Multi-GPU Implementation and Roofline Analysis of a Numerical Global Ocean Model

Takateru Yamagishi
Research Organization for Information Science and Technology
Minato, Tokyo, Japan
takateru.yamagishi@rist.or.jp

Masao Kurogi
Japan Agency for Marine-Earth Science and Technology
Yokohama, Kanagawa, Japan
m_kurogi@jamstec.go.jp

Takao Kawasaki
Japan Agency for Marine-Earth Science and Technology
Yokohama, Kanagawa, Japan
t_kawasaki@jamstec.go.jp

Yoshimasa Matsumura
National Institute for Environmental Studies
Tsukuba, Ibaraki, Japan
matsumura.yoshimasa@nies.go.jp

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

ABSTRACT

Numerical ocean models are essential tools for climate prediction and marine resource studies, requiring high resolution and realistic physical processes. We developed the global ocean model COCO and implemented it on GPUs using an OpenACC directive-based approach, while maintaining compatibility with CPUs. Performance was evaluated on the Miyabi Supercomputer, which includes GPU-based (NVIDIA GH200) and CPU-based (Intel Xeon MAX 9480) systems. Realistic ocean experiments with a 0.17° global grid showed that most components achieved faster execution on GPUs, with the tracer calculation accelerated by a factor of 2.9. Roofline analysis revealed that most loops were memory-bound, and GPU speedup was constrained by memory bandwidth rather than compute capability. Future improvements will require increasing arithmetic intensity and applying kernel-level optimizations, while ensuring compatibility between CPU- and GPU-based codes.

1 Introduction

Numerical ocean models are expected to play important roles in predicting climate change and investigating marine resources under various climatic conditions. GPUs have substantial computational power and wide memory bandwidth, making them suitable for realistic numerical ocean simulations with many grids. The implementation of ocean model to GPUs have been tackled with NIVIDIA GPUs with OpenACC[1]. In this study, we implemented and evaluated the performance of COCO with minimum modification on GH200 supercomputer system and compared with CPU with high memory bandwidth.

2 Model description

The model used in this study is an ice-ocean coupled model, named COCO[2]. COCO computes the ocean's dynamical fields by solving the three-dimensional Navier–Stokes equations, taking into account oceanic factors such as the Coriolis force, under the Boussinesq and hydrostatic approximations. Based on the computed dynamical fields, the distributions of temperature and salinity are obtained by solving the advection–diffusion equation. To avoid the problem of grid convergence near the poles in spherical coordinates, we adopt the tripolar grid systems[3] that allows the poles to be placed over land[Figure 1]. Most of the data arrays used in the computations are three-dimensional, and both the order of processing each array element and access to neighboring elements follow a systematic pattern.

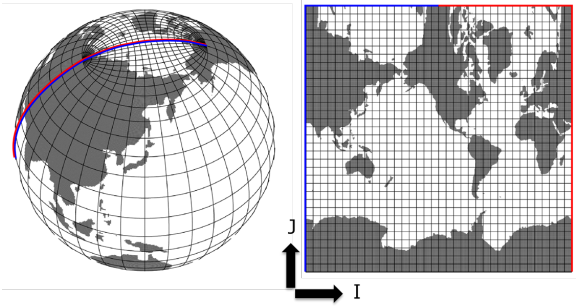

Figure 1: COCO's generalized curvilinear horizontal coordinate.

3 Implementation and Optimization on GPUs

We adopted OpenACC to implement COCO on NVIDIA GPUs. This directive-based approach reduces development cost and allows a single repository for CPU and GPU. Ocean models contain numerous code lines and computational loops; to efficiently exploit GPU acceleration and minimize CPU–GPU transfers, nearly all loops must be implemented on GPUs. Thus, low-cost GPU implementation is indispensable. We inserted basic OpenACC directives such as kernels or parallel (typically on outermost loops) and loop independent/seq directives, while

leaving GPU thread configurations (e.g., gang/vector clauses not inserted) to the compiler. Data directives were used to avoid redundant transfers.

The ocean computation code of COCO consists of about 5,000 lines and 426 computational loops. Of these, 417 loops were implemented on GPUs using the basic OpenACC style, while only 9 required algorithmic modifications. These correspond to advection–diffusion equations calculating tracer values (temperature, salinity). In the original ASIS code, only the vertical (K-axis) could be parallelized, with horizontal (I- and J-axes) executed sequentially [Figure 2]. To improve parallelism, we modified the algorithm so that horizontal loops are executed separately for positive and negative upwind components, enabling parallelization in both horizontal and vertical directions [Figure 2]. Although this increases the number of horizontal iterations, the performance gain from enhanced GPU thread parallelism outweighed the overhead.

