The Journal of Supercomputing (2021) 77:4698–4717
https://doi.org/10.1007/s11227-020-03451-3

GPUs-RRTMG_LW: high-efficient and scalable computing for a longwave radiative transfer model on multiple GPUs

Yuzhu Wang[1] · Mingxin Guo[1] · Yuan Zhao[1] · Jinrong Jiang[2]

Accepted: 11 October 2020 / Published online: 23 October 2020

Abstract

Atmospheric radiation physical process plays an important role in climate simulations. As a radiative transfer scheme, the rapid radiative transfer model for general circulation models (RRTMG) is widely used in weather forecasting and climate simulation systems. However, its expensive computational overhead poses a severe challenge to system performance. Therefore, improving the radiative transfer model's computational performance has significant scientific research and practical value. Numerous radiative transfer models have benefited from a widely used and powerful GPU. Nevertheless, few of them have exploited CPU/GPU cluster resources within heterogeneous high-performance computing platforms. In this paper, we endeavor to demonstrate an approach that runs a large-scale, computationally intensive, longwave radiative transfer model on a GPU cluster. First, a CUDA-based acceleration algorithm of the RRTMG longwave radiation scheme (RRTMG_LW) on multiple GPUs is proposed. Then, a heterogeneous, hybrid programming paradigm (MPI+CUDA) is presented and utilized with the RRTMG_LW on a GPU cluster. After implementing the algorithm in CUDA Fortran, a multi-GPU version of the RRTMG_LW, namely GPUs-RRTMG_LW, was developed. The experimental results demonstrate that the multi-GPU acceleration algorithm is valid, scalable, and highly efficient when compared to a single GPU or CPU. Running the GPUs-RRTMG_LW on a K20 cluster achieved a 77.78× speedup when compared to a single Intel Xeon E5-2680 CPU core.

Keywords High-performance computing · Graphics processing unit · Compute Unified Device Architecture · Radiative transfer

✉ Yuzhu Wang
wangyz@cugb.edu.cn

Extended author information available on the last page of the article

1 Introduction

Due to the massive number of calculations involved, climate models or earth system models need support from high-performance computing (HPC) [1, 2]. Radiative transfer models, which are employed to calculate atmospheric radiative fluxes and heating rates [3], also demand the HPC. Some of the most well-known radiative transfer models are the line-by-line radiative transfer model (LBLRTM) [4, 5], rapid radiative transfer model (RRTM) [6], and rapid radiative transfer model for general circulation models (RRTMG). As an accelerated version of RRTM, the RRTMG can perform computations more efficiently [7, 8]. However, it still demands enormous computing resources for long-term climatic simulation [9–11]. The Chinese Academy of Sciences-Earth System Model (CAS-ESM) [12–14] uses the Institute of Atmospheric Physics (IAP) of CAS Atmospheric General Circulation Model Version 4.0 (IAP AGCM4.0) [15, 16] as its atmospheric component model. Here, the IAP AGCM4.0 uses the RRTMG as its radiative parameterization scheme.

Large-scale numerical simulations are typically performed on CPU clusters. However, because of the low power consumption, high memory bandwidth, highly parallel processing, and many-core processor capabilities of graphics processing units (GPUs), the HPC has undergone a paradigm shift from CPU-based computing to GPU-based computing [17–21]. It has become increasingly more common to run climate models on GPUs to perform highly efficient computations with low energy consumption [22, 23]. For instance, the RRTM longwave radiation scheme (RRTM_LW) in the Weather Research and Forecasting (WRF) model was accelerated with CUDA (NVIDIA's Compute Unified Device Architecture) Fortran on C1060 GPUs and attained a 10× performance improvement [24]. A nearly 10× speedup for a computationally intensive portion of the WRF was obtained on an NVIDIA 8800 GTX [25]. In our previous study, the RRTMG longwave radiation scheme (RRTMG_LW) was accelerated on only one K20 GPU [26, 27].

In the aforementioned studies, radiation transfer models were accelerated on a single GPU. Currently, supercomputers or CPU/GPU heterogeneous HPC systems usually have thousands of CPU and GPU nodes. Radiation transfer models should be run on dozens of GPUs, at a minimum, to make full use of these GPU nodes. Moreover, running the current RRTMG on one GPU is still time-consuming when used for long-term simulations, so its acceleration algorithm should be studied to achieve more efficient computing on multiple GPUs. Thus, the present paper focuses on accelerating the RRTMG_LW on multiple GPUs. A CUDA-based acceleration algorithm of the RRTMG_LW on multiple GPUs is proposed. The proposed algorithm enables massively parallel calculations of the RRTMG_LW on multiple GPUs of a supercomputer. Then, a multi-GPU acceleration version of the RRTMG_LW, namely GPUs-RRTMG_LW, is built. The experimental results demonstrate that running the GPUs-RRTMG_LW on 16 K20 GPUs obtained a 77.78× speedup.

The main contributions of this study are as follows:

(1) To further accelerate the RRTMG_LW, a multi-GPU accelerating algorithm based on CUDA Fortran is proposed. The proposed algorithm adapts well to

multiple GPUs and nodes. Moreover, it can also be generalized to accelerate the RRTMG shortwave radiation scheme (RRTMG_SW).

(2) The GPUs-RRTMG_LW can run on a GPU cluster and shows excellent computational capability. To some extent, the more efficient computation of the GPUs-RRTMG_LW supports large-scale and real-time computing of the CAS-ESM. In addition, after implementing the GPUs-RRTMG_LW on multiple GPU nodes, the highly efficient parallel processing allows the CAS-ESM to run on a CPU/GPU heterogeneous supercomputer with thousands of nodes.

The remainder of this paper is organized as follows. Section 2 presents representative approaches that aim to accelerate physical parameterization schemes on multiple GPUs. Section 3 introduces the RRTMG_LW model and GPU environment. Section 4 describes the CUDA-based 3D acceleration algorithm for the RRTMG_LW on a single GPU. Section 5 details the MPI+CUDA acceleration algorithm for the RRTMG_LW on multiple GPUs. Section 6 evaluates the performance of the GPUs-RRTMG_LW in terms of runtime efficiency and speedup and discusses some of the problems arising during the experiments. The last section concludes the paper with a summary and proposal for future work.

2 Related work

In recent years, a fair amount of work has been devoted to accelerate physical parameterization schemes and climatic system models by using GPUs. Many GPU-based acceleration techniques have been proposed, and they can be divided into several categories: single GPU-based acceleration, multi-GPUs-based acceleration, and CPU/multi-GPU clusters-based acceleration. We are committed to speeding up the process on CPU/multi-GPU clusters. Here, we provide a brief summary of related categories about prior work.

Mielikainen et al. refactored the RRTMG_LW code. Without I/O transfer, their GPU version achieved a speedup of 127× on a single Tesla K40 GPU compared to its CPU version on an Intel Xeon E5-2603 [28]. The RRTMG_SW was written in CUDA C [29] instead of the previous Fortran code. Compared to its single-threaded Fortran counterpart running on an Intel Xeon E5-2603, the RRTMG_SW based on CUDA C had a 202× speedup on a single Tesla K40 GPU [30].

