The Journal of Supercomputing (2022) 78:17378–17402
https://doi.org/10.1007/s11227-022-04566-5

# CC-RRTMG_SW++: Further optimizing a shortwave radiative transfer scheme on GPU

**Fei Li[1] · Yuzhu Wang[1] · Zhenzhen Wang[1] · Xiaohui Ji[1] · Jinrong Jiang[2] · Xiaoyong Tang[3] · He Zhang[4]**

Accepted: 26 April 2022 / Published online: 18 May 2022


## Abstract

Atmospheric radiation is one of the most important atmospheric physics, and its expensive computation cost severely restricts the numerical simulation of atmospheric general circulation models. Therefore, it is necessary to study an efficient radiation parameterization scheme. Due to the powerful computing power of GPU, more and more numerical models are being transplanted to GPU. The CUDA C version (CC-RRTMG_SW) of the rapid radiative transfer model for general circulation models (RRTMG) shortwave radiation scheme (RRTMG_SW) has successfully run on GPU, but its computing efficiency is not yet very high, and the performance potential of GPU computing needs to be realized further. This paper is dedicated to optimizing CC-RRTMG_SW and exploring its maximum computing performance on GPU. First, a three-dimensional acceleration algorithm for CC-RRTMG_SW is proposed. Then, some optimization methods, such as decoupling data dependency, optimizing memory access, and I/O optimization, are studied. Finally, the optimized version of CC-RRTMG_SW is developed, namely CC-RRTMG_SW++. The experimental results demonstrate that the proposed acceleration algorithm and performance optimization methods are effective. CC-RRTMG_SW++ achieved good acceleration effects on different GPU architectures, such as NVIDIA Tesla K20, K40, and V100. Compared to RRTMG_SW running on a single Intel Xeon E5-2680 v2 CPU core, CC-RRTMG_SW++ obtained a speedup of 99.09× on a single V100 GPU without I/O transfer. Compared to CC-RRTMG_SW, the computing efficiency of CC-RRTMG_SW++ increased by 174.46%.

**Keywords** Graphics processing unit · Compute unified device architecture programming · Performance optimization · Shortwave radiative transfer

✉ Yuzhu Wang
wangyz@cugb.edu.cn

✉ He Zhang
zhanghe@mail.iap.ac.cn

Extended author information available on the last page of the article

## 1 Introduction

The weather and climate changes have a significant impact on people's productivity and lives. Using climate models to predict future climate conditions plays an important role in planning social production and formulating disaster prevention measures [1]. In climate models, whether the distribution and alteration of radiation from the Earth's surface can be simulated reasonably and accurately will greatly affect the prediction of future weather and climate changes [2]. Radiative transfer is one of the key issues in atmospheric physics; it consumes a large amount of computing resources among all atmospheric physical processes [3]. Therefore, developing a high-accuracy and high-speed radiative transfer model is very meaningful for global atmospheric modeling. The rapid radiative transfer model for general circulation models (RRTMG) is a related k-distribution model that calculates the longwave and shortwave radiative flux and heating rate in the atmosphere [4–6]. Due to the high accuracy of RRTMG, it has been applied in weather forecasts and climate models [7, 8]. For example, the Institute of Atmospheric Physics of CAS Atmospheric General Circulation Model Version 4.0 (IAP AGCM4.0) [9] uses RRTMG as its radiation parameterization scheme. Although RRTMG has a quite fast calculation speed, it still occupies 15% to 25% of the calculation time for the entirety of the atmospheric physics in IAP AGCM4.0. With the development of high-resolution atmospheric circulation models, the calculation amount of RRTMG will increase exponentially, which will seriously limit the performance of large-scale simulation calculations. Thus, it is necessary to further accelerate RRTMG.

GPUs (Graphics Processing Units) have powerful floating-point computing capabilities, high memory access bandwidth, and many computing cores. In today's era of green high-performance computing, the application of GPU-accelerated numerical computing is becoming more and more widespread [10]. The original RRTMG is with serial computing. Given the advantages of GPU-based parallel computing, using GPU to accelerate radiative transfer schemes has become a quite feasible and valuable research direction [11, 12].

At present, there have been some acceleration algorithms for radiative transfer scheme. In 2019, Wang et al. developed a GPU version of an RRTMG longwave radiation scheme (RRTMG_LW) using CUDA Fortran called G-RRTMG_LW, which achieved a speedup of 30.98× on a Tesla K40 GPU [13]. Later, CUDA Fortran and CUDA C versions of the RRTMG shortwave radiation scheme (RRTMG_SW) were developed: CF-RRTMG_SW and CC-RRTMG_SW. Compared with RRTMG_SW running on a single Intel Xeon E5-2680 CPU core, CC-RRTMG_SW could reach a speedup of 38.88× on a single NVIDIA GeForce Titan V GPU [14].

Although CC-RRTMG_SW implements GPU-based computing of RRTMG_SW, its speedup is fairly low and the cost of data transfer between CPU and GPU is relatively large. To fully utilize the computing performance of GPU, it is necessary to further optimize CC-RRTMG_SW. The main challenges are how to perform finer-grained parallelism and apply the optimization techniques of

CUDA [15] programming for CC-RRTMG_SW. To solve these problems, this paper first proposes two-dimensional (2D) and three-dimensional (3D) acceleration algorithms of RRTMG_SW based on CUDA C and then optimizes them. This includes decoupling data dependency, reduction of access of global memory, and optimization of I/O transfer. The optimized CC-RRTMG_SW with 3D acceleration is named CC-RRTMG_SW++. Experimental results show that compared to RRTMG_SW running on a single Intel Xeon E5-2680 v2 CPU core, CC-RRTMG_SW++ without I/O transfer can achieve a speedup of 99.09× on a single NVIDIA Tesla V100 GPU, and it still has a speedup of 24.07× with I/O transfer. Compared with CC-RRTMG_SW, the computing efficiency of CC-RRTMG_SW++ is increased by 174.46%.

The main contributions of this paper are as follows.

- A novel 2D acceleration algorithm is proposed for the *spcvmc* subroutine of RRTMG_SW and implements its GPU-based computing in the horizontal and *jp-band* dimensions. Then, a 3D acceleration algorithm is also proposed and implemented for RRTMG_SW in the horizontal, vertical, and *g-point* dimensions. The two algorithms further accelerate RRTMG_SW.
- Some performance optimization methods are also proposed, including effectively decoupling data dependency, using GPU registers to optimize global memory access, using CUDA streams to reduce I/O transfer time between host and device. These methods make full use of the computing performance of GPU and improve the computing performance of CC-RRTMG_SW++.

The rest of this paper is organized as follows. Section 2 presents some excellent work on accelerated climate models and radiative transfer schemes. Section 3 introduces the RRTMG_SW model and its parallel dimensions. Section 4 describes the 2D and 3D GPU-based acceleration algorithms of RRTMG_SW. Section 5 introduces some optimization methods for RRTMG_SW on GPU. Section 6 analyzes the results of numerical experiments. The last section summarizes this paper and proposes an outlook for future work.

## 2 Related work

In recent years, with the wide application of GPU, there has been considerable research on using GPU to accelerate the parameterization schemes of climate models. In this section, we will introduce some excellent works on accelerating climate models in recent years and then focus on the related research on using GPU to accelerate the radiation physics process.

Huang et al. implemented a WRF five-layer thermal diffusion scheme using GPU large-scale parallel architecture. Without considering I/O transfer, the accelerated WRF five-layer thermal diffusion scheme achieved a speedup of 311× on a Tesla K40 GPU [16]. Leutwyler et al. implemented a GPU version of the convection-resolving COSMO model in a climate model and completely transplanted it into a multi-core, heterogeneous atmospheric model. They demonstrated the applicability

of this approach to longer simulations by conducting a 3-month-long simulation [17]. Mielikainen implemented GPU acceleration for a WRF double-moment 6-class microphysics scheme. On a single GPU, a 150× speedup was achieved; if I/O overhead is not considered, a 206× speedup was achieved [18]. Cao implemented a highly scalable 3D atmospheric circulation model, AGCM-3DLF, based on leap format. The experimental results on different platforms showed that the model has good efficiency and scalability. On the CAS-Xiandao1 supercomputer, a speed of 11.1 simulated years per day (SYPD) was achieved at a high resolution of 25 km [19].

