Geosci. Model Dev. Discuss., 7, 7651–7691, 2014
www.geosci-model-dev-discuss.net/7/7651/2014/
doi:10.5194/gmdd-7-7651-2014
© Author(s) 2014. CC Attribution 3.0 License.

Geoscientific Model Development Discussions
Open Access

gpuPOM: a GPU-based Princeton Ocean Model

S. Xu[1], X. Huang[1], Y. Zhang[1], H. Fu[1], L.-Y. Oey[2,3], F. Xu[1], and G. Yang[1]

[1]Ministry of Education Key Laboratory for Earth System Modeling, Center for Earth System Science, Tsinghua University, 100084, and Joint Center for Global Change Studies, Beijing, 100875, China
[2]Institute of Hydrological & Oceanic Sciences, National Central University, Jhongli, Taiwan
[3]Program in Atmospheric & Oceanic Sciences, Princeton University, Princeton, New Jersey, USA

Received: 13 October 2014 – Accepted: 22 October 2014 – Published: 17 November 2014

Correspondence to: X. Huang (hxm@tsinghua.edu.cn)

Published by Copernicus Publications on behalf of the European Geosciences Union.

Abstract

Rapid advances in the performance of the graphics processing unit (GPU) have made the GPU a compelling solution for a series of scientific applications. However, most existing GPU acceleration works for climate models are doing partial code porting for certain hot spots, and can only achieve limited speedup for the entire model. In this work, we take the mpiPOM (a parallel version of the Princeton Ocean Model) as our starting point, design and implement a GPU-based Princeton Ocean Model. By carefully considering the architectural features of the state-of-the-art GPU devices, we rewrite the full mpiPOM model from the original Fortran version into a new Compute Unified Device Architecture C (CUDA-C) version. We take several accelerating methods to further improve the performance of gpuPOM, including optimizing memory access in a single GPU, overlapping communication and boundary operations among multiple GPUs, and overlapping input/output (I/O) between the hybrid Central Processing Unit (CPU) and the GPU. Our experimental results indicate that the performance of the gpuPOM on a workstation containing 4 GPUs is comparable to a powerful cluster with 408 CPU cores and it reduces the energy consumption by 6.8 times.

1 Introduction

High-resolution atmospheric, oceanic and/or climate modeling remains a significant scientific and engineering challenge because of the enormous computing, communication, and storage requirements. With the rapid development of computer architecture, in particular multi-core and many-core techniques, the computing power that can be applied to scientific problems has increased exponentially in recent decades. Some parallel computing techniques, such as the Message Passing Interface (MPI, Gropp et al., 1999) and Open Multi-Processing (OpenMP, Chapman et al., 2008) have been widely used to support the parallelization of numerous climate models. Moreover, as modern massive supercomputers become more and more heterogeneous because of the increasing number of different accelerating devices such as the GPU, the Intel many integrated core (Intel MIC) and reconfigurable computing based on field programmable gate array (FPGA), new approaches are required to more effectively utilize the emerging novel architecture, communication and input/output (I/O) to achieve order-of-magnitude acceleration required for climate models.

In recent years, a number of scientific codes have been ported to the GPU. Different levels of speedup were achieved for climate models using GPUs. Michalakes and Vachharajani (2008) accelerated a computationally intensive microphysics process of the Weather Research and Forecast (WRF) model with a speedup of nearly 25×; but the entire WRF model is sped up by only 1.23×. Shimokawabe et al. (2010) fully accelerated the ASUCA model – a non-hydrostatic weather model – on 528 Nvidia Tesla GT200 GPUs and achieved a speedup of 80×. Linford et al. (2009) accelerated a computationally intensive chemical kinetics kernel from the WRF model with Chemistry on an Nvidia Tesla C1060 and achieved a speedup of 8×. Leutwyler et al. (2014) accelerated a full huge operational weather forecasting model COSMO and achieved 2.8× speedup for its dynamic core. Carpenter et al. (2013) accelerated the spectral element dynamical core of the Community Earth System Model (CESM) using the GPU by 3×. Govett et al. (2010) ported the dynamics portion of the Non-hydrostatic Icosahedral (NIM) model to the GPU and achieved a speedup of 34×. Zhenya et al. (2010) adopted OpenACC Application Programming Interface (OpenACC API), which used simple compiler directives to accelerate some hot-spot functions, to accelerate the parallel ocean program (POP) by 2.2×.

Most existing GPU acceleration projects for climate models are only working on certain hot spots of the program, leaving a significant part of the program still running on CPUs. Therefore, there are usually frequent data exchange between CPUs and GPUs, which significantly reduces the overall performance.

The objective of our study is to shorten the high computation time of high-resolution ocean models by parallelizing their existing model structures using the GPU. Taking the parallel version of the Princeton Ocean Model (mpiPOM) as an example, we demonstrate how to parallelize an ocean model to make it run efficiently on a GPU architecture. Using the state-of-the-art GPU architecture, we first convert the mpiPOM from its original Fortran version into a new Compute Unified Device Architecture C (CUDA-C) version. CUDA-C is the dominant programming language for GPUs. We call the new version gpuPOM. Then, we design and implement several optimizing methods: (i) computation optimization in a single GPU; (ii) communication optimization among multiple GPUs, and (iii) I/O optimization between a hybrid GPU and CPU.

In terms of computing, we concentrate on memory access optimization and making better use of caches in GPU memory hierarchy. We improve memory usage by using read-only data cache, local memory blocking, loop fusion, function fusion and that disables error-correcting code memory (Error Checking & Correction, ECC memory). The experimental results demonstrate that high memory access optimization can achieve a speedup of approximately 100× when comparing a single GPU against a single CPU core.

In terms of communication, we concentrate on the overlapping between the inner-region computation and the outer-region communication and update. With the GPUDirect communication technology, multiple GPUs in one node can communicate directly and bypass the CPU. In addition, with the fine-grained control of the CUDA streams and its priority, inner-region computation can be executed concurrently with outer-region communication and updating.

In terms of I/O, we choose to split the MPI communicator into computation and I/O processes. One individual computation process and one individual I/O process are attached to one GPU. The computation process is responsible for launching kernels on the GPU and the I/O process is responsible for data copy back from the GPU and to write on disk. The computing process and the I/O process execute concurrently.

To understand the accuracy, performance and scalability of the gpuPOM, we build a customized workstation with four GPU K20X devices inside. The experimental results show that the performance of the gpuPOM running on this workstation is comparable to a powerful cluster with 408 CPU cores.

The paper is organized as follows. In Sect. 2, we review the mpiPOM model. In Sect. 3, we present detailed techniques about computation optimization in a single GPU, communication optimization among multiple GPUs, and I/O optimization between a hybrid GPU and CPU. We provide the corresponding experimental results about correctness, performance and scalability in Sect. 4 and conclude our work in Sect. 5.

2 The mpiPOM

The mpiPOM is a parallel version of the Princeton Ocean Model (POM) that is based on MPI. It retains most of the physics package of the original POM (Blumberg and Mellor, 1983, 1987; Oey et al., 1985a, b, c; Oey and Chen, 1992a, b), but includes also satellite and drifter assimilation schemes from the Princeton Regional Ocean Forecast System (Oey, 2005; Lin et al., 2006; Yin and Oey, 2007), as well as more recently advanced features such as wind-wave induced Stokes drift, wave-enhanced mixing and Localized Ensemble Transform Kalman Filter (Oey et al., 2013; Xu et al., 2013). The POM code was reorganized and MPI was implemented by Jordi and Wang (2012) using a two-dimensional data decomposition of the horizontal domain with a halo of ghost cells. The POM is a powerful ocean model that has been used in a wide range of applications: circulation and mixing processes in rivers, estuaries, shelf and slope, lakes, semi-enclosed seas and open and global oceans. It is also at the core of various real-time ocean and hurricane forecasting systems, for examples: Japan coastal ocean and Kuroshio (Isobe et al., 2012); Adiratic Sea Forecasting System (Zavatarelli and Pinardi, 2003); the Mediterranean Sea forecasting system (Korres et al., 2007); the GFDL Hurricane Prediction system (Kurihara et al., 1995, 1998), the US' Hurricane Forecasting System (Gopalakrishnan et al., 2010, 2011) and the Advanced Taiwan Ocean Prediction system (Oey et al., 2013). Additionally, the model has been used to study various geophysical fluid dynamical processes (e.g. Allen and Newberger, 1997; Newberger and Allen, 2007a, b; Kagimoto and Yamagata, 1997; Guo et al., 2006; Oey et al., 2003; Zavatarelli and Mellor, 1995; Ezer and Mellor, 1992; Oey, 2005; Xu and

Oey, 2011. For a more complete list please visit the POM website (http://www.ccpo.odu.edu/POMWEB).

The mpiPOM experiment that is used in this paper is one of the two designed and tested by Professor Oey and students; the codes and results are freely available at the FTP site (ftp://profs.princeton.edu/leo/mpipom/atop/tests/). The reader can see Chapter 3 of the Lecture Notes (Oey, 2014) for more detail. The test case is a dam-break problem in which warm and cold waters are initially separated in the middle of a zonally periodic channel 200 km × 50 km × 50 m on an f-plane, with walls at the northern and southern boundaries. Geostrophic adjustment then ensues and baroclinic instability waves amplify and develop into finite-amplitude eddies in 10 ~ 20 days. The horizontal grid sizes are 1 km and there are 50 vertical sigma levels. Although the problem is a test case, the code is the full mpiPOM version that is used in the ATOP forecasting system.

