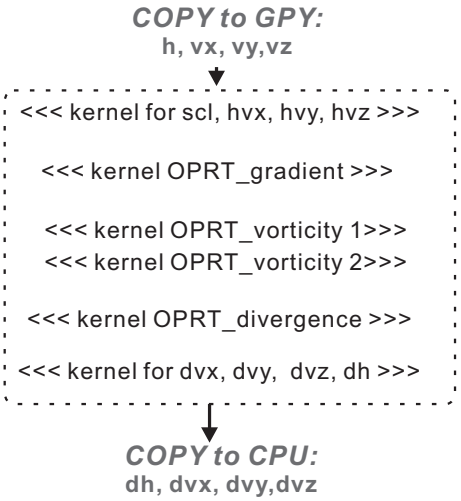

Fig. 8: "One large step" on GPU.

The data, which are variable inside the cycle and necessary for the GPU computations, are transferring to GPU before the "One large step" module on each time step. Then, data, computed on GPU and needed for farther computations on CPU, copying to the CPU at the end of the "One large step".

In the Figure 7 we present "One large step" module in detail. This module performs one full time step of the second-order Runge-Kutta scheme, which uses for the approximation of solutions of ordinary differential equations. "One large step" module consist of 3 main subroutines: OPRT_gradient, OPRT_vorticity and OPRT_divergence. If we port only this subroutines to GPU the communication overheads will be significant and affect to the performance. Therefore we also port initial data and output data computation modules to GPU and 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 created one kernel for OPRT_gradient and one for OPRT_divergence. For OPRT_vorticity module it was necessary to create 2 kernels for the data synchronization issue. Figure 8 shows how our numerical scheme is computed by executing 6 CUDA kernels in order.

Each module, ported to the kernels, based on a nested loop calculation. Each loop iteration computes 1 element of 2-dimensional array. Each thread of our GPU kernels calculates 1 element of the array.

The 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 the threads of a block in SIMT (single instruction, multiple thread) groups of 32 called warps.

We use 2-dimensional grid, which size depends on the of grid level and region level sizes. We use a block configuration of 256 threads where we have one thread per element. Our block size is a multiple of 32 which fits with the warp size and, therefore, allow us to achieve maximum efficiency.

We have h, vx, vy and vz arrays as an input data for the GPU computations, and dh, dvx, dvy, dvz as an output data. All the data, used for GPU computations, are stored in GPU Global memory during the entire NICAM model computation process. This allows different kernels access to the common data.

4. Evaluation

In this section we present results of implementation NICAM code on 1 node and single GPU.

4.1 Evaluation environment

Described approach was implemented on the TSUBAME 2.0 supercomputer, established at Tokyo Institute of Technology.

TSUBAME 2.0 consists of 1408 compute nodes (thin nodes) of two Intel Xeon Westreme-EP 2.9 GHz CPUs and tree NVIDIA M2050 GPUs with 52Gb and 3Gb of system

and GPU memory, running SUSE Linux Enterprise Server II SP1.

1 node has 2 sockets, 12 cores/node. Each node is interconnected by dual QDR Infinitband network with a full bisection- bandwidth fat-tree topology.

NICAM code consists of both FORTRAN and C++ modules. We use PGI CUDA FORTRAN $pgfortran$ version 2011 compiler for the GPU and FORTRAN code and $mpicc$ compiler for the C++ code.

4.2 Performance results

In purpose to compare NICAM CPU version performance with one for the GPU version we investigated initial CPU version for the code performance on TSUBAME 2.0 supercomputer. After performing the CPU code analysis we modified the code, porting the most time-consuming module to the single GPU, verified and validated our implementation. Then we compared MPI-parallel code performance with the single GPU code performance.

4.2.1 CPU performance

As it was described in section II, NICAM CPU code is based on 2d-domain decomposition with FLAT-MPI parallel programming model. Initial grid is divided by regions, managed by different MPI processes. The number of cores is restricted by the region division level, which can has limited number of the regions, that is $10*(2^n)^2$, where n - the region level order. Therefore, the total number of processes can be only the divisor of the total number of regions.