In terms of using GPUs to accelerate radiative physics, Lu et al. accelerated RRTMG_LW on GTX470, GTX480, and C2050, and obtained 23.2×, 27.6×, and 18.2× acceleration, respectively. They also performed performance analysis in terms of GPU clock frequency, execution configuration, use of registers, characteristics of RRTM_LW, etc. [20]. Mielikainen et al. used GPUs to accelerate the Goddard shortwave radiation parameterization scheme and achieved a 116× speedup on two NVIDIA GTX 590 GPUs, and a 259× speedup through a single precision calculation [21]. Price et al. rewrote the Fortran code of RRTMG_LW in C language and then implemented the GPU acceleration of RRTMG_LW using CUDA C. In the performance optimization, 19 optimization schemes were implemented. Regardless of I/O overhead, a 127× speedup was achieved on a Tesla K40 GPU compared to its computing time using an Intel Xeon E5-2603 single-core [22]. Mielikainen et al. used CUDA C to implement the GPU computation of RRTMG_SW in WRF. RRTMG_SW achieved a speedup of 202× on a single Tesla K40 GPU [2].

Significantly diverging from previous research, including Wang et al. [14], this paper applies parallel computing in the *g-point* dimension to RRTMG_SW for the first time and then creatively proposes the parallel scheme in the *jp-band* dimension. Moreover, this paper further accelerates RRTMG_SW by using certain optimization methods.

## 3 Model description and analysis

### 3.1 RRTMG_SW

The RRTMG model uses the two-stream approximation method to solve the radiation transfer equation and uses the correlation k-distribution method to calculate the radiation flux in the process of the molecular absorption and scattering. It decomposes the radiation spectrum into multiple bands, and the absorption intensity value of each band is divided into a cumulative distribution intensity function. In order to obtain the band's radiant flux, the distribution function is discretized in each band by $g$ iterative integration. For more details, please refer to [23].

Figure 1 shows the structure of RRTMG_SW. RRTMG_SW consists of two subroutines: *mcica_subcol_sw* and *rrtmg_sw*. *mcica_subcol_sw* is used to create Monte Carlo independent column approximation (McICA) stochastic arrays, enabling McICA to provide fractional cloudiness and cloud overlap capabilities to RRTMG_SW. *rrtmg_sw* is the driver of RRTMG_SW and is also the core computing part of

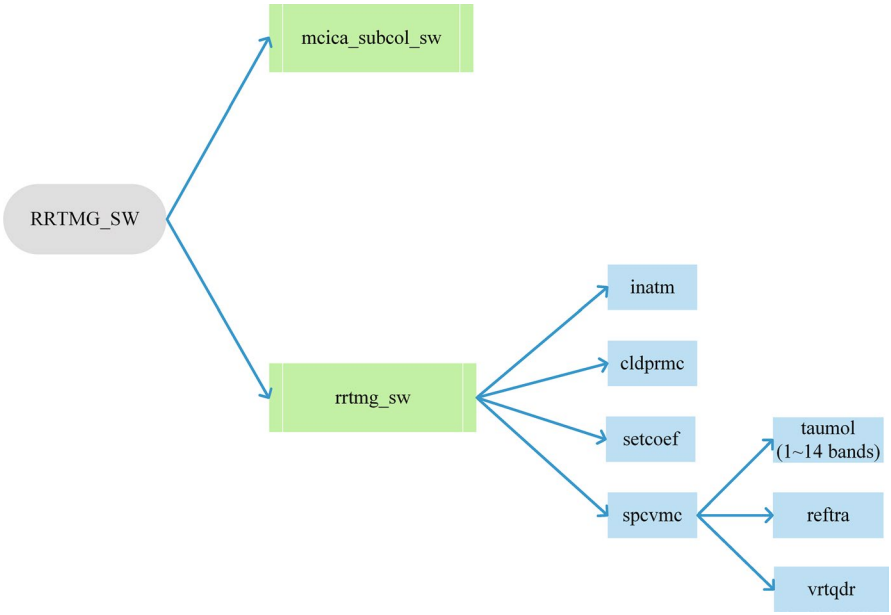


**Fig. 1** The structure of RRTMG_SW

the model. *rrtmg_sw* includes four main subroutines: *inatm* is used to read parallel plane atmospheric profile data ranging from the Earth's surface to the top of the atmosphere. *cldprmc* is used to select the parameters of the optical depth of cloud ice and liquid, and it uses cam shortwave cloud optical properties to set the cloud depth for McICA. *setcoef* is responsible for calculating the pressure correlation index and the temperature correlation score under the condition of fixed atmospheric data. *spcvmc* is responsible for calculating the spectral loop of shortwave radiative flux and realizes the overall solution of the two-stream approximation model. In addition, *spcvmc* calls *taumol*, *reftra*, and *vrtqdr*. *taumol* computes the optical depth and Planck fraction for each spectral band, *reftra* computes reflectance and atmospheric transmittance for the two-stream approximation model, and *vrtqdr* computes the vertical quadrature integral in the two-stream approximation model. Figure 2 shows the proportion of computing time for four subroutines in RRTMG_SW and CC-RRTMG_SW. As can be seen from Fig. 2, *spcvmc* accounts for a high proportion of computing time, so further improving its computing efficiency is the key to optimizing the model.

### 3.2 Parallelism analysis

RRTMG_SW uses 3D data to represent atmospheric shortwave radiation. The first dimension is the horizontal layer represented by latitude and longitude, the second dimension is the vertical layer in 3D space, and the third dimension is the special *g-point* dimension [24]. There are fourteen spectral bands in shortwave radiation.

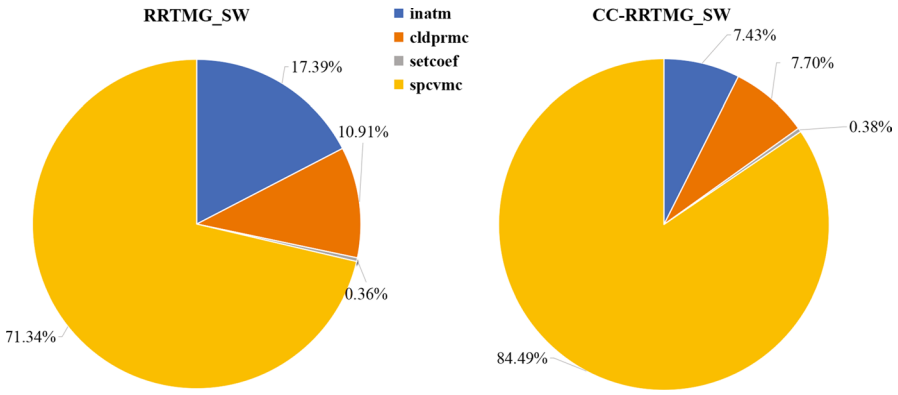


**Fig. 2** The proportion of computing time for the four subroutines in RRTMG_SW and CC-RRTMG_SW

When calculating the radiant flux of each spectral band, 112 *g-point* intervals are used to discretize the distribution function. By analyzing the model, this paper proposes a new parallel scheme in the *jp-band* dimension. *jp-band* is used to distinguish the above fourteen spectral bands. The calculation of radiative flux for each spectral band can be performed independently, so parallel computing in the *jp-band* dimension is workable.

Currently, CC-RRTMG_SW has achieved acceleration in the horizontal layer. To improve parallel scalability, this paper will tap the parallel potential of CC-RRTMG_SW in the vertical layer, *g-point*, and *jp-band* dimensions by designing appropriate methods to decouple data dependency.

## 4 Multi-dimensional acceleration algorithms

This section introduces the GPU-based 2D and 3D acceleration algorithms of RRTMG_SW, and their implementations.

### 4.1 Parallel strategy

The GPU-based acceleration algorithm of RRTMG_SW uses the CUDA programming model. Its main computing element is parallel in kernels. Figure 3 portrays its execution flow.

