IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 12, 2019, accepted July 23, 2019, date of publication August 1, 2019, date of current version November 4, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2932443

Porting LASG/ IAP Climate System Ocean Model to Gpus Using OpenAcc

JINRONG JIANG[1], PENGFEI LIN[3], JOEY WANG[2], HAILONG LIU[3], XUEBIN CHI[1], HUIQUN HAO[1], YUZHU WANG[4,5], WU WANG[1], AND LINGHAN ZHANG[1]

[1]Computer Network Information Center, Chinese Academy of Sciences, University of Chinese Academy of Sciences, Beijing 100190, China
[2]NVIDIA, Beijing 100004, China
[3]LASG, Institute of Atmospheric Physics, Chinese Academy of Sciences, University of Chinese Academy of Sciences, Beijing 100029, China
[4]School of Information Engineering, China University of Geosciences (Beijing), Beijing 100083, China
[5]Key Laboratory of Geological Information Technology, Ministry of Natural Resources, Beijing 100812, China

Corresponding authors: Jinrong Jiang (jjr@sccas.cn) and Pengfei Lin (linpf@mail.iap.ac.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2016YFB0200800, in part by the National Natural Science Foundation of China under Grant 61602477 and Grant 61432018, in part by the Strategic Priority Research Programme under Grant XDC01040000, in part by the Knowledge Innovation Program of the Chinese Academy of Sciences under Grant XXH13506-402 and Grant XXH13506-302, in part by the Open Research Project of the Key Laboratory of Geological Information Technology of Ministry of Natural Resources, and in part by the Open Research Project of the Hubei Key Laboratory of Intelligent Geo-Information Processing under Grant KLIGIP-2017A04.

ABSTRACT GPUs have become important solutions for accelerating scientific applications. Most of the existing work on climate models now use code rewritten using CUDA to achieve a limited speedup. This restriction also greatly limits followup development and applications. In this paper, we designed and implemented a GPU-based acceleration of the LASG/IAP climate system ocean model (LICOM) version 2, called LICOM2-GPU. Considering the extremely large codebase of the model and the occasional need to modify the code, we implemented the model completely in OpenACC. Several accelerated methods, including OpenACC data locality optimization, loop optimization, and interprocess communication optimization are presented. Developing for GPUs using OpenACC is substantially simpler than using the CUDA port. Thus, the OpenACC is a suitable GPU programming model for complex systems, such as the earth system model and its components. Our experimental results using 4 NVIDIA K80 cards achieved up to a 6.6x speedup compared with 4 Intel(R) Xeon(R) CPU E5-2690 v2 GPUs.

INDEX TERMS High performance computing, parallel algorithm, GPU, LICOM, parallel acceleration.

I. INTRODUCTION

There is a growing need for ever more accurate climate and weather simulations to be delivered at higher resolution and shorter timescales. Recently, the highest resolution has been approximately ten kilometers to one kilometer, but this is expected to decrease to hundreds of meters soon. One serious problem with high resolution modeling is that executing high-resolution climate models is more time consuming, and supercomputer and CPU time are expensive [19], [20]. The charges and execution times associated with these applications are two main concerns of the supercomputer users; thus, parallel implementations that can improve application execution speeds are important [18]. GPUs are increasingly used to construct complex distributed computing systems [22]–[25], such as SUMMIT and SIERRA, which are Top-1 and Top-2 in the TOP500 supercomputer list, respectively. Hardware acceleration, such as using GPUs, can potentially result in much shorter runtimes or in simulations with higher accuracy. Researchers have spent a great deal of energy porting GPU-compatible computer code well-suited to GPUs because GPUs have become a feasible approach to accelerating high -resolution models.

Many works have discussed porting only portions of model codes to GPUs with CUDA to obtain substantial speed improvements. For instance, Xiao H. et al. accelerated the Weather Research Forecast (WRF) Single Moment 6-class (WSM6) microphysics scheme, achieving a speedup of nearly 100x compared with a serial CPU version running on 1 CPU core [11]. Mielikainen J. et al. accelerated the Goddard solar radiative transfer module and achieved a speedup of 112x [13]. Michalakes et al. accelerated a

The associate editor coordinating the review of this manuscript and approving it for publication was Junaid Shuja.

computationally intensive microphysics process in the WRF model and achieved a speedup of nearly 25x compared with the MPI version running on 1 CPU [1]. R. Farina et al. accelerated the elliptic kernel for NEMO-OPA, which a sped it up by 53x compared with the serial version executing on 1 CPU core [21].

However, works to recode entire models is far less common; moreover, the achievable speed improvements are much smaller. For instance, Vanderbauwhede W. et al. accelerated the scalar advection module, achieving a speedup of 7x, but the speedup of the entire WRF model was less than 2x [10]. In [17] the authors accelerated a complete large operational weather forecasting model named COSMO and achieved a speedup of 2.8X speedups for its dynamical core. S. Xu et al. ported the Princeton Ocean Model to GPUs and achieved a 6.9x speedup [16].

According to our analysis, these results stem from two reasons. First, the cost of data transfer limits the performance of the entire application. Many scientific models suffer from a flat performance profile and boundary communication synchronization overhead during GPU acceleration. In CAM [3], the most single expensive subroutine accounts for less than 5%, and the synchronization overhead accounts for more than 15%, and the same is true for LICOM. According to Amdahl's law, it is not possible to speed up the whole model significantly, for example, the GPU acceleration of CAM [3], ROMS [4], WRF [4] and HOMME [5], especially their dynamical cores. Second, climate models have huge codebases and are mostly coded in FORTRAN and by multiple people, which makes it very difficult to understand the algorithms involved. Porting code to a GPU is arduous work and extremely difficult. However, because models develop fast and there is a need to change the code occasionally, readability is also very important.

Developing using the OpenACC for GPUs is substantially simpler than developing ports in CUDA; moreover, OpenACC is compatible with CUDA, which is sometimes a necessary compromise because of the huge codebase. Several good studies exist concerning porting models with OpenACC. For instance, Norman M. et al. accelerated the CAM-SE climate model in OpenACC and achieved an acceptable speedup [9]. Demeshko I. et al. accelerated the NICAM atmospheric model using OpenACC and achieved a speedup of 3x [12]. The authors of [7] showed that large parts of COSMO were ported based on OpenACC but its dynamical core was ported with CUDA, achieving a speedup of 2.8x.

The objective of our study is to shorten the computation time required to execute high resolution ocean models by parallelizing their current model structures using GPUs through OpenACC. Using LICOM as a representative oceanic model,, we demonstrate how to parallelize an oceanic model to make it run effectively on GPU architecture. We designed and implemented several different optimization methods, including OpenACC data locality optimization, loop optimization, algorithm implementation improvement and interprocess communication optimization, and also used conventional optimization methods such as Cuda for local memory blocking, loop fusion and subroutine fusion, and so forth.

The experimental results demonstrate that using 4 NVIDIA K80 cards can achieve up to a 6.6x speedup compared with 4 Intel(R) Xeon(R) E5-2690 v2 CPUs.

