GPU-Based Parallel Researches on RRTM Module of GRAPES Numerical Prediction System

Fang Zheng[1 2]
[1]School of Computer,Wuhan University,Wuhan,China, 430072
[2]School of Science, Huazhong Agricultural University, Wuhan, China, 430070
Email: zhengfang@mail.hzau.edu.cn

Xianbin Xu [1 3*]
[1]School of Computer,Wuhan University,Wuhan,China, 430072
[3]School of Computer,Wuhan Donghu University, Wuhan, China, 430212
Email: xbxu@whu.edu.cn

Dongdong Xiang [1] Zhuowei Wang[1] Ming Xu[1] Shuibing He[4]
[1]School of Computer,Wuhan University,Wuhan,China, 430072
[4]Department of Computer Science Illinois Institute of Technology Chicago, IL 60616
Email: {whuwzw,dongdongxiang,mrxuming}@gmail.com, hesbingxq@163.com

Abstract—**GRAPES (Global and Regional Assimilation and Prediction System) is a new generation of numerical weather prediction (NWP) system of China. As the system processes amount of data and requires high real-time,so it is always a hot research field of parallel computing.This is the first time that we use GPU (Graphics Processor Unit) general-purpose computing and CUDA technology on RRTM (Rapid Radiative transfer model) long-wave radiation module of GRAPES_Meso model for parallel processing, we rewrited the RRTM module with CUDA Fortran according to the characteristics of the GPU architecture.Enhancing the computational efficiency with optimization strategys such as the code tuning, asynchronous memory transfer,compiler option and etc. The optimization results indicate that a 14.3×speedup is obtained. Experiments are carried out on the multi-GPU platform,and can be easily extended to GPU clusters, the results show that the parallel computing algorithm is correct , stable and efficient.**

Index Terms—**GPU, CUDA, GRAPES system, RRTM, Parallel computing**

I. INTRODUCTION

GRAPES (Global and Regional Assimilation and Prediction Enhanced System) is a new generation of numerical weather prediction (NWP) system,it is developed by China Meteorological Academy. Radiation process is one of the most important physical parameterization schemes in numerical weather prediction models and climate models. In numerical weather and climate simulation, the executing time of LW flux (longwave radiation flux) is more than 1/3 of the total computation time. Therefore, a fast and an accurate long-wave radiation parameterization process is need. RRTM developed by AER retains the highest accuracy relative to line-by-line results for single column calculations,and it is a rapid radiative transfer model which utilizes the correlated-k approach to calculate fluxes and heating rates efficiently and accurately. RRTM is one of three long-wave radiation parameterizing schemes for the physical process of GRAPES .

For a large-scale numerical weather prediction system, in order to run real-time business, parallel computing is essential.And usually the simulation of the radiative transfer in the atmosphere were computed on Ultra-large CPU clusters with hundreds or thousands of nodes[1]. In the computation model of GRAPES, The atmosphere can be mathematically partitioned into a 2-D grid across the earth's surface, with a third dimension consisting of atmospheric layers. Therefore the GRAPES model has good fine-grained data parallelism and very suitable for parallel computing. At the same time , with the development of High-performance computing and parallel computing technology, GPU has been widely used for the general purpose with the release of CUDA(Compute Unified Device Architecture) by NVIDIA for its high performance of floating-point arithmetic operation,large memory bandwidth and powerful parallel processing capabilities. And it has attracted more and more attention of researchers in the field of science and engineering for its low-cost, low power and powerful computing capabilities. GPU will become an important branch of high-performance computing.Various applications have already achieved high performance on GPU,such as the simulation of seismic waves propagating in geophysical exploration [2] and accelerating the process of DBT(dynamic binary translation) on CPU/GPU architecture[3].

This paper presents an approach to port RRTM module of GRAPES to GPU platform. Enhancing the computational efficiency with optimization strategys. Experiments are carried out on the multi-GPU platform,and can be easily extended to GPU clusters.

The rest of the paper is organized as follows.Section 2 is related work.In section 3,introducing the construction

© 2013 ACADEMY PUBLISHER
doi:10.4304/jcp.8.3.550-558

of RRTM module.In section 4 ,we propose the parallel scheme of RRTM module in GRAPES system.Section 5 introducing the parallel strategy and optimization techniques of RRTM .Section 6 is the conclusion and the future work.

II. Related Work

WRF is a new generation mesoscale numerical weather forecast model and data assimilation system which is made by meteorological community of USA, 3DAR and WRF V.2 were released in 2004. Due to its highly modular features makes it very suitable for parallel computingAnd WRF model system has broad application in weather forecasts and in air quality forecasts.

J.Michalakes et al[4,5] in 2008 ported a computationally intensive physics module from the Weather Research and Forecast (WRF) model using GPU with CUDA, they achieved the 5× to 20× speedup in WSM5 with little optimization effort has been put into GPU code.

J.Delgado et al.[6] proposed and described a developed methodology to port WRF originally written in FORTRAN to NVIDIA CUDA. The contribution of their work was to save development effort by specifying a simple iterative approach to port that did not require knowledgements of the application being ported, but they didn't consider the improvement of performance.