Assuming that the global shortwave radiation is divided into 3D grids for calculation, Fig. 4 shows the calculation tasks of *rrtmg_sw* called each time during serial calculation and the calculation tasks per thread when using parallel computing in different dimensions. In Fig. 4a, *rrtmg_sw* performs computation tasks in the horizontal, vertical or *jp-band*, and *g-point* dimensions in a serial manner. In Fig. 4b, *rrtmg_sw* uses *ncol* threads to calculate horizontal columns in the grids in a 1D parallel computing manner. In Fig. 4c, *rrtmg_sw* uses *ncol*nlay* threads to calculate the horizontal and vertical layers in a 2D parallel computing manner

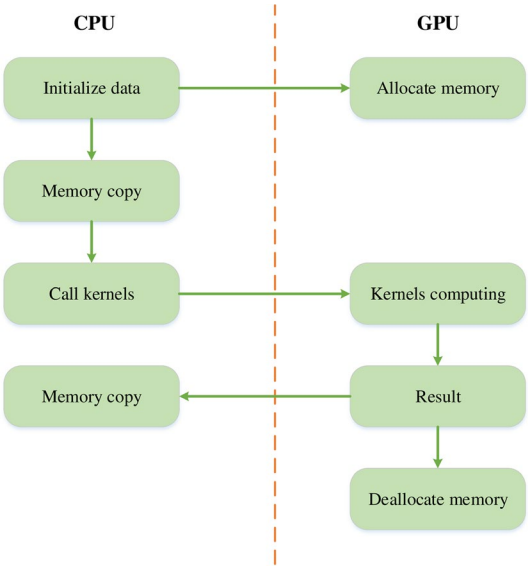


**Fig. 3** Execution flow of a parallel algorithm based on CUDA

or uses *ncol*nbndsw* threads to calculate the horizontal and *jp-band* dimensions. In Fig. 4d, *rrtmg_sw* uses *ncol*nlay*ngptsw* threads to calculate the horizontal, vertical, and *g-point* dimensions in a 3D parallel computing manner. After dividing the computing dimensions, the kernels can be launched for calculation.

### 4.2 Algorithm implementation

This section mainly describes the 2D and 3D acceleration algorithm implementation of *inatm*, *cldprmc*, *setcoef*, and *spcvmc*. Their kernels are *inatm_d*, *cldprmc_d*, *setcoef_d*, and *spcvmc_d*, respectively.

#### 4.2.1 inatm_d

Considering data dependency and data synchronization requirements, *inatm* needs to be divided into five kernels for parallel computing as fine-grained as possible, namely *inatm_d1*, *inatm_d2*, *inatm_d3*, *inatm_d4*, and *inatm_d5*, respectively. The computation of *inatm_d1* in the horizontal and vertical dimensions has no data dependency, so it uses 2D parallel computing. *inatm_d2* can perform 2D parallel computing like *inatm_d1*. *inatm_d3* has no data dependency in the horizontal, vertical, and *g-point* dimensions, so 3D parallel computing is used. In the branch sentences of *inatm*, the computing component that is not data-dependent on the horizontal and vertical dimensions uses 2D parallel computing. The kernel is named *inatm_d4*. Finally, *inatm_d5* uses 1D parallel computing for the other part of *inatm*.

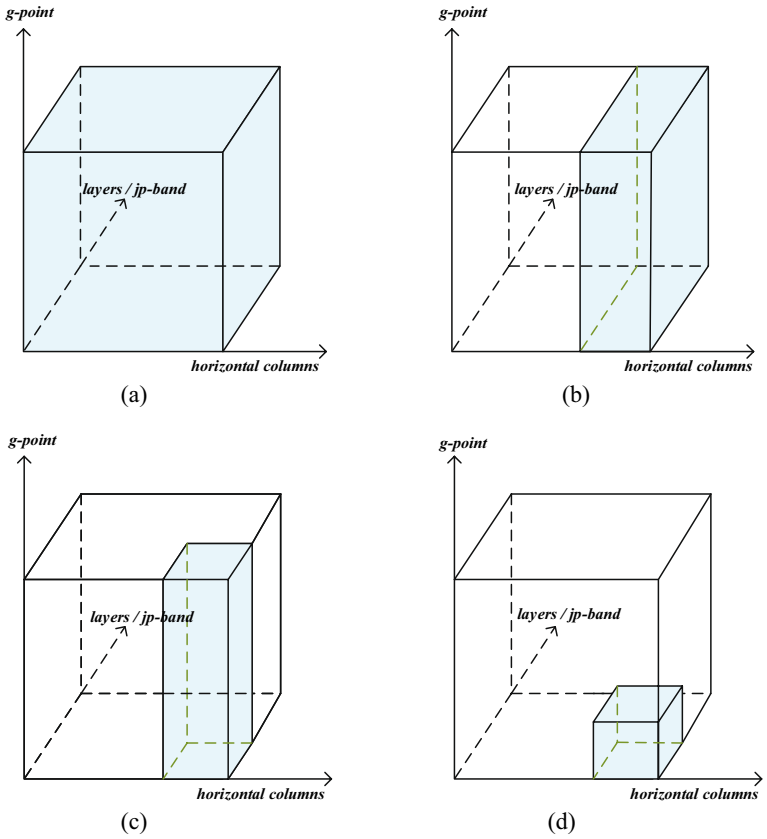


**Fig. 4** The division of computing tasks for *rrtmg_sw* in serial and parallel computing modes (a. serial computing, b. 1D parallel computing, c. 2D parallel computing, d. 3D parallel computing)

### 4.2.2 cldprmc_d

*cldprmc* is used to compute 3D arrays of cloud attribute parameters. It has no data dependency in the horizontal, vertical, and *g-point* dimensions, so it can use 3D parallel computing.

### 4.2.3 setcoef_d

To achieve multi-dimensional acceleration, *setcoef* is divided into two kernels: *setcoef_d1* and *setcoef_d2*. *setcoef_d1* does not have data dependence in the horizontal and vertical dimensions, so it uses 2D acceleration. *setcoef_d2* is with accumulation operations on the vertical dimension, so it can only be 1D parallel on the horizontal dimension.

#### 4.2.4 spcvmc_d

In RRTMG_SW, *spcvmc* is the most complicated subroutine and accounts for 71.4% of the computing time of *rrtmg_sw*. Thus, maximizing the acceleration of this subroutine is the key to improving the computing efficiency of RRTMG_SW.

*spcvmc* calls three sub-functions *taumol*, *reftra*, and *vrtqdr*, which are, respectively, used as three device functions in *spcvmc_d*. *taumol* calls fourteen sub-functions to calculate the data of fourteen bands. The calculation of fourteen bands is completely independent in the horizontal and vertical dimensions, so this paper considers *taumol_d* as a single kernel. That is, *taumol_d* is used for 2D parallel computing.

After separating *taumol_d*, there are many accumulation operations in the remaining part of *spcvmc_d*, so 2D acceleration in the horizontal and vertical dimensions is difficult. According to the analysis in Sect. 3.2, the front part of *spcvmc_d* calculates radiative flux for each band of shortwave and the computing for each band has no data dependency, so the computation for the 14 bands is 2D parallel in the horizontal and *jp-band* dimensions, as shown in *spcvmc_d1*. The remaining part of *spcvmc_d* is in *spcvmc_d2*, and it uses 1D parallel computing. Algorithm 1 shows 2D acceleration computing of *spcvmc_d1* in the horizontal and *jp-band* dimensions.

The 3D acceleration algorithm of *rrtmg_sw* is shown in Algorithm 2.

**Algorithm 1** 2D acceleration algorithm of *spcvmc_d1*

```
1: function spcvmc_d1(parameters)
2:     iplon = blockDim.x * blockIdx.x + threadIdx.x
3:     jpband = blockDim.y * blockIdx.y + threadIdx.y
       /*istart and iend are the beginning and ending of 14
       shortwave bands*/
4:     if iplon >= 0 && iplon < ncol && jpband >= istart && jpband <
   iend then
           //iw is cumulative counter
5:         iw = -1
           //ngc is the number of new g-intervals in each band
6:         igt = ngc[jpband - 15]
7:         for i ← 0, jpband - 15 do
8:             iw += ngc[i]
9:         end for
10:        for jg ← 0, igt do
11:            iw ++
12:            do necessary calculations
13:            for jk ← 0, nlayers do
14:                do necessary calculations
15:            end for
               //calculate clear sky reflectivities
16:            call device function reftra_sw
               //calculate total sky reflectivities
17:            call device function reftra_sw
18:            do some necessary calculations
19:        end for
20:    end if
21: end function
```