Figure 9 shows strong scalability results for the initial CPU version on TSUBAME 2.0 supercomputer. Problem size is grid level 11 with region level 5, which corresponds to 41943042 grid points. Strong scalability results demonstrate good speed up resulting on the number of cores up to 2560.

Figure 10 and Figure 11 present CPU performance results for the most time-consuming modules, ported to GPU.

From the CPU performance graph we can see that increasing grid level the scalability is rising due to increasing of the computational intensity per one process.

We assume 5 processes is a saturation point of the MPI parallelization and further compare CPU performance results on 5 processes with the GPU performance results on single GPU (see Figure 12)

4.2.2 GPU numerical performance

We compared single GPU version performance result with the performance for parallel MPI-parallel version (saturation point). We assume that the saturation point for performance of the MPI-parallel version achieves when the number of cores equal 5. Due to the fact that we have 6 cores in one socket on Intel Xeon X5670, we compare 1 CPU socket results versus 1 GPU socket.

This comparison results for the 3 main subroutines of the NICAM Shallow Water model are shown on the Figure 12. We can see that for the grid level 6 (gl06)

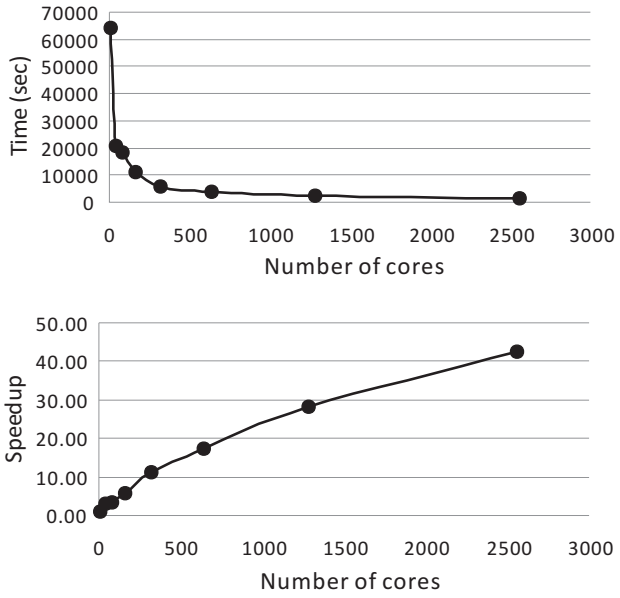

Fig. 9: Strong scalability of the CPU version. Average running time and speedup as a function of the number of cores for the grid level 11 and region level 5, which is correspond to horizontal size of grid cell around 3.5 km, 41943042 grid points

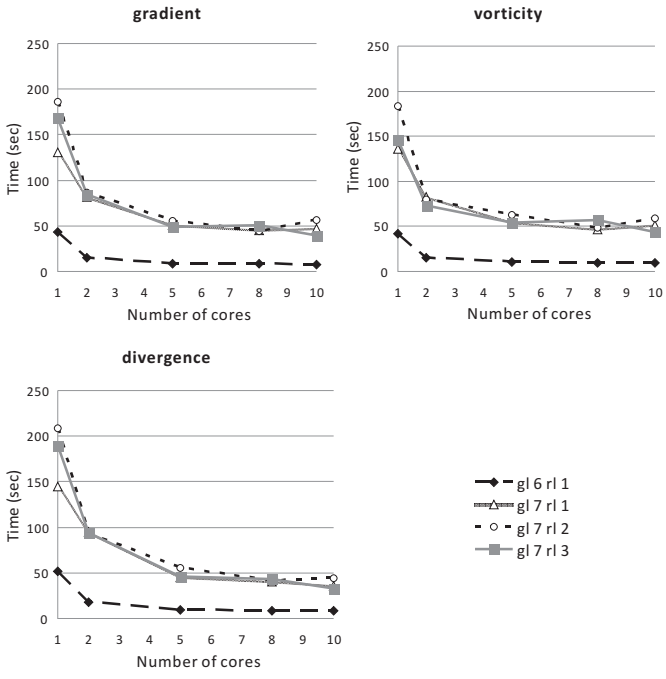

Fig. 10: CPU performance graph. Average running time as a function of the number of cores for the different configurations of the grid and region levels