The model solves the primitive equation under hydrostatic and boussinesq approximations. In the horizontal, spatial derivatives are computed either using centered-space differencing or Smolarkiewicz's positive definite advection transport algorithm (Smolarkiewicz, 1984) on a staggered Arakawa C-grid; both schemes have been tested, but the latter is reported here. In the vertical, the mpiPOM supports terrain-following sigma coordinates and a fourth-order scheme option to reduce the internal pressure-gradient errors (Berntsen and Oey, 2010). The mpiPOM uses the time-splitting technique to separate the vertically integrated equations (external mode) from the vertical structure equations (internal mode). The external mode calculation is responsible for updating surface elevation and the vertically averaged velocities. The internal mode calculation results in updates for velocity, temperature and salinity, as well as the turbulence quantities. The three-dimensional internal mode and the two-dimensional external mode are both integrated explicitly using a second-order leapfrog scheme. These two modules are the most computationally intensive kernels of the mpiPOM model.

3 Full GPU acceleration of the mpiPOM

The flowchart of the gpuPOM is illustrated in Fig. 1. The main difference between mpiPOM and gpuPOM is that the CPU in gpuPOM is only responsible for the initializing work and the outputting work. The gpuPOM begins with initializing the relevant arrays on the CPU host and then copies data from the CPU host to the GPU. The GPU does all the computations, including the external mode, the internal mode, and their interactions. In the 2-D external mode loop, the depth-averaged velocity UA, VA and sea surface height are calculated. In the 3-D internal model loop, the fields such as velocities (U,V), temperature (T), salinity (S), and various turbulence variables are time-stepped forward. Outputs such as velocity and sea surface height, are copied back to the CPU host and then written to disk at a user-specified time interval.

In the following sections, we introduce the general optimizations of the gpuPOM in a single GPU and the special optimizations of the gpuPOM according to state-of-the-art GPU architecture. Then, we present the design of communications for various processes and multiple GPUs within a node instead of using regular MPI functions. Finally, we describe the design of I/O overlapping for hybrid CPU and GPU architecture.

3.1 Computational optimizations in a single GPU

For current computers, GPU device can be connected to a host through a high-speed PCI-express interface. The Nvidia GPU has a number of multiprocessors which execute in parallel, and has its own device memory up to several gigabytes. The code is executed in groups of 32 threads, what Nvidia calls a warp.

In our implementation, the 3-D arrays of variables are stored sequentially in the order of x, y, z and the 2-D arrays are stored in the order of x, y, which is the same as the original code. The vertical diffusion is solved by the tridiagonal solver (the Thomas Algorithm) which is calculated sequentially in the z direction. For the sake of simplicity, the grid is divided along x and y directions (2-D block decomposition) in all kernel functions. Each GPU thread specifies a (x,y) point in the horizontal direction and performs all the calculations from surface to bottom. The thread blocks are divided as (32, 4). In the x direction, the block number should be a multiple of 32 threads to perform coalesced memory access within a warp. In the y direction, we tested many thread numbers, such as 4 and 8, and obtained similar performances. We finally choose 4 because we attempt to obtain more blocks to distribute the workload among stream multiprocessors (SM) more uniformly, and also to obtain enough occupancy (Volkov, 2010). Occupancy is the percentage of threads active per multiprocessor.

Because the high-resolution mpiPOM is memory intensive, the importance of efficiently using GPU memory cannot be overstated. The memory hierarchy of Nvidia Tesla K20X GPU is illustrated in Fig. 2. In the current K20X GPU, each SM owns 64K 32 bit registers; these registers are the fastest memory in the GPU memory hierarchy. At the same time, the shared memory and the L1 cache share a 64 KB on-chip fast memory and can be configured with artificial options such as 16/48 KB, 32/32 KB or 48/16 KB. A 48 KB read-only data cache can be directly accessed and is newly designed in each SM and L2 cache with 1.5 MB size that is shared by all SMs.

Managing the significant performance difference between off-chip and on-chip memory is the primary concern of a GPU programmer. As shown on the right side of Fig. 2, we propose five key optimizations to fully utilize the faster on-chip memory of the GPU and describe the relationships between the GPU memory hierarchy and each optimization in the following.

3.1.1 Read-only data cache utilization

Effective use of the new 48 KB directly-access and read-only data cache in the K20X GPU can improve the performance of memory intensive kernels. This feature will be automatically enabled and utilized as long as certain conditions are met. We add "const __restrict__" qualifiers to the parameter pointers in gpuPOM to explicitly allocate the read-only data cache for our program. The "LDG.E" instruction will then appear in the disassembling code, and Nvidia Visual Profiler(NVVP) software will show that the read-only data cache is actually being utilized.

As an example, consider the calculations of advection and the horizontal diffusion terms. Because mpiPOM adopts the Arakawa C-grid, the update of $T(i,j,k)$ requires the value of $u(i,j,k)$, $u(i+1,j,k)$, $v(i,j,k)$ and $v(i,j+1,k)$, in addition to the value of horizontal kinematic viscosity, aam, from four neighboring grid points. In one time step, the arrays of u and v must be accessed twice, and the aam array must be accessed four times. Therefore it is natural to use the read-only data cache to improve the data locality of gpuPOM. This optimization improves the performance of this part by 18.8 %.

3.1.2 Local memory blocking

Cache blocking is a common method to improve data reuse in parallel computing. In this method, a small subset of a dataset is loaded into the on-chip faster memory (e.g., the L1/L2 cache in the GPU and the CPU) and then the small data block is repeatedly accessed by the program. It is helpful to reduce the need to access the off-chip with high latency memory (e.g., global memory on the GPU). Because regular global memory access cannot be cached in L1 cache for K20X GPU, the method used here is to pull the data from local memory to the L1 cache.

For the subroutines about vertical diffusion and source/sink terms, the chasing method is used to solve a tridiagonal matrix along the vertical direction for each grid point individually. As shown in Algorithm 1, the 3-D temporary arrays in the original code, such as ee, gg, that store row transformation coefficients are streamed from memory. However, these arrays are too large to reside in the cache entirely; code efficiency is therefore decreased. We find that each thread performs a column calculation from surface to bottom and there is no communication. Thus, we declare 1-D arrays ee_new, gg_new in local memory to replace the original 3-D global arrays. Their size is equal to the level of ocean, $nz-1$, which is typically a very small value.

In the chasing method, these local arrays are accessed twice within one thread, one from $k=0$ to $k=nz-1$ and another from $k=nz-1$ to $k=0$. After blocking the vertical direction arrays in local memory, L1 cache is fully utilized although some of them may be spilled to global memory. The performance of the subroutines about vertical diffusion and source/sink terms is improved by 35.3 % when using the local memory blocking technique.

Algorithm 1 A simple example of local memory blocking.

```
/********************
* Origin CUDA-C code
********************/
//ee, gg are parameter pointers of the function
//that represent the use of global memory
for (k = 1; k < nz-2; k++){
  ee[k][j][i] = ee[k-1][j][i]*A[k][j][i];
  gg[k][j][i] = ee[k-1][j][i]*gg[k-1][j][i]-B[k][j][i];
}
for (k = nz-3; k >= 0; k++){
  uf[k][j][i] = (ee[k][j][i]*uf[k+1][j][i]+gg[k])*C[k][j][i];
}
/*******************
* After local memory blocking
*******************/
//ee_new, gg_new are 1-D array declared in function
//that represent the use of local memory
for (k = 1; k < kbm1; k++){
  ee_new[k] = ee_new[k-1]*A[k][j][i]
  gg_new[k] = ee_new[k-1]*gg_new[k-1]-B[k][j][i];
}
for (k = nz-3; k >= 0; k++){
uf[k][j][i] = (ee_new[k]*uf[k+1][j][i]+gg_new[k])*C[k][j][i];
}
```

3.1.3 Loop fusion

Loop fusion is an effective method to store scalar variables in registers for data reuse. Registers are the fastest memory in the GPU memory hierarchy. For example, as shown in Algorithm 2, if the variable drhox(k, j, i) must be read several times in multiple loops, we can fuse these loops into one. Therefore, the drhox(k, j, i) will be read from the global memory the first time and then repeatedly read from a register. This method can also be applied in a number of the mpiPOM subroutines.

Take the kernel profq as an example. After rewriting part of source code with loop fusion, the device memory transactions decrease by 57 %, while the registers used per thread increase from 46 to 72, as reported in NVVP. Although the occupancy achieved decrease from 61.1 to 42.7 %, the performance of this kernel is improved by 28.6 %.

3.1.4 Function fusion

Because we can fuse the loops in which the same arrays are accessed, we can also fuse functions in which similar formulas are calculated and the same arrays are accessed. For example, the advv and advu functions of the mpiPOM calculate advection in longitude and latitude, respectively, and they can be fused into one subroutine. This optimization benefits from the elimination of the redundancy calculations.

This optimization is also useful for the situation in which one function is called several times to calculate different tracers. For example, the proft functions of the mpiPOM is called twice – once for temperature and once for salinity. Their computing formulas are similar and certain common arrays are accessed; these functions were modified to calculate temperature and salinity simultaneously. The method of Function fusion improves the performance of these functions by 28.8 %.