The rest of this paper is organized as follows. Section II introduces the LICOM model and its control flow. In Section III, the OpenACC programming model is introduced. Section IV discusses the detailed optimizations that we employed for the LICOM model. Section V describes the experimental setups and the performance of the optimized model. Finally, Section VI provides conclusions concerning our optimization work of porting the LICOM model onto GPUs.

II. THE LICOM

The ocean plays a major role in regulating weather and climate on the earth. LICOM is a global ocean general circulation model (OGCM) developed since the 1980s by scientists at LASG, the Institute of Atmospheric Physics (IAP), and the Chinese Academy of Sciences (CAS). The goal of LICOM is to develop an oceanic component of a climate system model and a numerical tool to study the ocean circulation mechanism at different time scales. The first version of OGCM was established in the late 1980s (Zhang and Liang, 1989). Since then, the group of LASG/IAP ocean models has been expanded successively over the past 20 years. The LASG/IAP Climate system Ocean Model (LICOM) was developed based on the third version of the LASG/IAP OGCM (Jin et al., 1999). A new version of LICOM, version 2, was released in 2012 for coupled models, whose simulations were submitted to IPCC AR5 [15], and included improvements to several physical process models. The performance of LICOM2.0 has been extensively evaluated in [15]. Here, we implemented the LICOM2.0 model for GPUs. According to the Navier-Stokes formula, LICOM2.0 was written using modular Fortran code, and it can be executed on in different computer configurations. LICOM2.0 can predict the 3D ocean temperature, salinity, circulation and sea level undulations.

The primary features of LICOM include the η-coordinate, free surface, primitive equation, and mesoscale eddy parameterization from Gent and McWilliams (1990). Figure 1 shows a diagram of LICOM. The major processes in integral loop include barotropic, baroclinic and thermohaline processes, among others, and the Euler forward or leapfrog scheme is used. There are several modules with lengthy execution times, including READYT (compute density, baroclinic pressure and the relevant variables), READYC (compute momentum advection, diffusion and their vertical integrals), BARTOR (prediction of barotropic mode), BCLINC (prediction of baroclinic mode), TRACER (prediction of passive tracer), etc.

III. THE OpenACC

OpenACC is a directive-based programming model designed to provide a simple yet powerful approach to accelerators without significant programming effort. Using OpenACC,

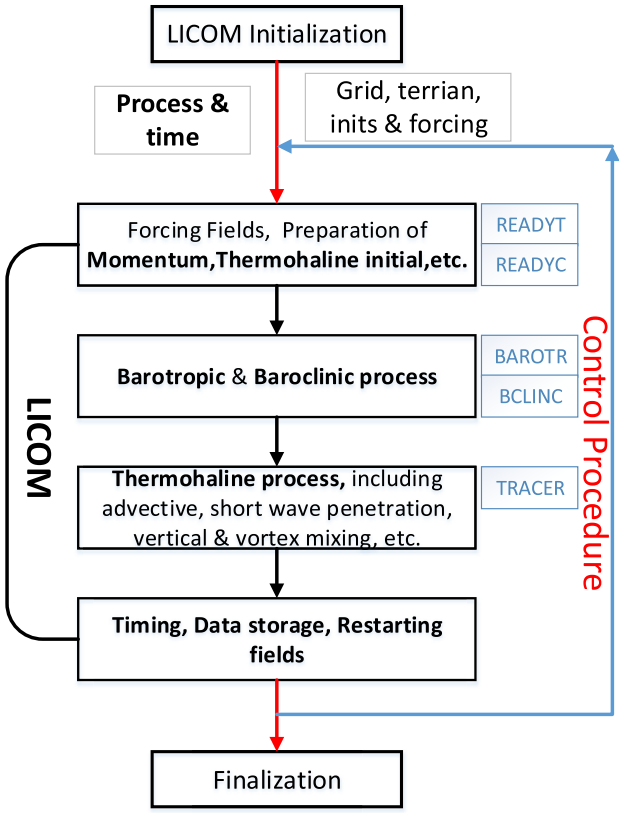


FIGURE 1. The diagram of LICOM.

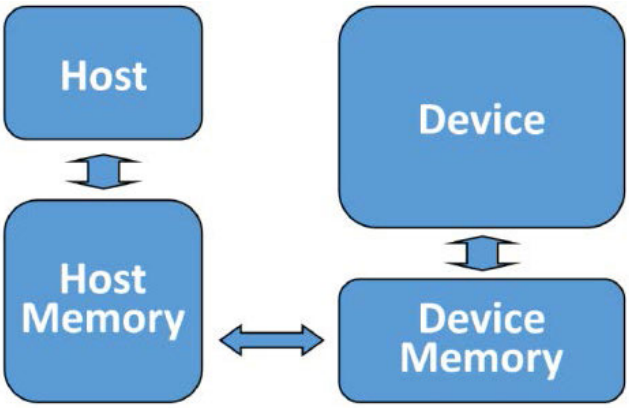


FIGURE 2. OpenACC's abstract accelerator model.

developers insert compiler hints in the form of OpenMP-like directives that are used to manage data movement, initiate parallel execution and optimize loop mapping into the compute-intensive portions of even the largest, most complex Fortran or C applications. The compiler automatically maps such code to an accelerator, including NVIDIA GPUs, to achieve higher performance. OpenACC is fully compatible and interoperates with OpenMP, CUDA and MPI.

At its core, OpenACC supports offloading both computation and data from a host device to an accelerator device. When the two devices have different types of memories, the OpenACC compiler and runtime will analyze the code and handle accelerator memory management and transferring

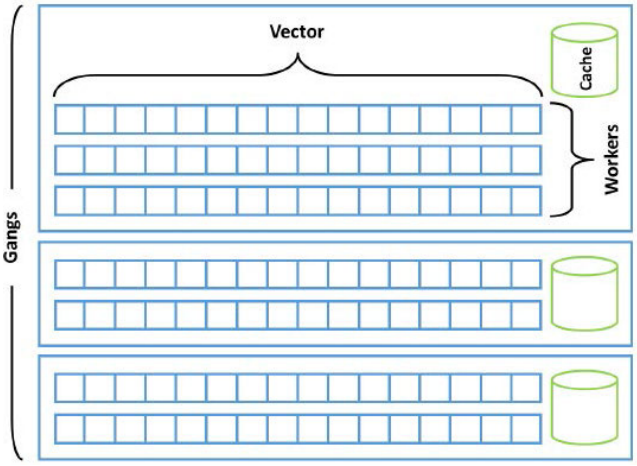


FIGURE 3. The map of gang, worker, vector.

data between host and device memory. Figure 2 shows a high-level diagram of the OpenACC abstract accelerator.

The OpenACC execution model has three levels: gang, worker and vector, which respectively correspond to the block, wrap and thread operations of CUDA. Figure 3 shows this mapping to any architecture consisting of a collection of processing elements (PEs), where each PE is multithreaded, and each thread can execute vector instructions.

IV. THE OPTIMIZATIONS

In addition to conventional optimization methods such as local memory blocking, loop fusion and communication optimization, we implemented four key optimizations in LICOM2-gpu.