Running the RRTM_LW on a GTX480 obtained a 27.6× speedup compared with the baseline wall-clock time [3]. The WRF Single Moment 5-class (WSM5) microphysics achieved a 9.4× performance increase even without systematically optimizing the GPU code [31]. The WRF Single Moment 6-class (WSM6) microphysics scheme was accelerated with CUDA C. Here, the CUDA programming model is used to convert the original WSM6 module into GPU programs. Its GPU version obtained a greater than 216× speedup when compared to its CPU serial version [32]. The GRAPES' WSM6 scheme, using the NVIDIA CUDA programming model, exploited its fine-grained data parallelism. The implementation achieved a greater than 140× performance improvement over a single CPU version [33].

The WRF Goddard shortwave radiance scheme was accelerated on two NVIDIA GTX 590s. Without taking I/O transfer times into account, the GPU implementation achieved a 141× speedup [34]. The RRTM_LW in the GRAPES_Meso model was rewritten in CUDA Fortran, and a 14.3× speedup was obtained. The experiments were carried out on a multi-GPU platform and could be extended to GPU clusters [9]. The double-precision version of the ODAS (a transmittance algorithm, which is available in the community radiative transfer model) obtained a 201× speedup on two NVIDIA GTX 590s compared to its single-threaded Fortran code [35]. The WRF Kessler cloud microphysics scheme obtained a 132× speedup on 4 GPUs compared to its single-threaded CPU version [36]. The WRF WSM5 microphysics scheme was accelerated by 357× on four GPUs [37]. The horizontal diffusion method in the WRF was accelerated approximately 3.5 times using two Tesla K40m GPUs compared with the single-GPU version [38]. Lu et al. utilized the MPI+OpenMP/CUDA programming pattern to simulate radiation physics on a large GPU cluster and investigated the computational efficiency of the RRTM_LW CPU/GPU implementation [39, 40].

Despite the excellent results of the aforementioned studies, few of them have exploited both the CPU and GPU computational resources within large GPU clusters. In this paper, a parallel programming model, MPI+CUDA, is presented when simulating the RRTMG_LW in a CPU/multi-GPU computing environment.

3 Model description and GPU overview

3.1 RRTMG_LW model

With an objective of higher efficiency while less loss of accuracy, the RRTM model was modified to create the RRTMG [41, 42], which is a correlated k-distribution band model for the calculation of longwave and shortwave atmospheric radiative fluxes and heating rates [43]. The correlated k-distribution method and *g* points in the RRTMG are described in [26, 44]. The radiation flux and heating/cooling rate for calculating radiative transfer through a planetary atmosphere is described detailedly in [28].

3.2 RRTMG_LW code structure

According to the code test, the subroutine *rrtmg_lw* in the RRTMG_LW module accounts for most of the time proportion consumed by each subroutine and is likely to become the bottleneck of system performance [26, 27]. The *rrtmg_lw* calls the following five subroutines.

(a) The subroutine *inatm* is used to read the atmospheric profile from the GCM for use in the RRTMG_LW and define other input parameters;
(b) The subroutine *cldprmc* is used to set cloud optical depth for the Monte Carlo-independent column approximation (McICA) based on the input cloud properties;

(c) The subroutine *setcoef* is used to calculate information needed by the radiative transfer routine which is specific to this atmosphere, especially some of the coefficients and indices needed to compute optical depths, by interpolating data from stored reference atmospheres;
(d) The subroutine *taumol* is used to calculate gaseous optical depths and Planck fractions for each of the 16 spectral bands;
(e) The subroutine *rtrnmc* (for both clear and cloudy profiles) is used to perform the radiative transfer calculation using the McICA to represent sub-grid-scale cloud variability.

Algorithm 1 shows the computing procedure of *rrtmg_lw*. The *rrtmg_lw* took most computing time of the RRTMG_LW, so the study target was to use GPUs to accelerate the *inatm*, *cldprmc*, *setcoef*, *taumol*, and *rtrnmc* subroutines.

Algorithm 1 Computing procedure of *rrtmg_lw*

```
subroutine rrtmg_lw(parameters)
    //iplon is the column loop index
    //ncol is the number of horizontal columns
1.  do iplon=1, ncol
2.     call inatm(parameters)
3.     call cldprmc(parameters)
4.     call setcoef(parameters)
5.     call taumol(parameters)
6.     if aerosol is active then
         //combine gaseous and aerosol optical depths
         //k is the layer loop index and ig is the g-point loop index
         //taug is the gaseous optical depths and taua is the aerosol optical depth
         //taut is the gaseous + aerosol optical depths
         //ngb is the band index for each new g-interval
7.       taut(k, ig) = taug(k, ig) + taua(k, ngb(ig))
8.     else
9.       taut(k, ig) = taug(k, ig)
10.    end if
11.    call rtrnmc(parameters)
12.    Transfer fluxes and heating rate to output arrays
13.end do
end subroutine
```

3.3 Overview of GPU and CUDA

Figure 1 illustrates the hardware architecture of a GPU. It is organized into an array of highly threaded streaming multiprocessors (SMs). Each SM has a number of streaming processors (SPs) that share control logic and instruction cache. As a general purpose parallel computing architecture, CUDA facilitates creating a software environment that fully utilizes the cores of many GPUs in a massively parallel fashion. CUDA defines functions or subroutines as 'kernels' executed on the GPU (the 'device'). Normally, the CPU (the 'host') invokes the kernels of an application. Each kernel is executed by CUDA threads, which are organized into a three-level hierarchy [45], as shown in Fig. 2. The top level is a grid consisting of thread blocks. Each thread block has a group of threads that share data efficiently through a fast shared memory.

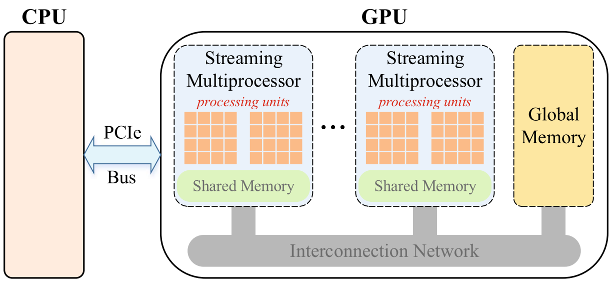


Fig. 1 Hardware architecture of a modern GPU

4 CUDA-based 3D acceleration of RRTMG_LW on a single GPU

The RRTMG_LW uses a collection of 3D cells to represent the atmosphere. Its 1D acceleration algorithm performs domain decompositions in the horizontal direction. Its 2D acceleration algorithm performs domain decompositions in the horizontal and vertical directions. In the RRTMG_LW, the total number of *g* points is 140.