Algorithm 2 A simple example of loop fusion.

```
/*******************
* Origin cuda-c code
*******************/
for (k = 1; k < kbm1; k++){
  drhox[k][j][i] = drhox[k-1][j][i] + A[k][j][i];
}
for (k = 0; k < kbm1; k++){
  drhox[k][j][i] = drhox[k][j][i] * B[k][j][i];
}
/******************
* After loop fusion
******************/
for (k = 1; k < kbm1; k++){
  drhox[k][j][i] = drhox[k-1][j][i] + A[k][j][i];
  drhox[k-1][j][i] = drhox[k-1][j][i] * B[k][j][i];
}
drhox[0][j][i] = drhox[0][j][i] + B[k][j][i];
```

3.1.5 ECC-off and GPU boost

Because ECC memory consumes some amount of memory bandwidth, we can improve the GPU global memory bandwidth by disabling the error checking and memory correcting features. Also, the memory bandwidth that can be achieved is improved by enhancing the clock of SM core. In our implementation, we overclock the default clock of K20X GPU from 732 to 784 MHz. The methods of ECC-off and GPU boost improves the performance of the whole application by 13.8 %.

We divide all the gpuPOM subroutines into different categories based on their different computation patterns. As shown in Table 1, in gpuPOM, we deploy different optimizations in different categories to achieve improved performance; these categories are now described.

1. Category 1: advection and horizontal diffusion(adv)

   This category has 6 subroutines, and calculates the advection and horizontal diffusion and in the case of velocity, the pressure gradient and Coriolis terms. Here it is possible to reuse data among adjacent threads, and the subroutines therefore benefit from using read-only data cache and shared-memory. Also, the variables are calculated in different loops of one function or in different functions, so the loop fusion and function fusion optimizations apply to this part.

2. Category 2: vertical diffusion(ver)

   This category has 4 subroutines, and calculates the vertical diffusion. In this part, chasing method is used in the tridiagonal solver in the k-direction. The main feature is that data is reused twice within one thread, while data is accessed once from $k = 0$ to $k = nz - 1$ and once from $k = nz - 1$ to $k = 0$. The subroutines are significantly sped up after grouping the k-direction variable in local memories.

3. Category 3: vorticity(vort), baroclinic(baro), continuity equation(cont) and equation of state(state)

   This category is less time consuming than the two categories above, but it also benefits from our optimizations. Because there exists data reuse with adjacent threads, the use of read-only data cache improves efficiency. For vort, there is data reuse within one thread, and loop fusion improves the efficiency.

3.2 Communication optimizations among multiple GPUs

In this section, we present the optimizing strategies used to harness the computing power of multiple GPUs. With multiple GPUs, the computing domain is divided into smaller blocks than with a single GPU. The performance of GPU computing is faster and the memory requirement for each GPU is reduced. To utilize multiple GPUs, an effective domain decomposition method and communication method should be employed. We split the domain along the x and y directions (2-D decomposition) and assign each MPI process for one subdomain, following Jordi and Wang (2012). Then, we attach the MPI process to one GPU and send messages from one GPU to another. Shimokawabe et al. (2010) and Yang et al. (2013) proposed some fine-grained overlapping methods of GPU computation and CPU communication to improve the simulation performance. An important common issue is that the communications between multiple GPUs explicitly require the participation of the CPU. In our work, we hope to implement the communication to bypass the CPU to fully employ the capability of the GPU.

State-of-the-art MPI libraries, such as OpenMPI and MVAPICH2, have announced their support for MPI communication directly from GPU memory, which is known as CUDA-aware MPI. We tried MVAPICH2 to implement direct communication among multiple GPUs at first. However, we found that the boundary operation and MPI communication occupied nearly 15 % of the total runtime after GPU porting.

To fully overlap the boundary operations and MPI communications with computation, we adopt the data decomposition method shown in Fig. 3. The data region is decomposed into three parts: the inner part, the outer part, and the halo part. The outer part includes east/west/north/south part, and the halo part also includes east/west/north/south halos to exchange data with neighbors. In CUDA, a stream is a sequence of commands that execute in order; different streams can also execute concurrently with different priorities. In our design, the inner part, which is the most time-consuming part with the largest workload is allocated to stream 1 in which to execute. The east/west outer part is allocated to stream 2 and the north/south outer part is allocated to stream 3. In the east/west outer part, the width is set to 32 to ensure coalesced memory access in a warp to improve performance. The halo part is also allocated to stream 2.

The workflow of multiple streams on the GPU is shown in Fig. 4. The east/west/north/south parts are normal kernel functions that can run in parallel with the inner part through different streams. The communication operations are implemented by cudaMemcpyAsync, which is an asynchronous CUDA memory copy function. The corresponding synchronization operation between the CPU and the GPU or among MPI processes are implemented with cudaStreamSynchronize function and MPI_barrier function. To hide the subsequent communication by the inner part, stream 2 and stream 3 for the outer part have higher priority to preempt the computing resource from stream 1 at any time.

Current CUDA-aware MPI implementation such as MVAPICH2 is not suitable for the "Comm." part in Fig. 3. We found the two-sided MPI functions MPI_Send and MPI_Recv will block the current stream so that the concurrency pipeline is broken. The probable cause is synchronous cudaMemcpy function is called in the current implementation of MPI_Send and MPI_Recv, according to Potluri et al. (2012). Moreover, the implementation of non-contiguous MPI datatype for communication between GPUs is not efficient enough for the gpuPOM. The computation time of many kernels is about a few hundred microseconds to a few milliseconds while MPI latency for our message size is about the same, which means the outer part update and communication can not be fully overlapped.

From CUDA 4.1, the Inter-Process Communication (IPC) feature has been introduced to facilitate direct data copy among multiple GPU buffers that are allocated by different processes. The IPC is implemented by creating and exchanging memory handles among processes and obtaining the device buffer pointers of others. This feature has been utilised in CUDA-aware MPI libraries to optimise communications within a node. Therefore, we decided to implement the communication among multiple GPUs by calling the low-level IPC functions and asynchronous CUDA memory copy functions directly, instead of using high-level CUDA-aware MPI functions. Our communication optimizations among multiple GPUs are mainly implemented with the following two optimizations.

First, we put the phases of creating, exchanging and opening relevant memory handles into the initialization phase of the gpuPOM, which is executed only once. This method can remove the overhead of IPC memory handle operations during each MPI communication operation. The cudaMemcpyAsync function with the corresponding device buffer pointers of neighbor processes replaces the original MPI functions.

Second, we take full consideration of the architecture of our platform in which 4 GPUs are connected with two I/O Hubs (IOHs). As illustrated in Fig. 5, there are two Intel SandyBridge CPUs that connect two GPUs. Both the CPUs are themselves connected through Intel QuickPath Interconnect (QPI). Notation ① means that the communications between GPUs are connected with the same IOH support Peer-to-Peer (P2P) access. Notation ② represents the communications in which P2P access is not supported. If MPI_Rank 0 (context on GPU-0) sends data to MPI_Rank 2 (context on GPU-2), rank 0 must switch its context to GPU-2 temporally and opens the corresponding memory handles to obtain the device buffer pointers of rank 2. For those GPUs that do not support P2P access between one another, we must switch context to the same GPU before opening the corresponding memory handles. We then call regular cudaMemcpyAsync functions to fulfill data communications. For communications between GPUs on the same IOH, the switching context is not necessary. Although the function cudaMemcpyAsync is used in the communication of both ① and ②, the NVVP software shows that ① does a device-to-device memory copy that bypasses the CPU, whereas ② does a device-to-host and a host-to-device memory copy that involves the CPU. The 2-D decomposition introduced in Fig. 5 is an example to demonstrate our design can easily extend to 8 or more GPUs within one node.

3.3 I/O optimizations between hybrid GPU and CPU

The time consumed for I/O in the original mpiPOM is not significant because the output frequency is relatively low. However, after we fully accelerate the model by GPU, the I/O overhead, which is approximately 30 % of the total runtime, cannot be ignored. As described in Sect. 3.2, each MPI process sets its context on one GPU and is responsible for launching kernel functions on this GPU, and the CPU is used to collect and output data. In fact, in most climate models, including the mpiPOM, the computing phase and I/O phase run alternately. In a sense, the computing phase and the I/O phase are serial, which means that the GPU will remain idle until the CPU finishes I/O operations. Huang et al. (2014) designed a fast I/O library for climate models and provided automatic overlapping of I/O and computing. Motivated by their work, we design a method so that computations on GPU and I/O operations on CPU can run in parallel.

Because MPI processes are blocked during the output phase and cannot launch kernels to GPUs, we choose to launch more MPI processes. We divide all the MPI processes into computing processes and I/O processes with different MPI communicators. The computing processes are responsible for launching kernel functions as usual, and the I/O processes are responsible for output. One I/O process attaches to one computing process and these two processes set their contexts on one single GPU through cudaSetDevice function. The total number of MPI processes are twice the size they were before.

Since the I/O processes must fetch data from the GPU, where the data are allocated by the computing processes, communication is necessary between them. Here, we again utilize the feature of CUDA IPC, as introduced in Sect. 3.2. Through CUDA IPC, the I/O processes obtain the device buffer pointers from the computing processes during the initialization phase. When there is a need to output data, the computing processes are blocked and kept idle for a short time while waiting for I/O processes to fetch data. Then, the computing processes continue their computation, and the I/O processes complete their output in the background, as illustrated in Fig. 6.

The advantage of this method is that it overlaps the I/O on the CPU with computation on the GPU. In the serial I/O, the computing processes are blocked while data are brought to the host and written to disk. In the overlapping I/O, the computing processes wait for the data to be brought to the host. In addition, the bandwidth of data brought to the host through the PCI-express bus is approximately 6 GBps, but the output bandwidth is approximately 100 MBps, as determined by the disk. Thus, the overlapping method significantly accelerates the entire application.