G. Ruetsch et al. [7] presented the approach and results of porting the Long-Wave Rapid Radiative Transfer Model (RRTM) component of the WRF code to the GPU using CUDA Fortran. The paper discussed that how the data structures have been modified for the GPU architecture,strategies for optimizing data movement, and determining how to partition the code into different kernels and how these kernels were configured. However, because RRTM relied heavily on lookup tables, performance optimization became very data dependent.

In this paper,we ported the GRAPES code to the GPU using CUDA Fortran. First,to improve the performance of GRAPES in the porting process, we optimized RRTM modules with using some optimizations. We implemented a standalone version of the individual module rather than performing an entire simulation to test a single module and compared our GPU output to the CPU output. Second, a parallel programming model (CUDA MPI) was presented in multi-GPU computing environment when simulating the RRTM. Finally, domain partitioning of GRAPES and asynchronous memory transfer mode were used for multi-GPU computations. we found that the experiment results which obtaining from the GPU code were accuracy and efficiency.

III. Backgroud

A. RRTM of GRAPES

Longwave (LW) radiation plays a crucial role in influencing the weather, climate and climate sensitivity to external radiation.Therefore,accurate long-wave radiation parameterization is very important for weather and climate research in atmospheric models.In the numerical computation of the weather and climate,the calculation time of LW flux (longwave radiation flux) accounted for 1/3 or more of the total time.Therefore, a great need for a rapid and accurate long-wave radiation parameterization.

The long-wave radiation parameterization of the RRTM module for GRAPES system was developed by Mlawer[16] of AER which is a rapid radiative transfer model which calculates fluxes and cooling rates for the longwaves pectral region (10-3000cm^{-1}) for an arbitrary clear atmosphere. The molecular species treated in the model are water vapor, carbon dioxide, ozone, methane, nitrous oxide, and the common halocarbons. And the radiative transfer in RRTM is performed using the correlated-k method: the k distributions are attained directly from the LBLRTM line-by-line model, which connects the absorption coefficients used by RRTM to high-resolution radiance validations done with observations. The correlated-k method is an approximate technique for the accelerated calculation of fluxes and cooling rates for inhomogeneous atmospheres.This method is capable of achieving an accuracy comparable with that of line-by-line models with an extreme reduction in the number of radiative transfer operations performed. The radiative transfer operations for a given homogeneous layer and spectral and are carried out using a small set of absorption coefficients that are representative of the absorption coefficients for all frequencies in the band.

B. Description of RRTM Module

The original serial CPU code implementation of RRTM were obtained from the Kernel Benchmark Page[6] at NCAR. The figure 1 is the pseudo-code for the highest level of module.

```
subroutine radiation_driver (parameter list )
    the part of defining variables
    the part of initializing data
    call radconst(parameter list)
    call cal_cldfra(parameter list)
    lwrad_select:
      select case(config_flags%ra_lw_physics)
                    ! longwave radiation physics process
          case (RRTMSCHEME)
                                 ! adopt RRTM scheme
             call rrtminit (parameter list)
                                 ! call rrtminit function
                times(1) = rsl_internal_microclock()
                                 ! timing function
             call rrtmlwrad (parameter list)
                                 ! call rrtmlwradfunction
                times(2) = rsl_internal_microclock()
                                 ! timing function
          case(other SCHEME)
      end select lwrad_select
    other code（such as the short wave raditation）
end subroutine radiation_driver
```

Figure 1. Pseudo-code for the RRTM module

© 2013 ACADEMY PUBLISHER

In the beginning of the module, the driver of the radiation initializes the data ,and according to the parameters a scheme of longwave radiation physics process is launched,then calling an initialization routine rrtminit( ), next calculating the long-wave radiation transfer process by calling the routine rrtmlwrad( ) and the rrtm( ) subroutine.The subroutine rrtm( ) are composed of these steps:

initrad( ): computes the ozone mixing ratio distribution；

mm5atm ( ): prepares atmospheric profiles；

setcoef ( ): calculates various quantities needed for the radiative transfer algorithm specific to this atmosphere

gasabs ( ):calculates gaseous optical depths

rtrn ( ):calculates the radiative transfer for both clear and cloudy columns

The gaseous optical depths and Planck fractions in *gasabs( ) calls* 16 subroutines taugb*n*( ) computing for 16 different long-wave spectral bands, where n = 1,…,16. The structure of function package for the RRTM module is shown in Fig. 2.

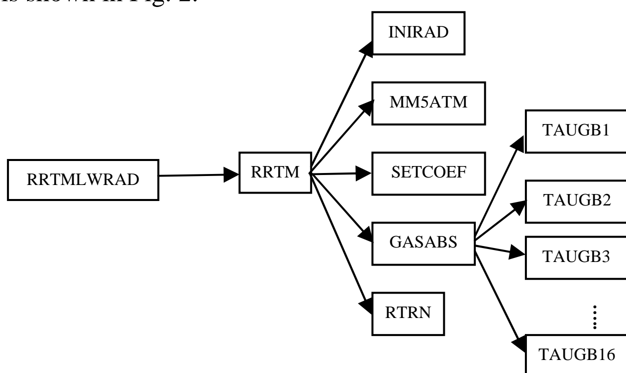

Figure 2 .function package structure of the RRTM module

IV. Accelerating Rrtm With Gpu

A. *Typical Parallel Mode of GRAPES*

The two dominant parallel models are message passing and shared memory,also MPI and OpenMP are used in the of GRAPES model for parallel processing,and GRAPES provides two parallel programming schemes for different computing platforms:

(1) patch parallel program for distributed memory computer system (multi-tasking) (as Fig.7 shows,patch corresponding to subdomain).