**Algorithm 2** 3D acceleration algorithm of *rrtmg_sw*

```
1: function rrtmg_sw(parameters)
2:     dim3 grid1, tBlock1, grid2, tBlock2
3:     parallel parameters assignment in 2D and 3D
       /*loop n times until atmospheric horizontal profile
       data computing is completed*/
4:     for i ← 0, n do
5:         copy input data to GPU device
           //call the kernel inatm_d1 with 2D decomposition
6:         inatm_d1<<<grid1,tBlock1>>>(parameters)
           //call the kernel inatm_d2 with 2D decomposition
7:         inatm_d2<<<grid1,tBlock1>>>(parameters)
           //call the kernel inatm_d3 with 3D decomposition
8:         inatm_d3<<<grid2,tBlock2>>>(parameters)
           //call the kernel inatm_d4 with 2D decomposition
9:         inatm_d4<<<grid1,tBlock1>>>(parameters)
           //call the kernel inatm_d5 with 1D decomposition
10:        inatm_d5<<<blockNum,threadNum>>>(parameters)
           //call the kernel cldprmc_d with 3D decomposition
11:        cldprmc_d<<<grid2,tBlock2>>>(parameters)
           //call the kernel setcoef_d1 with 2D decomposition
12:        setcoef_d1<<<grid1,tBlock1>>>(parameters)
           //call the kernel setcoef_d2 with 1D decomposition
13:        setcoef_d2<<<blockNum,threadNum>>>(parameters)
           //call the kernel taumol_d with 2D decomposition
14:        taumol_d<<<grid1,tBlock1>>>(parameters)
           //call the kernel spcvmc_d1 with 2D decomposition
15:        spcvmc_d1<<<grid1,tBlock1>>>(parameters)
           //call the kernel spcvmc_d2 with 1D decomposition
16:        spcvmc_d2<<<blockNum,threadNum>>>(parameters)
17:        copy result to host
18:    end for
19: end function
```

## 5 Optimization methods

CC-RRTMG_SW++ has 11 kernels. Each kernel needs to transfer I/O data. When kernels execute, the data need to be accessed through global memory. The time of these operations has become a bottleneck in improving computing efficiency, so further optimizing CC-RRTMG_SW++ is urgent. The optimized methods include decoupling data dependency to improve parallelism, using temporary registers to

improve memory access performance, avoiding unnecessary data transfers, and using CUDA streams [25] to implement asynchronous data transfer.

### 5.1 Decoupling data dependency

Data dependency often makes it so that algorithms cannot be parallel. In most cases, data dependency is inevitable. However, some appropriate methods can be utilized to decouple data dependency. Figure 5 shows the methods of decoupling data dependency. The initial decoupling method is to separate computing processes by increasing the number of kernels. However, the increase in the number of kernels will increase the time of kernels launch. Grid-level synchronization can be used to eliminate the time cost of kernels launch. Another decoupling method is to set temporary variables and increase the computing processes so that the dependent data are stored in temporary variables.

In CC-RRTMG_SW++, the calculation dependency in the horizontal dimension is decoupled by increasing the dimension of arrays so as to achieve the purpose of 1D parallel computing. Then, as shown in Sect. 4.2, by dividing *inatm_d*, *setcoef_d*, and *spcvmc_d* into smaller kernels, the calculation dependency in the vertical,

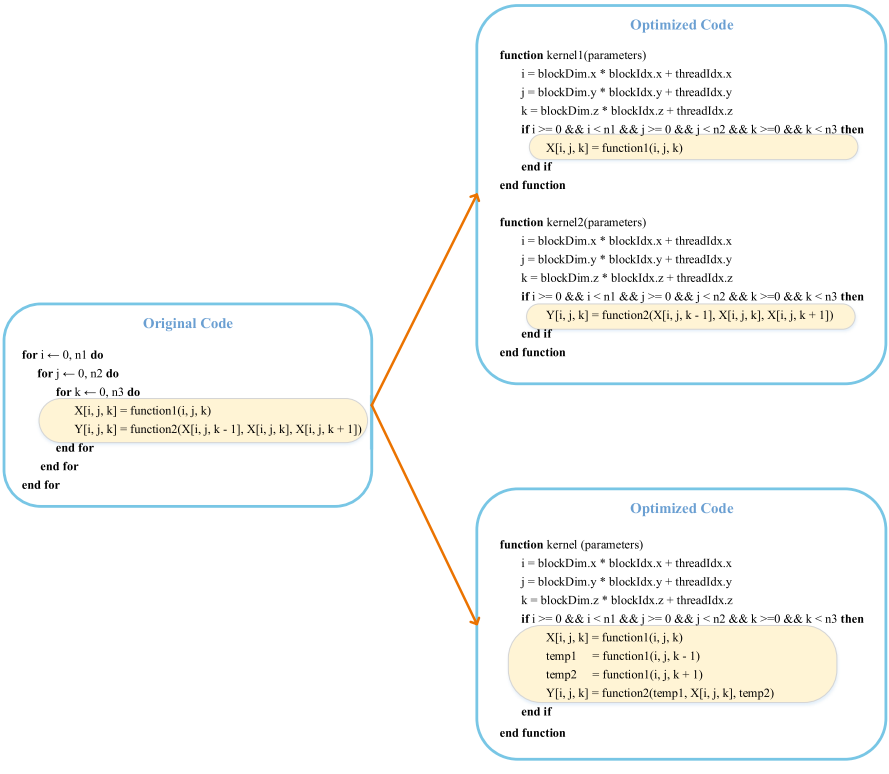


**Fig. 5** Decoupling data dependency

*jp-band*, and *g-point* dimensions is decoupled. Therefore, 2D and 3D parallel computing can be performed on these kernels.

### 5.2 Optimizing memory access

There are different kinds of memory in GPUs, whose size, access methods, and memory access speeds are all different. In CUDA programming, if the memory is not specified, the data of each thread need to be accessed from global memory. However, the speed of accessing global memory is much lower than that of other types of memory, such as texture memory, shared memory, and registers. In modern multi-core accelerators, memory access latency has become the main bottleneck to improving program performance [26]. Therefore, it is necessary to minimize the access latency of global memory.

In CC-RRTMG_SW++, *spcvmc_d* still accounts for more than 90% of the running time of *rrtmg_sw*, so it is meaningful to further optimize the kernel. After analyzing its code structure and different characteristics of GPU memories, in order to eliminate repeated global memory access, the data depot is used for optimization. The data in the global memory are temporarily stored in the data depot before being involved in the operation. The requirement of the data depot is that its access efficiency is much higher than that of the global memory, the shared memory and registers in GPU just meet this requirement. When using shared memory as a data depot, since shared memory is shared by threads in the same block, it is necessary to increase the calculation process in CC-RRTMG_SW++ to eliminate the memory writing conflicts among different threads, which weakens performance improvement by using the data depot. Therefore, this paper chooses to use registers as a data depot.

The specific method is to set more temporary variables in the kernel, and these temporary variables are stored in the registers of each streaming multiprocessor in GPU. Accessing registers is much faster than accessing global memory, but registers are finite resources. If the thread blocks use too many registers, the GPU kernel occupation will be reduced, thereby reducing the occupancy rate of the multi-core processors. Therefore, it is not that more registers being used lead to a more efficient program [27]. In *spcvmc_d2*, through many experiments and performance analysis, considering the best balance between register usage and occupancy, six arrays are temporarily stored in registers. To make better use of the registers, we reduce the original 2D arrays to 1D, which means that the unnecessary horizontal dimension of the arrays is removed when calculating inside the kernel. After finishing computation, their dimensionality will be upgraded. That is, the 1D arrays in the registers will be reassigned to 2D arrays in global memory. Figure 6 shows the memory access methods for *spcvmc_d2* before and after optimization.

### 5.3 I/O optimization

CC-RRTMG_SW++ uses heterogeneous hybrid computing, which is a method of "CPU+GPU" collaborative computing. The logic control is completed by CPU, and the core computation is completed by GPU, so the data transfer between host and

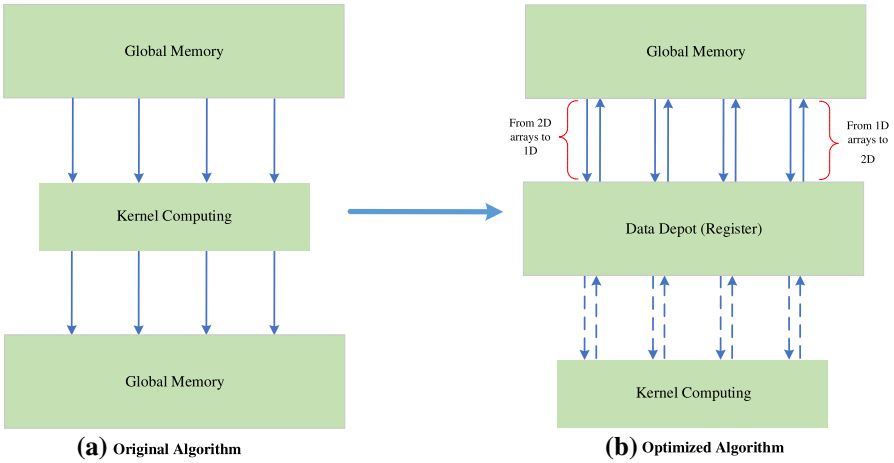


**Fig. 6** The optimization strategy of global memory access

device is inevitable [28]. Due to the limitation of PCIe bus bandwidth, data transfer takes up a large amount of runtime [29]. In CC-RRTMG_SW, the method of pinning memory has been used to optimize data transfer. However, the data transfer still occupies *rrtmg_sw* for nearly 50% of the running time, so further optimization is required.

