IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 15, 2021, accepted June 3, 2021, date of publication June 8, 2021, date of current version June 17, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3087507

GPU-RRTMG_SW: Accelerating a Shortwave Radiative Transfer Scheme on GPU

ZHENZHEN WANG[1], YUZHU WANG[1], XIAOCONG WANG[2], FEI LI[1], CHEN ZHOU[1], HANGTIAN HU[1], AND JINRONG JIANG[3]

[1]School of Information Engineering, China University of Geosciences, Beijing 100083, China
[2]Institute of Atmospheric Physics, Chinese Academy of Sciences, Beijing 100029, China
[3]Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China

Corresponding author: Yuzhu Wang (wangyz@cugb.edu.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 in part by the National Key Scientific and Technological Infrastructure Project "Earth System Science Numerical Simulator Facility" (EarthLab).

ABSTRACT As high performance computing technology continues to develop rapidly, graphics processing units (GPUs) are more widely used for daily computing tasks. However, some applications, such as weather forecasting, require large-scale computation. Thus, due to the computationally intensive characteristics of the rapid radiative transfer model for general circulation models (RRTMG) in the Earth system model, this study uses GPU-related technology to accelerate computation of the model. First, two kinds of algorithms using GPU technology to accelerate the RRTMG shortwave radiation scheme (RRTMG_SW) are proposed. Then, an optimization method for data transmission between host and device is proposed. Finally, after using CUDA Fortran and CUDA C to implement these algorithms, two GPU versions of RRTMG_SW, namely CF-RRTMG_SW (CUDA Fortran version) and CC-RRTMG_SW (CUDA C version), were developed. The experimental results demonstrate that the proposed acceleration algorithms are effective. Without I/O transfer, running CC-RRTMG_SW on a NVIDIA GeForce Titan V GPU achieved a 38.88× speedup when compared to a single Intel Xeon E5-2680 CPU core.

INDEX TERMS High performance computing, graphics processing unit, compute unified device architecture, shortwave radiative transfer.

I. INTRODUCTION

The Earth's weather and climate conditions are determined by the amount of solar radiation and its distribution. Modeling the radiative transfer process, one of the most crucial physical atmospheric processes, requires a high level of computing accuracy. The rapid radiative transfer model for general circulation models (RRTMG) is a radiation model used to calculate longwave and shortwave atmospheric radiant fluxes and heating rates. Moreover, the model utilizes the correlated k-distribution method to perform highly accurate radiation calculations [1]. RRTMG is compute-intensive, so it is necessary to use parallel computing technologies to improve its computing performance [2], [3].

Parallel computing technology based on graphics processing units (GPUs) has the characteristics of high parallelism, multi-threaded and multi-core processors, and high memory bandwidth. Due to its excellent performance, GPU technology is being applied in a growing number of fields [4]–[8]. As the computer unified device architecture (CUDA) [9] introduced by NVIDIA advances rapidly, GPU technology is more broadly utilized in applications with high computational density. Thus, GPU-based computing is currently a research hotspot.

Some studies have focused on using GPUs to accelerate computation in shortwave radiation models. *J. Mielikainen, et al.* used the C language to rewrite the original Fortran code of the RRTMG shortwave radiation scheme (RRTMG_SW). The CUDA C version of RRTMG_SW has a performance improvement of 202× on NVIDIA Tesla K40 compared to its single-threaded Fortran version running on Intel Xeon E5-2603 [10]. Similarly, two GPU-based acceleration algorithms of RRTMG_SW are proposed in this paper. Remarkably different from the previous work, the first acceleration algorithm is implemented by adopting CUDA Fortran [11] to build CF-RRTMG_SW. In the

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

Chinese Academy of Sciences–Earth System Model (CAS–ESM) [12]–[15], the Institute of Atmospheric Physics of CAS Atmospheric General Circulation Model Version 4.0 (IAP AGCM4.0) [16], [17] is its atmospheric component model and uses RRTMG as its radiative parameterization scheme. The CUDA C version of RRTMG_SW (known as CC-RRTMG_SW) is developed to enable CAS-ESM to run on a supercomputer with AMD GPUs. The experimental results demonstrate that the CC-RRTMG_SW without I/O transfer achieved a 38.88× speedup on a NVIDIA GeForce Titan V GPU. In this paper, the two algorithms are described and discussed.

The main contributions of this study are as follows:

(1) A GPU-based acceleration algorithm implemented in CUDA Fortran for RRTMG_SW is proposed. The algorithm can improve the computing efficiency of CAS–ESM RRTMG_SW.

(2) A GPU-based acceleration algorithm implemented in CUDA C for RRTMG_SW is also proposed. CC-RRTMG_SW has better computing performance than CF-RRTMG_SW.

(3) An optimization method for data transmission between the host and device is proposed. The method can significantly improve the performance of CC-RRTMG_SW.

(4) This work skillfully combines several concepts, approaches, techniques and components, such as high performance computing, many-core GPU, CUDA, acceleration technique, shortwave radiation scheme, and earth system model.

In summary, this work supports large-scale and real-time computing for CAS-ESM. The remainder of this paper is organized as follows. Section II introduces the related work on accelerating radiative transfer models. Section III describes the RRTMG_SW model and experimental platforms and then presents a detailed parallelization analysis of the model. Section IV details the two RRTMG_SW acceleration algorithms based on CUDA Fortran and CUDA C, as well as their implementations. Section V details the optimization method for data transfer between CPU and GPU. Section VI evaluates the two algorithms and discusses some problems encountered in the experiment. The final section summarizes the paper and makes suggestions for future work.

II. RELATED WORK

In recent years, researchers around the world have invested considerable effort into using GPU technology to accelerate physical parameterization schemes. GPU-based acceleration technologies, including single GPU acceleration, multi-GPU acceleration, and CPU/GPU cluster-based acceleration, have been proposed.

