GPU Acceleration of a Non-Hydrostatic Ocean Model with Lagrangian Particle Tracking

Takateru Yamagishi
Research Organization for Information Science and Technology
1-5-2, Minatojimaminami-machi,
Chuo-ku, Kobe-shi, Hyogo-ken,
650-0047, JAPAN.
+81-78-599-9511
yamagishi@rist.jp

Yoshimasa Matsumura
Institute of Low Temperature Science,
Hokkaido University
Kita-19, Nishi-8, Kita-ku, Sapporo,
060-0819, JAPAN
+81-11-706-5465
ymatsu@lowtem.hokudai.ac.jp

ABSTRACT
To achieve a detailed simulation of several particles in a numerical ocean simulation, we implemented and optimized a non-hydrostatic ocean model with Lagrangian particle tracking on a GPU. Changes to the particle-tracking algorithm, the use of a texture cache, and exploitation of thread-level parallelism were effective, and the GPU-implemented model was three times faster in terms of execution on a CPU-implemented model. The GPU output returned a good reproduction of the ocean particles' distribution.

1. INTRODUCTION
Lagrangian modeling of particles in the ocean is an effective tool for understanding and reproducing several significant multiphase flows, such as the formation of frazil ice. This method can track several tracers, such as nutrients or isotopes, and helps study ocean physics and dynamics. Matsumura and Ohshima [2] formulated an explicit expression of the particles via the application of Lagrangian particle tracking to a non-hydrostatic ocean model on a CPU and performed detail studies on frazil ice formation. The demand for conducting studies on numerous and various types of particles has been growing, and GPUs are expected to meet this demand because of their massive cores and their architecture, which is highly specialized for massive parallel computing. Despite these characteristics, no previous studies have tackled the simulation of particles in the ocean with a GPU.

This study aims to accelerate the non-hydrostatic ocean model with particle tracking developed by [2] using NVIDIA GPU and PGI CUDA Fortran and demonstrates some experimental cases and techniques that would be applicable to other studies in numerical ocean modeling on GPUs.

For the implementation to a GPU, we changed the particle-tracking algorithm with the idea of sorting all particles at every time step. We also optimized the kernels for ocean dynamics calculation and exploited the efficient thread-level parallelism to reduce the access to the GPU's global memory. We achieved a 3× speedup on the GPU and reproduced the ocean particles' nonlinear behavior.

2. MODEL DESCRIPTION
The core of this numerical ocean model is the three-dimensional non-hydrostatic Navier–Stokes equation in an orthogonal curvilinear coordinate system that simulates ocean dynamics. All equations are discretized on structured grids, and the approximated equation is solved via the conjugate gradient method with a multigrid preconditioner (MGCG). The core equation for ocean dynamics, with the simulation of tracers on Eulerian grids, has been implemented and optimized for GPUs ([3]).

In contrast to the Eulerian specification of the ocean dynamics, all particles are herein explicitly represented in a Lagrangian specification. They are individually advected via a fourth-order Runge–Kutta method via the summation of the Eulerian ocean current velocity.

In the original CPU code, the array of particles is represented using a linked list and the pointer of the list indicates physically close particles. With the linked list, we can achieve sequential access to a group of particles that are close to each other and we can reduce redundant accesses to the velocity array with the help of the CPU cache on updating the location of the particles.

3. IMPLEMENTATION AND OPTIMIZATION ON A GPU
While implementing particle tracking on a GPU, we changed the particle-tracking algorithms and optimized the kernels for the ocean dynamics calculation.

With a linked list, we cannot escape sparse accesses to the main CPU memory due to the irregularity of the particles in the ocean. While implementing particle tracking on the GPU, this array structure in the original CPU code impedes the coalesced access to the GPU global memory. In this study, we adopted the idea of sorting all particles on the global memory at every time step ([1]) and achieved a coalesced access to the global memory. For the sorting, we applied the fast radix sort provided by the Thrust library, which is highly optimized to the NVIDIA GPU. We assigned the arrays of the ocean current velocity field to the texture cache to reduce redundant accesses to the GPU global memory, especially in cases where the accesses are irregular and noncontiguous amongst the threads.

We optimized the kernels for the ocean dynamics calculation. The discretized form of the ocean dynamics equation requires access to values on adjacent grids. Figure 1 shows the pseudo code that denotes the essence of the ocean dynamics kernel. In this case, there is a data dependency between Loops 1 and 2, and Loop 1 needs to be finalized before the launch of Loop 2. In their implementation on a GPU, synchronization between the GPU threads is required between Loops 1 and 2 and the redundant store/load of array A to/from the GPU global memory is inevitable.

```
do j = 1, nj
  do i = 1, ni
    A(i, j) = Func( A(i, j) ) // Loop 1
  enddo
enddo

do j = 1, nj
  do i = 1, ni
    G(i, j) = A(i, j) + A(i-1, j) // Loop 2
  enddo
enddo
```

Figure 1. The pseudo CPU code indicating the essence of the ocean dynamics kernel.

