

Multi-GPU Accelerated Non-Hydrostatic Numerical Ocean Model with GPUDirect RDMA Transfers

Takateru Yamagishi

Research Organization for
Information Science and Technology
Kobe, Hyogo, Japan
yamagishi@rist.jp

Yoshimasa Matsumura

Atmosphere and Ocean Research
Institute, The University of Tokyo
Kashiwa, Chiba, Japan
ymatsu@aori.u-tokyo.ac.jp

Hiroyasu Hasumi

Atmosphere and Ocean Research
Institute, The University of Tokyo
Kashiwa, Chiba, Japan
hasumi@aori.u-tokyo.ac.jp

ABSTRACT

We have implemented our “kinaco” numerical ocean model on Tokyo University’s Reedbush supercomputer, which utilizes NVIDIA Pascal P100 GPUs with GPUDirect technology. We have also optimized the model’s Poisson/Helmholtz solver by adjusting the global memory alignment and thread block configuration, introducing shuffle functions to accelerate the creation of coarse grids and merging small kernels in the multigrid preconditioner. We also utilize GPUDirect RDMA transfers to improve MPI communication efficiency. By exploiting the GPUs’ capabilities, the GPU implementation is now twice as fast as the CPU version, and it shows good weak scalability to multiple GPUs. Most of the GPU kernels are accelerated, and the velocity diagnosis functions in particular are now approximately seven times faster. The performance of inter-node data transfers using a CUDA-aware MPI library with GPUDirect RDMA transfers is comparable to that on CPUs.

1 Introduction

Numerical ocean models play important roles in predicting climate change and investigating marine resources under various climatic conditions. Despite the fact that oceans cover a vast area, ocean circulation involves dynamics at small spatial scales. To conduct realistic simulations, we therefore have to resolve and simulate very small-scale dynamics over an extensive ocean. GPUs have great computational power and wide memory bandwidth, making them suitable for numerical simulations with large numbers of grids. Even so, multiple GPUs are required to simulate such huge areas. A previous study [1] implemented an ocean model on multiple GPUs using OpenACC, but this lacked the necessary functionality to fully optimize the model. In addition, no previous studies have applied NVIDIA GPUDirect remote direct memory access (RDMA) transfers to ocean models.

In this study, we implemented our “kinaco” numerical ocean model on Tokyo University’s Reedbush supercomputer using CUDA, which has significantly more functionality than OpenACC and allows for more flexible implementation and optimization techniques. We also compare our model’s performance with and without GPUDirect RDMA transfers. In addition, we implemented and optimized several original GPU kernels, including the GPU-specific optimizations introduced in

our previous single-GPU study [3]. We also applied four new optimizations to the GPU kernels in the Poisson/Helmholtz (P/H) solver. The resulting GPU-based model was twice as fast as the CPU version, and showed good scalability to multiple GPUs.

2 Model description and implementation

Our model simulates ocean dynamics using the three-dimensional non-hydrostatic Navier–Stokes equation in an orthogonal curvilinear coordinate system. All the equations are discretized on structured grids and approximated by Poisson/Helmholtz equations, which are solved via the conjugate gradient method with a multigrid preconditioner (MGCG). The MGCG solver is known to be scalable, with the number of iterations required remaining constant regardless of the number of grids. For more details about our kinaco model, please see [2].

We implemented all of kinaco’s components in CUDA Fortran and ran it on Tokyo University’s Reedbush supercomputer, which incorporates NVIDIA Pascal P100 GPUs with GPUDirect technology. This means no data is transferred between the CPU and GPU, except when it must be sent between computational nodes via MPI communication. We were able to exploit data parallelism on the GPUs due to the ocean model’s intrinsic characteristics, and could hide the latency by swapping GPU threads. In addition, we rewrote the original array of structures as ordinary arrays to enable coalesced access to the GPUs’ global memory. For more implementation details, please see our previous study on the single-GPU kinaco implementation [3]. To enable fast and efficient MPI communication, we adopted CUDA-Aware MPI and GPUDirect RDMA transfers, which enable direct data transfers between the GPUs’ global memories without going via the CPU’s main memory.

3 Poisson/Helmholtz Solver Optimizations

We had previously optimized the P/H solver for the core ocean dynamics equations for a single GPU [3]. In this study, we introduced the following four additional optimizations.

- 1) The GPUs’ global memory alignment was adjusted. Most prognostic values prepared the halo region for data transfer between adjacent computational nodes; for example, we usually allocated $(-1:n_1+2, -1:n_2+2, -1:n_3+2)$ to the values in

the practical region (n_1, n_2, n_3). However, this memory allocation led to misalignment and wasted 32-byte memory segments. Instead, we now eliminate the halo region and access the transferred data directly via MPI communication.