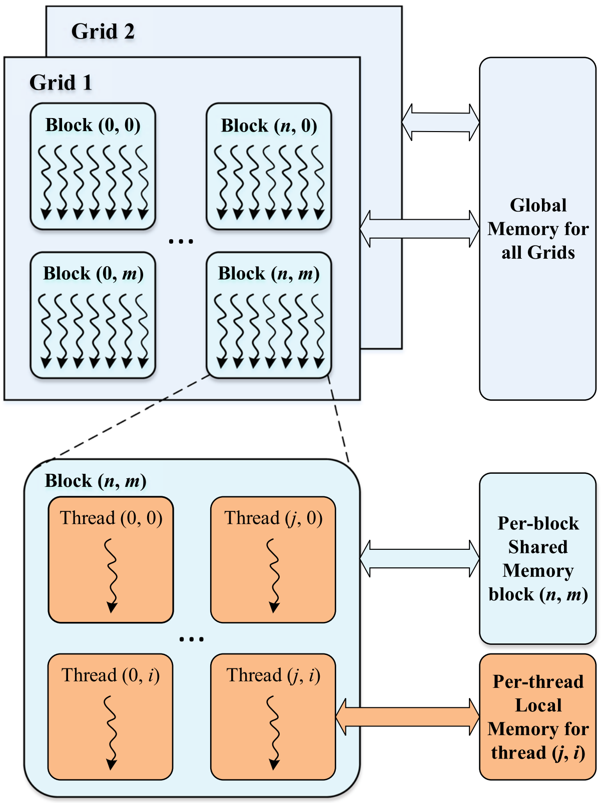


Fig. 2 Hierarchy of threads and memory in CUDA

Therefore, there are iterative computations for each *g* point in *inatm*, *taumol*, and *rtrnmc*. For example, the computation of 140 *g* points is executed by a do-loop in the GPU-based acceleration implementation of 1D *rtrnmc_d*. To achieve more fine-grained parallelism, 140 CUDA threads can be assigned to run the kernels *inatm_d*, *taumol_d*, and *rtrnmc_d*. Thus, on the basis of the 2D algorithm, the 3D parallel strategy is further accelerating *inatm_d*, *taumol_d*, and *rtrnmc_d* in the *g-point* dimension. Figure 3 illustrates the domain decomposition in the *g-point* dimension for the RRTMG_LW accelerated on a GPU. The 3D acceleration algorithm is illustrated in Algorithm 2 and described as follows:

(1) In the 3D acceleration algorithm, *inatm* consists of five kernels (*inatm_d1*, *inatm_d2*, *inatm_d3*, *inatm_d4*, and *inatm_d5*). Due to data dependency, a piece of code in *inatm* can be parallel only in the horizontal or vertical direction, so the kernel *inatm_d4* uses 1D decomposition. The kernels *inatm_d1*, *inatm_d2*, and *inatm_d5* use 2D decomposition, while the kernel *inatm_d3* uses 3D decomposition. Due to the requirement of data synchronization, *inatm_d1* and *inatm_d2* cannot be merged into one kernel.
(2) The kernel *cldprmc_d* still uses 1D decomposition.
(3) Similarly, the kernel *setcoef_d1* uses 2D decomposition, and the kernel *setcoef_d2* uses 1D decomposition.
(4) The kernel *taumol_d* uses 3D decomposition. In *taumol_d*, 16 subroutines with the device attribute are invoked.
(5) Similarly, *rtrnmc* consists of 11 kernels (*rtrnmc_d1*–*rtrnmc_d11*). Here, *rtrnmc_d1*, *rtrnmc_d4*, *rtrnmc_d8*, *rtrnmc_d10*, and *rtrnmc_d11* use 1D decomposition. Furthermore, *rtrnmc_d2* and *rtrnmc_d9* use 2D decomposition in the horizontal and vertical directions. In addition, *rtrnmc_d5* and *rtrnmc_d6* use 2D decomposition in the horizontal direction and *g-point* dimension. Finally, *rtrnmc_d3* and *rtrnmc_d7* use 3D decomposition.

In Algorithm 2, for 1D acceleration, n is the number of threads in each thread block, while $m = \lceil(\mathbf{real})ncol/n\rceil$ is the number of blocks utilized in each kernel grid. For 2D and 3D acceleration, the *tBlock* defines the number of threads utilized

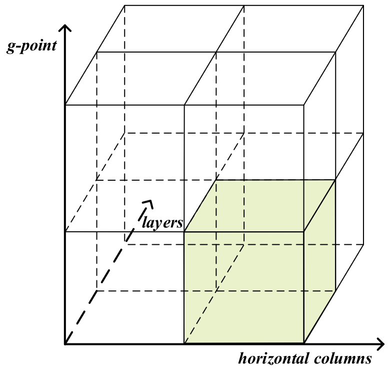


Fig. 3 Schematic diagram of the decomposition in the *g-point* dimension for the RRTMG_LW in the GPU acceleration

in each thread block of the *x*, *y*, and *z* dimensions by the derived type *dim3*. Furthermore, the *grid* defines the number of blocks in the *x*, *y*, and *z* dimensions by *dim3*.

The 1D, 2D, and 3D acceleration algorithms of the RRTMG_LW on one GPU were proposed in our previous study [26, 27]. After implementing the algorithms in CUDA Fortran, the GPU-RRTMG_LW has been developed and can run on a GPU. In the CAS-ESM, the IAP AGCM4.0 has a $1.4° \times 1.4°$ horizontal resolution and 51 levels in the vertical direction, so the RRTMG_LW has 128×256 horizontal grid points. If one GPU is applied, in theory, $128 \times 256 \times 51 \times 140$ CUDA threads will be required for each 3D kernel.

Algorithm 2 3D GPU-RRTMG_LW

```
subroutine rrtmg_lw_d3(parameters)
1. Copy input data to GPU device
    //Call inatm_d1 with 2D decomposition
2. call inatm_d1 <<< grid1, tBblock1 >>>(parameters)
    //Call inatm_d2 with 2D decomposition
3. call inatm_d2 <<< grid1, tBblock1 >>>(parameters)
    //Call inatm_d3 with 3D decomposition
4. call inatm_d3 <<< grid2, tBblock2 >>>(parameters)
    //Call inatm_d4 with 1D decomposition
5. call inatm_d4 <<< m, n >>>(parameters)
    //Call inatm_d5 with 2D decomposition
6. call inatm_d5 <<< grid1, tBblock1 >>>(parameters)
    //Call cldprmc_d with 1D decomposition
7. call cldprmc_d <<< m, n >>>(parameters)
    //Call setcoef_d1 with 2D decomposition
8. call setcoef_d1 <<< grid, tBblock >>>(parameters)
    //Call setcoef_d2 with 1D decomposition
9. call setcoef_d2 <<< m, n >>>(parameters)
    //Call taumol_d with 3D decomposition
10.call taumol_d <<< grid2, tBblock2 >>>(parameters)
    //Call rtrnmc_d1 with 1D decomposition
11.call rtrnmc_d1 <<< m, n >>>(parameters)
    //Call rtrnmc_d2 with 2D decomposition
12.call rtrnmc_d2 <<< grid1, tBblock1 >>>(parameters)
    //Call rtrnmc_d3 with 3D decomposition
13.call rtrnmc_d3 <<< grid2, tBblock2 >>>(parameters)
    //Call rtrnmc_d4 with 1D decomposition
14.call rtrnmc_d4 <<< m, n >>>(parameters)
    //Call rtrnmc_d5 with 2D decomposition in horizonal and g-point dimensions
15.call rtrnmc_d5 <<< grid3, tBblock3 >>>(parameters)
    //Call rtrnmc_d6 with 2D decomposition in horizonal and g-point dimensions
16.call rtrnmc_d6 <<< grid3, tBblock3 >>>(parameters)
    //Call rtrnmc_d7 with 3D decomposition
17.call rtrnmc_d7 <<< grid2, tBblock2 >>>(parameters)
    //Call rtrnmc_d8 with 1D decomposition
18.call rtrnmc_d8 <<< m, n >>>(parameters)
    //Call rtrnmc_d9 with 2D decomposition
19.call rtrnmc_d9 <<< grid1, tBblock1 >>>(parameters)
    //Call rtrnmc_d10 with 1D decomposition
20.call rtrnmc_d10 <<< m, n >>>(parameters)
    //Call rtrnmc_d11 with 1D decomposition
21.call rtrnmc_d11 <<< m, n >>>(parameters)
22.Copy result to host
    //Judge whether atmospheric horizontal profile data is completed//
23.if it is not completed goto 1
end subroutine
```

