GPU Acceleration of the Meso-scale Atmospheric Model "ASUCA"

Takashi Shimokawabe[1], Takayuki Aoki[1], Junichi Ishida[2], Chiashi Muroi[2]
[1] *Tokyo Institute of Technology, Japan;* [2] *Japan Meteorological Agency*
E-mail : shimokawabe@sim.gsic.titech.ac.jp

1 Introduction

Graphics Processing Units (GPUs) offer high performance of floating point calculation and wide memory bandwidth. Recently, general-purpose computation on GPUs (GPGPU) has become an active area of research in parallel computing because they provide high performance at relatively low cost for scientific computing applications. In the field of high performance computing, it was reported that various applications such as computational fluid dynamics [1] and astrophysical N-body simulations [2] ran dozens of times faster on a GPU than on a CPU core. In the field of numerical weather prediction, GPU acceleration of several modules from the Weather Research and Forecast (WRF) model were reported. Michalakes et al. reported a 20× speedup by GPU computing for WRF Single Moment 5-tracer (WSM5) microphysics, which is a computationally intensive physics module of the WRF model [3], but this speedup remains as a 1.3× overall improvement in the application. Linford et al. reported a 8.5× increase in speed on a GPU for a computationally expensive chemical kinetics kernel from WRF model with Chemistry (WRF-Chem) as compared to serial implementation [4]. Module-by-module acceleration is adopted as an approach to increase WRF speeds.

Full GPU application, in which all calculations are executed on a GPU using variables allocated on its memory, is essential in achieving more than ten times acceleration over the whole application compared to CPU application. This allows simulation to be run without frequent data transfer between the GPU and the host computer.

We are currently working on full GPU application for ASUCA [5] - a next-generation high resolution meso-scale atmospheric model being developed by the Japan Meteorological Agency. As a first step, we have implemented its dynamical core as a full GPU application, representing an important step toward establishing an appropriate framework for full GPU-based ASUCA. The GPU code is written from scratch in the CUDA (Compute Unified Device Architecture) [6] using its original code in Fortran as a reference. The Numerical results obtained from the GPU code agree with those from the CPU code within the margin of machine round-off error. In this paper, we report the results of GPU acceleration of the dynamical core in ASUCA, which has not yet been accelerated in WRF.

2 GPU implementation

In this study, we computed the dynamical core on an NVIDIA GTX 285 using CUDA 2.3 with an AMD Phenom 9750 Quad (2.4 GHz) and 8 GByte of memory as the host computer.

In the CUDA programming, CUDA kernels for the GPU are programmed. When a kernel is launched, it is executed by individual threads arranged into blocks with unique block and thread IDs. All blocks are grouped as a grid and all threads in the grid are able to access VRAM called as the global memory on the GPU. Access to the global memory takes 400 to 600 clock cycles, which corresponds to 159 GB/s, for example, in the case of the GTX 285. 16 kByte of shared memory is assigned in each block as scratchpad memory with an access time of about two cycles. Any part of the shared memory can be read and written by all threads in the block, which is utilized as a software-managed cache to reduce access to the global memory.

The implementations of some functions used in the dynamical core in ASUCA are explained here.

2.1 Advection

In order to improve calculation performance in ASUCA, access to the global memory is reduced by making use of the shared memory as a software-managed cache. To calculate advection for a given grid size (nx, ny, nz), the kernel functions for the GPU are configured for execution in $(nx/64, nz/4, 1)$ blocks with $(64, 4, 1)$ threads in each block. Each thread specifies an (x, z) point and performs calculations from $j = 0$ to $j = ny - 1$ marching in the y direction. In order to facilitate the implementation of kernel functions for domain decomposition with MPI, the z direction in numerical space is mapped to the y direction in CUDA.

The four-point stencil of a point in each direction is required to compute advection. To carry out calculations in the j th slice, the elements in the current slice are needed for calculations by more than one thread. On the other hand, elements preceding and succeeding the current y position are used only for the thread corresponding to the element's (x, z) position. Each block therefore has an array with $(64 + 3) \times (4 + 3)$ elements in its shared memory, which is utilized to accommodate 2D sub-domain data and halos for the current j th slice. The elements along the y axis are stored in registers on the corresponding thread (Figure 1). Data stored in both shared memory and registers to perform the j th computation are reused for the $j+1$ th calculation as far as possible.