### 5.3.1 Avoiding unnecessary data transfer

In CC-RRTMG_SW, data are transferred among different kernels through the device-host-device approach, which means that the intermediate data calculated by a kernel need to be transmitted back to CPU and the data are transmitted to the next kernel after doing simple calculations in CPU. This approach involves a large amount of meaningless intermediate data transfer, so data in CC-RRTMG_SW++ are kept in GPU and only the necessary data are transferred back to CPU. The calculations in CPU are also ported to kernels. In this way, the cost of data transfer can be effectively reduced.

### 5.3.2 CUDA stream

A CUDA stream is a series of operations issued by host and executed sequentially on device. After creating multiple CUDA streams, different tasks can be assigned to different streams. Each CUDA stream has to complete three processes in turn: copying the data from host to device, computing the kernel, and copying computing results to the host. For different streams, as long as the computing and data transfer do not depend on each other, they can be performed synchronously. In this way, the computing and data transfer can overlap. When using CUDA streams, pinned memory and the asynchronous copy function *cudaMemcpyAsync* need to be used.

This paper uses CUDA streams to optimize the I/O transfer of *spcvmc_d1* and *spcvmc_d2*, whose computation time and data transfer time account for a large proportion of the entire model. Four CUDA streams are used to divide the calculations of *spcvmc_d1* and *spcvmc_d2* in the horizontal grid, 1/4 of the horizontal column data (*ncol*/4) is transferred to the GPU's memory by a stream. The pointer offset address is set to calculate the horizontal column data, which have currently been transferred to the GPU. The time of the data transfer can be overlapped by transferring data at the same time as computing on different streams. Figure 7 illustrates the difference between using the default stream and using multiple streams in *spcvmc_d1* and *spcvmc_d2*.

## 6 Results and discussion

This paper conducts numerical experiments to evaluate the performance of the proposed acceleration algorithms and optimization methods.

### 6.1 Experimental setup

The experiments use three different GPU clusters: a K20 cluster, a K40 cluster, and a V100 cluster. The K20 cluster is located in the Computer Network Information Center of CAS, the V100 cluster is located in the Institute of Atmospheric Physics of CAS, and the K40 cluster is located in China University of Geosciences (Beijing). Their hardware configurations are listed in Table 1. The RRTMG_SW runs on a single Intel Xeon E5-2680 v2 CPU core of the K20 cluster. The CC-RRTMG_SW++ heterogeneous code runs on a single node of each cluster.

This paper evaluates the performance of CC-RRTMG_SW++ from three perspectives: parallel acceleration, memory access, and I/O transfer. The evaluation criterion is the speedup of CC-RRTMG_SW++ compared to RRTMG_SW and CC-RRTMG_SW. To make CC-RRTMG_SW++ achieve the best performance on three kinds of GPU, the block size for the 1D acceleration kernels is 128, and the block

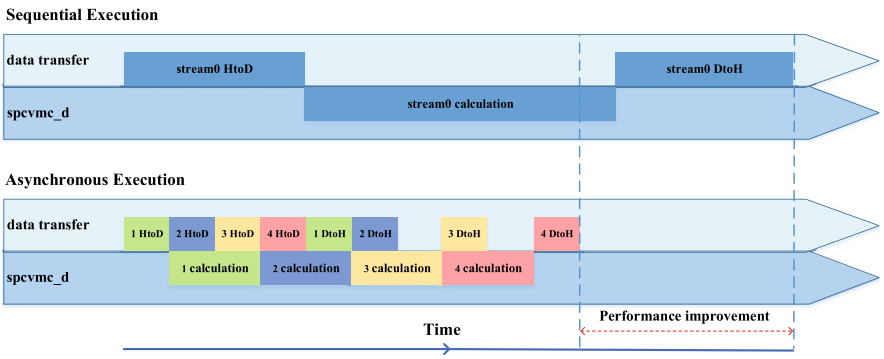


**Fig. 7** Execution flow chart of *spcvmc_d* using multiple CUDA streams

**Table 1** Configurations of GPU Clusters

| Specification of CPU | K20 cluster | K40 cluster | V100 cluster |
|---|---|---|---|
| CPU | Intel Xeon E5-2680 v2@2.8GHz | Intel Xeon E5-2620 v4@2.10GHz | Intel Xeon Gold 6240R @2.40GHz |
| Operating System | CentOS 6.4 | Red Hat 4.8.5-36 | Red Hat 4.8.5-44 |
| **Specification of GPU** | **K20 cluster** | **K40 cluster** | **V100 cluster** |
| GPU | NVIDIA Tesla K20 | NVIDIA Tesla K40 | NVIDIA Tesla V100 |
| CUDA Cores | 2496 | 2880 | 5120 |
| CUDA Version | 6.5 | 10.0 | 10.1 |
| Standard Memory | 5 GB | 12 GB | 32 GB |
| Memory Bandwidth | 208 GB/s | 288 GB/s | 900 GB/s |
| GFLOPS in double precision | 1.17TFLOPS | 1.43TFLOPS | 7.5TFLOPS |

size for the 2D and 3D acceleration kernels is 512. RRTMG_SW has 128×256 horizontal grid points when simulating the global shortwave radiation transfer process, with a resolution of $1.4° \times 1.4°$. When the size of *ncol* is set to 1024, the model needs to be called repeatedly: (128×256/1024)×24=768 times, for the simulation of one model day (24 hours). When the size of *ncol* increases, the number of calls will decrease accordingly. The following simulation experiments are all carried out for one model day.

## 6.2 Evaluation of optimization methods

### 6.2.1 3D acceleration

First, taking the running time of serial RRTMG_SW as the standard, a comparative experiment is conducted for CC-RRTMG_SW and CC-RRTMG_SW++. The speedup is the ratio of serial computing time to parallel computing time. The time of *rrtmg_sw* ($T_{rrtmg\_sw}$) is calculated with the following formula:

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

$T_{\text{inatm}}$ is the computing time of the subroutine *inatm* or the accumulation of kernels *inatm_d*1, *inatm_d*2, *inatm_d*3, *inatm_d*4, and *inatm_d*5. $T_{\text{cldprmc}}$, $T_{\text{setcoef}}$, $T_{\text{taumol}}$, and $T_{\text{spcvmc}}$ are the corresponding computing times of the subroutines or kernels.

The experiment uses one K20 GPU, and the size of *ncol* is set to 1024. Table 2 shows the speedup of CC-RRMTG_SW++ optimized with 3D parallelism and decoupling data dependency. As shown in Table 2, compared with RRTMG_SW, in the kernels with a higher degree of parallelism, such as *inatm*, *cldprmc*, and *setcoef*, the speedup increases significantly compared to 1D parallelism. It also has a speedup of 3.80× for the most time-consuming kernel, *spcvmc*, so the overall speedup of CC-RRTMG_SW++ with 3D acceleration computing is 5.05×; this is 1.84 times faster than CC-RRTMG_SW. Therefore, the 3D parallel method is very efficient.

**Table 2** Runtime (s) and speedup of CC-RRTMG_SW and CC-RRTMG_SW++ on a single K20 GPU when *ncol*=1024

| Subroutines | Serial time | 1D acceleration | Speedup | 3D acceleration | Speedup |
|---|---|---|---|---|---|
| *inatm* | 131.99 | 20.5307 | 6.43 | 4.2936 | 30.74 |
| *cldprmc* | 82.76 | 21.2583 | 3.89 | 3.3132 | 24.98 |
| *setcoef* | 2.76 | 1.0595 | 2.61 | 0.2069 | 13.34 |
| *taumol* | – | 13.2928 | 2.32 | 2.2815 | 3.80 |
| *spcvmc* | 541.38 | 220.0960 | | 140.1200 | |
| *rrtmg_sw* | 758.89 | 276.2373 | **2.75** | 150.2152 | **5.05** |

The bold values are used to represent critical information

### 6.2.2 Memory access optimization

Table 3 compares the running time of *spcvmc* before and after memory access optimization. By using the method described in Sect. 5.2, the overall computing efficiency of *spcvmc* on one Tesla V100 GPU improves by 30.11%. Given the highly time-consuming proportion of the *spcvmc* in the model, this magnitude of performance improvement is very meaningful.