A. LOCAL MEMORY BLOCKING

Locating global parameters and data on the GPU to reduce the overhead of data exchange between the host and the GPU is important. In this paper, we use OpenACC to accelerate one iteration cycle within the month-cycle. The host executes the constant initializations, parameter readings, and initializes the MPI calculation grid, and then transmits the required data to the GPU. After the GPU completes the computation, it transfers the results back to the host. To avoid the system overhead caused by frequent memory copy operations between the CPU and the GPU, we need to allocate a fixed memory space on the GPU and ensure that it remains in the GPU's memory space during the LICOM2-gpu lifetime.

As shown in Listing 1, we generate the corresponding mapping variables on the GPU of the global variables used in LICOM2-gpu by calling !$acc declare create(list of variables). Then, we use the present clause to indicate the storage address on the GPU, thus avoiding duplicate memory requests and data copies. Within one iteration cycle, except for halo communications, no data is transferred between the CPU and the GPU. We use !$acc update device(list of variables) and !$acc update self(list of variables) to manage data transfers between iteration cycles.

```
module pconst_mod
#ifdef SPMD
    integer, dimension(jmt):: j_global
    !$acc declare create(j_global)
     ...
#endif
    real(r8),dimension(imt, jmt, km):: vit,viv
    !$acc declare create(vit, viv)
    integer,dimension(imt, jmt):: itnu
    !$acc declare create(itnu)
    real(r8),dimension(km):: zkt,dzp,odzp,odzt
    !$acc declare create(zkt,dzp,odzp,odzt)
    real(r8),dimension(kmp1):: zkp
    !$acc declare create(zkp)
    ...
end module pconst_mod
```

Listing 1. Data Management of global parameters.

```
!$acc parallel loop gang vector_length(1024)&
!$acc present(v,vp,div,u,up,diu)&
!$acc private(k) acc async(1)
do k = 1, km
 !$acc loop collapse(2) private(i,j)
 do j = jst, jet
  do i = 1, imt
   v(i, j, k) = vp(i, j, k)+div(i, j, k)*dtc
   u(i, j, k) = up(i, j, k)+diu(i, j, k)*dtc
  end do
 end do
end do
```

Listing 2. Typical loop acceleration.

B. LOOP FUSION

The main work involves accelerating the LICOM2-gpu loops. A typical code diagram is shown in Listing 2. The logical structure of each loop layer is relatively simple; there are two statements in the innermost loop (i loop), and there is no dependency between the variables, which makes this loop highly suitable for acceleration via OpenACC. There is a correspondence between OpenACC's and CUDA's threading model. In OpenACC, the loop decorated by a gang is compiled into blocks in CUDA, and the inner loop of the gang loop is compiled into a CUDA thread.

As shown in Listing 2, by decorating the outer loop (k loop) with a parallel loop gang, the OpenACC compiler generates the corresponding number of blocks based on the number of outer loop iterations (for example, 30 blocks in this case). The loopcollapse(2) clause is used to merge two inner layers into one because the two inner loops are independent; thus, they can be merged. If the inner loops are not merged this way, then there one layer of the two inner cycle must be directed to run sequentially, which greatly reduce that the effect and wastes GPU computing resources. The vector_length(1024) clause indicates that 1024 threads are used for each block.

As shown in Listing 3, some code diagrams consist of several loop tiers with statements that execute in the middle. At this point, the code should be observed carefully

```
do j = 2, jmm
 do k = 2, km-1
  m = kisrpl(k)! statement
  fxd = cle10 * p25 *dzr(k)! statement
  do i = 2, imm
   e(i,k,j,3) = fxd * rhoi(i,k-1,j,m)
  end do
 end do
end do
```

Listing 3. Refactoring source code 1A.

```
!$acc parallel loop gang vector_length(32)&
!$acc present(kisrpl,dxr,e,rhoi)&
!$acc private (k,m,fxd) async(1)
do j = 2,jmm
 !$acc loop vector private(k,m,fxd)
 do k = 2,km-1! km = 30
  m = kisrpl(k)! statement
  fxd = cle10* p25*dzr (k)! statement
  do i = 2,imm
   e(i,k,j,3) = fxd*rhoi(i,k-1,j,m)
  end do
 end do
end do
```

Listing 4. Refactoring source code 1B.

```
!$acc parallel loop gang vector_length(1024)
!$acc present(kisrpl,dxr,e,rhoi) &
!$acc private (k,m,fxd) async(1)
do j = 2, jmm
 !$acc loop collapse(2) private(k,m,fxd)
 do k = 2, km-1
  do i = 2, imm
   m = kisrpl (k) ! statement
   fxd = cle10* p25*dzr (k)! statement
   e(i,k,j,3) = fxd*rhoi(i,k-1,j,m)
  end do
 end do
end do
```

Listing 5. Refactoring source code 1C.

to determine whether it can be refactored to use a collapse clause. The code is refactored by moving statements from the intermediate layer loop (k loop) to the innermost loop (i loop). As shown in Listing 4, if we run this code as a serial program, the moved code will be executed extra times because its execution is only related to the loop variable k, regardless of the loop variable i. As shown in Listing 5, it is necessary to refactor to merge the two-tier loops (k loop and i loop) using the loop collapse mechanism (2); then, we can use the configuration of 1024 threads per block to run the merged loops.

The number of cycles in the inner loop is larger than the number of cycles of the outer loop, and is also a very common code pattern, as shown in the diagram in Listing 6.

```
      do ncy = 2, max_nn
       do j = jst, jmt
        if (nn(j) .ge. ncy) then
         do k = 1, kk
          do i = 1, imt
           xs(i) = x(i,j,k)*z(i,j,k)
          end do
          do i = 2, imm
           x(i,j,k) = xs(i)*z(i,j,k)
          end do
         end do
        end if
       end do
      end do
```

Listing 6. Refactoring source code 2A.

```
do ncy = 2, max_nn
 !$acc parallel loop gang vector_length(1024)&
 !$acc present(xs,x,z,nn) private (k)&
 !$acc async(1)
 do k = 1, kk
  !$acc loop collapse(2) private (j,i)
  do j = jst, jmt
   do i = 1, imt
    if (nn(j).ge.ncy) then
     xs(i,j,k) = x(i,j,k)*z(i,j,k)
    end if
   end do
  end do
 end do
 !$acc parallel loop gang vector_length(1024)&
 !$acc present(nn,xs,z) private (k)&
 !$acc async(1)
 do k = 1, kk
  !$acc loop collapse(2) private (j,i)
  do j = jst, jmt
   do i = 2, imm
    if (nn(j).ge.ncy) then
     x(i,j,k) = xs(i,j,k)*z(i,j,k)
    end if
   end do
  end do
 end do
end do
```

Listing 7. Refactoring source code 2B.

We need to examine whether it can be split into a number of independent cycles and accelerated. To split the two inner *i* loops, as shown in Listing 7 we expand the capacity of array *x* and then update *x* (*i*, *j*, *k*) corresponding to *i*, *j*, *k* simultaneously.