4 Experiments

In this section, we first describe the specification of our platform and comparison methods used to validate the correctness of the gpuPOM. Furthermore, we present the performance and scalability of the gpuPOM on the GPU platform in comparison with the mpiPOM on the CPU platform.

4.1 Platform setup

The GPU platform used in our experiments is a super workstation computer consisting of two CPUs and 4 GPUs, as illustrated in Fig. 5. The CPUs are 2.6 GHz 8-core Intel E5-2670 (architecture code-named SandyBridge), which can turbo to 3.0 GHz when all 8 cores are utilized. The peak single-precision performance of the Intel SandyBridge CPU is 384 GFlops and the peak memory bandwidth is 51.2 GBps. The GPUs are Nvidia Telsa K20X, equipped with 2688 GPU-cores and 6 GB GDDR5 fast on-board memory. The peak single-precision performance of K20X GPU is 3.95 TFlops and the peak memory bandwidth is 250 GBps. Therefore, the aggregated performance provided with 4 GPUs can reach 16 TFlops and 1 TBps memory bandwidth, which is sufficient to execute the general simulation research for regional ocean models thus far. The operating system is RedHat Enterprise Linux 6.3 x86_64. The programs on this platform are complied with Intel compiler v14.0.1, Intel MPI Library v4.1.3 and CUDA 5.5 Toolkit.

For the purposes of comparison, the CPU platform used in our experiments is the Tansuo100 supercomputer at Tsinghua University, which consists of 740 nodes, each of which has two 2.93 GHz 6-core Intel Xeon X5670 processors and 32 GB memory. The nodes are connected through an Infiniband network, which provides a maximum bandwidth of 40 Gbps. The node operating system is RedHat Enterprise Linux 5.5 x86_64. All the programs on this platform are compiled with Intel compiler v11.1, and the MPI environment is Intel MPI v4.0.2. The Original mpiPOM code is benchmarked with its initial compiler flags(i.e., -O3 -precise) and also with the same Intel compiler. We also use the GPUDirect technology within MVAPICH2 v1.9 to test the communication effects among multiple GPUs, and compare the results with our implementation.

4.2 The test case and the verification of accuracy

The "dam break" simulation (Oey, 2014) is conducted to verify the correctness and test the performance and the scalability of the gpuPOM. It is a baroclinic instability problem which simulates flows produced by horizontal temperature gradients. The model domain is configured as a straight channel with uniform depth of 50 m. Periodic boundary conditions are used in the east–west direction, and the channel is closed in the north and south. Its horizontal resolution is 1 km × 1 km. To test large computational grid, the domain size of this test case is increased to 962 × 722 horizontal grid points and 51 vertical sigma levels, which is limited by the capacity of on-board memory. Initially, temperature in the southern half of the channel is 15 and 25 °C in the northern half. The salinity is fixed at 35 psu. The fluid is then allowed to adjust. In the first 3–5 days, geostrophic adjustments occurs. Then unstable wave develops due to baroclinic instability. Eventually, eddies are generated. Figure 7 shows the sea-surface height (SSH), sea-surface temperature (SST), and currents after 39 days. The development of a gravity wave is manifest. Noticeably, the gravity wave is confined in the middle of the channel by Rossby radius deformation.

To verify accuracy, we check the binary output files output from the original mpiPOM and the gpuPOM. This testing method is also used in the GPU-porting of ROMs (Mak et al., 2011). As introduced in Whitehead and Fit-Florea (2011), the same inputs will give identical results for individual IEEE-754 operations except in a few special cases. These cases can be classified into three categories: different operations orders, different instructions and different implementations of math libraries. For the first in our study, the parallelization of the mpiPOM does not change the order of each floating point operation and we benefit from this. For the second case in our study, the GPUs have fused multiply-add (FMA) instruction while the CPU does not in our CPU platform. Because this instruction might cause a difference in the numerical results, we disable FMA instructions with the "-fmad=false" compiler flag for the GPUs. For the third case in our study, the value of exponent used in the GPU has a maximum of 2 rounding errors NVIDIA (2014). Fortunately, in the execution path of our dam break simulation, the power of the exponent functions remains unchanged over the entire simulation. Therefore, we accomplish this function on the CPU during the initialization phase and copy the results to the GPU for later data reuse. The experimental results demonstrate that the output variables regarding velocity, temperature, salinity and sea surface height are identical.

4.3 Performance

To understand the advantages of the optimizing methods introduced in Sect. 3, we test the dam break case with different experiments. The current dam break case uses single-precision format. The metrics of seconds per simulation day, which is the walltime it requires to obtain 24 h in the simulation, is measured and used to compare the performance.

In the first experiment, we compare the gpuPOM with the mpiPOM on different hardware platforms, including K20X GPU, the Intel Westmere 6-cores X5670 CPU and the Intel SandyBridge E5-2670 CPU. Figure 8 shows that one K20X GPU can compete with approximately 55 Intel SandyBridge CPU cores or 95 Intel Westmere CPU cores. Obtaining such a speedup on a pure CPU platform is reasonable. Taking the Sandybridge CPU platform as an example, the theoretical memory bandwidth of one 8-core E5-2670 CPU is 51.2 GBps, and the peak single-precision floating point performance is 384 GFlops with all 8 cores turbo to 3.0 GHz. However, for K20X GPU, the memory bandwidth and peak single-precision floating point performance are 250 GBps and 3.95 TFlops, respectively. The approximate ratio of memory bandwidth is 1 : 5 and the ratio of floating points performance is 1 : 10. Therefore, if an application is strictly memory intensive, one K20X GPU can compete with 5 CPUs (approximately 40 Sandy-Bridge CPU cores). In addition, if an application is strictly computing bound, it can compete with 10 CPUs (approximately 80 Sandybridge CPU cores). Our results are more than 50× because the mpiPOM is mostly memory intensive and we have performed several memory optimizations to improve the data locality.

The performance API tool (PAPI) shows that the performance of the gpuPOM on single K20X is 107.3 Gflops in single-precision for the 962 × 722 × 51 grid size. The low performance in Gflops reflects the memory-bound problem in climate models. Previous work such as time skewing (McCalpin and Wonnacott, 1999; Wonnacott, 2000) can make a stencil computation compute bound by making use of data locality between different time-steps. However, for real-world climate models including mpiPOM, the code is usually tens to hundreds of thousands lines and analyzing the dependency manually is tough. Designing an automated tool to further analyze and optimize the mpiPOM and the gpuPOM is a part of our future work.

In the second experiment, we test the communication overlapping method used in the gpuPOM and compare it with the MVAPICH2. In the current MVAPICH2, the communication and boundary operations are not overlapping with computing. Figure 9 shows the weak scaling performance of the gpuPOM on multiple GPUs. To maximize performance, the grid size for each GPU is set to 962 × 722 × 51. When using 4 GPUs with the implementation of MVAPICH2, 18 % of the total runtime is consumed in executing the communication and boundary operations. This overhead does not exist in our communication overlapping method. Figure 9 shows that it spends almost the same time when using different GPUs because the communication and boundary operations are almost fully overlapped with the inner part of the computation.

In the third experiment, we test the efficiency of the gpuPOM on multiple GPUs. Table 2 shows the strong scaling result of the gpuPOM on multiple GPUs. We fix the global grid size at 962 × 722 × 51 and increase the amount of GPUs gradually. The results show that the strong scaling efficiency is 99 % on 2 GPUs and 92 % on 4 GPUs.

A smaller subdomain will decrease the performance of the gpuPOM in two aspects. First, communication time can easily exceed the computation time in the inner part and cannot be overlapped. As the subdomain size decreases, the inner part computation time decreases, but the communication time will not decrease because latency is the dominant factor. Second, the latency of kernel launching and overhead of implicit synchronization after kernel execution will not decrease. There are a series of small kernels in the gpuPOM, and the execution time is close to launching latency and synchronization overhead. When the subdomain size decreases, the impact of these delays expands.

In the fourth experiment, we test the performance of the I/O overlapping method and compare it with the default parallel NetCDF (PnetCDF) method and NO-I/O method. NO-I/O means that all I/O operations are disabled in the program and the time measured is the pure computing time. We simulated the experiment for 20 days and output the history files daily in the netCDF format. The variables included in the output netCDF files are 2-dimensional arrays of size 722×482 and 3-dimensional arrays of size $722 \times 482 \times 51$. The final history files are approximately 12 GB. Figure 10 shows that the I/O overlapping method outperforms the default PnetCDF method. For 1 GPU and 2 GPUs, the overall runtime decreases from 1694/1142 to 1239/688 s, which is close to the NO-I/O method. The small difference between our design and NO-I/O is that the computing processes must be blocked until I/O processes bring data from the GPU. For the case of 4 GPUs, the output time is longer than computational time because the latter is fast and the I/O time is relatively large such that the I/O phase cannot fully overlap with the computing phase. The overall runtime equals the sum of the computation time and the non-overlapped I/O time.