F. Lu, et al. accelerated RRTM_LW on three different GPU platforms (GTX470, GTX480, and C2050). Its speedup on the three GPUs was 23.2×, 27.6×, and 18.2×, respectively [18]. Subsequently, an MPI + OpenMP/CUDA parallel programming model on large GPU clusters was proposed. By testing the CPU/GPU version of RRTM_LW on a Tianhe-1A supercomputer, the final measured time is much shorter than that of the CPU counterpart [19]. *E. Price, et al.* implemented a GPU-compatible version of RRTMG. Their RRTMG longwave radiation scheme (RRTMG_LW) achieves an acceleration speed of 69× when implemented on a NVIDIA GTX 680 GPU [20] and 127× on a single K40 GPU [21]. *Y. Wang, et al.* developed a GPU version of RRTMG_LW that was implemented in CUDA Fortran and integrated into CAS–ESM [22]–[24].

J. Mielikainen, et al. implemented a GPU version of the WRF Goddard shortwave radiative scheme in CUDA C. Their GPU-based Goddard shortwave radiative scheme with data I/O increased performance by 116× on two NVIDIA GTX 590s, and without I/O, the speedup was 141× [25]. *J. Mielikainen, et al.* also implemented a GPU version of RRTMG_SW in CUDA C. When running on a K40 GPU, their RRTMG_SW achieved 202× acceleration compared to the one-threaded Fortran version running on an Intel Xeon E5-2603 CPU [10]. However, the GPU version of RRTMG_SW has not been integrated into weather/climate models.

In other fields, the advantages of acceleration techniques on many-core GPU are particularly outstanding. *N. Hou, et al.* proposed an efficient GPU-based parallel tabu search algorithm for hardware/software partitioning. Moreover, an optimized transfer strategy was also proposed to minimize the transfer overhead between CPU and GPU [26]. *Y. Zhou, et al.* proposed a new parallel ant colony optimization (ACO) algorithm on GPUs and obtained an outstanding speedup in the traveling salesman problem [27].

Despite the considerable amount of previous research on this topic, no published studies have focused on accelerating RRTMG_SW using CUDA Fortran and optimizing data transfer between CPU and GPU. In GPU-based acceleration computing, one of the most important challenges is to minimize the data transfer between CPU and GPU. In this paper, a method for accelerating RRTMG_SW using CUDA Fortran is presented and compared with its CUDA C version and a performance optimization method for the data transfer of RRTMG_SW is introduced. Finally, its GPU version is integrated into CAS-ESM.

III. MODEL DESCRIPTION AND EXPERIMENT PLATFORM

A. RRTMG_SW MODEL

RRTMG, developed by the Atmospheric and Environmental Research (AER) organization, has high computing accuracy and low computing cost. It has been widely used in weather and climate models, such as ECWMF, GRAPES, NCAR–CESM, and CAS–ESM. The correlation K method is used in the RRTMG model, which can effectively and accurately calculate radiation fluxes and heating rates. RRTMG_LW has 16 spectral bands, which have a total of 140 quadrature points (g points). In RRTMG_SW, the total number of g points is 112 for 14 spectral bands; for further details, refer to [21], [22], [28], [29].

The code structure of RRTMG_SW is shown in Fig. 1. The overall code is encapsulated in the *rrtmg_sw* and

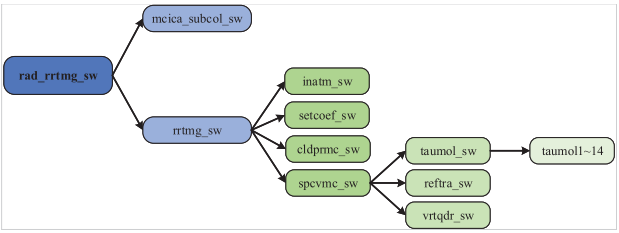

FIGURE 1. The code structure of RRTMG_SW in CAS–ESM.

mcica_subcol_sw subroutines. The *mcica_subcol_sw* subroutine is used to create Monte Carlo independent column approximation (McICA) stochastic arrays for the physical or optical properties of clouds. The *rrtmg_sw* subroutine is the driver for RRTMG_SW. It (a) calls the subroutine *inatm_sw* to read the atmospheric profile from the general circulation model (GCM), (b) calls the subroutine *cldprmc_sw* to set the cloud optical depth based on the input cloud properties, (c) calls the subroutine *setcoef_sw* to calculate various quantities needed for the radiative transfer algorithm, (d) calls the subroutine *spcvmc_sw* to call the two-stream model, which, in turn, calls the subroutine *taumol_sw* to calculate gaseous optical depths for each of the 14 spectral bands and to perform radiative transfer, and (e) passes the calculated fluxes and cooling rates back to the GCM. In this study, we will parallelize the four subroutines encapsulated in *rrtmg_sw*.

B. EXPERIMENT PLATFORM

The experiments in this article are conducted on three GPU clusters: K20, P100, and Titan. The three clusters are located in the computer network information center of CAS, and their specific configurations are shown in Table 1. The serial *rrtmg_sw* is run on an Intel Xeon E5-2680 v2 processor on a K20 cluster, and the GPU-based *rrtmg_sw* is run on all three clusters.

IV. GPU-BASED ACCELERATION ALGORITHMS

A. PARALLEL STRATEGY

In RRTMG_SW, the atmosphere is described in the form of three-dimensional (3D) cells. The three dimensions represent longitude, latitude, and the model layers in the vertical direction. To simplify the code, the longitude and latitude dimensions are combined into one dimension in the original Fortran code. Therefore, the first dimension of 3D arrays in CAS–ESM RRTMG_SW code represents the number of horizontal columns, the second dimension represents the number of model layers, and the third dimension represents the number of *g* intervals. In CAS–ESM, IAP AGCM4.0 has a 1.4° × 1.4° horizontal resolution and 51 model layers, so RRTMG_SW has $nx \times ny = 256 \times 128$ horizontal columns. Thus, the first dimension of arrays in RRTMG_SW has 256×128 elements at most.