Loops require special treatment when there are several subordinate inner loops and subroutines, as shown in Listing 8. Because the compiler does not know whether the called function is parallel or serial, the loop *i* clause will be ignored by the compiler, which will apply multithreading to the innermost loop *k*;however, this causes the canuto_2010_interface

```
!$acc parallel loop gang async(1)
do j = jsm, jem
 !$acc loop vector
 do i = 2, imm
  do k = 1, km
   ......
  end do
 do k = 1, itnu(i,j)-1
  ......
 end do
 ......
 call canuto_2010_interface(wk1,wk2,&
 wk3,wk4,amld(i,j),tau_mag,wp3,&
 wp4,wp5,wp6,wp7,1.0d0,wp8,&
 itnu(i,j),i,j,1,isc)
 do k = 1, km
  ......
 end do
 end do
end do
```

Listing 8. Refactoring source code 3A.

```
!$acc parallel loop gang async(1)
do j = jsm, jem
 !$acc loop seq
 do i = 2, imm
  !$acc loop vector
  do k = 1, km
   ......
  end do
  !$acc loop vector
  do k = 1, itnu(i,j)-1
   ......
  end do
  ......
  call canuto_2010_interface(wk1,wk2,&
  wk3,wk4,amld(i,j),tau_mag,wp3,&
  wp4,wp5,wp6,wp7,1.0d0,wp8,&
  itnu(i,j),i,j,1,isc)
  !$acc loop vector
  do k = 1, km
   ......
  end do
 end do
end do
```

Listing 9. Refactoring source code 3B.

to run serially in every block. Therefore the code in Listing 8 will be translated by the compiler to the one in Listing 9. However, the correct code in this case is shown in Listing 10. It needs to specify !$acc loop seq before *k* loop and !$acc routine(canuto_2010_interface) seq before the subroutine. Accordingly, we set a suitable number of vector_length.

```
!$acc parallel loop vector_length(256)&
!$acc async(1)
do j = jsm, jem
 !$acc loop vector
 do i = 2, imm
  !$acc loop seq
  do k = 1, km
   ......
  end do
  !$acc loop seq
  do k = 1, itnu(i,j)-1
   ......
  end do
  !$acc routine(canuto_2010_interface) seq
  call canuto_2010_interface(wk1,wk2,&
  wk3,wk4,amld(i,j),tau_mag,wp3,wp4,&
  wp5,wp6,wp7,1.0d0,wp8,&
  itnu(i,j),i,j,1,isc)
  !$acc loop seq
  do k = 1, km
   ......
  end do
 end do
end do
```

Listing 10. Refactoring source code 3C.

```
do j = jst,jmt
 ......
 computational code
 ......
 call exchange_1d_boundary(x,j,kk)
end do
!---------------------------------------
subroutine exchange_id_boundary(x,j,kk)
 real(r8) = x(imt,jmt,kk)
 real(r8) = send_buf(kk),recv_buf(kk)
 n2 = mod(mytid, nx_proc)
 n1 = (mytid-n2)/my_proc
 if (n2<nx_proc-1) then
  do k = 1,kk
   send_buf(k) = x(imt-1,j,k)
  end do
  call mpi_send(send_buf,kk, mpi_pr,&
  mytid-1,tag_2d,mpi_comm_ocn,ierr)
  do k = 1,kk
   x(1,j,k) = recv_buf(k)
  end do
 end if
end subroutine exchange_1d_boundary
```

Listing 11. Overlapping communication A.

```
do j = jst,jmt
 ......
 computational code
 ......
end do
call exchange_1d_boundary(x,j,kk)
!---------------------------------------
subroutine exchange_id_boundary(x,j,kk)
 real(r8)::send_buf(jmt,kk)
 real(r8)::recv_buf(jmt,kk)
 !$acc data create(send_buf,recv_buf)
 if (n2<nx_proc-1)then
  !$acc kernels loop gang private(k)&
  !$acc present(send_buf,x) async(1)
  do k = 1,kk
   !$acc loop vector private(j)
   do j = jst,jmt
    send_buf(j,k) = x(imt-1,j,k)
   end do
  end do
  !$acc wait(1)
  !$acc host_data use_device(send_buf)
  call mpi_send(send_buf,jmt*kk,mpi_pr,&
  mytid-1,tag_2d,mpi_comm_ocn,status,ierr)
  !$acc end host_data
 end if
 if (n2>0) then
  !$acc wait(1)
  !$acc host_data use_device(recv_buf)
  call mpi_recv(recv_buf,jmt*kk,mpi_pr,&
  mytid-1,tag_2d,mpi_comm_ocn,status,ierr)
  !$acc end host_data
  !$acc kernels loop gang private(k) &
  !$acc present(x,recv_buf) async(1)
  do k = 1,kk
   !$acc loop vector private(j)
   do j = jst,jmt
        x(1,j,k) = recv_buf(j,k)
   end do
  end do
 end if
end subroutine exchange_1d_boundary
```

Listing 12. Overlapping communication B.

TABLE 1. Description of platform for the experiments.

CPU	Intel(R) Xeon(R) CPU E5-2690 v2 , 3.00GHz dual socket 10-core FDR IB	2
GPU	Nvidia Tesla K80	2
CUDA Runtime	CUDA 8.0	1
Fortran Compiler	PGI 17.1	1
MPI Version	OpenMPI 1.10.5	1
NetCDF	4.4.4	1

C. OVERLAPPING COMMUNICATION OPTIMIZATION

Frequent communication reduces the speed of GPU computing resources.

As Listing 11 and Listing 12 show, in each iteration of the *j* cycle, the program executes the code and then uses the MPI to perform second-dimensional data exchange in the exchange function exchange_1d_boundary by specifying the *j* value for each iteration, using send_buf (kk)and recv_buf (kk) to exchange data. However, the data exchanged in each *j* cycle iteration are not used in the next *j* cycle iteration.

TABLE 2. Description of model for the experiments.

	LICOM1	LICOM2
Horizontal resolution	$1^\circ \times 1^\circ$	$1^\circ \times (0.5^\circ - 1^\circ)$
Vertical resolution	30 levels (25m in the upper 300m)	30 levels (10m in the upper 150m)
Advection scheme Vertical mixing	2-order central difference Pacanowski and Philander (1981)	A shape-preserving (Yu, 1994) Canuto et. al. (2001, 2002)
Mesoscale eddy parameterization	Gent and McWilliams (1990), $1000m^2s^{-1}$	Gent and McWilliams (1990), $500m^2s^{-1}$;Large et. al. (1997)
Horizontal viscosity	$2 \times 10^4 m^2 s^{-1}$	$3 \times 10^3 m^2 s^{-1}$
SW radiation penetration	Constant (Paulson and Simpson, 1977)	Chlorophyll dependent (Ohlmann, 2003)
Forcing formula and dataset	Restoring; Max-Planck-Institute Ocean Model Intercomparison Project forcing and World Ocean Atlas 98	Large and Yeager (2004); COREs

TABLE 3. Description of the forcing data.