(2) tile parallel program for shared memory computer system (multi-thread parallel computing).

In GRAPES computation model, the overhead of thread management is low, and the efficiency of thread synchronization is high. Exchanging and sharing information of the programs which running on different processors could be easily realized using on-chip memory, then GRAPES programs are suitable for fine-grained parallel computing.

In the CUDA programming model, the CPU is viewed as a control processor that is responsible for parameter setup, data initialization, and execution of the serial portions of the program. The GPU is viewed as a co-processor whose job is to accelerate data parallel computations. Threads within the same block to share data using high-speed on-chip shared memory. Threads from different blocks can share data via global memory. CUDA adopts a SIMD data-parallel model in which one instruction is executed multiple times in parallel on different data elements. So we can enhance the GRAPES system performance by using the GPU’s fine-grained parallel feature.

B. *Running mode of GRAPES based on GPU*

WRF model is supported by NCAR and free of charge released for public [8][9].The architecture of GRAPES is similar to the one of WRF system. Taking the advantage of GPU,CUDA and numeric computing technologies, we have completed its GPU parallel codes of GRAPES_Meso. Our researches of accelerating GRAPES meteorological model based on GPU are at the forefront in China.

Multi-GPU computing is now a part of the super-computing.The Message Passing Interface(MPI) is widely used for parallel programming on clusters.MPI can work well on both shared and distributed memory system. Using MPI with one node mapped to each GPU is the most straightforward way to use a multi-GPU.In our implementation, we extend our GRAPES system in multi-GPU environment(one node mapped to 4 GPUs). Each node run four CPU processes, and each CPU process control one GPU,Fig.3 is shown as the structure of multi-GPU .

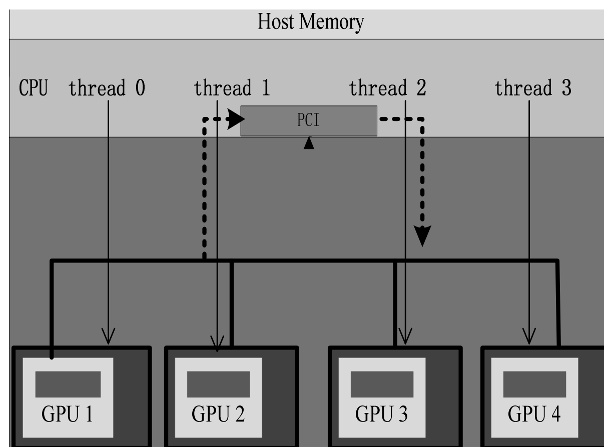

Figure 3 .Communication between multi-GPU

The parallel mode of GRAPES system on the GPU platform is described simply as follows:

step1: Setting the number of GPU which executing the CUDA parallel program of GRAPES system (communicating with MPI), setting the number of block in the GPU, etting the variables and parameters of computing region and parallel decomposition region , etc;

step2: Setting a GPU as a master node which is responsible for inputing initial data for GPU computing,

© 2013 ACADEMY PUBLISHER

and the master node also distribute and collect the results of other GPU computing nodes;
step3: Each GPU computing integral iteration part;
step4: Each GPU calling time integral calculation including the dynamical and physical processes of program. Exchanging data between each GPU;
step5: Each GPU determining whether to output intermediate results,if necessary,the data will be aggregated to master GPU node for output;
step 6: Each GPU judging whether the computation is over, if the computation is over,collecting the results and sending to the master node,ending the program. Otherwise, returns to *Step 4* and continue.

C. Parallel scheme of RRTM module based on GPU

In the version of CPU code of RRTM,the structure of most subroutine is as following :

```
do j=jts,jte
    do i=its,ite
        call RRTM Module;
    end do
end do
```

The above code contains a lot of looping structures, and iterative structures.In the Fortran version code, usually independent subroutines are called in the main function.

As the Fig.2 shows,the subroutine RRTM consists of five subroutines,we separate each subroutine into one kernel except the GASABS,and GASABS is split into 16 kernels[17,18].

For the characteristics of the procedure,there are three parallel strategys in the parallel computing of the RRTM on GPU.

(1) Parallelization between each iteration step .

(2) Parallelization between subroutines.

As the subroutine of the GASABS is encapsulated into different kernels,and the computational data for the 16 kernels is independent,so the subroutines can be executed currently.

(3) Parallelization within subroutines.

According to the characteristic of CUDA, instructions of one GPU kernel is executed by multiple threads in the same thread block,so the threads for the same subroutines is current.

Also, considering the parallel code, GPU architecture and other factors, there are two parallel layers.

(1) Parallelization of instruction-level.

We take some parallel strategys in some parts of the code,such as unrolling the loop,adjustmenting the scheduling policy of instruction.(as transferring more data to be calculated to a thread for hiding memory access latency.)

(2) Parallelization of thread-level.

In this level,every kernel execute their respective parallelized thread.(task partitions for data domain,each thread computes a region,there is little or no inter-regional data exchange.)

D. Mapping data Structure to GPU

CUDA provides an interface to program on the GPU for general purpose applications. In CUDA, the GPU can execute multiple concurrent threads. Threads are executed in SIMD thread blocks. Blocks and threads are indexed by block and thread ids, and the GPU can be viewed as a set of multiprocessors. One or more thread blocks is dispatched to each processor. Blocks are further organized into grids[10].