In the last experiment, we test different workloads with the gpuPOM and compare the results with the mpiPOM on Tansuo100 platform. The available global grid size are choosen from the three different high-resolution sets (Grid-1: $962 \times 722 \times 51$, Grid-2: $1922 \times 722 \times 51$, Grid-3: $1922 \times 1442 \times 51$). Figure 11 shows that our workstation with 4 GPUs is comparable to a powerful cluster with 408 CPU cores (34 nodes × 12 cores/node) for the simulation of mpiPOM. Since the Thermal Design Power(TDP) of one X5670 CPU(6-cores) is 95 W and that of one K20X GPU is 235 W, it means using 4 GPUs brings 6.8 times less energy consumption compared with 408 CPU cores. Small subdomains will decrease the performance of the gpuPOM as discussed in the strong scaling test, but it may greatly benefit the mpiPOM on the CPU. The last level cache of one SandyBridge CPU in our platform is 20 MB, whereas that of K20X GPU is only 1.5 MB. As the subdomain size for each MPI process decreases, the cache hit ratio will increase on a pure CPU platform, which can surely improve the performance especially for the memory-bound problem. However, for the simulation on 408 CPU cores, the MPI communication time may occupy more than 40 % of total execution time. With the number of cores increasing to over 450, the execution time may increase instead, as shown in Fig. 11. As a result, our GPU solution has an overwhelming advantage compared to the CPU because the communication overhead is less expensive and overlapped.

5 Conclusions

In this paper, we provide a full GPU accelerated solution of POM. Unlike partial GPU porting, such as WRF and ROMs, the gpuPOM does all the computations on the GPU. The main contribution of our work includes a better use of state-of-the-art GPU architecture, particularly regarding the memory subsystem, a new design of a communication and boundary operations overlapping approach and a new design of an I/O overlapping approach. With the workstation with 4 GPUs, we achieve over 400× speedup against a single CPU core, and provide equivalent performance to a powerful CPU cluster with 34 nodes and reduce the energy consumption by 6.8 times. This work provides cost-effective and efficient ways in ocean modeling.

The Supplement related to this article is available online at doi:10.5194/gmdd-7-7651-2014-supplement.

Acknowledgements. This work is supported in part by a grant from the Natural Science Foundation of China(41375102), the National Grand Fundamental Research 973 Program of China (No. 2014CB347800), and the National High Technology Development Program of China (2011AA01A203).

References

Allen, J. and Newberger, P.: Downwelling circulation on the Oregon continental shelf. Part I: response to idealized forcing, Oceanographic Literature Review, 44, 2011–2035, doi:10.1175/1520-0485(1996)026<2011:DCOTOC>2.0.CO;2, 1997. 7655

Berntsen, J. and Oey, L.-Y.: Estimation of the internal pressure gradient in σ-coordinate ocean models: comparison of second-, fourth-, and sixth-order schemes, Ocean Dynam., 60, 317–330, 2010. 7656

Blumberg, A. F. and Mellor, G. L.: Diagnostic and prognostic numerical circulation studies of the South Atlantic Bight, J. Geophys. Res.-Oceans, 88, 4579–4592, 1983. 7655

Blumberg, A. F. and Mellor, G. L.: A description of a three-dimensional coastal ocean circulation model, Coast. Est. S., 4, 1–16, 1987. 7655

Carpenter, I., Archibald, R., Evans, K. J., Larkin, J., Micikevicius, P., Norman, M., Rosinski, J., Schwarzmeier, J., and Taylor, M. A.: Progress towards accelerating HOMME on hybrid multi-core systems, Int. J. High Perform. C., 27, 335–347, 2013. 7653

Chapman, B., Jost, G., and Van Der Pas, R.: Using OpenMP: Portable Shared Memory Parallel Programming, vol. 10, The MIT Press, 2008. 7652

Ezer, T. and Mellor, G. L.: A numerical study of the variability and the separation of the Gulf Stream, induced by surface atmospheric forcing and lateral boundary flows, J. Phys. Oceanogr., 22, 660–682, 1992. 7655

Gopalakrishnan, S., Liu, Q., Marchok, T., Sheinin, D., Surgi, N., Tuleya, R., Yablonsky, R., and Zhang, X.: Hurricane Weather Research and Forecasting (HWRF) model scientific documentation, L Bernardet Ed, 75, 2010. 7655

Gopalakrishnan, S., Liu, Q., Marchok, T., Sheinin, D., Surgi, N., Tong, M., Tallapragada, V., Tuleya, R., Yablonsky, R., and Zhang, X.: Hurricane Weather Research and Forecasting (HWRF) model: 2011 scientific documentation, L. Bernardet, Ed, 2011. 7655

Govett, M., Middlecoff, J., and Henderson, T.: Running the NIM next-generation weather model on GPUs, in: Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on, 792–796, IEEE, 2010. 7653

Gropp, W. D., Lusk, E. L., and Thakur, R.: Using MPI-2: Advanced Features of the Message-Passing Interface, vol. 2, Globe Pequot, 1999. 7652

Guo, X., Miyazawa, Y., and Yamagata, T.: The Kuroshio onshore intrusion along the shelf break of the East China Sea: the origin of the Tsushima warm current, J. Phys. Oceanogr., 36, 2205–2231, 2006. 7655

Huang, X. M., Wang, W. C., Fu, H. H., Yang, G. W., Wang, B., and Zhang, C.: A fast input/output library for high-resolution climate models, Geosci. Model Dev., 7, 93–103, doi:10.5194/gmd-7-93-2014, 2014. 7667

Isobe, A., Kako, S., Guo, X., and Takeoka, H.: Ensemble numerical forecasts of the sporadic Kuroshio water intrusion (kyucho) into shelf and coastal waters, Ocean Dynam., 62, 633–644, 2012. 7655

Jordi, A. and Wang, D.-P.: sbPOM: A parallel implementation of Princenton Ocean Model, Environ. Modell. Softw., 38, 59–61, 2012. 7655, 7664

Kagimoto, T. and Yamagata, T.: Seasonal transport variations of the Kuroshio: an OGCM simulation, J. Phys. Oceanogr., 27, 403–418, 1997. 7655

Korres, G., Hoteit, I., and Triantafyllou, G.: Data assimilation into a Princeton Ocean Model of the Mediterranean Sea using advanced Kalman filters, J. Marine Syst., 65, 84–104, 2007. 7655

Kurihara, Y., Bender, M. A., Tuleya, R. E., and Ross, R. J.: Improvements in the GFDL hurricane prediction system, Mon. Weather Rev., 123, 2791–2801, 1995. 7655

Kurihara, Y., Tuleya, R. E., and Bender, M. A.: The GFDL hurricane prediction system and its performance in the 1995 hurricane season, Mon. Weather Rev., 126, 1306–1322, 1998. 7655

Leutwyler, D., Fuhrer, O., Cumming, B., Lapillonne, X., Gysi, T., Lüthi, D., Osuna, C., and Schär, C.: Towards cloud-resolving European-scale climate simulations using a fully GPU-enabled prototype of the COSMO Regional Model, in: EGU General Assembly Conference Abstracts, vol. 16, p. 11914, 2014. 7653

Lin, X., Xie, S.-P., Chen, X., and Xu, L.: A well-mixed warm water column in the central Bohai Sea in summer: effects of tidal and surface wave mixing, J. Geophys. Res.-Oceans, 111, 2156–2202, doi:10.1029/2006JC003504, 2006. 7655

Linford, J. C., Michalakes, J., Vachharajani, M., and Sandu, A.: Multi-core acceleration of chemical kinetics for simulation and prediction, in: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, p. 7, ACM, 2009. 7653

Mak, J., Choboter, P., and Lupo, C.: Numerical ocean modeling and simulation with CUDA, in: OCEANS 2011, IEEE, 1–6, 2011. 7669

McCalpin, J. and Wonnacott, D.: Time skewing: a value-based approach to optimizing for memory locality, Tech. rep., Technical Report DCS-TR-379, Department of Computer Science, Rugers University, 1999. 7671

Michalakes, J. and Vachharajani, M.: GPU acceleration of numerical weather prediction, Parallel Processing Letters, 18, 531–548, 2008. 7653

Newberger, P. and Allen, J. S.: Forcing a three-dimensional, hydrostatic, primitive-equation model for application in the surf zone: 1. Formulation, J. Geophys. Res.-Oceans, 112, C08018, doi:10.1029/2006JC003472, 2007a. 7655

Newberger, P. A. and Allen, J. S.: Forcing a three-dimensional, hydrostatic, primitive-equation model for application in the surf zone: 2. Application to DUCK94, J. Geophys. Res.-Oceans, 112, C08019, doi:10.1029/2006JC003474 2007b. 7655

NVIDIA: CUDA C Programming Guide Version 5.5, available at http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html (last access: January 2014), 2014. 7670

Oey, L., Chang, Y.-L., Lin, Y.-C., Chang, M.-C., Xu, F.-H., and Lu, H.-F.: ATOP-the Advanced Taiwan Ocean Prediction System based on the mpiPOM Part 1: model descriptions, analyses and results, Terr. Atmos. Ocean. Sci., 24, 137–158, 2013. 7655

Oey, L.-Y.: A wetting and drying scheme for POM, Ocean Model., 9, 133–150, 2005. 7655

Oey, L.-Y.: Geophysical Fluid Modeling with the mpi version of the Princeton Ocean Model (mpiPOM), Lecture Notes, 70 pp., available at: ftp://profs.princeton.edu/leo/lecture-notes/OceanAtmosModeling/Notes/GFModellingUsingMpiPOM.pdf (last access: January 2014), 2014. 7656, 7669

Oey, L.-Y. and Chen, P.: A model simulation of circulation in the northeast Atlantic shelves and seas, J. Geophys. Res.-Oceans, 97, 20087–20115, 1992a. 7655

Oey, L.-Y. and Chen, P.: A nested-grid ocean model: with application to the simulation of meanders and eddies in the Norwegian Coastal Current, J. Geophys. Res.-Oceans, 97, 20063–20086, 1992b. 7655