### 6.2.3 I/O optimization

Table 4 shows the time and speedup of I/O transfer before and after optimizing in CC-RRTMG_SW++. *CUDA Memory HtoD* represents the data transfer from host to device, and *CUDA Memory DtoH* represents the data transfer from device to host. Through the I/O optimization methods specified in Sect. 5.3, the time of data transfer has been greatly reduced. By avoiding unnecessary data transfer, the transfer time from device to host is greatly reduced, even negligible. The transfer time from host to device is 1.62 times faster by using CUDA streams. In addition, the speed of data transfer is related to the PCIe bus bandwidth between host and device, so the

**Table 3** The performance of optimizing memory access for *spcvmc*

| Kernel | Original (K20) | Optimization (K20) | Improvement | Original (V100) | Optimization (V100) | Improvement |
|---|---|---|---|---|---|---|
| *spcvmc* | 140.1200 | 119.7340 | **17.03%** | 44.7590 | 34.4013 | **30.11%** |

The bold values are used to represent critical information

**Table 4** The time and speedup of I/O transfer before and after optimizing for CC-RRTMG_SW++

| Transfer direction | Original (K20) | Optimization (K20) | Speedup | Original (V100) | Optimization (V100) | Speedup |
|---|---|---|---|---|---|---|
| CUDA Memcpy HtoD | 78.9733 | 48.6722 | **1.62** | 38.6182 | 23.7652 | **1.62** |
| CUDA Memcpy DtoH | 66.8794 | 0.9980 | **67.01** | 34.0014 | 0.5072 | **66.04** |

The bold values are used to represent critical information

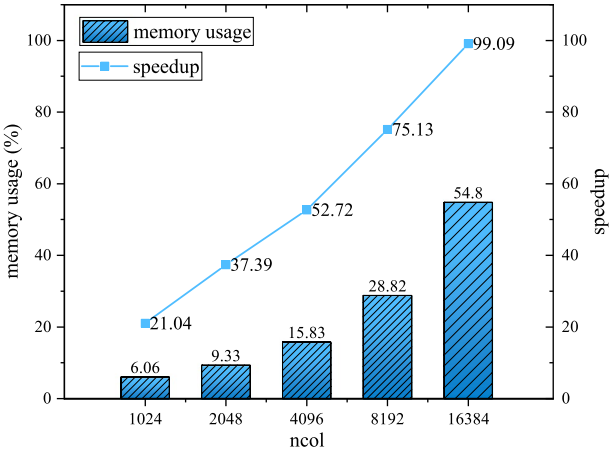


**Fig. 8** The influence of *ncol* on the speedup and memory usage of CC-RRTMG_SW++ on a V100 GPU

speedup of the data transfer time on K20 GPU and V100 GPU is almost the same when the PCIe bus bandwidth is unchanged.

## 6.3 Overall performance evaluation

### 6.3.1 Evaluation on different GPUs

This paper conducts experiments on the overall performance of CC-RRTMG_SW++ on three different GPUs and compares its speedup relative to RRTMG_SW. The experimental conditions are specified in Sect. 6.1. The results are given in Tables 5, 6, and 7. For the calculation method of the parameters in the three tables, please refer to Sect. 6.2.1. The experimental results show that CC-RRTMG_SW++ has reached a speedup of 99.09× on a single Tesla V100 GPU. For the kernels *inatm* and *cldprmc* with high parallelism, the speedup is more than 150×. The results demonstrate the effectiveness and efficiency of 3D parallel computing and the performance optimization methods for RRTMG_SW.

Due to the differences in video memory size, the maximum value of *ncol* on different types of GPUs is different. Recalling the instructions in Sect. 6.1, increasing

**Table 5** CC-RRTMG_SW++ runtime (s) and speedup on a single K20 GPU

| Subroutines | Serial time | *ncol*=1024 | speedup | *ncol*=2048 | speedup | *ncol*=4096 | speedup |
|---|---|---|---|---|---|---|---|
| *inatm* | 131.99 | 4.2872 | 30.79 | 4.0958 | 32.23 | 4.1206 | 32.03 |
| *cldprmc* | 82.76 | 3.3078 | 25.02 | 3.1775 | 26.05 | 3.2582 | 25.40 |
| *setcoef* | 2.76 | 0.2014 | 13.70 | 0.1379 | 20.01 | 0.1092 | 25.27 |
| *taumol* | – | 2.2848 | 4.44 | 2.1787 | 7.33 | 2.1018 | 10.69 |
| *spcvmc* | 541.38 | 119.7340 | | 71.6782 | | 48.5506 | |
| *rrtmg_sw* | 758.89 | 129.8152 | **5.85** | 81.2681 | **9.34** | 58.1404 | **13.05** |

The bold values are used to represent critical information

**Table 6** CC-RRTMG_SW++ runtime (s) and speedup on a single K40 GPU

| Subroutines | Serial time | ncol=1024 | speedup | ncol=2048 | speedup | ncol=4096 | speedup | ncol=8192 | Speedup |
|---|---|---|---|---|---|---|---|---|---|
| *inatm* | 131.99 | 1.7262 | 76.46 | 1.6577 | 79.62 | 1.6683 | 79.12 | 1.6476 | 80.11 |
| *cldprmc* | 82.76 | 1.0794 | 76.67 | 1.0991 | 75.30 | 1.0859 | 76.21 | 1.0870 | 76.14 |
| *setcoef* | 2.76 | 0.09814 | 28.12 | 0.06240 | 44.23 | 0.04721 | 58.46 | 0.03916 | 70.48 |
| *taumol* | – | 0.5944 | 9.37 | 0.5561 | 15.10 | 0.5362 | 23.40 | 0.5199 | 33.33 |
| *spcvmc* | 541.38 | 57.2048 | | 35.2991 | | 22.6001 | | 15.7245 | |
| *rrtmg_sw* | 758.89 | 60.7029 | **12.50** | 38.6744 | **19.62** | 25.9377 | **29.26** | 19.0182 | **39.90** |

The bold values are used to represent critical information

*ncol* reduces the number of kernel calls and indirectly increases the parallelism of the kernel in the horizontal dimension. Therefore, it can be seen from the experimental results that with the increase of *ncol*, the speedup of *setcoef* and *spcvmc* has been greatly improved. For *inatm* and *cldprmc*, the calculation process is simple and there are many memory access operations. With the increase of *ncol*, the higher memory access cost limits the parallel efficiency of kernels, so there will be cases where the speedup is not significantly improved or even slightly decreased. In addition, increasing the value of *ncol* also means that the horizontal dimension of the arrays increases, and more GPU memory is required for each call to *rrtmg_sw*. Figure 8 shows the impact of *ncol* on the speedup and memory usage of *rrtmg_sw* on a V100 GPU. When *ncol* is set to 16384, the memory usage reaches 54.80%, but the speedup of *rrtmg_sw* is 4.71 times higher than that of *ncol*=1024.

Because of the differences in video memory, bandwidth, and double-precision floating-point arithmetic capability for different GPUs, the computing performance of CC-RRTMG_SW++ is highly diverse. Figure 9 compares the performance of CC-RRTMG_SW++ on the K20, K40, and V100 GPUs. On a single Tesla V100 GPU, the highest speedup of CC-RRTMG_SW++ is 7.59 times faster than K20 and is 2.48 times faster than K40.

### 6.3.2 With I/O transfer

With I/O transfer, the overall performance of CC-RRTMG_SW++ is given in Table 8. The time of I/O transfer ($T_{I/O}$) and rrtmg_sw ($T_{rrtmg\_sw}$) is calculated with the following formulas:

$$T_{\mathrm{I/O}} = T_{\mathrm{HtoD}} + T_{\mathrm{DtoH}}, T_{rrtmg\_sw} = T_{\mathrm{computing}} + T_{\mathrm{I/O}}$$

$T_{HtoD}$ is the data transfer from host to device, $T_{DtoH}$ is the data transfer from device to host, and $T_{computing}$ is the sum of the calculation times of all kernels. Due to the limitations of PCIe bus bandwidth and frequent communication, the data transfer between host and device still costs plenty of time after optimizing I/O. However, after the use of optimization methods, the speedup of CC-RRTMG_SW++ is still up to 24.07×, which is better than CC-RRTMG_SW.