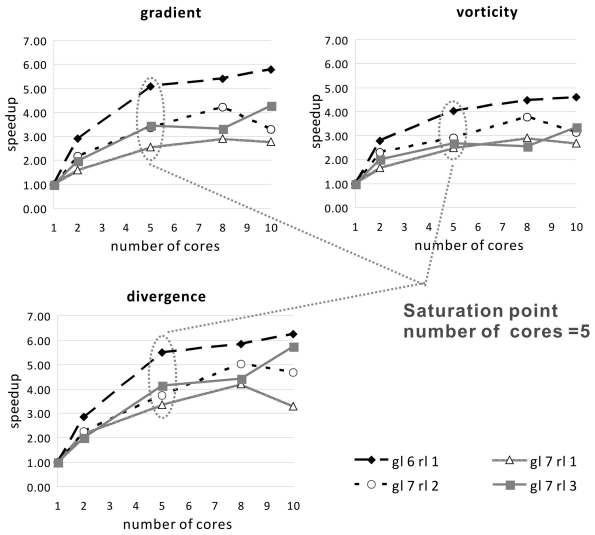

Fig. 11: CPU performance graph. Average speedup as a function of the number of cores for the different configurations of the grid and region levels

GPU versions performs 3 times faster then multi-CPU one. Increasing grid level to gl07 GPU version performs about 5 times faster then parallel-CPU one. It can be explained by the increasing computational intensity per kernel. We get 5x acceleration also for the grid level 8.

Due to limitations in GPU global memory, we were unable to scale NICAM code to a grid level higher then 8. The NVIDIA M2050 GPU only provides 3 GByte of Global memory. This limitation can be relaxed by using GPUs with higher amount of global memory. Also, as a future work, we are going to reduce amount of memory to be allocated per one GPU by splitting kernels between multiple GPUs, sharing 1 node.

4.2.3 GPU performance model

In purpose to measure the performance of our GPU NICAM implementation, we used "roofline" model of Samuel Willams [15]. This model compares achieved performance to a "roofline" graph of peak data streaming bandwidth and peak FLOP/s capacity.

We calculated performance by the next formula (1):

$$Performance = \frac{FLOP}{\frac{FLOP}{Fpeak} + \frac{Byte}{Bpeak} + \alpha} = \\ = \frac{FLOP/Byte}{\frac{FLOP}{Byte} + \frac{Fpeak}{Bpeak} + \alpha\frac{Fpeak}{Byte}} Fpeak \qquad (1)$$

Here $FLOP$ - number of floating-point operations for applications, $Byte$ - byte number of memory access for applications, $Fpeak$ - peak performance of floating-point operation, $Bpeak$ - peak memory bandwidth, α - time taken by other OPs except both FP and memory access.

For TSUBAME 2.0 supercomputer $Fpeak = 515GFlops$, $Bpeak = 148GByte/sec$ in double precision, we assume $\alpha = 0$.