In the framework of GRAPES model, The atmosphere is divided into horizontal and vertical three-dimensional grids, where x axis representing longitude, y axis representing latitude, and the z axis representing the vertical direction,the decomposition of the region is illustrated in Fig.3. Mapping to the RRTM numeric computing scheme,the atmosphere is divided into 28 layers vertically and each layer is divided into a horizontal 73×60 grids.As Fig.4 shows, the value of k is 28,x is 73 and y is 60.Each grid point corresponds to a set of data, so the calculation is the process of interaction between the data. As the process of k direction is independent, the basic parallelism idea is that a thread deal with every processing task of each column on GPU indexing *i* and *j* two-dimensional directions. NVIDIA GPUs consist of blocks of “multiprocessors” each containing a number of “thread processors”. So each vertical atmospheric column is assigned to a multiprocessor, each atmospheric layer within a column is assigned to a thread processor.

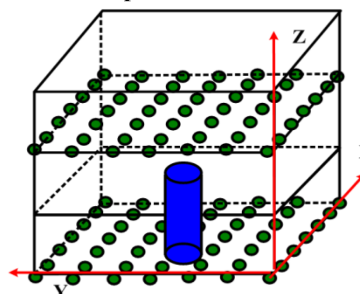

Figure.4 3D physical space

In the language of Fortran and C, plane coordinate can be expressed using two-dimensional array,and 3D coordinate can be expressed using three-dimensional array. Therefore, transformation of the original coordinate in the CPU is needed for only one-dimensional array could be used in CUDA.

In the CUDA FORTRAN code [9], built-in variable *threadIdx.x* identify a unique thread inside a thread block. Likewise, each kernel determines which block to be executed with the built-in variables *blockIdx.x* and *blockIdx.y*. The size of the block is identified by using the built-in variable *blockdim.x*. The build-in variables of CUDA are used to identify a unique thread, which is then used in dividing the work, so that each CUDA kernel thread computes one linear spatial (1 dimension).

idxy variable represents coordinate of the horizontal grid points X, Y, which can be expressed as expression *idxy=(blockIdx%x-1)*blockDim%x+threadIdx%x,* lay

© 2013 ACADEMY PUBLISHER

variable represents coordinate of the vertical dimension Z, which can be expressed as expression *lay = (blockIdx%y-) *blockDim%y+threadIdx%y*. So a point in a three-dimensional grid can be identified by an idx and a lay variable as one-dimensional linear data.

E. GPU-based Implementation of RRTM Module

In the implementation of RRTM module based on GPU, first initializing rrtminit_cuda ( ) subroutine, calling the CPU version of the subroutine rrtminit ( ), transfering the calculating data to the storage of GPU, and then calling rrtmlwrad ( ) routine, starting the computation of 5 basis subroutines in rrtmlwrad ( ).

As large amount of data should be processed in the computation, and for the limitation of the numbers of registers and capacity of memory, the data transmission between GPU and CPU is needed for next computing step. In order to avoid the exchange of data too frequently, resulting low efficiency of parallel programs, loop unrolling, iteration are commonly used.

GPU can execute thousands of concurrent threads simultaneously for its SIMT (Single Instruction, Multiple Thread) characteristics. As GPU's computations rely on the scheduler of CPU, for the limitations of the storage capacity,data transmission between GPU and CPU is inevitable. Data exchange between the GPU and CPU lead to memory latency. Therefore, sufficient number of active threads need to be run for hiding the memory latency and improving computational efficiency.

In order to perform as much concurrent threads as possible, we adopt two parallel strategies. As we mentioned above, atmosphere can be depicted as a three-dimensional grid,according to the Fig. 4. Given input data is a (X, Y, Z) = (73,60,28) three-dimensional mesh data set.In the calculation,we implement two different data parallelisms.The one is coarse-grained parallelism , each thread calculate a separate column data ,there are 73*60=4380 threads in the horizontal dimension. The other one is fine-grained parallelism, horizontal dimension is divided into 28 layers, there are 73*60*28=122640 threads executing currently,each thread calculate a grid point. Thread allocation scheme is illustrated in Fig. 5.

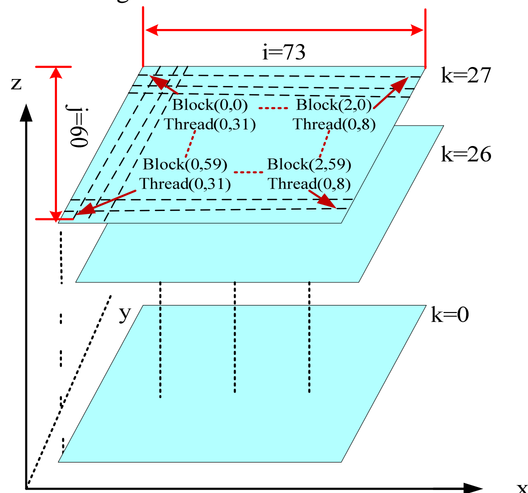

Figure 5. schematic visualization of computational kernel

In addition to the above-mentioned parallel strategies, parallelization between subroutines has also been used.In the implementation of the coarse-grained parallelism,each thread calculate a vertical column data, so some threads block the execution of other threads for they are interdependence of each other's computational data. Considering these factors, We can not just launch one kernel to perform rrtmlwrad_cuda ( ), but should launch multiple different kernels and different executing configurations for caculation of different kernel. We can benefit from the above method.(1) If the computing data of a kernel is independence, threads of the kernel is lanched to calculate a grid point of 3D mesh.(2) If the computational data of a kernel is dependence, threads of the kernel is lanched to calculate a vertical column of 3D mesh. Using of multiple kernels can also reduce pressure of register,and reducing the transfer of data from local memory to the register ,thereby enhancing the efficiency of execution.In addition, in order to reduce the use of registers, the configuration of the thread block size for different kernel functions can be adjusted according to circumstances in the implementation.