1-D parallel code (ASIS) CPU

```
do k = kstr, kend : vertical axis, parallelizable
  do ij = ijstr, ijend : horizontal axis, sequential
    if ( uv(ij, k) .gt. 0.d0 ) then
      sm(ij  , k, n) = sm(ij  , k, n) + fm(ij-1, k)
    else
      sm(ij-1, k, n) = sm(ij-1, k, n) + fm(ij-1, k)
    end if
  end do
end do
```

3-D parallel code (TUNED) GPU

```
do k = kstr, kend : vertical axis, parallelizable
  do ij = ijstr, ijend : horizontal axis, parallelizable
    if ( uv(ij  , k) .gt. 0.d0 ) then
      sm(ij, k, n) = sm(ij, k, n) + fm(ij-1, k)
    end if
  end do
  do ij = ijstr-1, ijend-1 : horizontal axis, parallelizable
    if ( uv(ij+1, k) .le. 0.d0 ) then
      sm(ij, k, n) = sm(ij, k, n) + fm(ij  , k)
    end if
  end do
end do
```

Figure 2: Change of algorithms on advection terms. (a) ASIS. (b) TUNED.

4 Performance Evaluation

We compared GPU and CPU performance using a representative experiment that included typical processes on a realistic global ocean geometry. The horizontal grid resolution was 0.17° (1/6°) with [2160, 1680, 63] grid points, divided into 32 areas. The model was evaluated on 32 GPUs or CPUs.

It was executed on the Miyabi Supercomputer, which has two systems: a GPU-based system Miyabi-G with NVIDIA GH200 and a CPU-based system Miyabi-C with Intel Xeon MAX 9480. Most components achieved better elapsed times on GPUs (Table 1). The TRACER part, the most time-consuming component, was accelerated by a factor of 2.9, highlighting the effect of algorithmic modifications [Figure 2]. In contrast, the BRTRO part showed performance degradation because its two-dimensional values provided insufficient parallelism to fully utilize GPU threads.

Table 1: Performance of CPUs and GPUs on each component

	CPU	GPU	ratio of speedup
COEFF	1.10	0.40	2.7
BRCLI	0.23	0.08	2.7
BRTRO	0.15	0.24	0.6
TDIAG	0.04	0.02	2.3
TRACER	1.42	0.50	2.9
VDIAG	0.09	0.01	6.8
TOTAL	3.03	1.25	2.4

Elapsed time, unit: second

Although GPUs demonstrated better overall performance, the acceleration ratio was still lower than the ratio of their theoretical fp64 peak performance (67TFLOPS/6.8TFLOPS = 9.6 times). To investigate this, we performed a Roofline analysis to identify the essential performance-limiting factors on CPUs[Figure 3]. Most of the time-consuming loops on CPUs were memory-bound. The Roofline analysis of the GPU results was generally the same as that of the CPU. Therefore, the speedup on GPUs is largely dependent on the ratio of memory bandwidth of the processors (4.0TB/s/3.2TB/s = 1.25 times).

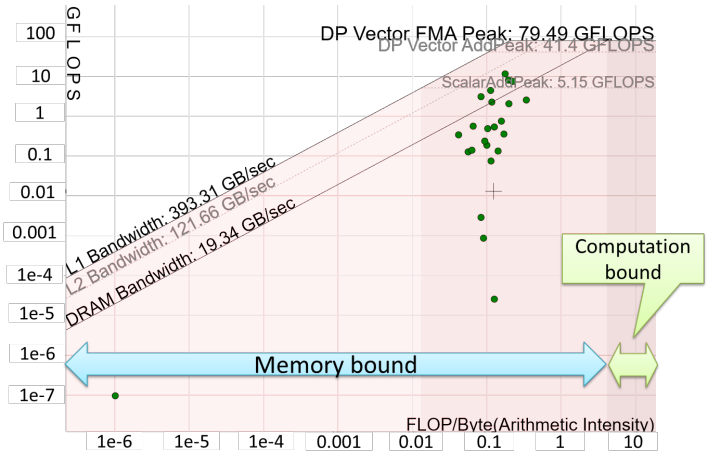

Figure 3: Roofline model analysis of the time-consuming loops on CPUs. Horizontal axis: arithmetic intensity; Vertical axis: computational performance (GFLOPS).

.

5 Discussion and Conclusions

In this study, we found that most computational loops exhibited low arithmetic intensity, and that the theoretical memory bandwidth ratio had a significant impact on performance. As future work, we plan to modify the algorithms of ocean models, which have traditionally been developed for the CPU execution model, to better align with the GPU execution model. Increasing arithmetic intensity through temporal blocking will be crucial, as will incorporating techniques such as tuning GPU thread configurations, leveraging GPU registers by optimizing workload allocation to GPU threads or loop exchange, and applying cache optimizations, in order to fully exploit the high computational capabilities of GPUs.

REFERENCES

[1] J. Jiang (2019). Porting LASG/ IAP Climate System Ocean Model to Gpus Using OpenAcc. IEE Access.

[2] H. Hasumi (2006). CCSR Ocean Component Model (COCO) version 4.0, Center for Climate System Research Rep., 25.

[3] R. J. Murray (1996). Explicit generation of orthogonal grids for ocean models, J. Comput. Phys., 126, 251–273.