**Table 7** CC-RRTMG_SW++ runtime (s) and speedup on a single V100 GPU

| Subroutines | Serial time | ncol=1024 | Speedup | ncol=2048 | Speedup | ncol=4096 | Speedup | ncol=8192 | Speedup | ncol=16384 | Speedup |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *inatm* | 131.99 | 0.8625 | 153.03 | 0.8200 | 160.96 | 0.7756 | 170.18 | 0.7844 | 168.27 | 0.7584 | 174.04 |
| *cldprmc* | 82.76 | 0.5347 | 154.78 | 0.5631 | 146.97 | 0.5485 | 150.88 | 0.5725 | 144.56 | 0.5491 | 150.72 |
| *setcoef* | 2.76 | 0.04914 | 56.17 | 0.03122 | 88.40 | 0.02256 | 122.34 | 0.01863 | 148.15 | 0.01656 | 166.67 |
| *taumol* | – | 0.2265 | 15.63 | 0.1824 | 28.67 | 0.1682 | 41.49 | 0.1564 | 62.04 | 0.1617 | 85.47 |
| *spcvmc* | 541.38 | 34.4013 | | 18.7002 | | 12.8798 | | 8.5695 | | 6.1725 | |
| *rrtmg_sw* | 758.89 | 36.0741 | **21.04** | 20.2969 | **37.39** | 14.3947 | **52.72** | 10.1014 | **75.13** | 7.6583 | **99.09** |

The bold values are used to represent critical information

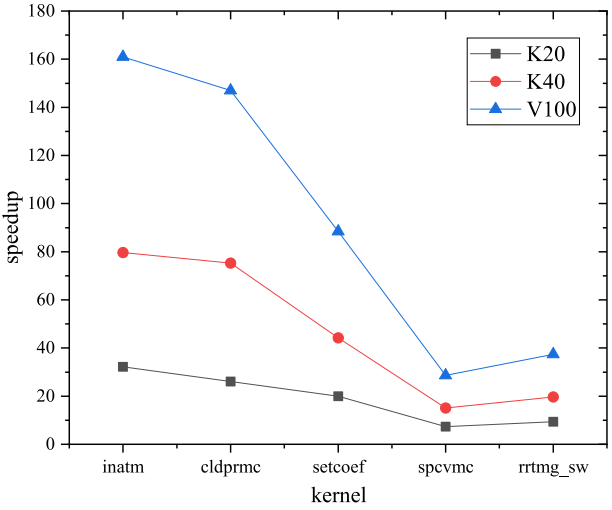


**Fig. 9** The influence of different GPUs on the speedup of CC-RRTMG_SW++ when *ncol*=2048

**Table 8** Running time (s) and speedup of CC-RRTMG_SW++ with I/O transfer on different GPUs

| GPU | ncol | Computing time | CUDA Memcpy HtoD | CUDA Memcpy DtoH | I/O transfer | *rrtmg_sw* | Speedup |
|---|---|---|---|---|---|---|---|
| K20 | 4096 | 58.1404 | 47.1573 | 0.8047 | 47.9620 | 106.1024 | **7.15** |
| K40 | 8192 | 19.0182 | 24.6021 | 0.5238 | 25.1259 | 44.1441 | **17.19** |
| V100 | 16384 | 7.6583 | 23.3652 | 0.5012 | 23.8664 | 31.5247 | **24.07** |

The bold values are used to represent critical information

## 6.4 Accuracy verification

In order to ensure the accuracy of the CC-RRTMG_SW++, this paper carried out an error experiment of CC-RRTMG_SW++ and RRTMG_SW. The method is to call *rrtmg_sw* in the serial Fortran version and the CUDA C version, respectively, and subtract the results after the calls to obtain the error. The average error of CC-RRTMG_SW++ obtained by this method compared to RRTMG_SW is *-0.000455583* $W/m^2$, which is consistent with CC-RRTMG_SW. The reason for error is the difference in computing accuracy between the CPU and GPU, and the difference is magnified by many cumulative calculations in the code. Due to the error in accuracy of the numerical model itself, it cannot simulate the atmospheric radiative transfer process with absolute accuracy. Therefore, the error in accuracy between CC-RRTMG_SW++ and RRTMG_SW is acceptable.

### 6.5 Discussion

The overall performance of CC-RRTMG_SW++ and CC-RRTMG_SW is compared in Fig. 10. In CC-RRTMG_SW++, both the calculation time of the kernels and the I/O transfer time are much less than in CC-RRTMG_SW. Taking the calculation time as a measure of computing efficiency, the average computing efficiency of CC-RRTMG_SW++ on three different types of GPUs is 174.46% higher than that of CC-RRTMG_SW. Compared to Mielikainen et al. [2], their CUDA C-based 1D parallel RRTMG_SW achieved a speedup of 202× on a single Tesla K40 GPU, but they compared the serial RRTMG_SW with the grid size of 425×308, which is much higher than the grid size of 128×256 in this paper; thus, they could achieve higher horizontal dimension parallelism. If this paper used RRTMG_SW with a higher grid resolution as the comparison, the speedup would be further improved. Thus, the CC-RRTMG_SW++ proposed in this paper is very effective, and it provides a more efficient scheme for accelerating atmospheric physics on GPUs.

In the future, if a GPU version of the entire atmospheric general circulation model or all atmospheric physics processes is developed, the initialization data in CC-RRTMG_SW++ only need to be transferred once from the host to the device. This means that the cost of data transfer is almost negligible, so CC-RRTMG_SW++ will achieve a high computing efficiency.

## 7 Conclusions and future work

High-efficiency computation on GPU is always challenging. This paper proposes a multi-dimensional acceleration algorithm for RRTMG_SW and some GPU-based performance optimization methods. Then, CC-RRTMG_SW++, an optimized

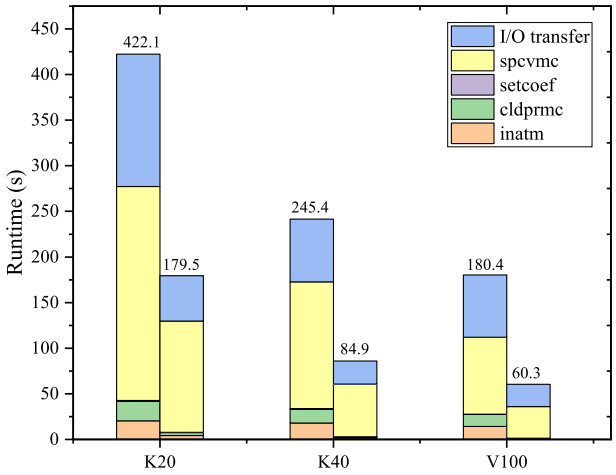


**Fig. 10** The runtime (s) of CC-RRTMG_SW (left) and CC-RRTMG_SW++ (right) on three different GPUs when *ncol*=1024

version of CC-RRTMG_SW, is developed. CC-RRTMG_SW++ further exploits the advantages of GPU multi-core computing capabilities. The experimental results prove the effectiveness and high efficiency of CC-RRTMG_SW++. Thus, CC-RRTMG_SW++ can support the higher-resolution computation of shortwave radiative transfer. This is of great significance to the development of the atmospheric general circulation model.

Future work will begin from multi-GPU acceleration and mixed-precision computing. (1) Supercomputers usually have hundreds of thousands of CPU and GPU nodes. In order to make full use of these nodes, the "MPI+CUDA" hybrid programming will be utilized to further accelerate CC-RRTMG_SW++ [30, 31]. (2) The mixed-precision computing of CC-RRTMG_SW++ will also be considered. In recent years, to reduce computing cost and improve computing efficiency, mixed-precision computing has become a hot research topic in high-performance computing. When accuracy errors exist objectively in many applications, most of the variables are actually not necessary to use double-precision computation [32, 33]. Therefore, half-precision, single-precision, and double-precision mixed computing [34, 35] will be quite promising work in CC-RRTMG_SW++.

**Acknowledgements** This work was supported in part by the National Natural Science Foundation of China under Grant 41931183, in part by the National Key Scientific and Technological Infrastructure project "Earth System Science Numerical Simulator Facility" (EarthLab), and in part by the GHFUND A under Grant ghfund202107013661.

## Declarations

**Code availability** The code generated and analyzed during this study is available in the Github repository: https://github.com/guirenbenxin/Heterogeneous-RRTMG_SW.