V. Experimental Results And Optimization Techniques Of RRTM

A . GPU Device and CUDA Conception

In this section, we introduce our experiment environment, available device resource. The GPU-based RRTM scheme experiments were performed on Intel Xeon 5500 connect to centOS 2.6.18-164.el5 x86_64 version system (36GB of memory). The system is also equipped with 4×NVIDIA Tesla C1060 cards. Table I shows its specifications. The Fortran compiler is the PGI Fortran compiler version 10.4 that has support for CUDA Fortran and the development environment is mpich 2-1.2.1p1. For post-processing and displaying of GRAPES system, you can selectively install netcdf or other post-processing software GrADS[15].

TABLE 1

SPECIFICATIONS OF THE NVIDIA TESLAC1060

Number of thread processors	512
Frequency of thread processors	1.296Ghz
Single Precision Floating Point Capability	933GFlops
Double Precision Floating Point Capability	78 GFlops
Global memory	512bit,800MHz GDDR3
Global memory bandwidth	102 GB/s
Shared memory per streaming multiprocessor (SM)	16K
Number of 32-bit registers per SM	16K
Max power consumption	188W

Testing CUDA programs are more difficult than

© 2013 ACADEMY PUBLISHER

traditional programs, an entire simulation to test the single module. To obtain input data for testing, the module can be modified to print the values of its input variables while performing a full simulation. This output can be directed to a file that can later be used as input data. Our testing method is to compare our GPU output to the CPU output from the GRAPES Fortran version. The procedure is showed in Fig. 6. The CPU and GPU version of the code are separately executed. When the difference between GPU and CPU data values is close to 0, the procedure is over.

In addition to parallelism of the RRTM based-on GPU platform, optimization techniques within the kernel are also carried out such as optimizing data scheduling, using constant memory, improving circulation structure and ect.

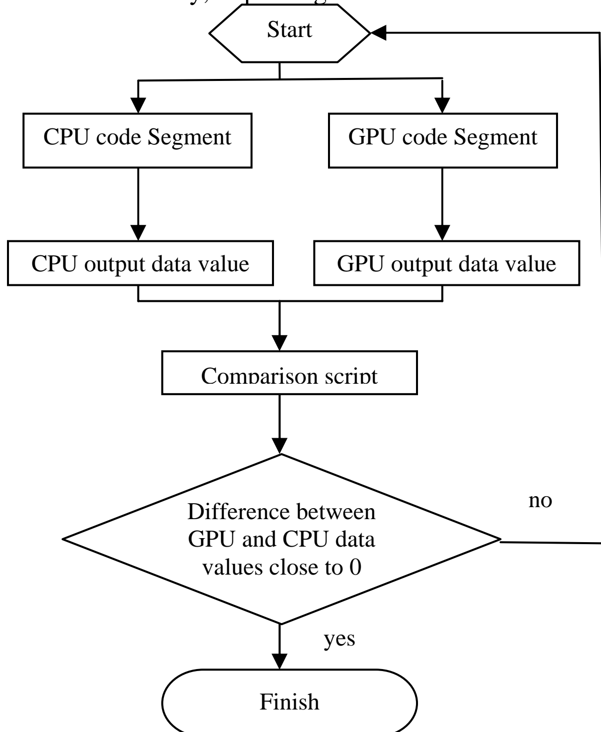

Figure 6 .the flow diagram of test

B. *Asynchronous Memory Transfer*

For a large numerical weather prediction (NWP) system，not only computation speed，but also memory size is important.Multiple GPUs is an available solution for both speed and storage.We extend our implementation to multiple GPUs .

These GPUs cannot share their global memory，so we must manage them by CPU threads and exchange data between neighboring GPUs with CPU memory.It is a higher layer coarse grained parallel like classic distributed memory parallel system that requires domain decomposition.

The computational domain of GRAPES system is divided into four subdomains which mapping to four GPUs for computing. Fig.7 shows the domain partitioning of GRAPES. Each subdomain is surrounded by a boundary region,this is because the correlation of computing data between their neighboring subdomain.

As GPUs cannot control the data transfer between themselves,the data communication process should be realized via CPU.First,the data are copied from CPU to GPU by CUDA API(*cudamemcpy( )*),CPU lanch the GPU kernel executing GPU program,at last copying the results from GPU to CPU when the GPU program is over.The efficiency of this approch is not very high since the GPU will be idle during communication.A better approach is that the computations can be carried out concurrently with communications.