Oey, L.-Y., Mellor, G. L., and Hires, R. I.: A three-dimensional simulation of the Hudson-Raritan estuary. Part I: Description of the model and model simulations, J. Phys. Oceanogr., 15, 1676–1692, 1985a. 7655

Oey, L.-Y., Mellor, G. L., and Hires, R. I.: A three-dimensional simulation of the Hudson-Raritaqn estuary. Part II: Comparison with observation, J. Phys. Oceanogr., 15, 1693–1709, 1985b. 7655

Oey, L.-Y., Mellor, G. L., and Hires, R. I.: A three-dimensional simulation of the Hudson-Raritan estuary. Part III: Salt flux analyses, J. Phys. Oceanogr., 15, 1711–1720, 1985c. 7655

Oey, L.-Y., Lee, H.-C., and Schmitz, W. J.: Effects of winds and Caribbean eddies on the frequency of Loop Current eddy shedding: a numerical model study, J. Geophys. Res.-Oceans, 108, 3324, doi:10.1029/2002JC001698, 2003. 7655

Potluri, S., Wang, H., Bureddy, D., Singh, A. K., Rosales, C., and Panda, D. K.: Optimizing MPI communication on multi-GPU systems using CUDA inter-process communication, in: Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International, 1848–1857, IEEE, 2012. 7665

Shimokawabe, T., Aoki, T., Muroi, C., Ishida, J., Kawano, K., Endo, T., Nukada, A., Maruyama, N., and Matsuoka, S.: An 80-fold speedup, 15.0 TFlops full GPU acceleration of non-hydrostatic weather model ASUCA production code, in: High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for, 1–11, IEEE, 2010. 7653, 7664

Smolarkiewicz, P. K.: A fully multidimensional positive definite advection transport algorithm with small implicit diffusion, J. Comput. Phys., 54, 325–362, 1984. 7656

Volkov, V.: Better performance at lower occupancy, in: Proceedings of the GPU Technology Conference, GTC, vol. 10, 2010. 7658

Whitehead, N. and Fit-Florea, A.: Precision & performance: Floating point and IEEE 754 compliance for NVIDIA GPUs, rn $(A+B)$, 21, available at: https://developer.nvidia.com/sites/default/files/akamai/cuda/files/NVIDIA-CUDA-Floating-Point.pdf (last access: January 2014), 2011. 7669

Wonnacott, D.: Using time skewing to eliminate idle time due to memory bandwidth and network limitations, in: Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International, 171–180, IEEE, 2000. 7671

Xu, F.-H. and Oey, L.-Y.: The origin of along-shelf pressure gradient in the Middle Atlantic Bight, J. Phys. Oceanogr., 41, 1720–1740, 2011. 7655

Xu, F.-H., Oey, L.-Y., Miyazawa, Y., and Hamilton, P.: Hindcasts and forecasts of Loop Current and eddies in the Gulf of Mexico using local ensemble transform Kalman filter and optimum-interpolation assimilation schemes, Ocean Model., 69, 22–38, 2013. 7655

Yang, C., Xue, W., Fu, H., Gan, L., Li, L., Xu, Y., Lu, Y., Sun, J., Yang, G., and Zheng, W.: A peta-scalable CPU-GPU algorithm for global atmospheric simulations, in: Proceedings of the 18th ACM SIGPLAN symposium on Principles and practice of parallel programming, 1–12, ACM, 2013. 7664

Yin, X.-Q. and Oey, L.-Y.: Bred-ensemble ocean forecast of Loop Current and rings, Ocean Model., 17, 300–326, 2007. 7655

Zavatarelli, M. and Mellor, G. L.: A numerical study of the Mediterranean Sea circulation, J. Phys. Oceanogr., 25, 1384–1414, 1995. 7655

Zavatarelli, M. and Pinardi, N.: The Adriatic Sea modelling system: a nested approach, Ann. Geophys., 21, 345–364, doi:10.5194/angeo-21-345-2003, 2003. 7655

Zhenya, S., Haixing, L., Xiaoyan, L., and Zhao, W.: The Application of GPU in Ocean General Circulation Mode POP, Computer Applications and Software, 27, 27–29, 2010. 7653

Table 1. Different subroutines adopt different optimizations in gpuPOM.

Subroutines	A	B	C	D	E	Speedup
Adv and Hor diff	√		√	√	√	2.05×
Ver diff		√	√	√	√	2.82×
Baroclinic	√		√		√	2.08×
Continuity equ	√				√	1.39×
Vorticity	√		√		√	3.19×
State equ	√				√	1.35×

Table 2. The strong scaling result of gpuPOM.

Number of GPUs	1 GPU	2 GPUs	4 GPUs
Time (s)	97.2	48.7	26.3
Efficiency	1.00	0.99	0.92

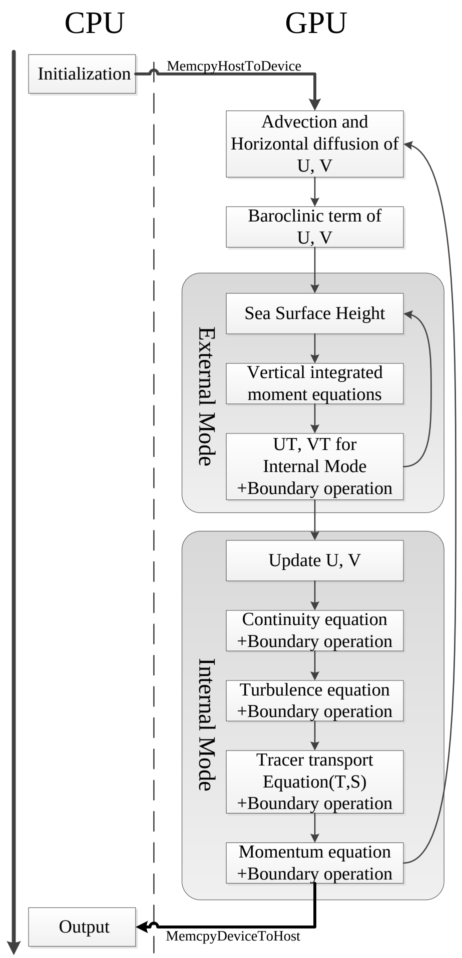
CPU
GPU
Initialization
MemcpyHostToDevice
Advection and Horizontal diffusion of U, V
Baroclinic term of U, V
External Mode
Sea Surface Height
Vertical integrated moment equations
UT, VT for Internal Mode +Boundary operation
Internal Mode
Update U, V
Continuity equation +Boundary operation
Turbulence equation +Boundary operation
Tracer transport Equation(T,S) +Boundary operation
Momentum equation +Boundary operation
Output
MemcpyDeviceToHost

Figure 1. gpuPOM flowchart.

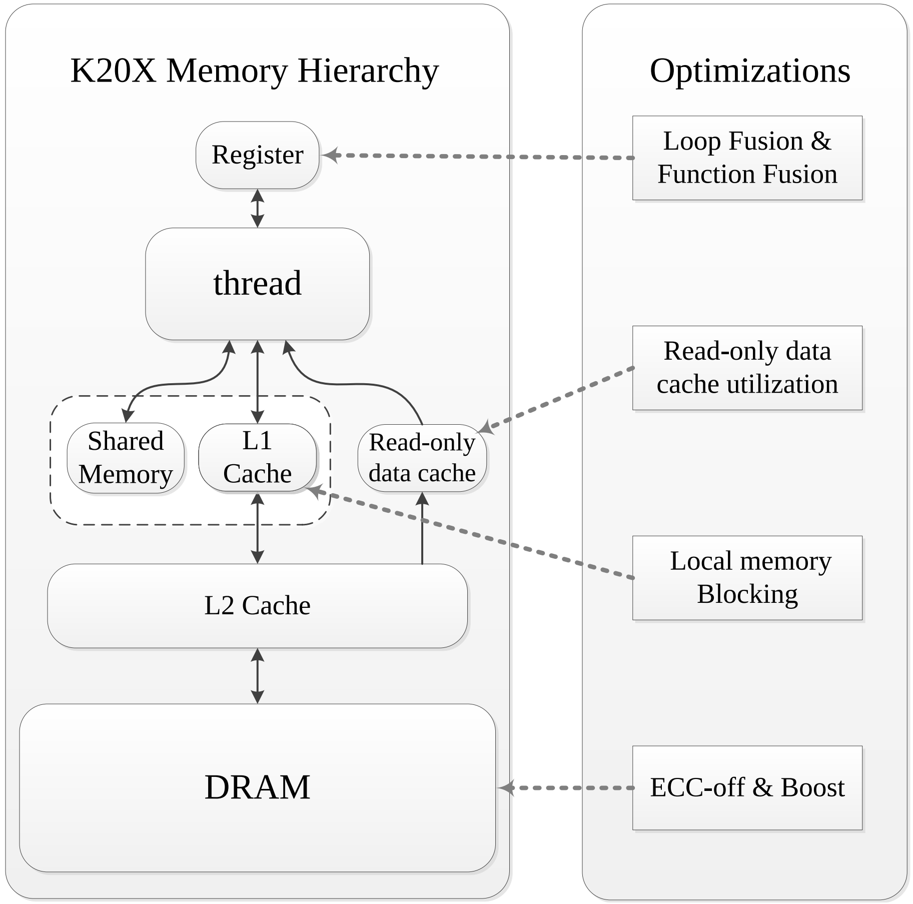

Figure 2. The memory hierarchy of K20X GPU and the relationships with each optimizations.