5 MPI+CUDA acceleration algorithm of RRTMG_LW on multiple GPUs

5.1 Parallel architecture

The current CAS-ESM, which is implemented by adopting MPI technology, typically runs on dozens of compute nodes. Once the GPU-RRTMG_LW is integrated into the CAS-ESM, it also has to run on multiple compute nodes and GPUs. Generally, supercomputers or large-scale clusters have hundreds of compute nodes, each having two or more GPUs. To make full use of multi-core and multi-GPU supercomputers and further improve the computational performance of the GPU-RRTMG_LW, this study adopted a parallel architecture with an MPI+CUDA hybrid paradigm, as shown in Fig. 4. Hence, the GPU-RRTMG_LW can run on multiple GPUs, whereas the other code of the CAS-ESM can run on multiple CPUs.

5.2 GPUs-RRTMG_LW algorithm

In the IAP AGCM4.0 of the CAS-ESM system, the computation of its physical parameterizations features the characteristics of a vertical single-column model. Therefore, when running the CAS-ESM on multiple CPU cores, the computation task of physical parameterizations is decomposed in the horizontal direction. As one of these physical parameterizations, the 3D global domain of the GPU-RRTMG_LW is decomposed by latitude and longitude. More specifically, the decomposition of the 3D atmospherical global grid is implemented by using MPI technology. Then, each MPI rank will finish the computation task on its own sub-grid points. For example, if the CAS-ESM is run on 4 CPU cores, the number of each sub-grid point in the horizontal direction is $128 \times 256/4 = 8192$. Each CPU core drives a GPU, so 4 GPUs will be required here, as shown in Fig. 5. It means that each GPU will finish the computing of 8192 grid points in the horizontal direction at each time step. Due to the limitation of global memory on a GPU, a K20 GPU can only compute 2048 horizontal grid points. Thus, the 8192 points will be divided into 4 chunks, each having 2048 points. In other words, a K20 GPU will perform the computation of 8192 points in 4 iterations at each time step.

After decomposing the 3D atmospherical global grid, this study uses MPI to implement collaboration and communication among multiple GPUs. Figure 6 illustrates the flow of the GPUs-RRTMG_LW acceleration algorithm on multiple GPUs and nodes. The detailed acceleration algorithm is as follows.

(1) The CAS-ESM runs concurrently on multiple CPU cores using MPI. The 3D atmospherical global grid is decomposed into sub-grids. Each CPU core is responsible for the computing task on its own sub-grid points. Then, each CPU core starts a GPU and sends the input data of the GPUs-RRTMG_LW to its corresponding GPU.
(2) Each GPU initializes the run environment and allocates space for its variables or arrays. After receiving the input data from its corresponding CPU core, the *ncol*

CUDA threads in each GPU will execute the computation of radiative transfer concurrently for each grid point on its own chunk. Then, each GPU will send the computing results to its corresponding CPU.

(3) Each CPU core will receive the computing results. If all of the computing tasks are not finished at a time step, the algorithm will continue to run from the first step.

5.3 GPUs-RRTMG_LW implementation

The implementation of the RRTMG_LW acceleration algorithm on multiple GPUs is illustrated in Table 1 and described as follows.

(1) If each GPU node invokes two or more GPUs, arrays with device attribution for input or output data must be allocated memory dynamically. Thus, the dynamic allocation memory is adopted in the algorithm implementation.

(2) The NVIDIA CUDA library *cudaSetDevice* sets which device (GPU) to be used for GPU code executions. Calling the library is to realize multi-GPU computation on each GPU node.

(3) The NVIDIA CUDA library *cudaDeviceSynchronize* is used to wait for compute device to finish. Calling the library is to realize multi-GPU synchronous computation.

6 Experimental results and discussion

To evaluate the performance of the proposed algorithm, experimental studies were conducted. The results are described below.