In the GPU implementation, we eliminated the data dependency between Loops 1 and 2 by adding a calculation for the value on the adjacent grid (Figure 2); therefore, we could exploit thread-level parallelism over the entire kernel. Furthermore, we stored the values to the GPU register implementing scalar values to eliminate the synchronization cost and unnecessary accesses to the GPU global memory.

```
i = threadidx%x; j = threadidx%y

A_ij = Func( A(i, j) ); A_im1j = Func( A(i-1, j) )

G(i, j) = A_ij + A_im1j
```

Figure 2. The pseudo GPU implemented and optimized code of the ocean dynamics kernel.

4. EXPERIMENTS AND RESULTS

The GPU implementation was run and evaluated on a workstation with an Intel Core i7 3930K and NVIDIA K20C. In addition, we evaluated the basic performance of a CPU with a single node of a Fujitsu FX10 incorporating a Fujitsu SPARC64 IXfx.

The size of the domain was set to (256, 256, 64), and the number of particles was set to 4194304 ($256 \times 256 \times 64$). The total numbers of time steps were 120 and 21600 for performance comparisons and validation of outputs, respectively. Each simulated time step was a period of 1 s. For the execution on the CPU, we set 16 CPU threads.

Compared to the execution on a Fujitsu SPARC64 IXfx, the GPU-implemented model ran 3 times faster on the NVIDIA K20C, and the MGCG solver, dynamics, and particle tracking are 2.8, 3.2, and 3.1 times faster, respectively (Table 1). The cost of sorting the (256, 256, 64) grids was approximately 7 ms per sorting, which is approximately 5% of a particle-tracking cycle. The cost of sorting was minor and the speed up with the change in the algorithm more than compensated for the increase in the cost. The metrics, which indicate the basic performances, were consistent with the speedup on the GPU (Table 2).

Table 1. Elapsed time (in s) for each component of the CPU/GPU executions.

	CPU	GPU	speed up
All	138.8	45.8	3.0
MGCG solver	44.0	16.0	2.8
Dynamics	47.9	14.9	3.2
Particle tracking	46.9	15.0	3.1

Table 2. Basic performance metrics for the CPU/GPU executions (dp: double precision, sp: single precision).

	CPU	GPU
Computational performance (GFLOPS)	7.2	34.2(dp)/2.3(sp)
GFLOPS/PEAK (%)	3.4	2.9(dp)/0.1(sp)
Memory transfer (GB/S)	27.5	84.5

The outputs from the GPU-implemented model successfully reproduced the distribution of the particles in the ocean, and we could observe a streaky pattern on the surface and a chaotic distribution in the middle layer, which are essential characteristics of the experiments (Figure 3).

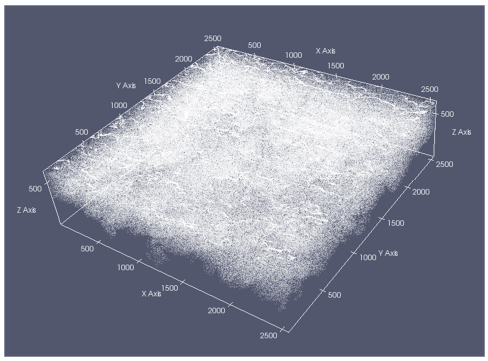

Figure 3. Distribution of the particles after 21600 time steps.

5. DISCUSSION AND CONCLUSIONS

This study demonstrates that GPUs are useful not only for ocean dynamics but also for particle tracking in the ocean, which has not been represented in previous studies. The change in the algorithm and the use of a texture cache can accelerate particle tracking in the ocean. Exploiting thread-level parallelism with the help of an additional calculation and the use of registers is effective for the ocean dynamics kernels. The output from the GPU showed a good reproduction of the ocean particles' distribution.

Numerical ocean models intrinsically show chaotic and nonlinear behavior. Therefore, further performance analyses and validations with various experimental settings are required. Basic metric profiles (Table 2) show that the values of the metrics are less than the theoretical peak performances and that there is still room for improvement on both the CPU and GPU.

6. REFERENCES

[1] Green, S., 2012. Particle simulation using CUDA. *Technical Report, NVIDIA*.

[2] Matsumura, Y. and Ohshima, K. I., 2015. Lagrangian modelling of frazil ice in the ocean. *Annals of Glaciology 56*, 69, 373-382. DOI= http://dx.doi.org/10.3189/2015AoG69A657.

[3] Yamagishi, T. and Matsumura, Y., 2016. GPU Acceleration of a Non-hydrostatic Ocean Model with a Multigrid Poisson/Helmholtz Solver. *Procedia Computer Science 80*, 1658-1669. DOI=http://dx.doi.org/http://dx.doi.org/10.1016/j.procs.2016.05.502.