Variable	Dataset (original)	Period	Resolution	Frequency (original)
$2-m$ temperature	COREs v2 NCEP/NCAR[1] reanalysis I	1948 - 2007 –	T62 –	Daily (6h) –
$2-m$ specific humidity	–	1948 - 2007	T62	Daily (6h)
Surface level pressure	–	1948 - 2007	T62	Daily (6h)
$10-m$ wind	–	1948 - 2007	T62	Daily (6h)
Downward shortwave radiation	COREs v2(ISCCP[2])	1948 - 2007	T62	Daily (daily)
Downward longwave radiation	–	1948 - 2007	T62	Daily (daily)
Precipitation Runoff	COREs v2 (merging of 3 datasets[3]) COREs v2	1948 - 2007 1948 - 2007	T62 1°	Daily (monthly) Annual mean (annual mean)
Sea ice concentration	NSIDC	1979 - 2006	1°	Monthly (monthly)
SST and SSS	WOA05	Before 2005	1°	Monthly (monthly)

1. National Ceters for Environmental Prediction/National Center for Atmospheric Research
2. International Satellite Cloud Climatology Project
3. Global Precipitation Climatology Project (GPCP), CPC (Climate Prediction Center) Merged Analysis of Precipitation (CMAP), and the dataset of Serrez and Hurst (2000) and Yang (1999)

Therefore, to improve the communication efficiency, we need to overlap the communication using split computation and communication. We extend the buffered arrays send_buf (kk) and recv_buf (kk) to send_buf (jmt, kk), recv_buf (jmt, kk) to allow communications to complete all at one time. We use the wait (1) clause to ensure that the data on the GPU are ready; then, we use the host_datause_device clause to indicate the data used by the communication on the GPU to ensure that the data being manipulated are correct.

D. IMPROVING THE CODE AND ALGORITHM TO FIT THE OPENACC

Some complex algorithms and modules exist in the LICOM, such as the GISS turbulent vertical mixing module (Canuto_2010_interface) and GFDL's full convective adjustment module convadj, each of which have many exit, goto and while branch calculations. Consequently, the compiler is unable to obtain the logical dependencies between different blocks and threads. Therefore, we need to rewrite these models to clarify the logical dependencies.

V. EXPERIMENTS

A. THE PLATFORM AND INITIAL DATA SETUP

The hardware components used for the experiments is listed in Table 1. The GPU platform used for the experiments is a work station consisting of 2 computational nodes. Each node has 2 10-core Intel E5-2690 v2 CPUs and 2 Nvidia Tesla K80s. We tested the programs on this platform with the PGI compiler v17.1, OpenMPI Library v1.10.5, and the CUDA 8.0 Toolkit.

We used the LICOM version 2 model for the experiments. The primary settings of LICOM2.0 used in the experiments is listed in Table 2. We conducted the experiment from the initial condition with WOA05 temperature and salinity and no current, forced by the daily corrected NYF of Large and Yeager (2004). The simulation was conducted to verify the correctness of the code and to test the performance and

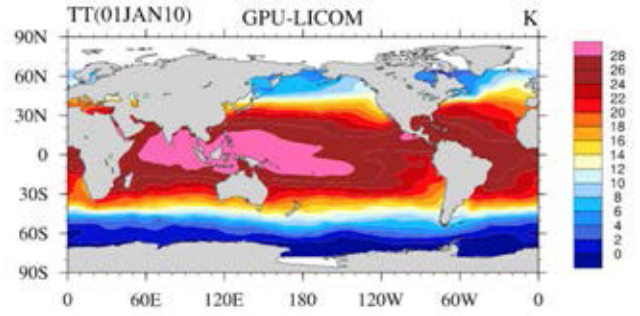


FIGURE 4. LICOM2-gpu.

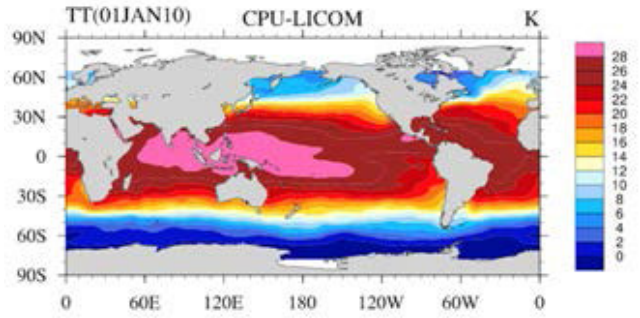


FIGURE 5. LICOM 2.0.

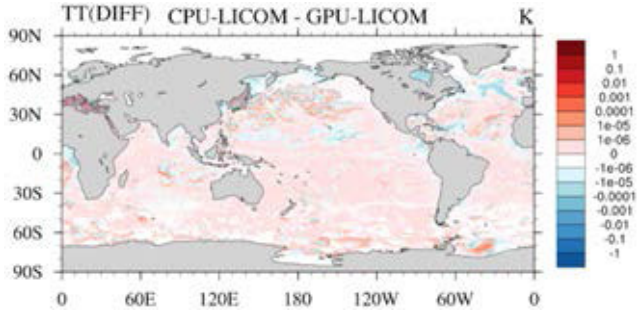


FIGURE 6. Difference between LICOM2-gpu and LICOM 2.0.

scalability of the LICOM2-gpu model. More details about the forcing data can be found in Table 3.

B. THE VERIFICATION OF ACCURACY

We executed two separate experiments using LICOM2.0 and LICOM2-gpu. The test results demonstrate that variables such as velocity, temperature, salinity and sea-surface height are all identical, and the error between the LICOM2.0 and LICOM2-gpu results is small and acceptable. The error may be caused by differences in mathematical precision between the GPU and CPU. Figure 4- 6 show the sea-surface temperature (SST) results of the two experiments and their differences after 10 days.

C. THE PERFORMANCE

In this section, we conducted a series of experiments to illustrate the improvement and benefit of the LICOM2-gpu model compared to the LICOM2.0 model. We tested the different modules of LICOM2-gpu separately and compared them with

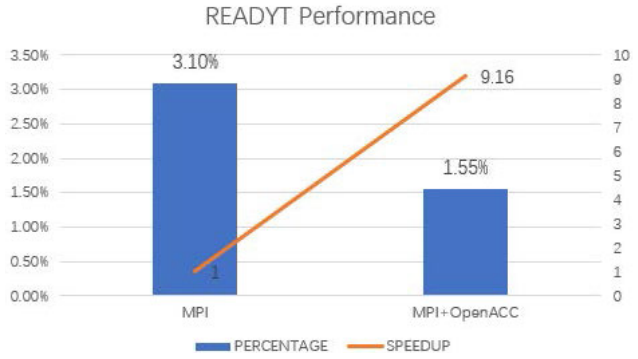


FIGURE 7. READYT performance.

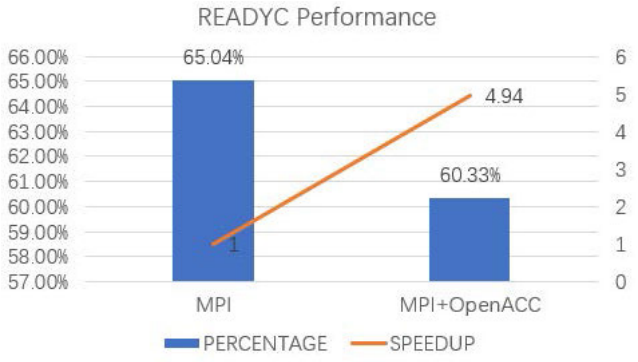


FIGURE 8. READYC performance.

the performances of LICOM2.0 to investigate the accrued optimization from employing OpenACC.