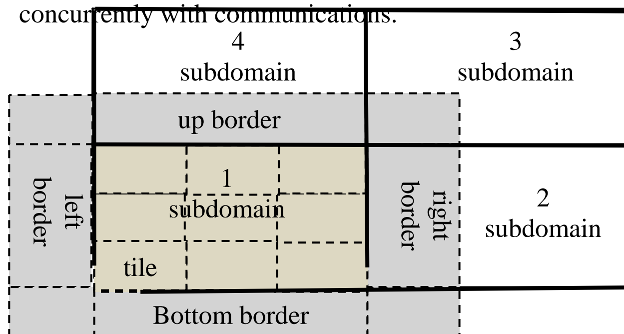

Figure 7. Domain partitioning of GRAPES

CUDA provides an asynchronous transfer mode to make executing a kernel and copying data between CPU and GPU possible simultaneously,this mode is realized by using a *stream*, two streams in one GPU are created,and be managed as follows:

(1) Copying left border and subdomain from host to Device(stream1);
(2) Launching GPU kernel for computing(stream1);
(3) Copying right border from host to Device(stream 2);
(4) Copying left border from Device to host for other GPU's computation (stream 1);
(5) Launching GPU kernel for computing(stream2);
(6) Copying right border from Device to host (stream 2);

The execution time for memory transfers and GPU kernel execution is illustrated in Fig.8. GPU execution and memory transfers executed simultaneously,the communicational time is hidden.

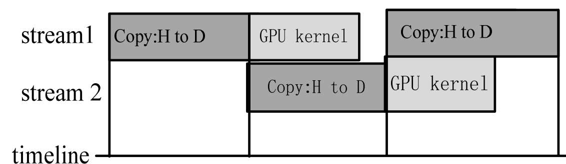

Figure 8.Asynchronous memory transfer mode

Table 2 shows the performances of the computations by asynchronous memory transfer and synchronous memory transfer modes .It is obvious that using asynchronous memory, the performance increases,the speedup is 1.21.

TABLE 2.
PERFORMANCES OF THE ASYNCHRONOUS MEMORY TRANSFER

	CPU time(us)	GPU+I/O time(us)
Asyn-transfer	690524	50256

© 2013 ACADEMY PUBLISHER

syn-transfer	690524	60809
speedup	1	1.21

C. Dimensions of the Thread Block and Compiler Optimization

Maximizing utilization of available computing resources should be consider to set the thread block dimensions when dividing the thread block. Therefore, the number of thread blocks should be at least the same number of SM in the GPU card. In order to avoid the thread synchronization and idle SM while effectively hidden pipeline delay when accessing memory device .The number of thread blocks in the device should be at least more than twice the number of processors.

The Tesla C1060 as an example, should contain at least 64 thread blocks, and the number of blocks should be set to a multiple of 64. Shared memory which is assigned to each thread block should be at least the half of the total shared memory available for each SM.If the number of thread blocks is sufficient, the number of thread in each thread block should be multiple of warp (32 threads per a warp) to avoid of a waste of computing resources by lack filling the warp[13].In the parallel algorithm of RRTM module, we tested the module in different size of thread blocks .the results are illustrated in Table 3 and Fig. 9.

In the testing process, block size was set to 128, 256 and 512, Table 3 is the test results of the comparison. In the test, the CPU code of of RRTM module was compiled using both -O3 and -fast optimization option.- O3 option increased the cost of storage and extended compiling time, but it reduced executing time; and-fast option is to vector the input data, it makes the code run faster, but may bring some loss of precision.

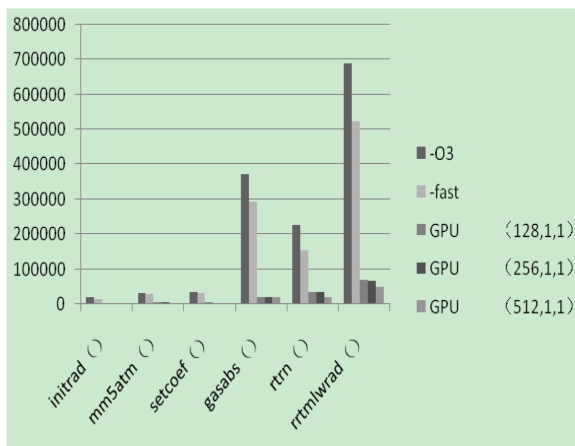

Figure 9. Performance comparison of the routine operation in the RRTM module .

Table 3 and Fig. 9 shows executing performance comparison of each subroutine in the RRTM modules .we can see intuitively that the execution time of each subroutine and the proportion of the total execution time with different block thread.

Execution time of rtrn ( ) and gasabs ( ) routine is about 86% of the total execution time.However,in the GPU version , execution time of these two subroutines is about 80% of the total time, slightly less than the CPU's executing time. The main reason is that part of the time was consumed for data transmission from the CPU-to-GPU equipment, Thus the proportion of time used to calculate decline relatively.

Meanwhile, in two different compiler optimization options of CPU code, gasabs ( ) are the most time-consuming part, accounting for 50% of the total time, while rtrn ( ) is about 30%.However, the GPU version is not the case, the most time-consuming subroutine is rtrn( ) (accounting for 52% of the total time) rather than gasabs( )(accounting for 28% of the total time) after GPU acceleration of RRTM module. The cause of rtrn ( ) consuming more time in the GPU is GPU's low utilization.As the compuation depends on the physical data in the 3D grid , executing thread in rtrn ( ) kernel only cover a limited horizontal grid. Number of the executing threads is far less than a total of 4380 and half of the maximum occupancy number of concurrent threads in Tesla C1060 - about 15,000.In contrast, taugbn ( ) kernel in gasabs ( ) subroutine, there is no data dependence on the vertical coordinate, so it can have enough threads to cover the entire computing grid.