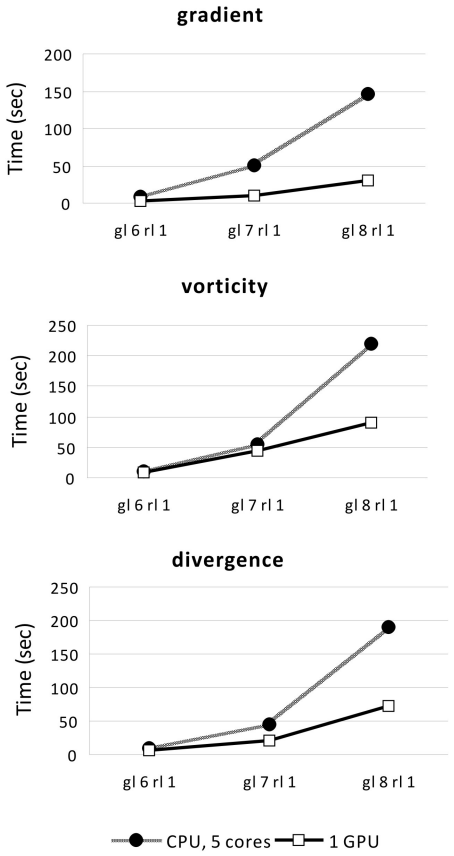

Fig. 12: Average running time for MPI-parallel CPU module versus time for GPU kernel as a function of the problem configuration for 3 main modules of the NICAM code.

Arithmetic intensity was calculated as $FLOP/Byte$.

Figure 13 shows "roofline" performance model for all GPU kernels of the GPU NICAM implementation. It is shown that output data calculations kernel, both vorticity kernels and gradient kernel give performance, close to theoretical one. This results indicate that we get maximum performance from the kernels. Performance of the input calculation kernel and divergence kernel we get is not the maximum one, but we will try to increase it by applying some additional optimization in the following work.

5. Conclusion

In this paper we have described our strategy of mapping to GPU a ultra-high resolution atmospheric model NICAM. We outlined the main steps of the mapping process and reported about results we got on TSUBAME 2.0 supercomputer.

We ported the most time-consuming modules of the initial NICAM Fortran code to a single GPU by using PGI CUDA Fortran. In this work we investigated 2-dimensional (Shallow water) case of NICAM model, and we plan to implement our GPU algorithm to the 3- dimensional case in a future work.

The results of our evaluation show a 5x speedup for a

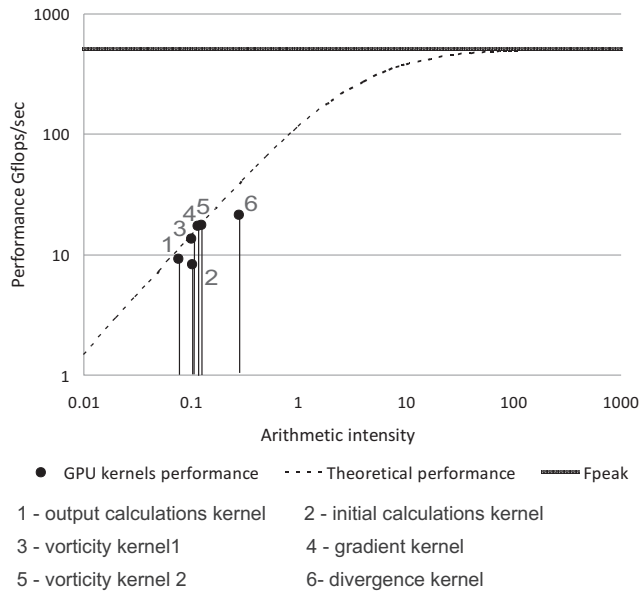

Fig. 13: Roofline graph: Visual GPU kernels performance model.

computationally intensive part of shallow water simulation model on a single GPU in comparison with a parallel CPU implementation.

In purpose to investigate performance of our GPU NICAM implementation, we created a "roofline" performance model for the GPU kernels. It was shown, that 4 of our 6 kernels give us performance close to the maximum one and, therefore, do not need any optimization. Two kernels still can be optimized to get better performance.

As our work progresses we will optimize current GPU implementation to get better performance. Then, we plan to apply our strategy to the multiple GPUs on one node, in order to reduce the amount of memory to be stored on one GPU. After that we plan to finish our MPI-CUDA implementation and run on multiple GPU nodes. We hope to see additional significant performance gains.