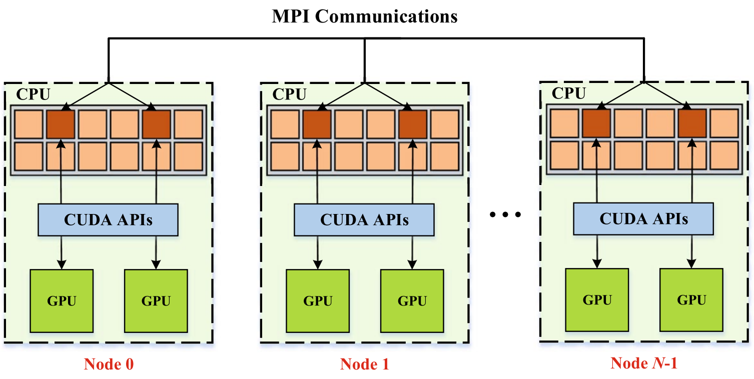


Fig. 4 Parallel architecture of the GPU-RRTMG_LW on multiple GPUs

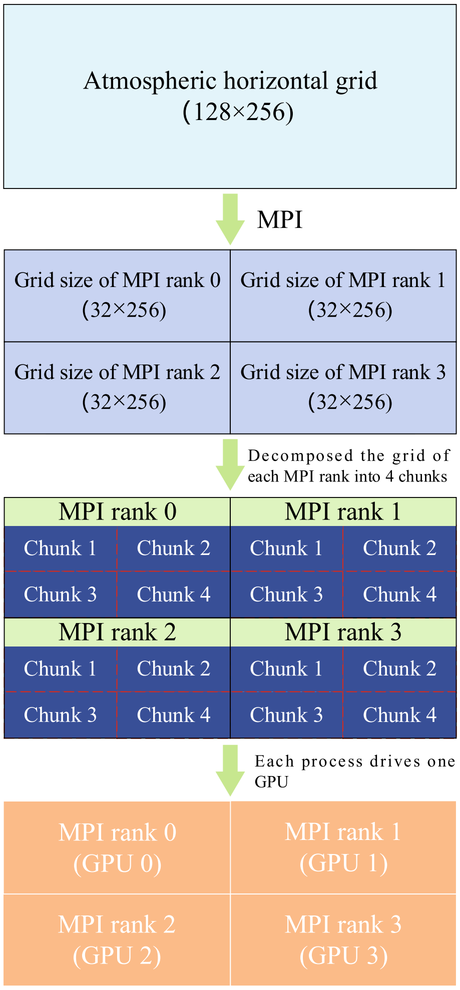

Atmospheric horizontal grid (128×256)
MPI
Grid size of MPI rank 0 (32×256)
Grid size of MPI rank 1 (32×256)
Grid size of MPI rank 2 (32×256)
Grid size of MPI rank 3 (32×256)
Decomposed the grid of each MPI rank into 4 chunks
MPI rank 0
MPI rank 1
Chunk 1
Chunk 2
Chunk 3
Chunk 4
MPI rank 2
MPI rank 3
Each process drives one GPU
MPI rank 0 (GPU 0)
MPI rank 1 (GPU 1)
MPI rank 2 (GPU 2)
MPI rank 3 (GPU 3)

Fig. 5 Decomposition of the global grid in the horizontal direction when running the GPUs-RRTMG_LW on four CPU cores and four GPUs

6.1 Experimental setup

This paper conducted an ideal global climate simulation for one model day to fully investigate the proposed algorithm. In this experiment, the time step of the GPUs-RRTMG_LW was one hour. The experiment ran on a K20 cluster in the Computer Network Information Center of CAS, which has 30 GPU nodes. Each GPU node has two Intel Xeon E5-2680 v2 processors and two NVIDIA Tesla K20 GPUs. Twenty CPU cores in each GPU node share 64 GB DDR3 system memory through Quick-Path Interconnect. The basic compiler is the PGI Fortran compiler Version 14.10 that supports CUDA Fortran. Table 2 lists its detailed configurations. The serial RRTMG_LW was executed on an Intel Xeon E5-2680 v2 processor of the K20 cluster.

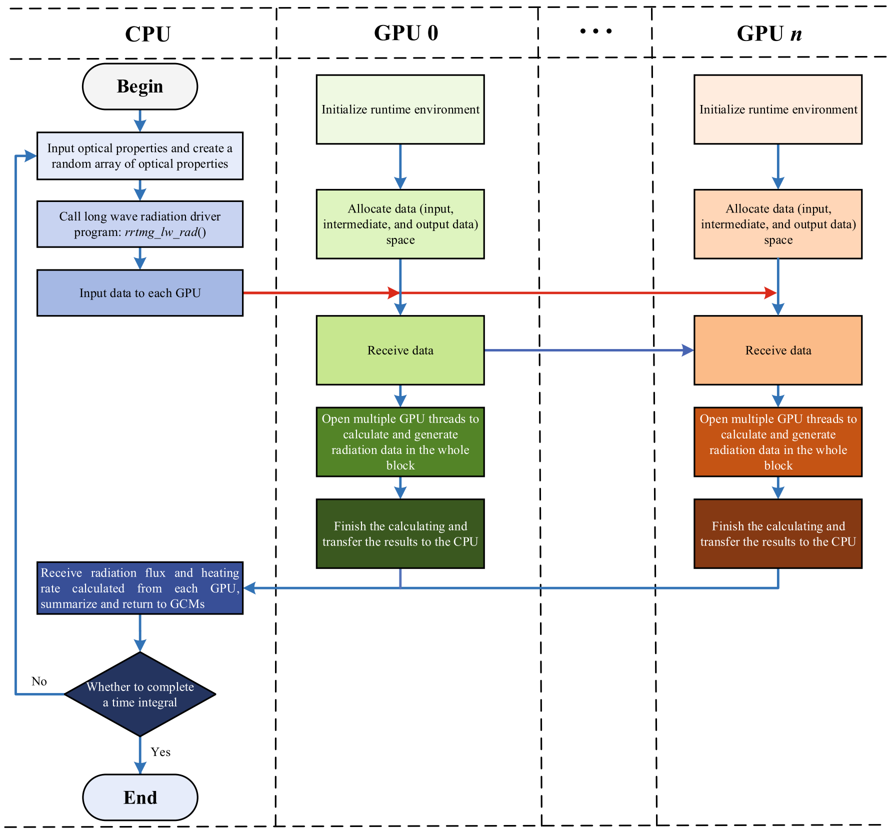

CPU
GPU 0
• • •
GPU *n*
Begin
Input optical properties and create a random array of optical properties
Call long wave radiation driver program: *rrtmg_lw_rad*()
Input data to each GPU
Receive radiation flux and heating rate calculated from each GPU, summarize and return to GCMs
Whether to complete a time integral
No
Yes
End
Initialize runtime environment
Allocate data (input, intermediate, and output data) space
Receive data
Open multiple GPU threads to calculate and generate radiation data in the whole block
Finish the calculating and transfer the results to the CPU

Fig. 6 Flow of the GPUs-RRTMG_LW acceleration algorithm

6.2 Performance comparison of 1D and 3D GPUs-RRTMG_LW

Table 3 shows the runtime of the serial RRTMG_LW on one core of an Intel Xeon E5-2680 v2 processor. The computing time of the RRTMG_LW on the CPU or GPU, T_{rrtmg_lw}, is calculated with the following formula:

$$T_{rrtmg\_lw} = T_{inatm} + T_{cldprmc} + T_{setcoef} + T_{taumol} + T_{rtrnmc} + T_{I/O},$$

where T_{inatm} is the computing time of the subroutine *inatm* or kernel *inatm_d*; moreover, $T_{cldprmc}$, $T_{setcoef}$, T_{taumol}, and T_{rtrnmc} are the corresponding computing time of the other kernels; $T_{I/O}$ is the I/O transfer time between the CPU and GPU.