In this study, we optimized and tested five major LICOM modules to investigate the performance of OpenACC acceleration. The percentages of the complete elapsed time accounted for by these modules are shown below, and the performances of LICOM2.0 and LICOM2-gpu are compared. We executed LICOM2.0 on 16 MPI processes, while LICOM2-gpu employed 8 MPI processes for each NVIDIA Tesla K80 chipâĂ"16 MPI processes in total for the 2 NVIDIA Tesla K80 chips.

1) READYT PERFORMANCE

Figure 7 shows the performance of the READYT module. The OpenACC version achieved a speedup of 9.16*x*. The percentage of the total elapsed time occupied by the READYT module was reduced from 3.1% to 1.55%. Because READYT is a computationally intensive module, it would be advantageous to fully unroll the loops in the calculations. As shown in Figure 7, the READYT module requires only 3.1% of the total elapsed time because it is called only once during each daily iteration in a month loop. Thus, although the module itself achieved a speedup of 9.16*x*, the overall performance did not improve much.

2) READYC PERFORMANCE

Figure 8 shows the performance of the READYC module, which achieved a speedup of 4.94*x* by applying OpenACC acceleration. Additionally, the percentage of total

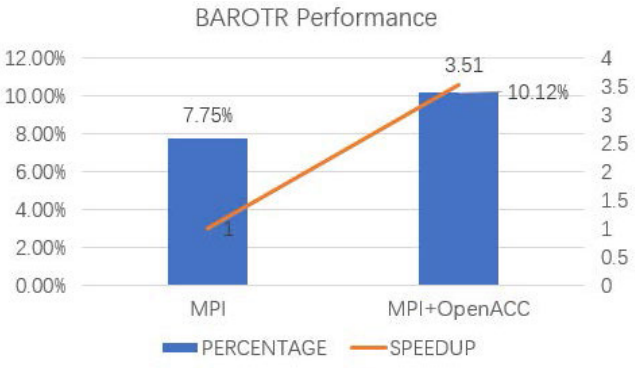


FIGURE 9. BAROTR performance.

time that the READYC modulus occupies fell from 65% to 60%. Because READYC is a computation-dominated module, it requires less communication than computation. Thus, loop optimization produces a speedup on the READYC module. In addition, the optimization of the READYC module is critical to the entire LICOM model because it occupies a dominant position among all the LICOM modules.

3) BAROTR PERFORMANCE

As shown in Figure 9, a speedup of 3.51x was achieved by applying OpenACC to the BAROTR module-the lowest of the five major modules of LICOM. Meanwhile, the proportion that the BAROTR module occupies in the total elapsed time increased from 7.75% to 10.12%. This performance loss occurs because the BAROTR module performs more communication than calculation. The main structure of BAROTR is shown in Listing 13.The outer layer is a loop that executes 24 times. In each loop, a series of subroutines, including the part of code shown in line 3 to line 9-boundary communications concerning some array variables, velocity field, thermohaline field, sea-surface fluctuation field and the one-dimensional threepoint smoothing modules of SMUV, SMTS and SMZO-run successively.

This alternating subroutine process that has a low calculation proportion is different from the situation in the other modules, where most of the loops are iterations over i, j, k indexes. The major iteration in BAROTR is the part shown in Figure 13, from line 3 to line 9. After implementing OpenACC acceleration, every two-layer loop requires only several hundreds of rounds to complete. We optimized 25 loops in this work. The percentage of time occupied by each loop in BAROTR is shown in Table 4.

Almost all optimized iterations in BAROTR achieve considerable speedups. However, the calculation involves only 10% of the total time. Thus, as the MPI processes increase, the proportion that the calculation occupies decreases. This explains why BAROTR achieves only a small speedup.

4) BCLINC PERFORMANCE

As shown in Figure 10, BCLINC module achieves a speedup of 5.64x after OpenACC acceleration is implemented, and its proportion of the total time decreases from 11.95% to 9.71%. Similar to READYT, the high communication proportion prevents the BCLINC module from achieving a higher speedup ratio. Because the BCLINC module occupies

```
!***Main Structure of BAROTR Module***
!***baro_loop***
DO NC = 1, NBB + IEB_loop
!***COMPUTING H0 AT UB/VB/H0 POINTS***
 DO J = JSM,JEM
  DO I = 1,IMT
    UB (I,J)= UBP (I,J) + WKA (I,J,3)* DTB
    VB (I,J)= VBP (I,J) + WKA (I,J,4)* DTB
    H0 (I,J)= H0P (I,J) + WORK (I,J) * DTB
  END DO
 END DO
 call exch_boundary_acc2(h0,1)
 ......
 !***COMPUTING DU & DV***
 call exch_boundary_acc2(wka(1,1,3),1)
 ......
 !***COMPUTING DH0***
 ......
 call exch_boundary_acc2(wka(1,1,1),1)
 CALL SMUV_acc (WKA(1,1,3),VIV,1,fil_lat1)
 CALL SMZ0_acc (WORK,VIT,fil_lat1)
 ......
 !***FILTER FORCING AT HIGT LATITUDES*
 !***many SMUV,SMZ0,SMTS exchange boundary*
END DO baro_loop
```

Listing 13. Main structure of BAROTR.

TABLE 4. Performance of loops.

	CPU Elapsed Time	GPU Elapsed Time
barotr_86_gpu	1003us	188us
barotr_102_gpu	1382us	230us
barotr_123_gpu	968us	172us
barotr_135_gpu	768us	88us
barotr_155_gpu	1419us	256us
...	...	...
Time Occupation of Barotr	10.55%	1.92%

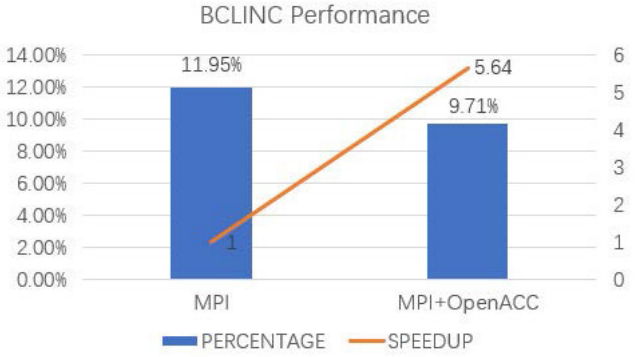


FIGURE 10. BCLINC performance.

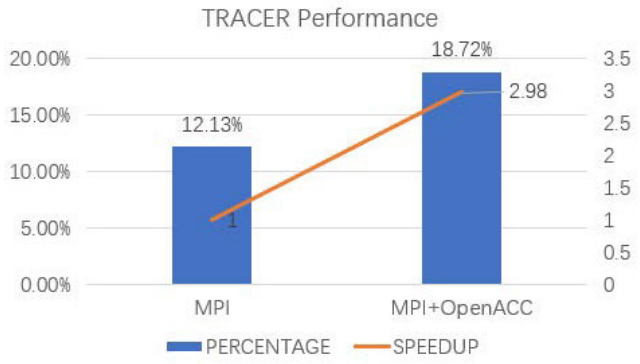


FIGURE 11. TRACER performance.