We also plan to investigate behavior of the NICAM code with OpenACC and the 3 compilers: PGI, CAPS/OMPP and Cray. Then it might be interesting to look at the NICAM code at the Cray XK6, which has been installed at Tokyo Tech this spring.

Acknowledgment

This work is done under the G8 ECS (Enabling Climate Simulation at Extreme Scale) project[16], focus on investigating ways to run efficiently climate simulations on future Exascale systems and get correct results.

References

[1] J. Michalakes, M. Vachharajani, "GPU Acceleration of Numerical Weather Prediction", *Parallel Processing Letters* , vol. 18 No. 4, pp. 531-548, 2008.

[2] M. Satoh, T. Matsuno, H. Tomita, H. Miura, T. Nasuno, S. Iga, "Nonhydrostatic Icosahedral Atmospheric Model (NICAM) for global cloud resolving simulations", *Journal of Computational Physics, the special issue on Predicting Weather, Climate and Extreme events*, vol. 227, pp. 3486-3514, 2007.

[3] M. W. Govett , J. Middlecoff, T. Henderson, Running the NIM next-generation weather model on GPUs", *Proc. 10th IEEE/ACM Int. Conf. Cluster, Cloud and Grid Computing (CCGrid)* , pp. 792-796, 2010.

[4] T. Shimokawabe, T. Aoki, Tomohiro 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 conference on Supercomputing (SC'11)*, Nov 2011.

[5] H. Tomita, M. Satoh, "A new dynamical framework of nonhydrostatic global model using the icosahedral grid", *Fluid Dynamics Research*, vol. 34, pp. 357-400, 2004.

[6] H. Tanaka, T. BOKU, M. Satoh, "Historical Progress of the Dynamical Core of the General Circulation Model of the Atmosphere", *NAGARE: Journal of Japan Society of Fluid Mechanics*, vol. 29, N 1, pp. 26-32, 2010.

[7] R. Sadourny, A. Arakawa and Y. Mintz, "Integration of the nondivergent barotropic vorticity equation with an icosa- hedral-hexagonal grid for the sphere", *Mon. Wea. Rev.*, vol.96, pp.35-356, 1968.

[8] D.L. Williamson, J.B. Drake, J.J. Hack, R. Jakob, P.N. Swarztrauber, "A standard test set for numerical approximations to the shallow water equations in spherical geometry", *Journal of Computational Physics* , vol. 102, p. 211, 1992.

[9] H. Tomita, M. Tsugawa, M. Satoh, K. Goto, "Shallow water model on a modified icosagedral geodesic grid by using spring dynamics", *J. Comp. Phys.*, vol. 174, pp. 579-613, 2001.

[10] M. Satoh, H. Tomita, H. Miura, S. Iga, and T. Nasuno, "Development of a global cloud resolving model – a multi-scale structure of tropical convections", *J. Earth Simulator*, vol.3, pp. 11-19, 2005.

[11] E. F. Toro, emphShock-Capturing Methods for Free-Surface Shallow Flows, Wiley and Sons Ltd., 2001, 309 pages.

[12] (2012) Portland Group International CUDA Fortran Compiler. [Online]. Available: http://www.pgroup.com/resources/cudafortran.htm

[13] (2012) NVIDIA Corporation. NVIDIA CUDA Programming Guide, version 4.1.. [Online]. Available: http://www.nvidia.com/cuda

[14] (2012) The Scalasca performance toolset. [Online]. Available: http://www.scalasca.org/

[15] S. Williams, A. Waterman, D. Patterson, "Roofline: an insightful visual performance model for multicore architectures", *Commun. ACM 52*, 4, pp. 65-76, 2009.

[16] (2012) G8 ESC – Enabling Climate Simulations at Extreme Scale. [Online]. Available: http://www.inria.fr/en/news/news-from-inria/g8-enabling-climate-simulation