Algorithm 1 shows the calculation process of the original *rrtmg_sw*. Every time *rrtmg_sw* is called, *inatm_sw*, *cldprmc_sw*, *setcoef_sw*, and *spcvmc_sw* are called for *ncol* times to finish the computations on all the horizontal

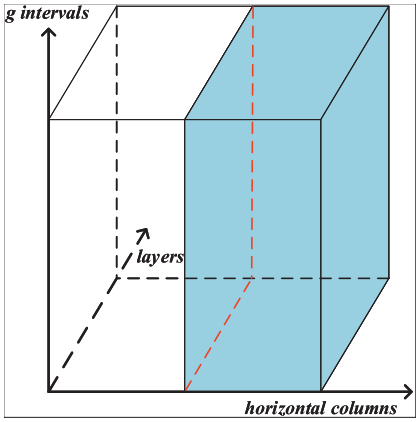

FIGURE 2. Schematic diagram of accelerating RRTMG_SW on GPU.

columns. The term *ncol* refers to the number of horizontal columns. The calculations in the horizontal direction of RRTMG_SW are independent of each other, so the parallel strategy is to utilize *ncol* CUDA threads (or CUDA cores) to execute *rrtmg_sw* simultaneously. Thus, each CUDA thread is assigned the computation of one horizontal column. Fig. 2 illustrates the GPU-based acceleration algorithm of RRTMG_SW with the one-dimensional (1D) domain decomposition.

Algorithm 1 Computing Procedure of the Original *rrtmg_sw*

```
subroutine rrtmg_sw(parameters)
1.  do i=1, ncol
2.     call inatm_sw(parameters)
3.     call cldprmc_sw(parameters)
4.     call setcoef_sw(parameters)
5.     if (iaer .eq. 0) then
6.        ztaua(nlay+1, nbndsw)=0._r8
7.     else
8.        ztaua(i, ib)=taua(i, ib)
9.     end if
10.    call spcvmc_sw(parameters)
11.    {
12.       call taumol_sw(parameters)
13.       call reftra_sw(parameters)
14.       call reftra1_sw(parameters)
15.       call vrtqdr_sw(parameters)
16.       call vrtqdr1_sw(parameters)
17.    }
18.    Transfer fluxes and heating rate to output arrays
19.end do
end subroutine
```

During GPU-based accelerated computing, many CUDA threads execute specific calculation processes in parallel. The usual computing processes are as follows.

(1) Define and initialize the required data on CPU.

(2) The GPU allocates appropriate memory space to store calculation parameters and results.

TABLE 1. Configurations of GPU clusters.

Specification of CPU	K20 cluster	P100 cluster	Titan cluster
CPU	Intel Xeon E5-2680 v2 at 2.8GHz	Intel Xeon E5-2650 v4 at 2.2GHz	Intel Xeon Gold 6148 at 2.4GHz
Operating System	CentOS 6.4	Red Hat 4.8.3-9	Red Hat 4.8.5-16
Specification of GPU	K20 cluster	P100 cluster	Titan cluster
GPU	NVIDIA Tesla K20	NVIDIA Telsa P100	NVIDIA GeForce GTX Titan V
CUDA Cores	2496	3584	5120
Standard Memory	5 GB	16 GB	12 GB
Memory Bandwidth	208 GB/s	732 GB/s	651.3 GB/s
CUDA Version	6.5	8.0	11.0

(3) Copy the calculation parameters from the host memory to the GPU's global memory.

(4) Start CUDA threads and call kernel functions to perform parallel computing.

(5) Pass the computing results from the GPU's memory to the host memory.

(6) Release the GPU's memory.

B. CUDA FORTRAN-BASED ACCELERATION ALGORITHM

Algorithm 2 demonstrates the CUDA Fortran-based acceleration algorithm for RRTMG_SW. Here, n is the block size, which is the number of threads per thread block; $m = \lceil (real)ncol/n \rceil$ refers to each kernel block used in the grid. In Algorithm 1, *inatm_sw*, *cldprmc_sw*, *setcoef _sw*, and *spcvmc_sw* are all called iteratively *ncol* times. Because *ncol* CUDA threads are started, they are only called once in Algorithm 2. In theory, their computational efficiency will be improved *ncol* times.

Algorithm 2 CUDA Fortran-Based Acceleration Algorithm of RRTMG_SW With 1D Domain Decomposition and the Implementation of CF-RRTMG_SW

subroutine *rrtmg_sw_d*(parameters)
1. Copy input data to GPU device
   //Call the kernel *inatm_sw_d*
2. **call** *inatm_sw_d*$\lll m, n \ggg$(parameters)
   //Call the kernel *cldprmc_sw_d*
3. **call** *cldprmc_sw_d*$\lll m, n \ggg$(parameters)
   //Call the kernel *setcoef_sw_d*
4. **call** *setcoef_sw_d*$\lll m, n \ggg$(parameters)
   //Call the kernel *spcvmc_sw_d*
5. **call** *spcvmc_sw_d*$\lll m, n \ggg$(parameters)
6. Copy results to host
   //Judge whether atmospheric horizontal profile data is completed
7. **if** it is not completed **goto** 1

end subroutine

Fig. 3 shows a comparison of the Fortran and CUDA Fortran codes in *spcvmc_sw*. The right panel of Fig. 3 illustrates the acceleration implementation in CUDA Fortran. Here, *iplon* is both the index of the horizontal column and the ID of the global thread in CUDA; *threadIdx%x* is the index of the thread in the thread block; *blockIdx%x* is the index of thread blocks within the kernel grid; and *blockDim%x* represents the block size (i.e., the thread count) within a thread block. According to this design, the kernel *spcvmc_sw_d* will be executed by *ncol* threads concurrently. The CUDA Fortran-based algorithms and implementations of the *inatm_sw_d*, *cldprmc_sw_d*, and *setcoef_sw_d* kernels are not dissimilar to *spcvmc_sw_d*, so no further description is given here.

C. CUDA C-BASED ACCELERATION ALGORITHM

Algorithm 3 demonstrates the acceleration algorithm of RRTMG_SW based on CUDA C. The specific algorithm implementations of the four kernels *inatm_sw_d*, *cldprmc_sw_d*, *setcoef_sw_d*, and *spcvmc_sw_d* based on CUDA C are shown in Algorithms 4–7 of A.