To evaluate the acceleration performance of the GPUs-RRTMG_LW on multiple GPUs, the performance of the 1D GPUs-RRTMG_LW on multi-GPUs was also evaluated. Table 3 also portrays the runtime and speedup of the 1D GPUs-RRTMG_LW on multiple K20 GPUs when each GPU node of the cluster invokes one K20 GPU. Table 4 presents the runtime and speedup of the 3D GPUs-RRTMG_LW on

Table 1 Implementation of the GPUs-RRTMG_LW

```
    //myrank is the process ID or MPI rank
1. Integer : : myrank, dev, ierr, istat
2. Dynamically allocate memory for related 1D, 2D, and 3D arrays
3. dev = mod(myrank,2)
4. ierr = cudaSetDevice(dev)
5. Copy input data to GPU device
6. call inatm_d1<<< grid1, tBblock1 >>>(parameters)
7. call inatm_d2<<< grid1, tBblock1 >>>(parameters)
8. call inatm_d3<<< grid2, tBblock2 >>>(parameters)
9. call inatm_d4<<< m, n >>>(parameters)
10.call inatm_d5<<< grid1, tBblock1 >>>(parameters)
11.call cldprmc_d<<< m, n >>>(parameters)
12.call setcoef_d1<<< grid, tBblock >>>(parameters)
13.call setcoef_d2<<< m, n >>>(parameters)
14.call taumol_d<<< grid2, tBblock2 >>>(parameters)
15.call rtrnmc_d1<<< m, n >>>(parameters)
16.call rtrnmc_d2<<< grid1, tBblock1 >>>(parameters)
17.call rtrnmc_d3<<< grid2, tBblock2 >>>(parameters)
18.call rtrnmc_d4<<< m, n >>>(parameters)
19.call rtrnmc_d5<<< grid3, tBblock3 >>>(parameters)
20.call rtrnmc_d6<<< grid3, tBblock3 >>>(parameters)
21.call rtrnmc_d7<<< grid2, tBblock2 >>>(parameters)
22.call rtrnmc_d8<<< m, n >>>(parameters)
23.call rtrnmc_d9<<< grid1, tBblock1 >>>(parameters)
24.call rtrnmc_d10<<< m, n >>>(parameters)
25.call rtrnmc_d11<<< m, n >>>(parameters)
26.istat = cudaDeviceSynchronize()
27.Copy result to host
28.Deallocate memory for related 1D, 2D, and 3D arrays
      //Judge whether atmospheric horizontal profile data is completed
29.if it is not completed goto 2
```

multiple K20 GPUs when each GPU node of the cluster invokes one K20 GPU. Some conclusions and analysis are described as below.

(1) Increasing the number of GPUs can reduce the runtime and improve speedup. When the 1D GPUs-RRTMG_LW ran on 16 K20 GPUs, it achieved a speedup of 51.28× as compared to its counterpart running on one CPU core of an Intel Xeon E5-2680 v2.
(2) With incremental increases in the number of GPUs, the 3D GPUs-RRTMG_LW resulted in a similar rule. When the 3D GPUs-RRTMG_LW ran on 16 K20 GPUs, it achieved a speedup of 77.78×. The 3D GPUs-RRTMG_LW has better

Table 2 Configurations of the K20 GPU cluster

Items	Specification of CPU
CPU	E5-2680 v2@2.8 GHz
Operating system	CentOS 6.4
Items	Specification of GPU
GPU	Tesla K20
CUDA Cores	2496
Standard memory	5 GB
Memory bandwidth	208 GB/s
CUDA version	6.5

Table 3 Runtime and speedup of the CAS-ESM 1D GPUs-RRTMG_LW on multiple GPUs when each GPU node of the cluster invokes one K20 GPU

Nodes (CPU Cores)	GPUs	Runtime (s)	Speedup
1 (1)	0	647.12	1
4 (4)	4	44.04	14.69
8 (8)	8	23.61	27.41
16 (16)	16	12.62	51.28

Here, the block size = 512 and *ncol* = 2048

acceleration algorithm than the 1D GPUs-RRTMG_LW, so it can obtain a higher speedup.

6.3 Performance evaluation with different GPU configurations

In the K20 cluster, each GPU node has two Intel Xeon E5-2680 v2 processors (20 CPU cores) and two K20 GPUs. In the experiment of Sect. 6.2, each GPU node invokes one K20 GPU. To make full use of the cluster, each GPU node will invoke two K20 GPUs in the following experiment. Table 5 presents the runtime and speedup of the 3D GPUs-RRTMG_LW on multiple K20 GPUs when each GPU node invokes two K20 GPUs. Some conclusions and analysis are described as below.

(1) Increasing the number of GPUs can reduce the runtime and improve speedup. When the 3D GPUs-RRTMG_LW ran on 16 and 32 K20 GPUs, it achieved a speedup of 60.88× and 76.13×, respectively.
(2) As shown in Tables 4 and 5, the 3D GPUs-RRTMG_LW running on the same number of GPUs obtains a higher speedup when each GPU node invokes one K20 GPU. This is because the data transfer between the CPU and GPU is slower and a communication overhead is produced when two GPUs are invoked in a GPU node.

Table 4 Runtime and speedup of the CAS-ESM 3D GPUs-RRTMG_LW on multiple GPUs when each GPU node of the cluster invokes one K20 GPU

Nodes (CPU Cores)	GPUs	Runtime (s)	Speedup
1 (1)	0	647.12	1
4 (4)	4	35.41	18.28
8 (8)	8	17.24	37.54
16 (16)	16	8.32	77.78

Here, the block size = 512 and *ncol* = 2048; *inatm* and *rtrnmc* are with a 3D decomposition; *cldprmc* is with a 1D decomposition; *setcoef* and *taumol* are with a 2D decomposition

(3) Although the 3D GPUs-RRTMG_LW does not show a perfect performance improvement when each GPU node invokes two K20 GPUs, it can utilize more GPUs and has a stronger scalability.
(4) When 16 nodes and 32 GPUs are utilized in Table 5, the maximum value of the *ncol* is 1024 ($128 \times 256/32$) because of the low resolution of the IAP AGCM4.0 in the CAS-ESM. In theory, if the IAP AGCM4.0 with a higher resolution is developed, the value of the *ncol* can be 2048 and the 3D GPUs-RRTMG_LW will have a speedup of about $120\times$. Therefore, the proposed algorithm can fully support the CAS-ESM with a higher resolution.

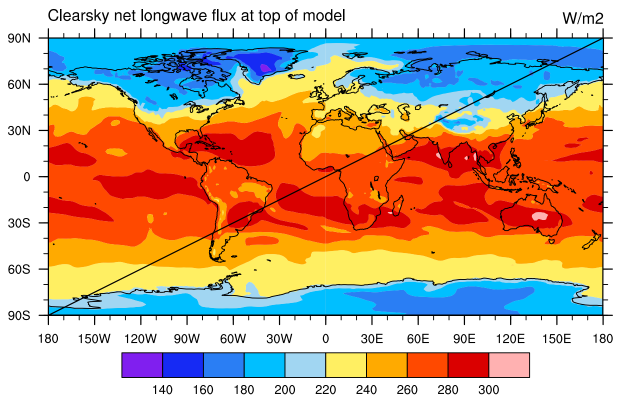
6.4 Error analysis

When accelerating the computational performance of a climate system model, it is of vital importance to ensure that running the model on multiple GPUs can generate the same results within a small tolerance threshold. In a simulation experiment of two model days, Fig. 7 illustrates the impact on the longwave flux at the top of the atmosphere in a clear sky. The outgoing longwave flux achieved by running the CAS-ESM entirely on CPUs is shown in Fig. 7a. The longwave flux differences between the simulations running the CAS-ESM only on CPUs and running the CAS-ESM RRTMG on 16 GPUs are shown in Fig. 7b. The results show that there