According to Table 3, we can obtain that the subroutine execution time under the thread block division of (128,1,1) and (256,1,1) are not very different, except initrad ( ) subroutine. The execution time of other routines in the thread block dimensions for the (512,1,1) are significantly reduced than the other two dimensions,these indicate that the design of thread dimension has a certain influence on parallel performance of the algorithm.

TABLE 3.

THE SUBROUTINE IN DIFFERENT SIZE OF THREAD BLOCKS

subroutine	running time（us）				
	-O3 option	-fast option	GPU 128	GPU 256	GPU 512
initrad（）	20716	15721	3052	3021	2995
mm5atm（）	33426	29203	5254	4382	4169
setcoef（）	35628	32156	5671	4352	3256
gasabs（）	372882	292961	19385	19921	19005
rtrn（）	227872	153960	35623	35216	20831
rrtmlwrad（）	690524	524001	68985	66892	50256

Thread block size is also different in different Thread dimensions. so the GPU memory access scheduling time is different. If thread dimension is too small, thread

© 2013 ACADEMY PUBLISHER

block unit data is transferred to GPU memory more frequently(such as lobal memory, constant memory, texture memory).If the data offset of data accessing in thread block is large, memory access latency will be not been hidden, thus affecting the overall execution time.

Therefore, increasing the thread block size (the value of the thread block dimensions) can hide some memory access latency in some degree and increase performance of the parallel algorithm. But, if the definition of thread block size is too large, for the all threads within the same block are shared with a constant memory, texture memory and shared memory, When the applying storage space of these threads is greater than the hardware configuration , it may cause kernel failed to start.Through the above analysis, algorithm performance can be improved effectively when the thread block dimension of the RRTM module is defined as (512,1,1) ,and ultimately RRTM module reached 13.7x speedup.

D. Constant Memory

The common variables used in the module are stated as followings:

integer, constant :: kts, kte, ktep1, nlayers

integer, constant :: nx, ny, nxy, nxyPad

! Variables used to input associated dimension data values in the Module， identified with the keyword constant are stored in constant memory.

real, constant :: preflog_c(59),tref_c(59)

! Constants used in setcoef（）subroutine

After compiling in the CUDA Fortran these variables will be stored in the GPU constant memory.The constant memory is read-only and has cache, If the data is buffered in cache, its access latency is only one clock cycle, however if the memory without cache, accessing efficiency of memory is the same to global memory, about 400 to 600 clock cycles.Therefore, the read-only variables are stored in the GPU's constant memory, constant cache can effectively reduce the access time of global memory, so as to enhance the computing speed.With setting the size of thread blocks(512,1,1) and constant memory optimization,the results are shown in Table 4 as following.

TABLE 4

OPTIMIZATION RESULTS AFTER USING CONSTANT MEMORY

running time（us）	
without constant memory	with constant
50256	48375

By comparing with previous execution time, reducing (50256-48375) / 50256 = 3.7% using constant memory, and ultimately to a comprehensive acceleration ratio 690524/48375 = 14.3.

VI. Conclusion And Future Work

Based on the study on the GRAPES global / regional assimilation prediction system principles, structures, parallel programs, We propose a basic parallel ideas for RRTM module in GPU platform. With performance analysis of the program, based on the characteristics of the architecture for the GPU, from code optimization, memory optimization, compiler options, etc. to optimize the performance of the program, and achieved 14X speedup.

As RRTM module depends on the look-up table, the improvement of program performance is also correlate with the input data, Especially with storage of variable. (stored in the register or constant memory, etc.) The global memory and constant memory are used in our Optimization, but this is not necessarily the best option for a particular input data set. As to the lookup table, coalesced memory access should be considered. If using texture memory to store the lookup table is more appropriate than using constant memory with limited size. As CUDA Fortran does not support texture memory, we do not adopt this method. Experiments are carried out on the GPU platform, the results show that the parallel computing algorithm is correct, stable and efficient for operational implementation of GRAPES in near future.

Acknowledgment

We would like to thank the support of Fundamental Research Funds for the Central Universities (Grant No.3101012), and the Key Laboratory of High Confidence Software Technologies Program (Grant No.HCST201104).

References

[1] J.Michalakes, J. Hacker, R.Loft, M. O. McCracken, A.Snavely, N.Wright, T. Spelce, B. Gorda, and B. Walkup. Wrf nature run. In proceedings of the 2007 ACM/IEEE conference on Super-computing,pages 1-6, 2007.

[2] Zhangang Wang, Suping Peng, Tao Liu. GPU Accelerated 2-D Staggered-grid Finite Difference Seismic Modelling [J],Joural of Software, Vol 6, No 8 (2011) 1554-1561.

[3] Erzhou Zhu, Haibing Guan,etc. A Translation Framework for Executing the Sequential Binary Code on CPU/GPU Based Architectures[J], JOURNAL OF SOFTWARE, VOL. 6, NO 12,(2011) 2331-2341.