Algorithm 3 CUDA C-Based Acceleration Algorithm of RRTMG_SW With 1D Domain Decomposition and the Implementation of CC-RRTMG_SW

__global__ void *rrtmg_sw_d*(parameters){
1. Copy input data to GPU device
   //Call the kernel *inatm_sw_d*
2. *inatm_sw_d*$\lll m, n \ggg$(parameters);
   //Call the kernel *cldprmc_sw_d*
3. *cldprmc_sw_d*$\lll m, n \ggg$(parameters);
   //Call the kernel *setcoef_sw_d*
4. *setcoef_sw_d*$\lll m, n \ggg$(parameters);
   //Call the kernel *spcvmc_sw_d*
5. *spcvmc_sw_d*$\lll m, n \ggg$(parameters);
6. Copy results to host
   //Judge whether atmospheric horizontal profile data is completed
7. **if** it is not completed **goto** 1

}

V. OPTIMIZATION METHOD FOR DATA TRANSFER

A well-known problem is that data transmission between CPU (host) and GPU (device) is a fundamental performance bottleneck in GPU-based accelerated computing. For CF-RRTMG_SW or CC-RRTMG_SW, data transfer is quite time-consuming, so it is necessary to optimize it. When allocating memory space for variables on the host, pageable memory is used by default. As illustrated in Fig. 4, when transferring data between the host and device, the operating system will first allocate a temporary pinned host buffer, then copy data from pageable memory to a temporary pinned buffer, and finally transfer data to the device. A pinned buffer

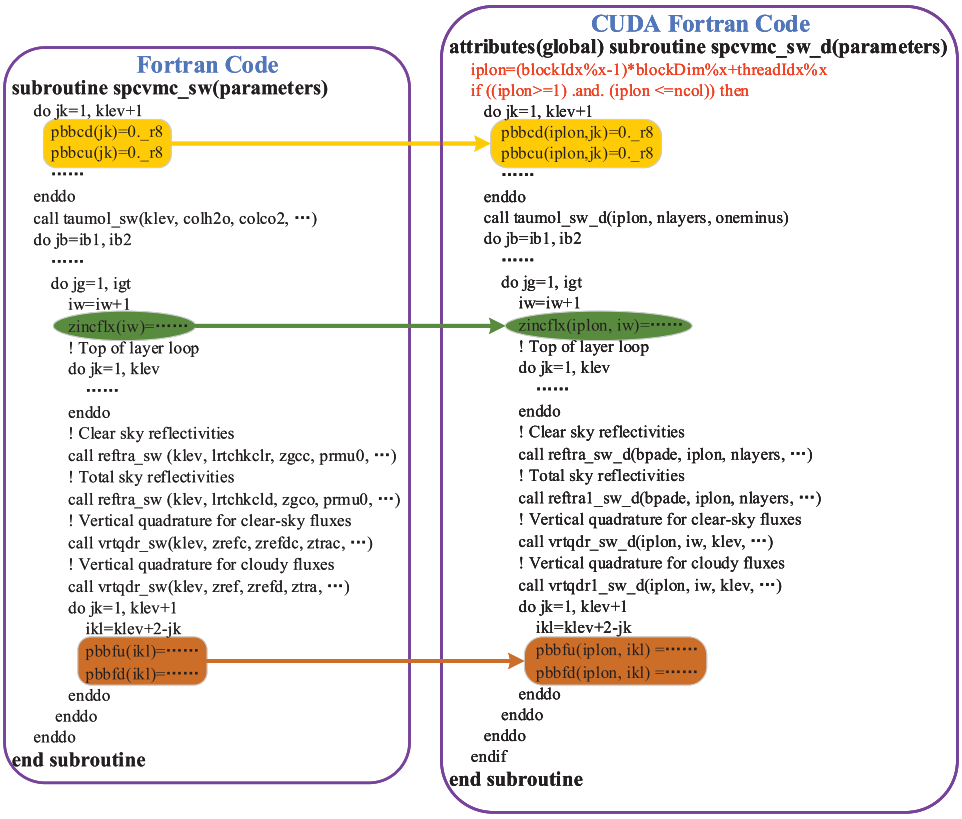

FIGURE 3. Comparison of Fortran and CUDA Fortran codes of *spcvmc_sw*.

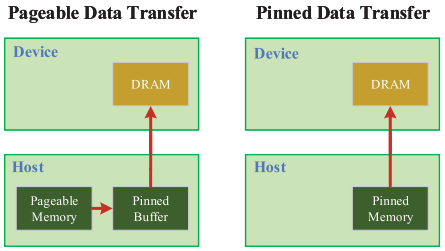

FIGURE 4. Illustration for host-to-device data transfer from pageable memory (left) and pinned memory (right).

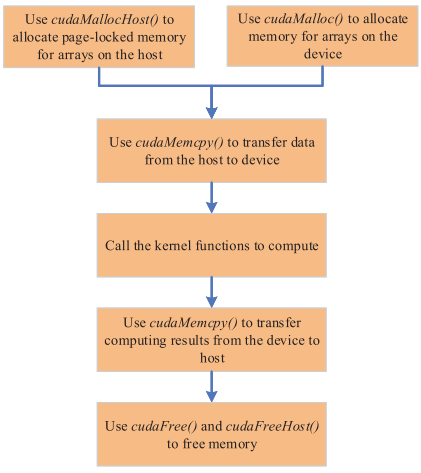

FIGURE 5. Computing flowchart of CC-RRTMG_SW by adopting the pinned memory technology.

may be smaller than pageable memory holding host data, in which case copying data occurs in multiple stages. Data transfer from the device to the host is similar. If pinned memory is used, the system will ensure that the data in pinned memory always resides in random access memory and is not allocated to a hard drive. When adopting pinned memory technology to transfer data between a host and device, the cost of data transfer between paging and pinned host arrays can be avoided by allocating the host array directly in the pinned memory [30]. Thus, data transfer overhead between the host and device will be reduced.

Fig. 5 describes the computing flowchart of CC-RRTMG_SW by adopting pinned memory technology. Here, the application program interface (API) *cudaMallocHost* provided by CUDA allocates page-locked memory on the host; the API *cudaMalloc* allocates memory on the device; the API *cudaMemcpy* copies data between the host and device; the API *cudaFree* frees memory on the device; and the API *cudaFreeHost* frees page-locked memory.

VI. RESULTS AND DISCUSSION

To comprehensively study the proposed acceleration algorithms, an ideal climate simulation experiment was carried

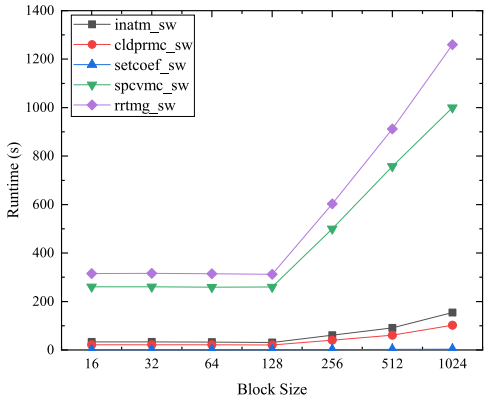

FIGURE 6. Runtime (s) of CF-RRTMG_SW on one K20 GPU.