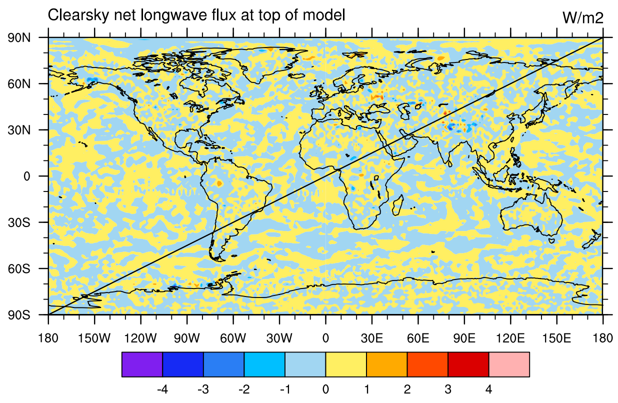
Table 5 Runtime and speedup of the CAS-ESM 3D GPUs-RRTMG_LW on multiple GPUs when each GPU node of the cluster invokes two K20 GPUs

Nodes (CPU Cores)	GPUs	Runtime (s)	Speedup
1 (1)	0	647.12	1
2 (4)	4	43.47	14.89
4 (8)	8	21.48	30.13
8 (16)	16	10.63	60.88
16 (32)	32	8.50	76.13

Here, the block size = 512 and *ncol* = 2048; *inatm* and *rtrnmc* are with a 3D decomposition; *cldprmc* is with a 1D decomposition; *setcoef* and *taumol* are with a 2D decomposition. When the 3D GPUs-RRTMG_LW runs on 32 K20 GPUs, the *ncol* = 1024



(a) Longwave flux simulated by the CAS-ESM RRTMG on CPUs.



(b) Longwave flux differences between the simulations running the CAS-ESM RRTMG only on CPUs and running the CAS-ESM RRTMG on GPUs.

Fig. 7 Impact on the longwave flux at the top of the atmosphere in a clear sky

are minor and negligible differences. Besides the impact of running the 3D GPUs-RRTMG_LW on GPUs, the impact of the slight physics change by running the 3D GPUs-RRTMG_LW code on GPUs also results in these differences.

6.5 Discussion

(1) Zheng et al. proposed an acceleration algorithm for the RRTM_LW in the GRAPES_Meso model on multiple GPUs. Their CUDA Fortran version obtained a 14.3× speedup on 4×NVIDIA Tesla C1060 cards [9]. Compared to their algorithm, our proposed algorithm for the RRTMG_LW in the CAS-ESM has a better speedup. Moreover, our algorithm can run on multiple GPU nodes.

(2) In fact, our algorithm does not attain an ideal speedup when running on multiple nodes and GPUs. There are two main reasons. First, the current IAP AGCM4.0 in the CAS-ESM is with a low resolution, so the RRTMG_LW calculation amount assigned on each GPU is less and the GPU hardware performance is inefficiently utilized. Second, the inevitable I/O transfer cost between the CPU and GPU reduces performance improvement. Thus, the proposed acceleration algorithm will be optimized further to achieve a better performance.

7 Conclusions and future work

Large-scale numerical simulation places an ever-growing demand on the computational performance of HPC infrastructure. Consequently, it is critical to make full use of the computational resources of CPU/GPU clusters. In this paper, a multi-GPU acceleration algorithm for the RRTMG_LW is proposed, and its hybrid programming paradigm (MPI+CUDA) is presented. After implementing the algorithm, the GPUs-RRTMG_LW was developed and integrated into the CAS-ESM as its longwave radiation transfer module, which realized the CPU/GPU heterogeneous parallel computing of the CAS-ESM. Moreover, we performed a simulation by exploiting the computational capacities of both CPU and GPU clusters. The experimental results demonstrate that the multi-GPU acceleration algorithm is valid and highly efficient. During a climate simulation of one model day, the GPUs-RRTMG_LW obtained a speedup of 77.78× on a K20 GPU cluster.

The future work mainly includes the following two aspects: (1) The acceleration algorithm will be optimized to further harness the GPU performance. For example, using pinned memory reduces I/O transfer between the CPU and GPU. (2) To fully utilize CPU cores and GPUs, we will adopt the MPI+OpenMP+CUDA hybrid paradigm to improve the acceleration algorithm.

Acknowledgements We would like to acknowledge the contributions of Prof. Minghua Zhang for insightful suggestions on algorithm design. 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 41931183, and in part by the National Key Scientific and Technological Infrastructure project "Earth System Science Numerical Simulator Facility" (EarthLab).

References

1. Xue W, Yang C, Fu H et al (2015) Ultra-scalable CPU-MIC acceleration of mesoscale atmospheric modeling on tianhe-2. IEEE Trans Comput 64(8):2382–2393
2. Wang Y, Jiang J, Zhang J et al (2018) An efficient parallel algorithm for the coupling of global climate models and regional climate models on a large-scale multi-core cluster. J Supercomput 74(8):3999–4018

