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

Takateru Yamagishi(1) Masao Kurogi(2) Takao Kawasaki(3) Yoshimasa Matsumura(4) Hiroyasu Hasumi(5)

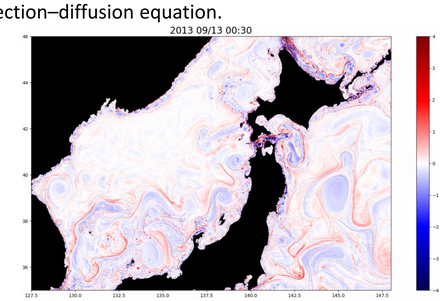
1: Research Organization for Information Science and Technology, *takateru.yamagishi@rist.or.jp* 2: Japan Agency for Marine-Earth Science and Technology, *m_kurogi@jamstec.go.jp* *3*: Japan Agency for Marine-Earth Science and Technology, *t_kawasaki@jamstec.go.jp*
4: National Institute for Environmental Studies, *matsumura.yoshimasa@nies.go.jp* 5: Atmosphere and Ocean Research Institute, The University of Tokyo, *hasumi@aori.u-tokyo.ac.jp*

Abstract

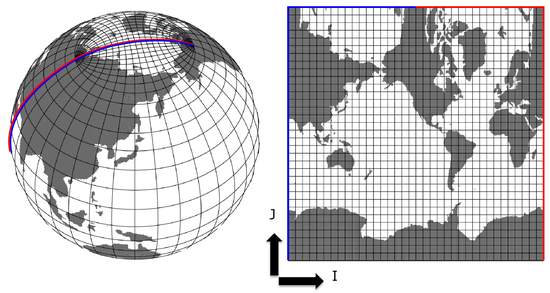
- Numerical ocean models are key for climate prediction and marine resource studies.
- We developed the global ocean model COCO with an OpenACC GPU implementation compatible with CPUs.
- Performance was tested on the Miyabi Supercomputer (GPU: NVIDIA GH200, CPU: Intel Xeon MAX).
- A 0.17deg global grid experiment showed up to 2.9 times speedup in tracer calculations on GPUs.
- Roofline analysis found GPU performance limited by memory bandwidth, suggesting kernel-level optimizations as future work.

"COCO" ice-ocean coupled model and Supercomputer Miyabi

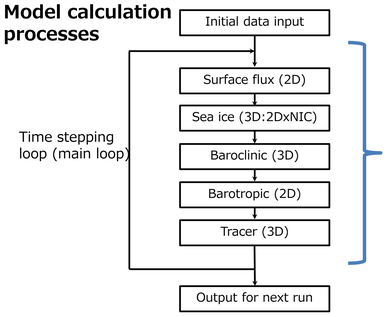
- The model is an ice–ocean coupled system (COCO)【1】.
- COCO solves 3D Navier–Stokes equations with Coriolis force under Boussinesq and hydrostatic approximations.
- Temperature and salinity fields are computed using the advection–diffusion equation.



- A tripolar grid system【2】avoids pole convergence by placing poles over land (Fig. 1).
- Data arrays are mainly 3D, with systematic access patterns across neighboring elements.



Model calculation processes



- Ocean models involves many processes
- Minimize CPU–GPU data transfer is necessary
- GPU acceleration requires implementing nearly all loops
- OpenACC enables low-cost approach for ocean models.

Why OpenACC?
- Many COCO users are oceanographers, not HPC experts
- Accessibility is the top priority
- Low development cost, single CPU–GPU code base
- OpenMP-like style, familiar and user-friendly

Supercomputer Miyabi

- COCO implemented and evaluated on the Miyabi Supercomputer, operated by JCAHPC (University of Tokyo & University of Tsukuba)
- Miyabi-G: GPU-based (NVIDIA GH200 Grace-Hopper)
- Miyabi-C: CPU-based (Intel Xeon MAX 9480)
- Performance: 80.1 PFLOPS (fp64).