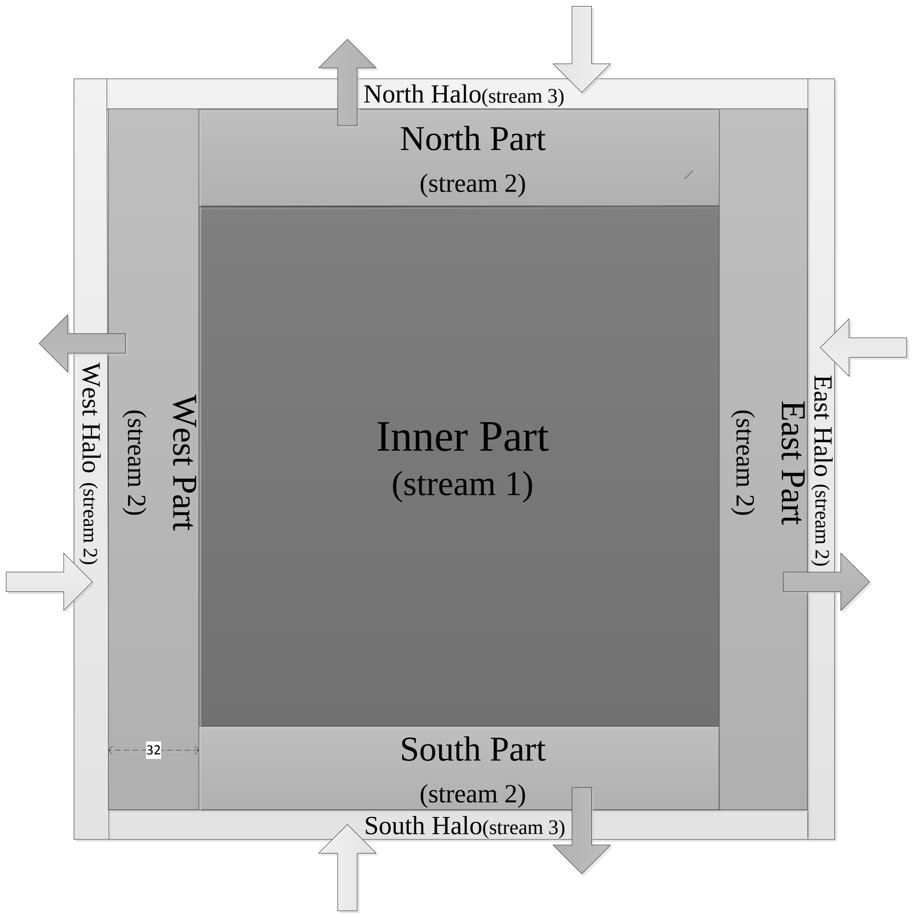

Figure 3. Data decomposition in gpuPOM.

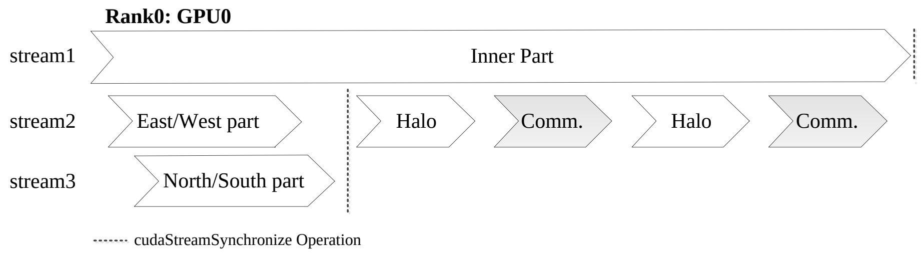

Figure 4. The workflow of multiple streams on the GPU. The "inner/east/west/north/south part" and "Halo" refer to computation and update of corresponding part. "Comm." refers to communication between processes, which implies synchronization.

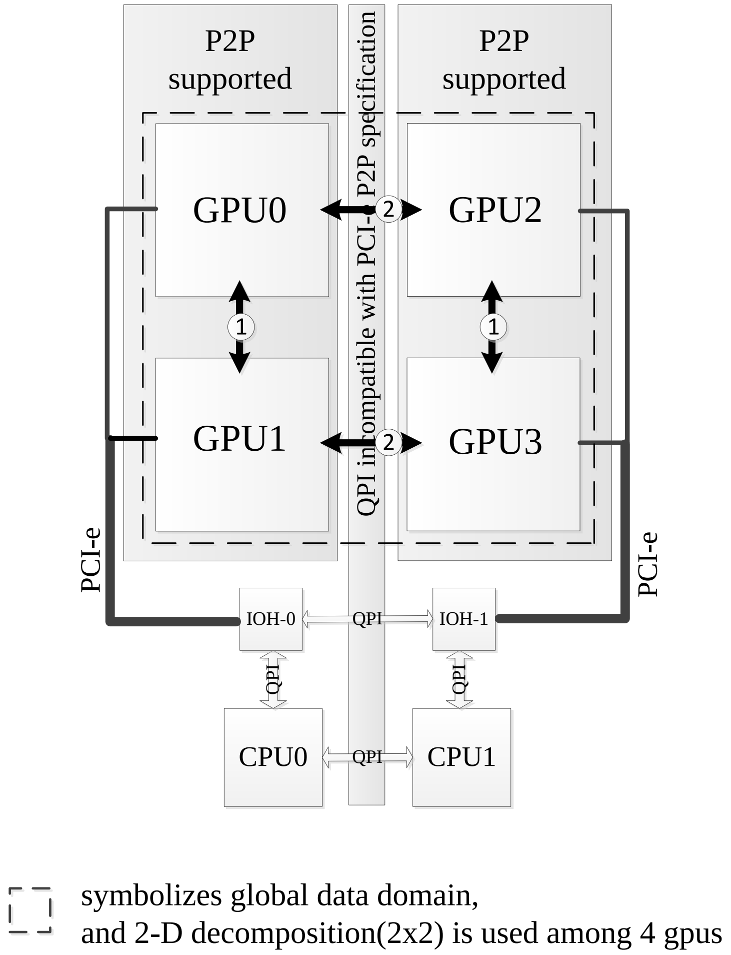
P2P supported
P2P supported
QPI incompatible with PCI-e P2P specification
GPU0
GPU2
GPU1
GPU3
1
1
2
2
PCI-e
PCI-e
IOH-0
QPI
IOH-1
QPI
QPI
CPU0
QPI
CPU1
symbolizes global data domain,
and 2-D decomposition(2x2) is used among 4 gpus

Figure 5. Communications pattern among multiple GPUs in one node.

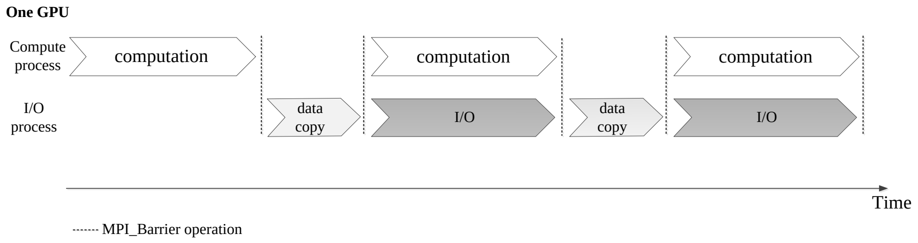

Figure 6. One computing process and one I/O process both set their contexts on the same GPU. During the data copy phase, the computing process remains idle and the I/O process will copy data from the GPU to the CPU host through the cudaMemcpy function.

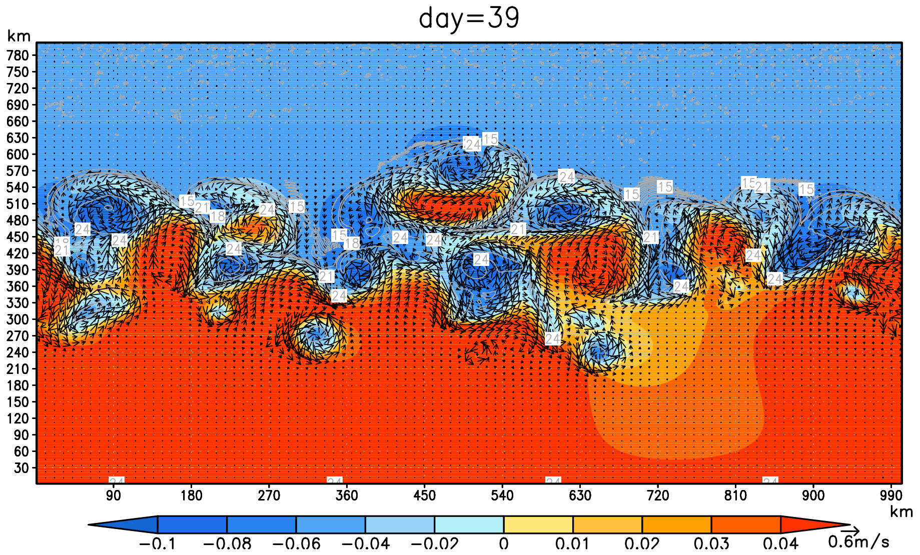

Figure 7. The sea-surface height (SSH), sea-surface temperature (SST), and currents after 39 days simulation.