- 2) The GPU thread block configuration was adjusted so as to utilize the L1 cache efficiently. Due to the core equations' physical nature, their matrix coefficients are vertically uniform in the middle ocean layer. For example, matrix coefficient b , which is allocated a (256, 256, 32) memory block, is vertically uniform from $b(i, j, 3)$ to $b(i, j, 30)$. We therefore set the GPU thread block to (32, 1, 16) to utilize the L1 cache efficiently for b . With this configuration, vertically uniform values are assigned to the same GPU streaming multiprocessor, and hence make better use of the L1 cache.
- 3) The GPU's intrinsic shuffle function is utilized to create coarse grids. The multigrid preconditioner converts fine grids into coarse grids, for example converting a (32, 32, 1) grid into a (16, 16, 1) grid by averaging adjacent values. However, this meant each thread made uncoalesced accesses to global memory with a stride of 2 words. We now use shuffle functions to avoid this. First, we make coalesced accesses to global memory and store the values in GPU registers. Then, we average the values by utilizing shuffle functions to directly access the adjacent thread's register values.
- 4) Small kernels are merged in the multigrid preconditioner. The multigrid preconditioner invokes many GPU kernels to deal with small grids, each typically running for a few microseconds. However, launching a new kernel incurs an overhead of about 20 microseconds, which is comparable to its runtime. We therefore merge several small kernels into one to eliminate these launching costs.

4 Performance Evaluation

Each GPU's domain was set to a size of (256, 256, 32), and we evaluated the model using 2, 4, 8, or 16 GPUs or CPUs. Since the total number of grids increased linearly with the number of GPUs, this was an evaluation of the model's weak scalability. We considered a test case with idealistic and systematic forcing, and with boundary conditions that induced baroclinic instability, which is a geophysically important phenomenon. This was done to eliminate nonlinear reactions, which might otherwise have complicated this evaluation. We used NVIDIA Pascal P100 GPUs and Intel Xeon E5-2695v4 CPUs, both installed in the Reedbush supercomputer, and we utilized the NVIDIA Visual Profiler (NVVP) to investigate the GPUs' behavior in detail.

Our kinaco model ran twice as fast on GPUs as it did on CPUs for all numbers of processors between 2 and 16 (Fig. 1). It also showed good weak scalability on both CPUs and GPUs. After investigating the performance of the model's individual components (Fig. 2), we found that most of them ran faster on GPUs than on CPUs, especially the velocity diagnosis functions, which were approximately seven times faster. The four new P/H solver optimizations also yielded good performance, and

analyzing the NVVP data made it clear that they were working as intended. In addition, inter-node data transfers using the CUDA-Aware MPI library with GPUDirect RDMA transfers performed similarly to those on CPUs when transferring halo velocity data. Only the tracer component, which was under-optimized, showed no advantage compared with on CPUs.

5 Discussion and Conclusions

This study has shown the applicability of multiple GPUs to numerical ocean modeling. Our four new optimization methods and GPUDirect RDMA transfers are both good matches to kinaco's numerical characteristics, and we believe these ideas could also be applied to other numerical ocean models. The MGCG solver is known to be scalable, and kinaco has been shown to scale up to 8000 CPUs [4]. We believe the same ideas can be applied to GPUs, so it can also scale to thousands of GPUs. Although the speed of MPI communication with GPUDirect RDMA transfers is comparable to that on CPUs, considering our final goal of making the multi-GPU kinaco implementation significantly faster than the CPU version, MPI communication could become a bottleneck.

In future work, we plan to optimize the tracer component and find ways to improve the performance of kinaco's GPU implementation on huge numbers of grids. This will involve evaluating and optimizing it for thousands of GPUs. To reduce the MPI communication cost, we will attempt to overlap it with GPU kernel execution.

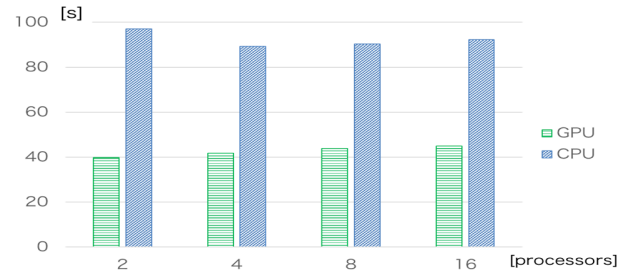


Figure 1: Elapsed times for all components in the main timing loop on GPUs and CPUs, showing the weak scaling.

CPU	GPU	CPU-GPU (>0 is better)	components
92.81	44.99	47.82	All components
3.14	0.37	2.76	density, richardson
7.35	1.76	5.59	nonlinear terms, coriolis
0.43	0.13	0.30	buoyancy
52.29	19.90	32.39	Poisson/Helmholtz solver
8.53	1.13	7.40	velocity diagnosis
0.50	0.45	0.05	halo data transfer (velocity)
17.51	19.33	-1.82	tracer advection-diffusion
3.06	1.92	1.14	others

Figure 2: Comparison of elapsed times for each component on GPUs and CPUs.

REFERENCES

- [1] M. Milakov (2013). Accelerating NEMO with OpenACC. NVIDIA GTC 2013.
- [2] T. Yamagishi and Y. Matsumura (2016). GPU Acceleration of a Non-hydrostatic Ocean Model with a Multigrid Poisson/Helmholtz Solver. *Procedia Computer Science*, 80, 1658-1669, Jun 2016.
<http://dx.doi.org/http://dx.doi.org/10.1016/j.procs.2016.05.502>
- [3] Y. Matsumura and H. Hasumi (2008). A non-hydrostatic ocean model with a scalable multigrid Poisson solver. *Ocean Modelling* 24(1–2): 15-28.
<http://dx.doi.org/10.1016/j.ocemod.2008.05.001>
- [4] Y. Matsumura, H. Hasumi, E. Tomiyama, T. Yamagishi, T. Inoue, S. Inoue, K. Minami and K. I. Ohshima (2012). Numerical simulation of oceanic small scale processes by a non-hydrostatic ocean model. *K computer symposium* 2012.