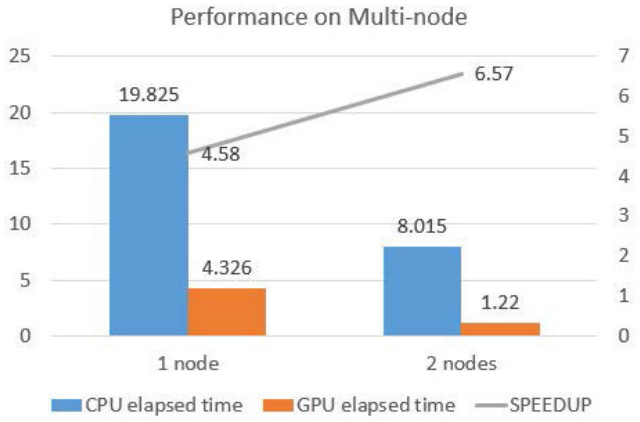


FIGURE 12. Performance on Multi-node with 1D decomposition.

11.95% of the total time, it has good potential for optimization.

5) TRACER PERFORMANCE

Figure 11 shows the performance of the TRACER module, which achieved a speedup of 2.98x, but its proportion of the total time increased from 12.13% to 18%.

6) OVERALL PERFORMANCE

In the overall performance test, we ran 1-day simulations separately 10 times using the LICOM2.0 and LICOM2-gpu models. We compare the best performances of LICOM2.0 and LICOM2-gpu on 1 and 2 nodes, as shown in Figure 12 and Figure 13, respectively. Figure 12 shows the performance of the employed 1D decomposition scheme, while Figure 13 shows the performance of the employed 2D decomposition scheme. We achieved speedups of 4.58x and 6.57x using 1D decomposition and 3.89x and 4.48x using 2D decomposition, respectively. A superlinear speedup occurs when moving from 1 node to 2 nodes, which is quite common in atmospheric and oceanic models. We believe this occurs when the working set of the LICOM exceeds the cache size when executed on 1 node and 2 GPUs but fits nicely in each available cache when executed on 2 nodes and 4 GPUs. It also occurs because memory copy and MPI communication cost when executed on 2 nodes and 4 GPUs is half that

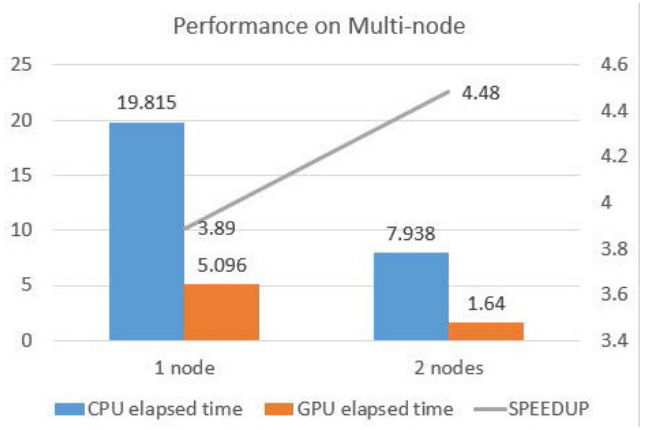


FIGURE 13. Performance on Multi-node with 2D decomposition.

when executed on 1 node and 2 GPUs and fit nicely in PCIE bandwidth. Meanwhile, because the 1D decomposition LICOM2-gpu has no MPI communication in the west-east direction grids, it requires fewer memory copies between the CPU and the GPU than in the 2D decomposition; consequently, the LICOM2-gpu runtime with 1D decomposition is less than that of the LICOM2-gpu with 2D decomposition. The 1D decomposition parallel is better than the 2D decomposition for the LICOM2-gpu model.

VI. CONCLUSION

In this paper, we designed and implemented a GPU-based accelerated version of the LASG/IAP Climate system Ocean Model (LICOM) version 2, called LICOM2-gpu. LICOM2-gpu was completely implemented using OpenACC, and it distributes all the model computations to the GPUs. Our main contributions include OpenACC data locality optimization, optimizing the code on each of the GPUs, and optimizing the communications between GPUs. The experimental results on 4 NVIDIA K80 cards achieved a speedup of up to 6.6 times compared with 4 Intel(R) Xeon(R) E5-2690 v2 CPUs. GPUs have become an important acceleration solution for scientific applications. OpenACC programming can be implemented with few changes to the source code, which means the original structure and readability of the program can be maintained while obtaining good speedup values. Development using OpenACC for GPUs is substantially simpler than development through CUDA porting. Thus, it is a suitable GPU programming model for complex systems, such as the Earth System Model and its component models.

In future work, we plan to adjust the code structure of LICOM2-gpu and adopt more aggressive fusion functions to further improve the performance. Meanwhile, we will improve the horizontal resolution of the LICOM2-gpu model and adjust the modeling schemes and the numerical methods of the equations, which will cause new computation problems. Using OpenACC instead of CUDA or OpenCL reduces the workload and promotes interdisciplinary overlap.