[4] Michalakes, J., J. Dudhia, D. Gill, T.Henderson, J. Klemp,W.Skamarock, and W. Wang, 2005: The Weather Research and Forecast Model: Software architecture and performance. 11th Workshop on High Performance Computing in Meteorology, World Scientific, 156-168.

[5] V. Simek, R. Dvorak, F. Zboril, and J. Kunovsky, "Towards accelerated computation of atmospheric equations using CUDA,"in Proceeding of 11th International Conference on Computer Modelling and Simulation, UKSIM '09, Cambridge, 2009, pp. 449-45.

[6] B.Huang, J Mielikainen, H Oh,etc. Development of a GPU-based high-performance radiative transfer model for the Infrared Atmospheric Sounding Interferometer (IASI)[J], Journal of Computational Physics,2010, 230 (2011) 2207-2221.

[7] J.Michalakes and M. Vachharajani. GPU acceleration of numerical weather prediction. In Workshop on Large Scale Parallel Processing 2008 (IPDPS2008), Miami, FL, April 18, 2008.

© 2013 ACADEMY PUBLISHER

[8] J. Delgado, J. Gazolla, E. Clua, and S. Masoud Sadjadi, An Incremental Approach to Porting Complex Scientific Allications to GPU/CUDA. EScience 2010.

[9] G. Ruetsch, E. Phillips, and M. Fatica, GPU Acceleration of the Long-Wave Rapid Radiative Transfer Model in WRF using CUDA FORTRAN, Many-Core and Reconfigurable Supercomputing Conference 2010 (MRSC 2010), CASPUR, Rome, March 2010.

[10] GPU Acceleration of NWP: Benchmark Kernels Web Page: http://www.mmm.ucar.edu/wrf/WG2/GPU/

[11] Roe, K., Stevens, D.: 'Maximizing Multi-core Performance of the Weather Research and Forecast Model over the Hawaiian Islands',Proceedings of the Advanced Maui Optical and Space Surveillance Technologies Conference, 2010. Ed.: S. Ryan, The Maui EconomicDevelopment Board., p.E66.

[12] Baghsorkhi, S.S., Delahaye, M., Patel, S.J., Gropp, W.D., and Hwu,W.-m.W.: 'An adaptive performance modeling tool for GPU architectures', SIGPLAN Not., 45, (5), pp. 105-114.

[13] Ryoo, S., Rodrigues, C.I., Baghsorkhi, S.S., Stone, S.S., Kirk, D.B.,and Hwu, W-m.W.: 'Optimization principles and application performance evaluation of a multithreaded GPU using CUDA',Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming, 2008, pp. 73-82.

[14] Hong, S., and Kim, H.: 'An integrated GPU power and performance model'. Proc. Proceedings of the 37th annual international symposium on Computer architecture, Saint-Malo, France pp. Pages280-289.

[15] GrADS ocumentation[EB/OL]. http://grads.iges.org/grads/gadoc/.

[16] Mlawer, E.J., Taubman, S.J., Brown, P.D., Iacono, M.J., Clough, S.A.: 'Radiative transfer for inhomogeneous atmospheres: RRTM, a validated correlated-k model for the longwave. Journal of Geophysical Research, 1997,102 (D14), 16663–16682.

[17] F.sh. Lu, J.Song, X. Cao, X.q. Zhu: 'GPU Computing for Longwave Radiation Physics: A RRTM_LW Scheme Case Study ', Proc. Proceedings of the 9th IEEE International Symposium on Parallel and Distributed rocessing with Applications Workshops (ISPAW), 2011,Busan,pages 71-76.

[18] F.sh. Lu, J.Song, X. Cao, X.q. Zhu: 'CPU/GPU computing for long-wave radiation physics on large GPU clusters', volume41, 2012, pp47-51.

Fang Zheng is a Ph.D. candidate of the School of Computer,Wuhan University, Wuhan, China.She received a B.A. degree in 2002 and a M.S. degree in 2005 from Central China Normal University,Wuhan,China . She is especially interested in high performance computing, and distributed system and etc.

Xianbin Xu graduated from the department of system architecture in Huazhong University of science and technology and worked for teaching in Huazhong University of science and technology from 1977 to 1985.He got Ph.D. of school of computer in Wuhan University. He is now president of institute of computer in Wuhan University. His research interests focus on network storage,data grid and distributed system.

Dongdong Xiang receives his B.A and M.S degree from the School of Computer,Wuhan University, Wuhan,China. His research interests focus on high performance computing,data grid and distributed system and etc.

Zhuowei Wang is a Ph.D. candidate of the School of Computer,Wuhan University, Wuhan, China.She received a B.A. degree in 2007 from China University of Geosciences,China and a M.S. degree in 2009 from Wuhan University.She is especially interested in high performance computing, data grid,distributed system and etc.

Ming Xu received the B.Sc. degree from the Hubei University of Technology in 2001 and M.Sc.degree in the Department of Computer from National University of Defense Technology in 2007.He is currently a Ph.D.student in the Wuhan University. His research areas include high performance computing.

Shuibing He is a postdoctoral researcher at the SCS laboratory of Illinois Institute of Technology. He received his Ph.D. in Computer Science in Dec. 2009 from the Huazhong University of Science and Technology. His research areas include parallel computer architecture, distributed storage and file systems, embedded system. He is currently working on I/O performance evaluation and optimization.

© 2013 ACADEMY PUBLISHER