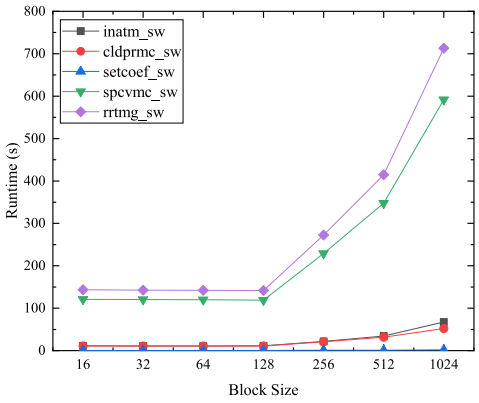

FIGURE 7. Runtime (s) of CC-RRTMG_SW on one K20 GPU.

out on a model day. In this experiment, the time increment of RRTMG_SW was 1 h.

A. INFLUENCE OF BLOCK SIZE

Due to GPUs' limited hardware resources, the block size in CUDA has a significant impact on the performance of acceleration algorithms. The running time of *rrtmg_sw* is obtained by using the following formula in the case of no data transfer:

$$T_{rrtmg\_sw} = T_{inatm\_sw} + T_{cldprmc\_sw} + T_{setcoef\_sw} + T_{spcvmc\_sw}$$

Fig. 6 portrays the running time of CF-RRTMG_SW on one K20 GPU. When the block size is 128, CF-RRTMG_SW achieves the best performance. Similarly, when the block size is 128, CC-RRTMG_SW on one K20 GPU also achieves the best performance, as shown in Fig. 7. Usually, moderately increasing the block size can hide the latency of memory access and helps improve the computing performance of CUDA code. However, if the block size is larger, each thread will have fewer hardware resources. Therefore, when the block size is 1024, CF-RRTMG_SW or CC-RRTMG_SW does not achieve the best performance. Moreover, comparing the results shown in Figs. 6 and 7, reveals that CC-RRTMG_SW achieves better computing performance than CF-RRTMG_SW.

B. INFLUENCE OF NCOL

When the block size is fixed, the size of *ncol* also has a significant impact on the performance of CC-RRTMG_SW. RRTMG_SW has $128 \times 256 = 32{,}768$ horizontal columns. Because global memory is limited in GPUs, a single K20 GPU can only compute 2048 horizontal columns of RRTMG_SW, which means that the maximum value of *ncol* is 2048. During each integration for RRTMG_SW, CC-RRTMG_SW will be invoked 32,768/2048=16 times if *ncol*=2048. In this simulation of one model day, integration computations are performed 24 times. This means that CC-RRTMG_SW will be invoked repeatedly $16 \times 24 = 384$ times. Likewise, if *ncol*=1024, it will be invoked repeatedly 768 times. As shown in Table 2, when *ncol*=2048, CC-RRTMG_SW has shorter runtimes. Some conclusions can be drawn.

(1) One K20 GPU has 2496 cores, so there are enough cores to compute CC-RRTMG_SW when *ncol*=2048. If *ncol*=1024, these cores will be not fully utilized.

(2) When *ncol*=2048, there are fewer invoking times for CC-RRTMG_SW.

TABLE 2. Runtime (s) of CC-RRTMG_SW on one K20 GPU, where the block size = 128.

Subroutines	ncol=1024	ncol=2048
inatm_sw	20.56	11.31
cldprmc_sw	21.27	10.86
setcoef_sw	1.06	0.54
spcvmc_sw	232.14	118.96
rrtmg_sw	275.03	141.67

C. EVALUATIONS PERFORMED ON DIFFERENT GPUs

Table 3 shows the runtime and speedup values when running CC-RRTMG_SW on K20, P100, and Titan GPUs. When *ncol* is 2048, in the condition without I/O transfer, the speedups when CC-RRTMG_SW running on the three kinds of GPUs are 5.36×, 8.39×, and 13.73×, respectively. The speedups of CC-RRTMG_SW running on P100 and Titan GPUs are shown in Table 4 and Table 5 when *ncol* is 4096 and 8192. From the results presented in Tables 3-5, the following conclusions can be drawn.

(1) CC-RRTMG_SW produces faster speeds on a Titan GPU than on K20 and P100 GPUs because a Titan GPU has more CUDA cores, as shown in Table 1.

(2) As *ncol* increases, the speed of CC-RRTMG_SW on K20, P100, and Titan GPUs also increases.

(3) Without I/O transfer, CC-RRTMG_SW on one Titan GPU has a speedup of 38.88× when *ncol* is 8192.

D. I/O TRANSFER

As mentioned above, if *ncol*=1024, CC-RRTMG_SW will be invoked 768 times, which means there will be 1536 data transfers between CPU and GPU. Thus, to improve performance when frequently transferring data between CPU and GPU, the pinned memory technology described in Section V was

TABLE 3. Runtime (s) and speedup of CC-RRTMG_SW on different GPUs, where the block size = 128 and *ncol* = 2048.

Subroutines	CPU time	K20 time	Speedup
inatm_sw	131.99	11.31	11.67
cldprmc_sw	82.76	10.86	7.62
setcoef_sw	2.76	0.54	5.11
spcvmc_sw	541.38	118.96	4.55
rrtmg_sw	758.89	141.67	**5.36**
Subroutines	CPU time	P100 time	Speedup
inatm_sw	131.99	9.76	13.52
cldprmc_sw	82.76	7.94	10.42
setcoef_sw	2.76	0.32	8.63
spcvmc_sw	541.38	72.45	7.47
rrtmg_sw	758.89	90.47	**8.39**
Subroutines	CPU time	Titan time	Speedup
inatm_sw	131.99	7.10	18.59
cldprmc_sw	82.76	5.95	13.91
setcoef_sw	2.76	0.15	18.40
spcvmc_sw	541.38	42.06	12.87
rrtmg_sw	758.89	55.26	**13.73**

TABLE 4. Runtime (s) and speedup of CC-RRTMG_SW on different GPUs, where the block size = 128 and *ncol* = 4096.