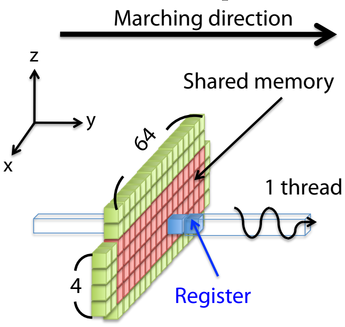

Figure 1: $(64+3)\times(4+3)$ elements in shared memory and 3 elements in registers along the y axis

2.2 1D Helmholtz-like elliptic equation

The 1D Helmholtz-like elliptic equation is solved in the vertical direction because the HE-VI scheme is adopted in ASUCA. Through discretization of the equation, a tridiagonal matrix is obtained. The basic strategy of implementation for the solver for this matrix is the same as that for advection. However, because sequential computation in the z direction is required for it, threads should not march along the y axis in view of the efficiency of parallel computing by threads. Thus, $(nx/64, ny/4, 1)$ blocks with $(64, 4, 1)$ threads are configured to the given grid size (nx, ny, nz). The threads march in the z direction for the Helmholtz-like elliptic equation.

3 Results

Figure 2 shows the performance of the dynamical core in ASUCA in both single- and double-precision floating-point calculation for six different grid sizes. With nx set as 32 and nz set as 64, the value of ny is varied from 16 to 56. In order to measure the performance on a GPU, we count the number of floating-point operations in ASUCA running on a CPU with the Performance API (PAPI) [7]. This code is implemented in C/C++ language corresponding to GPU code. Using the obtained count and the GPU computation time, the performance on the GPU is evaluated. The performance of 67.1 GFlops in single precision for a $320\times56\times64$ mesh on a single GPU has been achieved. It is found that the dynamical core in ASUCA implemented on the GPU runs 51.5 times faster than the original code for CPU performed by the serial implementation in Fortran on the Intel Core i7 920 2.67 GHz. In the case of the computation in double precision, the speed is increased by a factor of 15.8.

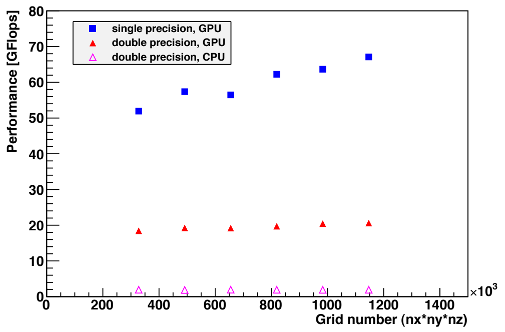

Figure 2: Performance of ASUCA on a GPU (GTX 285) and a CPU. The solid blue and red points indicate the performance of the GPU version in single and double precision respectively. The magenta outline points show the performance of the original Fortran code running on a CPU core.

4 Conclusion and future work

We are currently developing a full GPU version of ASUCA. As a first and key step, we have implemented its dynamical core on a GPU. The effective utilization of shared memory in the GPU for optimization has resulted in the performance of 67.1 GFlops, which is 51.5 times faster than the original code on a CPU. Implementation for multi-GPUs will be a subject of future work.

Acknowledgments This research was supported in part by the Global Center of Excellence Program "Computationism as a Foundation for the Sciences" from the Ministry of Education, Culture, Sports, Science and Technology of Japan and in part by the Japan Science and Technology Agency CREST research program "Ultra Low-Power (ULP-HPC)".

References

[1] J. C. Thibault and I. Senocak. CUDA implementation of a navier-stokes solver on multi-GPU desktop platforms for incompressible flows. In *Proceedings of the 47th AIAA Aerospace Sciences Meeting*, number AIAA 2009-758, jan 2009.

[2] Tsuyoshi Hamada, Tetsu Narumi, Rio Yokota, Kenji Yasuoka, Keigo Nitadori, and Makoto Taiji. 42 tflops hierarchical n-body simulations on gpus with applications in both astrophysics and turbulence. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–12, New York, NY, USA, 2009. ACM.

[3] John Michalakes and Manish Vachharajani. GPU acceleration of numerical weather prediction. In *IPDPS*, pages 1–7. IEEE, 2008.

[4] John C. Linford, John Michalakes, Manish Vachharajani, and Adrian Sandu. Multi-core acceleration of chemical kinetics for simulation and prediction. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–11, New York, NY, USA, 2009. ACM.

[5] Junichi Ishida, Chiashi Muroi, Kohei Kawano, and Yuji Kitamura. Development of a new nonhydrostatic model "ASUCA" at JMA. *CAS/JSC WGNE Reserch Activities in Atmospheric and Oceanic Modelling*, 2010.

[6] CUDA Programming Guide 2.3. `http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_CUDA_Programming_Guide_2.3.pdf`, 2009.

[7] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. A portable programming interface for performance evaluation on modern processors. *Int. J. High Perform. Comput. Appl.*, 14(3):189–204, 2000.