REFERENCES

[1] J. Michalakes and M. Vachharajani, "GPU acceleration of numerical weather prediction," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, vol. 18, Apr. 2008, pp. 1–7.
[2] T. Shimokawabe, T. Aoki, C. Muroi, J. Ishida, K. Kawano, T. Endo, A. Nukada, N. Maruyama, and S. Matsuoka, "An 80-fold speedup, 15.0 TFlops full GPU acceleration of non-hydrostatic weather model ASUCA production code," in *Proc. ACM/IEEE Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC)*, Nov. 2010, pp. 1–11.
[3] R. Kelly, "GPU computing for atmospheric modeling," *Comput. Sci. Eng.*, vol. 12, no. 4, pp. 26–33, 2010.
[4] J. Mak, P. Choboter, C. Lupo, "Numerical ocean modeling and simulation with CUDA," in *Proc. OCEANS MTS/IEEE KONA*, Sep. 2011, pp. 1–6.
[5] I. Carpenter, R. K. Archibald, K. J. Evans, J. Larkin, P. Micikevicius, M. Norman, J. Rosinski, J. Schwarzmeier, and M. A. Taylor, "Progress towards accelerating HOMME on hybrid multi-core systems," *Int. J. High Perform. Comput. Appl.*, vol. 27, no. 3, pp. 335–347, 2013.
[6] M. W. Govett, J. Middlecoff, T. Henderson, "Running the NIM next-generation weather model on GPUs," in *Proc. IEEE/ACM Int. Conf. Cluster, Cloud Grid Comput.*, 2010, pp. 792–796.
[7] X. Lapillonne and O. Fuhrer, "How to achieve performance portable code using openacc compiler directives?" in *Proc. EGU Gen. Assem. Conf.*, 2014, p. 1.
[8] *CUDA C Programming Guide*, document Version 7.5, Nvidia, Santa Clara, CA, USA, Sep. 2015.
[9] M. Norman, J. Larkin, A. Vose, and K. Evans, "A case study of CUDA FORTRAN and OpenACC for an atmospheric climate kernel," *J. Comput. Sci.*, vol. 9, pp. 1–6, Jul. 2015.
[10] W. Vanderbauwhede and T. Takemi, "An investigation into the feasibility and benefits of GPU/multicore acceleration of the weather research and forecasting model," in *Proc. Int. Conf. High Perform. Comput. Simulation (HPCS)*, Jul. 2013, pp. 482–489.
[11] H. Xiao, J. Sun, X. Bian, and Z. Dai, "GPU acceleration of the WSM6 cloud microphysics scheme in GRAPES model," *Comput. Geosci.*, vol. 59, no. 3, pp. 156–162, 2013.
[12] I. Demeshko, N. Maruyama, H. Tomita, and S. Matsuoka, *Multi-GPU Implementation of the NICAM Atmospheric Model*. Berlin, Germany: Springer, 2013, pp. 175–184.
[13] J. Mielikainen, B. Huang, H.-L. A. Huang, and M. D. Goldberg, "GPU acceleration of the updated Goddard shortwave radiation scheme in the weather research and forecasting (WRF) model," *IEEE J. Sel. Topics Appl. Earth Observat. Remote Sens.*, vol. 5, no. 2, pp. 555–562, Apr. 2012.
[14] *OpenACC Programming and Best Practices Guide*, Nvidia, Santa Clara, CA, USA, 2015.
[15] H. Liu, P. Lin, Y. Yu, and X. Zhang, "The baseline evaluation of LASG/IAP climate system ocean model (LICOM) version 2," *J. Meteorological Res.*, vol. 26, no. 3, pp. 318–329, 2012.
[16] S. Xu, X. Huang, L.-Y. Oey, F. Xu, H. Fu, Y. Zhang, and G. Yang, "POM.gpu-v1.0: A GPU-based princeton ocean model," *Geosci. Model Develop.*, vol. 8, no. 9, pp. 2815–2827, 2015.
[17] O. Fuhrer, C. Osuna, X. Lapillonne, T. Gysi, M. Bianco, and T. Schulthess, "Towards GPU-accelerated operational weather forecasting," in *Proc. GPU Technol. Conf. (GTC)*, 2013, pp. 1–34.
[18] G. B. Barone, V. Boccia, D. Bottalico, R. Campagna, L. Carracciuolo, G. Laccetti, and M. Lapegna, "An approach to forecast queue time in adaptive scheduling: How to mediate system efficiency and users satisfaction," *Int. J. Parallel Program.*, vol. 45, no. 5, pp. 1164–1193, 2017. doi: 10.1007/s10766-016-0457-y.
[19] R. Farina, S. Dobricic, A. Storto, S. Masina, and S. Cuomo, "A revised scheme to compute horizontal covariances in an oceanographic 3D-VAR assimilation system," *J. Comput. Phys.*, vol. 284, pp. 631–647, Mar. 2015.
[20] R. Farina, S. Dobricic, and S. Cuomo, " Some numerical enhancements in a data assimilation scheme," in *Proc. AIP Conf.*, 2013, vol. 1558, no. 1, p. 2369. doi: 10.1063/1.4826017.
[21] R. Farina, S. Cuomo, P. De Michele, and F. Piccialli, "A smart GPU implementation of an elliptic kernel for an ocean global circulation model," *Appl. Math. Sci.*, vol. 7, no. 61, pp. 3007–3021, 2013.
[22] Z. Deng, D. Chen, Y. Hu, X. Wu, W. Peng, and X. Li, "Massively parallel non-stationary EEG data processing on GPGPU platforms with Morlet continuous wavelet transform," *J. Internet Services Appl.*, vol. 3, no. 3, pp. 347–357, 2012.
[23] D. Chen, L. Wang, M. Tian, J. Tian, S. Wang, C. Bian, and X. Li, "Massively parallel modelling & simulation of large crowd with GPGPU," *J. Supercomput.*, vol. 63, no. 3, pp. 675–690, 2013.
[24] D. Chen, X. Li, L. Wang, S. U. Khan, J. Wang, K. Zeng, C. Cai, "Fast and scalable multi-way analysis of massive neural data," *IEEE Trans. Comput.*, vol. 64, no. 3, pp. 707–719, Mar. 2015.
[25] D. Chen, D. Li, M. Xiong, H. Bao, and X. Li, "GPGPU-aided ensemble empirical-mode decomposition for EEG analysis during anesthesia," *IEEE Trans. Inf. Technol. Biomed.*, vol. 14, no. 6, pp. 1417–1427, Nov. 2010.

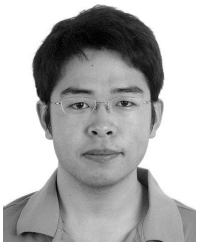
JINRONG JIANG received the B.S. degree in computational mathematics and economics from Peking University, Beijing, China, in 1999, the M.S. degree in computational mathematics from the Institute of Computing, Chinese Academy of Sciences, Beijing, China, and the Ph.D. degree in computer software and theory from the Institute of Software, Chinese Academy of Sciences, where he is currently a Research Fellow with the Computer Network Information Center. His research interests include the development of earth system model, parallel computing, and computer architecture. Dr. Jiang's awards and honors include the computational earth science young talents award from Tsinghua University and silver award of best application in PAC2017.

PENGFEI LIN received the Ph.D. degree from the Institute of Atmospheric Physics, Chinese Academy of Sciences, where he is currently with the LASG Laboratory, Institute of Atmospheric Physics. His research interests include ocean model, and the development and application of ocean-atmosphere coupling model.

JOEY WANG received the M.S. degree from the Beijing University of Posts and Telecommunications. He is currently an Engineer in high performance computing with NVIDIA. His research interests include the development and optimization of CUDA algorithm.

HAILONG LIU received the Ph.D. degree in dynamical meteorology from the Institute of Atmospheric Physics, Chinese Academy of Sciences, where he is currently a Research Fellow with the State Key Laboratory of Numerical Modeling for Atmospheric Sciences and Geophysical Fluid Dynamics. His research interests include the development, application, and research of ocean circulation model.

XUEBIN CHI received the Ph.D. and M.S. degrees in computational mathematics from the Computing Center, Chinese Academy of Sciences, where he is currently a Research Fellow and the Deputy Director of the Computer Network Information Center. His research interests include high performance computing, machine learning, grid technology, and visualization.



HUIQUN HAO received the M.S. degree in computer science from The University of Manchester. He is currently an Engineer with the Computer Network Information Center, Chinese Academy of Sciences. His research interests include high performance computing and the parallel optimization of earth system model.

YUZHU WANG received the Ph.D. degree in engineering from the University of Chinese Academy of Sciences. He is currently a Lecturer with the School of Information Engineering, China University of Geosciences (Beijing), Beijing, China. His research interests include high performance computing and big data.

WU WANG received the Ph.D. degree in computer software and theory from the Computer Network Information Center, Chinese Academy of Sciences, where he is currently an Associate Research Fellow. His research interests include high performance computing and parallel algorithm.

LINGHAN ZHANG received the M.S. degree in computer software and theory from the Computer Network Information Center, Chinese Academy of Sciences. His research interests include high performance computing and CUDA programming.