Specification of Miyabi-G/C

	Miyabi-G	Miyabi-C
Processor	NVIDIA GH200	Intel Xeon MAX 9480
Theoretical calculating performance	3.5TFLOPS (Grace CPU)	6.8TFLOPS
	67TFLOPS (Hopper GPU)	
Memory bandwidth	512GB/s (Grace CPU)	3.2TB/s
	4.0TB/S (Hopper GPU)	
Memory capacity	120GB (Grace CPU)	64GiB
	96GB (Hopper GPU)	

3-D parallelization of tracer equations

- Advection–diffusion equation discretized with upwind differencing
- Original ASIS code: only vertical direction parallelized → insufficient GPU parallelism
- Algorithm modified: horizontal loops split into positive/negative upwind components
- Enables parallelization in horizontal direction
- Loop iterations double, but GPU thread parallelism yields higher speedup

1-D parallel code (ASIS)

```
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
```

In the ij-axis loop, memory access during the TRUE condition conflicts with that during the ELSE condition.

3-D parallel code (TUNED)

```
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
```

Experiment and Evaluation

- Idealistic and systematic forcing assuming baloclinic instability
- GPU vs. CPU performance compared using a realistic global ocean experiment
- Resolution: 0.17 degree with [2160, 1680, 63] grid points, divided into 32 areas
- Evaluated on 32 GPUs or CPUs of the Miyabi-G: GPU (NVIDIA GH200) and Miyabi-C: CPU (Intel Xeon MAX 9480)
- Most components faster on GPUs
- TRACER: 2.9 times speedup (largest gain, due to algorithmic modifications)
- BRTRO: slower on GPUs, limited parallelism from 2D values

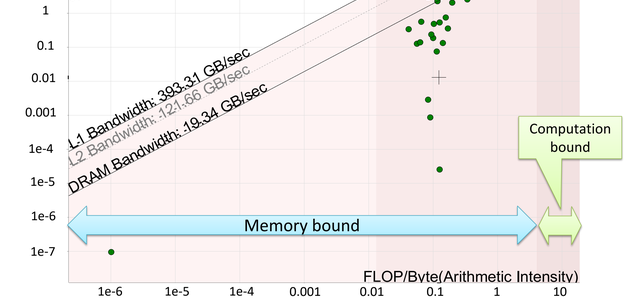
Performance of CPUs and GPUs on each component

	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

Roofline Analysis

Roofline model analysis of the execution results on CPU



Horizontal axis: arithmetic intensity; Vertical axis: computational performance (GFLOPS).

- Acceleration ratio of GPU is below theoretical peak ratio
- Roofline analysis performed to identify limiting factors
- Most CPU loops memory-bound
- GPU speedup limited by memory bandwidth

Implementation of ocean model to GPUs with "the basic OpenACC style"

- Inserting basic OpenACC directives
  - Kernels or parallel directives
    - Basically, inserted into the outermost of nested loops
  - loop independent/seq directives
    - Inserted according to the loop algorithms
  - Not specify any quantitative configurations for GPU threads
    - Gang or vector clauses are not applied, leaving the compiler's decision.
  - Data directives
    - Remove redundant data transfer between CPU and GPU
- Minimize modification of original CPU code as possible
  - Ideal: If you remove the lines including "!$acc" automatically, you can get the code essentially the same as the original
- Out of 417 loops, only 9 loops of the advection–diffusion equation could not be accelerated with the above OpenACC basic implementation and required algorithmic modifications.

Example of implementation with the basic OpenACC style

```
!$acc data copy(fz)
!$acc data copyin(mask, kpa, a, idz)
 !$acc kernels
DO j=1, jsize
  DO i=1, isize
   !$acc loop seq
    DO k=0, ksize
      fz(i,j,k) = - mask(i,j,k) * mask(i,j,k+1) *  &
                    ( kpa(i,j,k) + kpa(i,j,k+1)) *  &
                    ( a(i,j,k+1) - a(i,j,k)) * idz(k)
    END DO
  END DO
END DO
!$acc end kernels
!$acc end data
!$acc end data
```

Discussion and future work

- In our ocean model COCO, most computational loops showed low arithmetic intensity
- Memory bandwidth had a significant impact on performance
- Future work: modify algorithms developed for CPU execution model to align with GPU execution model
- Key directions:
  - Increase arithmetic intensity (e.g., temporal blocking)
  - Tune GPU thread configurations
  - Leverage GPU registers via optimized workload allocation or loop exchange
  - Apply cache optimizations following GPU execution model
- Goal: fully exploit the high computational capability of GPUs

References

[1] H. Hasumi (2006). CCSR Ocean Component Model (COCO) version 4.0, Center for Climate System Research Rep., 25.
[2] R. J. Murray (1996). Explicit generation of orthogonal grids for ocean models, J. Comput. Phys., 126, 251–273.