3. Lu F, Cao X, Song J, et al (2011) GPU computing for longwave radiation physics: a RRTM_LW scheme case study. In: IEEE 9th international symposium on parallel and distributed processing with applications workshops (ISPAW), pp 71–76
4. Clough SA, Iacono MJ, Moncet JL (1992) Line-by-line calculations of atmospheric fluxes and cooling rates: application to water vapor. J Geophys Res Atmos 97(D14):15761–15785
5. Clough SA, Iacono MJ (1995) Line-by-line calculation of atmospheric fluxes and cooling rates II: application to carbon dioxide, ozone, methane, nitrous oxide and the halocarbons. J Geophys Res Atmos 100(D8):16519–16535
6. Mlawer EJ, Taubman SJ, Brown PD et al (1997) Radiative transfer for inhomogeneous atmospheres: RRTM, a validated correlated-k model for the longwave. J Geophys Res Atmos 102(D14):16663–16682
7. Iacono MJ, Mlawer EJ, Clough SA et al (2000) Impact of an improved longwave radiation model, RRTM, on the energy budget and thermodynamic properties of the NCAR community climate model, CCM3. J Geophys Res Atmos 105(D11):14873–14890
8. Iacono MJ, Delamere JS, Mlawer EJ et al (2008) Radiative forcing by long-lived greenhouse gases: calculations with the AER radiative transfer models. J Geophys Res Atmos 113(D13)
9. Zheng F, Xu X, Xiang D et al (2013) GPU-based parallel researches on RRTM module of GRAPES numerical prediction system. J Comput 8(3):550–558
10. Iacono MJ (2015) Enhancing cloud radiative processes and radiation efficiency in the advanced research weather research and forecasting (WRF) model. Atmospheric and Environmental Research, Lexington
11. Morcrette JJ, Mozdzynski G, Leutbecher M (2008) A reduced radiation grid for the ECMWF integrated forecasting system. Mon Weather Rev 136(12):4760–4772
12. Dong X, Su T, Wang J et al (2014) Decadal variation of the Aleutian low-icelandic low seesaw simulated by a climate system model (CAS-ESM-C). Atmos Ocean Sci Lett 7(2):110–114
13. Wang Y, Jiang J, Ye H et al (2016) A distributed load balancing algorithm for climate big data processing over a multi-core CPU cluster. Concurr Comput Pract Exp 28(15):4144–4160
14. Wang Y, Hao H, Zhang J et al (2019) Performance optimization and evaluation for parallel processing of big data in earth system models. Cluster Comput 22:2371–2381
15. Zhang H, Zhang M, Zeng Q (2013) Sensitivity of simulated climate to two atmospheric models: interpretation of differences between dry models and moist models. Mon Weather Rev 141(5):1558–1576
16. Wang Y, Jiang J, Zhang H et al (2017) A scalable parallel algorithm for atmospheric general circulation models on a multi-core cluster. Future Gener Comput Syst 72:1–10
17. Nickolls J, Dally WJ (2010) The GPU computing era. IEEE Micro 30(2):56–69
18. Deng Z, Chen D, Hu Y et al (2012) Massively parallel non-stationary EEG data processing on GPGPU platforms with Morlet continuous wavelet transform. J Internet Serv Appl 3(3):347–357
19. Chen D, Wang L, Tian M et al (2013) Massively parallel modelling & simulation of large crowd with GPGPU. J Supercomput 63(3):675–690
20. Chen D, Li X, Wang L et al (2015) Fast and scalable multi-way analysis of massive neural data. IEEE Trans Comput 64(3):707–719
21. Candel F, Petit S, Sahuquillo J et al (2018) Accurately modeling the on-chip and off-chip GPU memory subsystem. Future Gener Comput Syst 82:510–519
22. Norman M, Larkin J, Vose A et al (2015) A case study of CUDA FORTRAN and OpenACC for an atmospheric climate kernel. J Comput Sci 9:1–6
23. Schalkwijk J, Jonker HJ, Siebesma AP et al (2015) Weather forecasting using GPU-based large-eddy simulations. Bull Am Meteorol Soc 96(5):715–723
24. Ruetsch G, Phillips E, Fatica M (2010) GPU acceleration of the long-wave rapid radiative transfer model in WRF using CUDA Fortran. In: Many–Core and reconfigurable supercomputing conference
25. Michalakes J, Vachharajani M (2008) GPU acceleration of numerical weather prediction. Parallel Process Lett 18(04):531–548
26. Wang Y, Zhao Y, Li W et al (2019) Using a GPU to accelerate a longwave radiative transfer model with efficient CUDA-based methods. Appl Sci 9(19):4039
27. Wang Y, Zhao Y, Jiang J et al (2020) A novel GPU-based acceleration algorithm for a longwave radiative transfer model. Appl Sci 10(2):649
28. Price E, Mielikainen J, Huang M et al (2014) GPU-accelerated longwave radiation scheme of the rapid radiative transfer model for general circulation models (RRTMG). IEEE J Sel Topics Appl Earth Obs Remote Sens 7(8):3660–3667

29. NVIDIA, CUDA C Programming Guide v10.0, Technical Document (2018). Available:https://docs.nvidia.com/pdf/CUDA_C_Programming_Guide.pdf
30. Mielikainen J, Price E, Huang B et al (2016) GPU compute unified device architecture (CUDA)-based parallelization of the RRTMG shortwave rapid radiative transfer model. IEEE J Sel Topics Appl Earth Obs Remote Sens 9(2):921–931
31. Huang M, Huang B, Chang YL et al (2015) Efficient parallel GPU design on WRF five-layer thermal diffusion scheme. IEEE J Sel Topics Appl Earth Obs Remote Sens 8(5):2249–2259
32. Huang M, Huang B, Gu L et al (2015) Parallel GPU architecture framework for the WRF Single Moment 6-class microphysics scheme. Comput Geosci 83:17–26
33. Xiao H, Sun J, Bian X et al (2013) GPU acceleration of the WSM6 cloud microphysics scheme in GRAPES model. Comput Geosci 59:156–162
34. Mielikainen J, Huang B, Huang HLA et al (2012) GPU acceleration of the updated Goddard shortwave radiation scheme in the weather research and forecasting (WRF) model. IEEE J Sel Topics Appl Earth Obs Remote Sens 5(2):555–562
35. Mielikainen J, Huang B, Huang HLA et al (2015) Performance and scalability of the jcsda community radiative transfer model (crtm) on nvidia gpus. IEEE J Sel Topics Appl Earth Obs Remote Sens 8(4):1519–1527
36. Mielikainen J, Huang B, Wang J et al (2013) Compute unified device architecture (CUDA)-based parallelization of WRF Kessler cloud microphysics scheme. Comput Geosci 52:292–299
37. Mielikainen J, Huang B, Huang HLA et al (2012) Improved GPU/CUDA based parallel weather and research forecast (WRF) single moment 5-class (WSM5) cloud microphysics. IEEE J Sel Topics Appl Earth Obs Remote Sens 5(4):1256–1265
38. Solano-Quinde L, Gualan-Saavedra R, Zuiga-Prieto M (2016) Multi-GPU implementation of the Horizontal diffusion method of the weather research and forecast model. In: ACM proceedings of the 7th international workshop on programming models and applications for multicores and Manycores, pp 98–103
39. Lu F, Song J, Cao X et al (2012) CPU/GPU computing for long-wave radiation physics on large GPU clusters. Comput Geosci 41:47–55
40. Lu F, Song J, Yin F et al (2012) Performance evaluation of hybrid programming patterns for large CPU/GPU heterogeneous clusters. Comput Phys Commun 183(6):1172–1181
41. Iacono MJ, Delamere JS, Mlawer EJ et al (2003) Evaluation of upper tropospheric water vapor in the NCAR Community Climate Model (CCM3) using modeled and observed HIRS radiances. J Geophys Res Atmos 108(D2):ACL-1
42. Morcrette JJ, Barker HW, Cole JNS et al (2008) Impact of a new radiation package, McRad, in the ECMWF integrated forecasting system. Mon Weather Rev 136(12):4773–4798
43. Clough SA, Shephard MW, Mlawer EJ et al (2005) Atmospheric radiative transfer modeling: a summary of the AER codes. J Quant Spectrosc Radiat Transf 91(2):233–244
44. Mlawer EJ, Iacono MJ, Pincus R et al (2016) Contributions of the ARM program to radiative transfer modeling for climate and weather applications. AMS Meteorol Monogr 57:15.1–15.19
45. Chen D, Li D, Xiong M et al (2010) GPGPU-aided ensemble empirical-mode decomposition for EEG analysis during anesthesia. IEEE Trans Inf Technol Biomed 14(6):1417–1427

Affiliations

Yuzhu Wang[1] · **Mingxin Guo**[1] · **Yuan Zhao**[1] · **Jinrong Jiang**[2]

Mingxin Guo
guomx@cugb.edu.cn

Yuan Zhao
zhaoyuan_cugb@163.com

Jinrong Jiang
jjr@sccas.cn

1 School of Information Engineering, China University of Geosciences, Beijing 100083, China

2 Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China