

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

Takateru Yamagishi(1) Yoshimasa Matsumura(2) Hiroyasu Hasumi(3)

1: Research Organization for Information Science and Technology, yamagishi@rist.jp 2: Atmosphere and Ocean Research Institute, The University of Tokyo, ymatsu@aori.u-tokyo.ac.jp 3: Atmosphere and Ocean Research Institute, The University of Tokyo, hasumi@aori.u-tokyo.ac.jp

Introduction

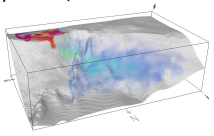
- The objective is to accelerate high-spatial-resolution numerical ocean simulations with multiple GPUs.
- Pascal GPUs and efficient GPUDirect RDMA transfer technology are expected to meet this demand.
- We implemented our “kinaco” ocean model on the latest supercomputer with Pascal GPUs and GPUDirect.
- Flexible optimizations of Poisson/Helmholtz solver are achieved with CUDA Fortran.
- GPU-implemented kinaco runs 2 times faster than CPU and shows good weak scaling.

“Kinaco” non-hydrostatic ocean model

- Kinaco [1] resolves vertical convection and eddy mixing with non-hydrostatic approximation on ~1 km scale.
- Pressure field is solved using MGCG (multigrid preconditioned conjugate gradient method) solver, which can be scaled up to 8k processors.
- Implemented on and optimized for a single NVIDIA Kepler GPU [2].

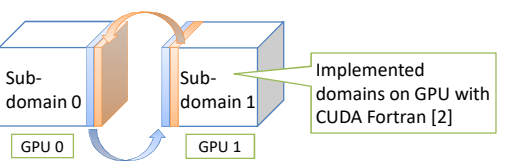
Extended to multiple Pascal GPUs!
New Optimization of P/H solver

Deep convection experiment
in the Southern Weddell Sea



Multi-GPU implementation

Domain decomposition and halo data exchange



Halo region data required by adjacent domain passed to additional layer by CUDA-aware MPI using GPUDirect RDMA transfer

CUDA-aware MPI with GPUDirect RDMA transfer

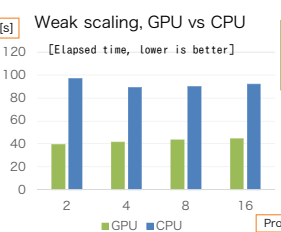
- Data transfer is pipelined and latency is hidden
- Direct transfer between GPU global memories without accessing CPU memories

Specifications of Reedbush and experiments

Hardware	Reedbush-L	SGI Rackable C1102-GP8
CPU	Xeon E5-2695v4, [605 GF, 2.1 GHz, 18 core] × 2	
GPU	Tesla P100 with NVLink, [5.3 TF, 16 GB] × 4	
Interconnect	InfiniBand EDR, 4x[100 Gbps]	
Software	Fortran	PGI 17.1, CUDA Fortran
CUDA-aware MPI	Mvapich 2.2 GDR (GPUDirect)	
CUDA	CUDA 8.0.44	
Application	Kinaco	Matsumura and Hasumi (2008)
	Kinaco-CUDA-MPI	Kinaco + CUDA + MPI (GPUDirect)
Experiments	Setup	Deep convection/Visbeck et al. (1996)
	Grid config.	256 × 256 × 32 cells/processor [weak scaling]
	Time steps	120 sec × 300 cycles = 10 hours

Performance evaluation

Performance comparison for all components



Number of cells:
[256, 256, 32]
× processors

For CPUs, 16
cores/processor
were used

- Twice as fast as GPUs as it did on CPUs.
- Good weak scalability on GPUs (GPUDirect works well)

Performance comparisons for each component [16PEs]

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

2.6 times faster on GPUs

7 times faster on GPUs

GPU data transfer costs comparable to those for CPUs

CG method is 4 times faster on GPUs

Poisson/Helmholtz solver optimization

Memory alignment adjustment

Halo region for MPI communication is prepared:
 $A(-1:N+2, -1:N+2, -1:N+2)$

Access to A always cause memory misalignment. (N is set to 2**m)
32byte memory segment is wasted

Eliminate halo region and Access directly buffer data.

$B(i, j, k) = A(i-1, j, k) + A(i, j, k) + A(i, j-1, k) + \dots$

ASIS code loads halo data from $A(-1:N+2, -1:N+2, -1:N+2)$

Elapsed Time: 215us → 177us

Global Load L2 Transactions $9 \rightarrow 8$ (8 is an ideal value)

haloim1 = recvj(j) or A(i-1, j, k)
halojm1 = recvi(i) or A(i, j-1, k)

$B(i, j, k) = \text{haloim1} + A(i, j, k) + \text{halojm1} + \dots$

TUNED code loads halo data from buffered data (recvi/recvj)

Global load/store efficiency (Kernel)
66.45%/87.18% → 85.38%/100.00%

Thread block adjustment

Due to model's physical nature, Matrix coefficients are vertically uniform in the middle ocean layer.

$kk = \text{indx}(k)$

Reuse by L1 cache

$B(i, j, k) = F(i, j, kk, -3) * C(i, j, k-1) + F(i, j, kk, -2) * C(i, j-1, k) + \dots$

Vertically uniform

Thread block configuration is optimized to reuse F vertically

$(32, 8, 1) < (32, 1, 16)$

Vertically uniform, reuse by L1 cache

Elapsed Time: 323us → 177us

L1 hit rate: 19.79% → 56.44%

Shuffle function to make coarse grids

Multigrid preconditioner converts fine grids into coarse grids.

$F[32, 32, 1] \rightarrow C[16, 16, 1]$

Averaging adjacent values

$C(i, j) = \frac{F(i, j) + F(i+1, j) + F(i, j+1) + F(i+1, j+1)}{4}$

Uncoalesced accesses to Global Memory

Elapsed Time: 14.8us (ASIS) → 14.3us (TUNED)

Comparison of Data Loads by NVVP

	ASIS	TUNED
Global Loads	131K	131K
L2 Global Loads	131K	131K
L1 Global Loads	262K	131K

L1 loads are increased due to stride accesses

L1 access cost (ASIS) VS shuffle function cost (TUNED)

Merging small kernels

Multigrid preconditioner invokes GPU kernels with small grids many times.

Kernels smaller than [8, 8] are 200 times called per 1 time step.

Kernel launch overhead cannot be ignored.

Kernel launch ~20μs

Small kernels (few μs) are launched

Kernel fusion

cudaLaunchKernel is reduced

Merged to 1 kernel

Summary and future work

- We implemented our “kinaco” numerical ocean model on Tokyo University's Reedbush supercomputer, which incorporates the latest NVIDIA P100 GPUs and GPUDirect technology.
- GPU-implemented kinaco runs 2 times faster than CPU and shows good weak scaling.
- Most components are faster than CPUs and especially the velocity diagnosis functions are approximately seven times faster.
- P/H solver optimized by newly introduced four methods with CUDA Fortran contributes for good performance.
- Inter-node data transfers by CUDA-aware MPI library with GPUDirect RDMA show comparable performance to those for CPUs.
- In future work, we will adopt the following optimization techniques for detailed experiments with a large number of cells.
- Tracer components are optimized, brush up codes.
- Overlap kernel computation with inter-node data transfers with MPI

References

[1] Matsumura, Y. and Hasumi, H., "A non-hydrostatic ocean model with a scalable multigrid Poisson solver," Ocean Modelling, 2008. DOI=<http://dx.doi.org/10.1016/j.ocemod.2008.05.001>.

[2] 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/10.1016/j.procs.2016.05.502>.