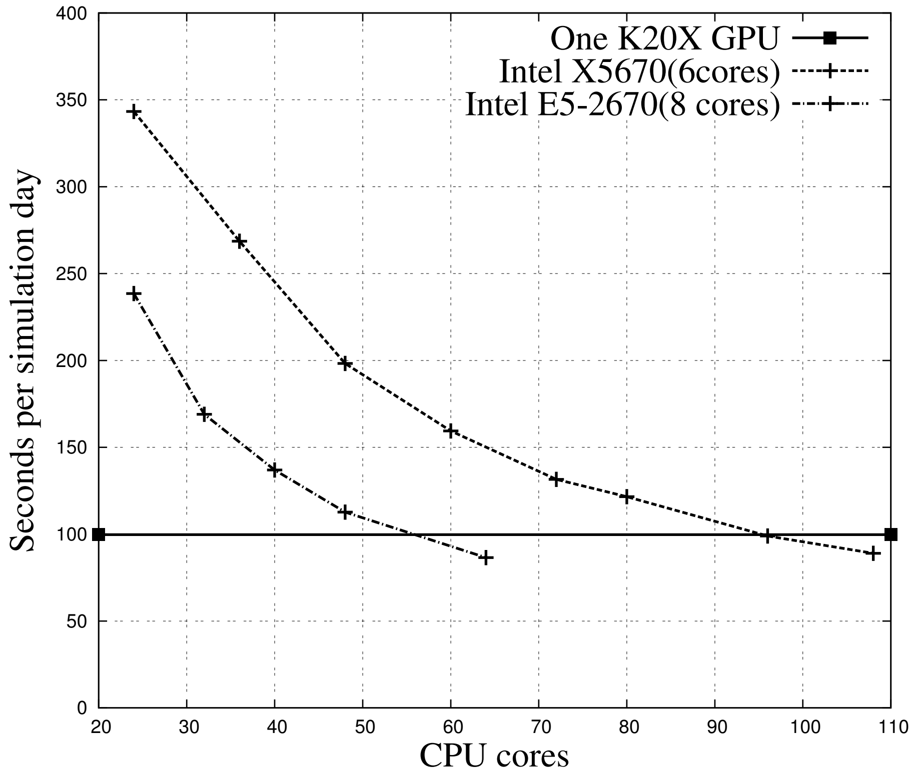

Figure 8. Performance comparison with different hardware platform.

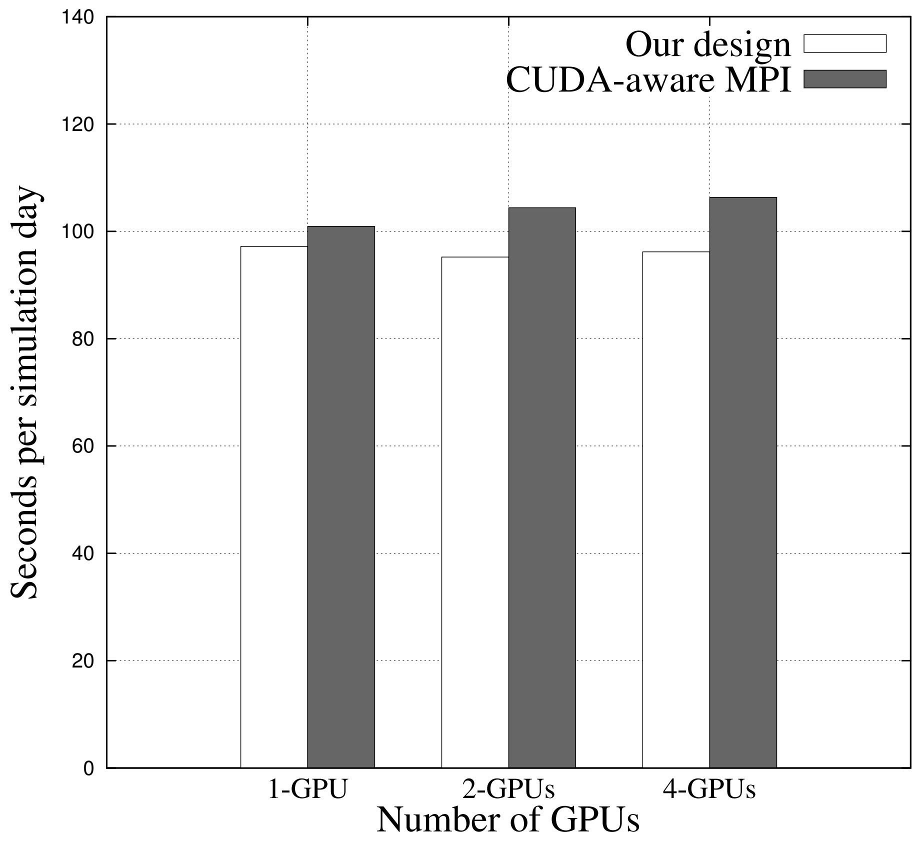

Figure 9. The weak scaling test between our communication overlapping method and the MVA-PICH2 subroutines.

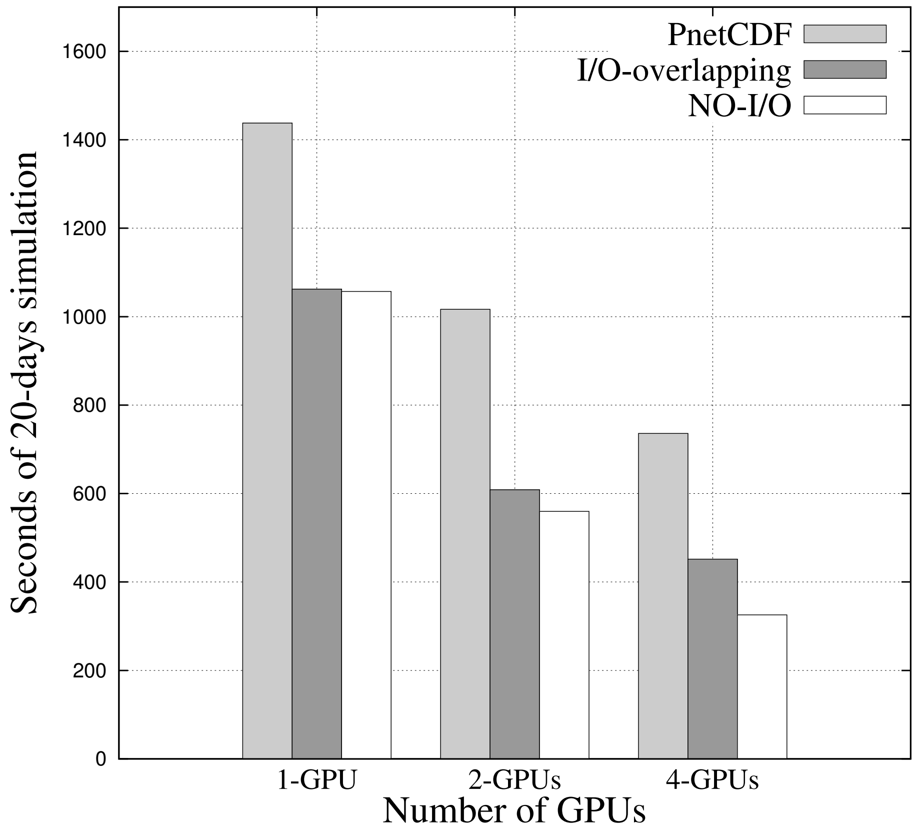

Figure 10. I/O Test for gpuPOM.

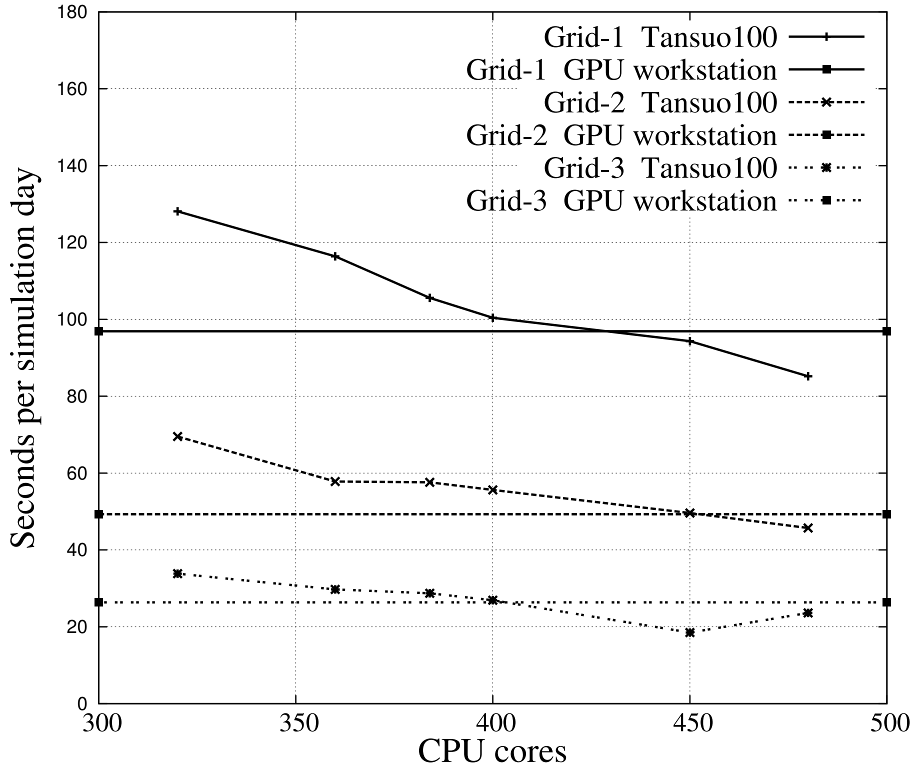

Figure 11. Four GPUs performance test compared with Tansuo100 clusters (Intel Westmere CPUs).