Subroutines	CPU time	P100 time	Speedup
inatm_sw	131.99	5.66	23.32
cldprmc_sw	82.76	4.29	19.29
setcoef_sw	2.76	0.17	16.24
spcvmc_sw	541.38	40.52	13.36
rrtmg_sw	758.89	50.64	**14.99**
Subroutines	CPU time	Titan time	Speedup
inatm_sw	131.99	3.96	33.33
cldprmc_sw	82.76	3.15	26.27
setcoef_sw	2.76	0.08	34.50
spcvmc_sw	541.38	24.83	21.80
rrtmg_sw	758.89	32.02	**23.70**

TABLE 5. Runtime (s) and speedup of CC-RRTMG_SW on different GPUs, where the block size = 128 and *ncol* = 8192.

Subroutines	CPU time	P100 time	Speedup
inatm_sw	131.99	3.70	35.67
cldprmc_sw	82.76	2.49	33.24
setcoef_sw	2.76	0.10	27.6
spcvmc_sw	541.38	24.59	22.02
rrtmg_sw	758.89	30.88	**24.58**
Subroutines	CPU time	Titan time	Speedup
inatm_sw	131.99	2.59	50.96
cldprmc_sw	82.76	1.82	45.47
setcoef_sw	2.76	0.04	69.00
spcvmc_sw	541.38	15.07	35.92
rrtmg_sw	758.89	19.52	**38.88**

adopted. Table 6 shows the computing performance of CC-RRTMG_SW with and without pinned memory technology on one K20 GPU. With this technology, transferring data between CPU and GPU is sped up by approximately 86%. When the total performance of CC-RRTMG_SW on one K20 GPU is analyzed, there is a 44% improvement.

Table 7 and Table 8 show the elapsed time and acceleration of CC-RRTMG_SW for I/O transfers on GPUs. Some conclusions can be drawn from these results, as follows.

TABLE 6. Runtime (s) of CC-RRTMG_SW with and without the pinned memory technology on one K20 GPU, where the block size = 128 and *ncol* = 2048. ***CUDA memcpy DtoH*** is the time of transferring data from GPU to CPU; ***CUDA memcpy HtoD*** is the one from CPU to GPU.

Subroutine	Without optimization	With optimization	Speedup
inatm_sw	11.31	11.30	–
cldprmc_sw	10.86	10.86	–
setcoef_sw	0.54	0.54	–
spcvmc_sw	118.96	119.05	–
CUDA memcpy DtoH	124.38	66.15	**1.88**
CUDA memcpy HtoD	144.14	77.93	**1.85**
rrtmg_sw	410.19	285.83	**1.44**

TABLE 7. Runtime (s) and speedup of CC-RRTMG_SW with I/O transfer on GPUs, where the block size = 128 and *ncol* = 2048.

GPU	Computing time	DtoH	HtoD	I/O	*rrtmg_sw*	Speedup
K20	141.75	66.15	77.93	144.08	285.83	**2.66**
P100	90.47	34.07	39.84	73.91	164.38	**4.62**
Titan	55.26	39.11	43.80	82.91	138.17	**5.49**

TABLE 8. Runtime (s) and speedup of CC-RRTMG_SW with I/O transfer on GPUs, where the block size = 128 and *ncol* = 8192.

GPU	Computing time	DtoH	HtoD	I/O	*rrtmg_sw*	Speedup
P100	30.88	34.06	39.61	73.67	104.55	**7.26**
Titan	19.52	39.14	43.69	82.83	102.35	**7.41**

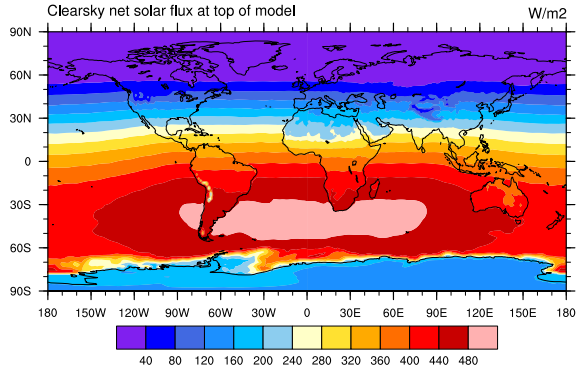

FIGURE 8. Radiant flux simulated by CAS–ESM RRTMG_SW on CPU.

(1) CC-RRTMG_SW with I/O transfer on a Titan GPU still has the best performance, with a speedup of 7.41× when *ncol* is 8192.

(2) A P100 GPU has a higher memory bandwidth than K20 and Titan GPUs, so its I/O time is the shortest.

(3) Despite adopting pinned memory technology, data transmission between CPU and GPU is still a significant bottleneck at higher speeds with CC-RRTMG_SW. Thus, further optimizing data transfer remains an urgent problem.

E. ERROR ANALYSIS

When accelerating RRTMG_SW on a GPU, it is necessary to ensure that errors are minimized. In an experiment for one model day, Fig. 8 shows the net solar flux at the top of the model in a clear sky, which is the result of running CAS–ESM RRTMG_SW on CPU. The global averaged difference between the two simulations running CAS–ESM

Algorithm 4 Implementation of *inatm_sw* Based on CUDA C

```
__global__ void inatm_sw_d(parameters){
1. iplon=blockDim.x * blockIdx.x + threadIdx.x;
2. if ((iplon>=0 && iplon<ncol){
   /* Initializes all the data needed for calculation.*/
3.    for (j=0; j<nlayers; j++){
4.       do some corresponding work;
5.    }
6.    for (j=0; j<ngptsw; j++){
7.       do some corresponding work;
8.    }
9. }
}
```

Algorithm 5 Implementation of *cldprmc_sw* Based on CUDA C

```
__global__ void cldprmc_sw_d(parameters){
1. iplon=blockDim.x * blockIdx.x + threadIdx.x;
2. if ((iplon>=0 && iplon<ncol){
3.    //Main g-point interval loop.
4.    for (lay=0; lay<nlayers; lay++){
5.       for (ig=0; ig<ngptsw; ig++){
6.          do some corresponding work;
7.       }
8.    }
9. }
}
```

RRTMG_SW only on CPU and running it on one K20 GPU is -0.000455583 W/m^2. The bias is minor and acceptable. The reasons for the deviation are as follows: (1) When run on CPU and GPU, CAS-ESM RRTMG_SW has different language implementations (Fortran and CUDA C); (2) the round-off error on CPU and GPU is inevitable.