## References

1. Javadinejad S, Eslamian S, Ostad-Ali-Askari K (2021) The analysis of the most important climatic parameters affecting performance of crop variability in a changing climate. Int J Hydrol Sci Technol 11(1):1–25
2. Mielikainen J, Price E, Huang B, Huang HLA, Lee T (2015) GPU compute unified device architecture (CUDA)-based parallelization of the RRTMG shortwave rapid radiative transfer model. IEEE J Selected Topics Appl Earth Observ Remote Sens 9(2):921–931
3. Michalakes J, Vachharajani M (2008) GPU acceleration of numerical weather prediction. Parallel Process Lett 18(04):531–548
4. Clough S, Shephard M, Mlawer E, Delamere J, Iacono M, Cady-Pereira K, Boukabara S, Brown P (2005) Atmospheric radiative transfer modeling: a summary of the AER codes. J Quantit Spectroscopy Radiative Transf 91(2):233–244
5. Mlawer EJ, Taubman SJ, Brown PD, Iacono MJ, Clough SA (1997) Radiative transfer for inhomogeneous atmospheres: RRTM, a validated correlated-k model for the longwave. J Geophys Res: Atmos 102(D14):16663–16682
6. Iacono MJ, Delamere JS, Mlawer EJ, Shephard MW, Clough SA, Collins WD (2008) Radiative forcing by long-lived greenhouse gases: calculations with the AER radiative transfer models. J Geophys Res: Atmos 113:13
7. Pervin L, Gan TY (2021) Sensitivity of physical parameterization schemes in WRF model for dynamic downscaling of climatic variables over the MRB. J Water Clim Change 12(4):1043–1058

8. Bae SY, Hong SY, Lim KSS (2016) Coupling WRF double-moment 6-class microphysics schemes to RRTMG radiation scheme in weather research forecasting model. Adv Meteorol 2016:84
9. Zhang H, Zhang M, Zeng QC (2013) Sensitivity of simulated climate to two atmospheric models: interpretation of differences between dry models and moist models. Monthly Weather Rev 141(5):1558–1576
10. Wang Y, Yan X, Zhang J (2021) Research on GPU parallel algorithm for direct numerical solution of two-dimensional compressible flows. J Supercomput 77(10):10921–10941
11. Ramon D, Steinmetz F, Jolivet D, Compiègne M, Frouin R (2019) Modeling polarized radiative transfer in the ocean-atmosphere system with the GPU-accelerated SMART-G Monte Carlo code. J Quantit Spectroscopy Radiative Transf 222:89–107
12. Kelly R (2010) GPU computing for atmospheric modeling. Comput Sci Eng 12(4):26–33
13. Wang Y, Zhao Y, Li W, Jiang J, Ji X, Zomaya AY (2019) Using a GPU to accelerate a longwave radiative transfer model with efficient CUDA-based methods. Appl Sci 9(19):4039
14. Wang Z, Wang Y, Wang X, Li F, Zhou C, Hu H, Jiang J (2021) GPU-RRTMG_SW: Accelerating a Shortwave Radiative Transfer Scheme on GPU. IEEE Access 25:6681
15. Ghorpade, J., Parande, J., Kulkarni, M., Bawaskar, A.: GPGPU processing in CUDA architecture. http://arxiv.org/abs/1202.4347 (2012)
16. Huang M, Huang B, Chang YL, Mielikainen J, Huang HLA, Goldberg MD (2015) Efficient parallel GPU design on WRF five-layer thermal diffusion scheme. IEEE J Selected Topics Appl Earth Observ Remote Sens 8(5):2249–2259
17. Leutwyler D, Fuhrer O, Lapillonne X, Lüthi D, Schär C (2016) Towards European-scale convection-resolving climate simulations with GPUs: a study with COSMO 4.19. Geosci Model Develop 9(9):3393–3412
18. Mielikainen J, Huang B, Huang HL, Goldberg M, Mehta A (2013) Speeding up the computation of WRF double-moment 6-class microphysics scheme with GPU. J Atmos Oceanic Technol 30(12):2896–2906
19. Cao, H., Yuan, L., Zhang, H., Zhang, Y.: AGCM-3DLF: Accelerating Atmospheric General Circulation Model via 3D Parallelization and Leap-Format. http://arxiv.org/abs/2103.10114 (2021)
20. Lu, F., Cao, X., Song, J., Zhu, X.: GPU computing for longwave radiation physics: A RRTM_LW scheme case study. In: 2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications Workshops, pp. 71–76. IEEE (2011)
21. Mielikainen J, Huang B, Huang HLA, Goldberg MD (2012) GPU acceleration of the updated goddard shortwave radiation scheme in the weather research and forecasting (WRF) model. IEEE J Selected Topics Appl Earth Observ Remote Sens 5(2):555–562
22. Price E, Mielikainen J, Huang M, Huang B, Huang HLA, Lee T (2014) GPU-accelerated longwave radiation scheme of the rapid radiative transfer model for general circulation models (RRTMG). IEEE J Selected Topics Appl Earth Observ Remote Sens 7(8):3660–3667
23. Shi, G.Y.: On the k-distribution and correlated k-distribution models in the atmospheric radiation calculations. Scientia Atmospherica Sinica (Special Issue Dedicated to the 70 < th> Anniversary of the Founding of the Institute of Atmospheric Physics, Chinese Academy of Sciences) **22**(4), 555–576 (1998)
24. Wang Y, Zhao Y, Jiang J, Zhang H (2020) A novel GPU-based acceleration algorithm for a longwave radiative transfer model. Appl Sci 10(2):649
25. Li, X., Ye, H., Zhang, J.: Redesigning Peridigm on SIMT accelerators for High-performance Peridynamics Simulations. In: 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 433–443. IEEE (2021)
26. Xu J, Fu H, Luk W, Gan L, Shi W, Xue W, Yang C, Jiang Y, He C, Yang G (2019) Optimizing finite volume method solvers on NVIDIA GPUs. IEEE Trans Parallel Distrib Syst 30(12):2790–2805
27. Fu, H., Xu, J., Gan, L., Yang, C., Xue, W., Zhao, W., Shi, W., Wang, X., Yang, G.: Unleashing the performance potential of CPU-GPU platforms for the 3D atmospheric Euler solver. In: 2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP), pp. 41–49. IEEE (2016)
28. Yang C, Xue W, Fu H, Gan L, Li L, Xu Y, Lu Y, Sun J, Yang G, Zheng W (2013) A peta-scalable CPU-GPU algorithm for global atmospheric simulations. ACM SIGPLAN Notices 48(8):1–12
29. Ashcraft MB, Lemon A, Penry DA, Snell Q (2019) Compiler optimization of accelerator data transfers. Int J Parallel Program 47(1):39–58
30. Wang Y, Guo M, Zhao Y, Jiang J (2021) GPUs-RRTMG_LW: high-efficient and scalable computing for a longwave radiative transfer model on multiple GPUs. J Supercomput 77(5):4698–4717

31. Farhatuaini, L., Pulungan, R.: Parallelization of Uniformization Algorithm with CUDA-Aware MPI. In: 2019 7th International Conference on Information and Communication Technology (ICoICT), pp. 1–6. IEEE (2019)
32. Jia, W., Wang, H., Chen, M., Lu, D., Lin, L., Car, R., Weinan, E., Zhang, L.: Pushing the limit of molecular dynamics with ab initio accuracy to 100 million atoms with machine learning. In: SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–14. IEEE (2020)
33. Váňa F, Düben P, Lang S, Palmer T, Leutbecher M, Salmond D, Carver G (2017) Single precision in weather forecasting models: an evaluation with the IFS. Monthly Weather Rev 145(2):495–502
34. Thornes T, Düben P, Palmer T (2017) On the use of scale-dependent precision in Earth system modelling. Q J R Meteorol Soc 143(703):897–908
35. Klöwer M, Düben P, Palmer T (2020) Number formats, error mitigation, and scope for 16-bit arithmetics in weather and climate modeling analyzed with a shallow water model. J Adv Model Earth Syst 12(10):246



## Authors and Affiliations

**Fei Li[1] · Yuzhu Wang[1] · Zhenzhen Wang[1] · Xiaohui Ji[1] · Jinrong Jiang[2] · Xiaoyong Tang[3] · He Zhang[4]**

Fei Li
lifei2021@email.cugb.edu.cn

Zhenzhen Wang
zhen020227@cugb.edu.cn

Xiaohui Ji
xhji@cugb.edu.cn

Jinrong Jiang
jjr@sccas.cn

Xiaoyong Tang
tangxy@csust.edu.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

[3] School of Computer and Communication Engineering, Changsha University of Science and Technology, Changsha 410004, China

[4] Institute of Atmospheric Physics, Chinese Academy of Sciences, Beijing 100029, China