VII. CONCLUSION AND FUTURE WORK

The accelerated calculation of radiation physics based on a GPU is quite challenging. This paper introduces RRTMG_SW acceleration algorithms on GPU. Two GPU versions RRTMG SW (i.e., CF-RRTMG_LW and CC-RRTMG_SW) were built. In CC-RRTMG_SW, data transfer between the host and device was optimized by adopting pinned memory technology. The experimental results indicate that running CC-RRTMG_SW on a GPU produces satisfactory speedup rates.

However, the current work does have some shortcomings. To address them, future work will include the following. First, CC-RRTMG_SW will be further accelerated on the model layer and g intervals dimensions. A GPU-based acceleration algorithm for RRTMG_SW with a 3D domain decomposition method will be proposed. Second, utilizing an MPI+CUDA hybrid paradigm to accelerate RRTMG_SW on multiple GPUs will also be considered. Third, accelerating RRTMG_SW based on AMD GPUs will also be studied. However, implementing the algorithms under a heterogeneous-compute interface for portability (HIP) framework will be a significant challenge. Undoubtedly, the future work will be helpful in improving the computational performance of RRTMG_SW.

Algorithm 6 Implementation of *setcoef_sw* Based on CUDA C

```
__global__ void setcoef_sw_d(parameters){
1. iplon=blockDim.x * blockIdx.x + threadIdx.x;
2. if ((iplon>=0 && iplon<ncol){
      /*Find the two reference pressures on either side of the
layer pressure.*/
3.    for (lay=0; lay<nlayers; lay++){
4.       if (plog<=4.56){
5.          do some corresponding work;
6.       }
7.       else{
8.          do some corresponding work;
9.       }
10.      do some corresponding work;
11.   }
12.}
}
```

Algorithm 7 Implementation of *spcvmc_sw* Based on CUDA C

```
__global__ void spcvmc_sw_d(parameters){
1. iplon=blockDim.x * blockIdx.x + threadIdx.x;
2. if ((iplon>=0 && iplon<ncol){
3.    taumol_sw_d(parameters);
4.    for (jb=ib1−1; jb<ib2; jb++){
5.       ibm=jb−15;
6.       igt=ngc[ibm];
7.       if(iout>0 && ibm>=1) iw=ngs[ibm-1]−1;
8.       for(jg=0; jg<igt; jg++){
9.          do some corresponding work;
10.         reftra_sw_d(parameters);
11.         reftra1_sw_d(parameters);
12.         do some corresponding work;
13.         vrtqdr_sw_d(parameters);
14.         vrtqdr1_sw_d(parameters);
15.         for(jk=0; jk<klev+1; jk++){
16.            do some corresponding work;
17.         }
18.      }
19.   }
20.}
}
```

APPENDIX

A. IMPLEMENTATION OF THE KERNELS BASED ON CUDA C

See Algorithms 4–7.

ACKNOWLEDGMENT

The authors would like to acknowledge the contributions of Prof. Minghua Zhang for insightful suggestions on algorithm design.

REFERENCES

[1] S. A. Clough, M. W. Shephard, E. J. Mlawer, J. S. Delamere, M. J. Iacono, K. Cady-Pereira, S. Boukabara, and P. D. Brown, "Atmospheric radiative transfer modeling: A summary of the AER codes," *J. Quant. Spectrosc. Radiat. Transf.*, vol. 91, no. 2, pp. 233–244, Mar. 2005.

[2] M. J. Iacono, *Enhancing Cloud Radiative Processes and Radiation Efficiency in the Advanced Research Weather Research and Forecasting (WRF) Model*. Lexington, MA, USA: Atmospheric and Environmental Research, 2015.

[3] J.-J. Morcrette, G. Mozdzynski, and M. Leutbecher, "A reduced radiation grid for the ECMWF integrated forecasting system," *Monthly Weather Rev.*, vol. 136, no. 12, pp. 4760–4772, Dec. 2008.

[4] J. Nickolls and W. J. Dally, "The GPU computing era," *IEEE Micro*, vol. 30, no. 2, pp. 56–69, Mar. 2010.

[5] 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, Dec. 2012.

[6] 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.

[7] D. Chen, X. Li, L. Wang, S. U. Khan, J. Wang, K. Zeng, and C. Cai, "Fast and scalable multi-way analysis of massive neural data," *IEEE Trans. Comput.*, vol. 64, no. 3, pp. 707–719, Mar. 2015.

[8] F. Candel, S. Petit, J. Sahuquillo, and J. Duato, "Accurately modeling the on-chip and off-chip GPU memory subsystem," *Future Gener. Comput. Syst.*, vol. 82, pp. 510–519, May 2018.

[9] NVIDIA. (2019). *CUDA C Programming Guide v10.0*. Accessed: Sep. 26, 2019. [Online]. Available: https://docs.nvidia.com/pdf/CUDA_C_Programming_Guide.pdf

[10] J. Mielikainen, E. Price, B. Huang, H.-L.-A. Huang, and T. Lee, "GPU compute unified device architecture (CUDA)-based parallelization of the RRTMG shortwave rapid radiative transfer model," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 9, no. 2, pp. 921–931, Feb. 2016.

[11] NVIDIA. (2019). *CUDA Fortran Programming Guide and Reference*. Accessed: Sep. 26, 2019. [Online]. Available: https://www.pgroup.com/resources/docs/19.1/pdf/pgi19cudaforug.pdf

[12] D. Xiao, S. Tong-Hua, W. Jun, and L. Ren-Ping, "Decadal variation of the aleutian low-icelandic low seesaw simulated by a climate system model (CAS-ESM-C)," *Atmos. Ocean. Sci. Lett.*, vol. 7, no. 2, pp. 110–114, Jan. 2014.

[13] Y. Wang, J. Jiang, H. Ye, and J. He, "A distributed load balancing algorithm for climate big data processing over a multi-core CPU cluster," *Concurrency Comput., Pract. Exper.*, vol. 28, no. 15, pp. 4144–4160, Oct. 2016.

[14] Y. Wang, J. Jiang, J. Zhang, J. He, H. Zhang, X. Chi, and T. Yue, "An efficient parallel algorithm for the coupling of global climate models and regional climate models on a large-scale multi-core cluster," *J. Supercomput.*, vol. 74, no. 8, pp. 3999–4018, Aug. 2018.

[15] Y. Wang, H. Hao, J. Zhang, J. Jiang, J. He, and Y. Ma, "Performance optimization and evaluation for parallel processing of big data in Earth system models," *Cluster Comput.*, vol. 22, no. S1, pp. 2371–2381, Jan. 2019.

[16] H. Zhang, M. Zhang, and Q.-C. Zeng, "Sensitivity of simulated climate to two atmospheric models: Interpretation of differences between dry models and moist models," *Monthly Weather Rev.*, vol. 141, no. 5, pp. 1558–1576, May 2013.

[17] Y. Wang, J. Jiang, H. Zhang, X. Dong, L. Wang, R. Ranjan, and A. Y. Zomaya, "A scalable parallel algorithm for atmospheric general circulation models on a multi-core cluster," *Future Gener. Comput. Syst.*, vol. 72, pp. 1–10, Jul. 2017.

[18] F. Lu, X. Cao, J. Song, and X. Zhu, "GPU computing for longwave radiation physics: A RRTM_LW scheme case study," in *Proc. IEEE 9th Int. Symp. Parallel Distrib. Process. With Appl. Workshops*, May 2011, pp. 71–76.

[19] F. Lu, J. Song, X. Cao, and X. Zhu, "CPU/GPU computing for long-wave radiation physics on large GPU clusters," *Comput. Geosci.*, vol. 41, pp. 47–55, Apr. 2012.

[20] E. Price, J. Mielikainen, B. Huang, H. A. Huang, and T. Lee, "GPU acceleration experience with RRTMG long wave radiation model," *Proc. SPIE*, vol. 8895, Oct. 2013, Art. no. 88950H.

[21] E. Price, J. Mielikainen, M. Huang, B. Huang, H.-L.-A. Huang, and T. Lee, "GPU-accelerated longwave radiation scheme of the rapid radiative transfer model for general circulation models (RRTMG)," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 8, pp. 3660–3667, Aug. 2014.

[22] Y. Wang, Y. Zhao, W. Li, J. Jiang, X. Ji, and A. Y. Zomaya, "Using a GPU to accelerate a longwave radiative transfer model with efficient CUDA-based methods," *Appl. Sci.*, vol. 9, no. 19, p. 4039, Sep. 2019.

[23] Y. Wang, Y. Zhao, J. Jiang, and H. Zhang, "A novel GPU-based acceleration algorithm for a longwave radiative transfer model," *Appl. Sci.*, vol. 10, no. 2, p. 649, Jan. 2020.

[24] Y. Wang, M. Guo, Y. Zhao, and J. Jiang, "GPUs-RRTMG_LW: high-efficient and scalable computing for a longwave radiative transfer model on multiple GPUs," *J. Supercomput.*, vol. 77, no. 5, pp. 4698–4717, May 2021.

[25] 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 Observ. Remote Sens.*, vol. 5, no. 2, pp. 555–562, Apr. 2012.

[26] N. Hou, F. He, Y. Zhou, and Y. Chen, "An efficient GPU-based parallel tabu search algorithm for hardware/software co-design," *Frontiers Comput. Sci.*, vol. 14, no. 5, Oct. 2020.

[27] Y. Zhou, F. He, and Y. Qiu, "Dynamic strategy based parallel ant colony optimization on GPUs for TSPs," *Sci. China Inf. Sci.*, vol. 60, no. 6, Jun. 2017, Art. no. 068102

[28] E. J. Mlawer, M. J. Iacono, R. Pincus, H. W. Barker, L. Oreopoulos, and D. L. Mitchell, "Contributions of the ARM program to radiative transfer modeling for climate and weather applications," *Meteorolog. Monographs*, vol. 57, no. 15, pp. 1–19, 2016.

[29] H. Zhang, M. Zhang, J. Jin, K. Fei, D. Ji, C. Wu, J. Zhu, J. He, Z. Chai, J. Xie, and X. Dong, "Description and climate simulation performance of CAS-ESM version 2," *J. Adv. Model. Earth Syst.*, vol. 12, no. 12, Dec. 2020, Art. no. e2020MS002210.

[30] G. Ruetsch and M. Fatica, *CUDA Fortran for Scientists and Engineers: Best Practices for Efficient CUDA Fortran Programming*. Amsterdam, The Netherlands: Elsevier, 2013.

ZHENZHEN WANG received the B.Sc. degree in electronic information science and technology from the School of Information Science and Technology, Hebei Agricultural University, in 2019. She is currently pursuing the M.Eng. degree in computer science with the School of Information Engineering, China University of Geosciences, Beijing. Her research interests include high performance computing and parallel algorithm.

YUZHU WANG received the Ph.D. degree in engineering from the University of Chinese Academy of Sciences, in 2015. He is an Associate Professor with the School of Information Engineering, China University of Geosciences, Beijing, China. His research interests include high performance computing and parallel algorithm.

XIAOCONG WANG received the Ph.D. degree in meteorology from the Chinese Academy of Sciences, Beijing, China, in 2012. He is an Associate Professor with the Institute of Atmospheric Physics, Chinese Academy of Sciences. His research interests include parameterizations of cumulus convection and cloud radiation in climate models.

FEI LI is currently pursuing the B.Sc. degree in computer science with the School of Information Engineering, China University of Geosciences, Beijing. His research interests include high performance computing and parallel algorithm.

HANGTIAN HU is currently pursuing the B.Sc. degree in computer science with the School of Information Engineering, China University of Geosciences, Beijing. His research interests include parallel and distributed computing.

CHEN ZHOU received the M.Eng. degree from the School of Information Engineering, China University of Geosciences, Beijing, in 2020. Her research interests include high performance computing and parallel algorithm.

JINRONG JIANG received the Ph.D. degree in engineering from the Institute of Software, Chinese Academy of Sciences, Beijing, China, in 2007. He is a Full Professor with the Computer Network Information Center, Chinese Academy of Sciences. His research interests include high performance computing and parallel algorithm.