基于GPU的GRAPES数值预报系统中RRTM模块的并行化研究

郑 芳[1,2] 许先斌[1,3] 向冬冬[1] 王卓薇[1] 徐 鸣[1]

(武汉大学计算机学院 武汉 430072)[1] (华中农业大学理学院 武汉 430070)[2]

(武汉东湖学院计算机科学学院 武汉 430212)[3]

摘 要 GRAPES(Global and Regional Assimilation and Prediction System)是由中国气象科学研究院自主研究开发的中国新一代数值天气预报系统,由于其处理的数据量非常庞大以及对实时性的要求较高,因此一直是并行计算领域研究的热点。首次运用GPU(图形处理器)通用计算及CUDA技术对GRAPES_Meso模式中物理过程的RRTM(快速辐射传输模式)长波辐射模块进行并行化处理。在性能分析的基础上,针对GPU体系结构的特点,从代码优化、存储器优化、编译选项等方面对程序性能进行优化,并取得了14X倍的加速比。经过测试表明,长波辐射RRTM模块在GPU上并行计算过程正确、稳定而且有效,并为GRAPES系统未来在GPU平台上的并行化发展奠定了一定的基础。

关键词 GPU,CUDA,GRAPES系统,RRTM,并行计算

中图法分类号 TP399 **文献标识码** A

GPU-based Parallel Rearches on RRTM Module of GRAPES Numerical Prediction System

ZHENG Fang[1,2] XU Xian-bin[1,3] XIANG Dong-dong[1] WANG Zhuo-wei[1] XU Ming[1]

(School of Computer, Wuhan University, Wuhan 430072, China)[1]

(School of Science, Huazhong Agricultural University, Wuhan 430070, China)[2]

(School of Computer Science, Wuhan Donghu University, Wuhan 430212, China)[3]

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, 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. Based on the analysis of computing performance, and according to the characteristics of the GPU architecture, the RRTM module parallel computational efficiency was optimized from the aspect of code tuning, memory, compiler options and etc. The optimization results indicate that the performance obtains a speedup of 14.3×. Experiments were 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.

Keywords GPU, CUDA, GRAPES system, RRTM, Parallel computing

1 引言

GRAPES(Global/Regional Assimilation and Prediction System是中国气象局组织本系统有关科研与业务力量,经过多年的科技攻关,自主研发的新一代全球/区域多尺度统一的同化与数值预报系统[1]。该系统的核心部分包括模式动力框架、模式物理过程、资料同化系统(即3D-Var三维变分资料同化系统)和模式标准初始化系统。对于一个大型、先进的数值预报系统而言,为了满足实时业务的运行,并行计算已成为其必需的结构特征[2]。

随着新的并行计算技术的出现,特别是近年来能耗/性价比较高的GPU通用计算和CUDA技术的出现,GPU可以在科学计算领域和工程应用领域实现2～3个数量级的加速。GPU巨大的计算能力已经吸引了越来越多的科学家和工程师将其作为一个高效益低成本的高性能计算平台。GPU在大气领域的应用也取得了成效,如GPU上平流扩散方程的实现获得约10倍的加速比[3];美国国家中心空间研究院(CNES)与欧洲开发气象卫星组织(EUMETSAT)研究的红外大气探测干涉仪获得364倍加速比[4];美国Colorado大学国家大气研究中心开展的WRF模式GPU并行化的研究[5],取得了单个WSM5模块获得9.4倍加速比,目前,该中心已经初步完成了WRF模式wsm5、Scalar Advection、WRF-Chem等模块或功能的GPU并行化工作,还有RRTM长波辐射物理过程、SWRAD短波辐射物理过程、wsm6等模块还有待进一步研究和开发。

辐射处理过程是大气物理过程中一个非常重要的部分,

本文受中央高校基本科研业务费专项资金(3101012),高可信软件技术教育部重点实验室开放课题(HCST201104)资助。

郑 芳(1980－),女,硕士生,讲师,主要研究方向为并行计算、高性能计算,E-mail:zf22_00@163.com。

同时 RRTM 模块在 GRAPES 系统辐射物理过程方案中的应用耗时较多，适合 GPU 的并行运算。本文以 AGAC 实验室针对 GRAPES 系统 Meso 模式的并行化工作为参考和基础，通过认真研究 GRAPES_Meso 模式的特点和工作模式，完成了对 Grapes 模式中模式物理过程中的辐射传输过程 RRTM 模块的 GPU 并行化工作。试验结果表明：GRAPES 模式 RRTM 模块的 GPU 并行加速的结果准确有效。

2 相关工作

长波(LW)辐射在影响天气、气候和气候对外部辐射强迫的敏感性上起到至关紧要的作用[6]。因此，在用于天气、气候研究的大气模式中，精确的长波辐射参数化是很重要的。在天气和气候的数值模拟中，LW 通量(长波辐射通量)的计算耗费总计算时间的 1/3，或更多[7]。因此，非常需要快速和精确的长波辐射参数化。由美国大气环境研究中心的 Mlawer等研制的 RRTM(A rapid radiative transfer model，快速辐射传输模型)方案就是一个在通量和冷却率计算精度上与逐线模式一致的快速辐射传输模式，它已被广泛地应用于各种大气模式(包括大气环流模式，中尺度模式和单柱模式等)的大气辐射传输研究中。

在 GRAPES 系统中也采用了 RRTM 长波辐射方案，即将整个大气层划分成水平方向和垂直方向的三维网格，水平方向为横跨地球表面的二维网格，垂直方向则表示大气的层数，如图 1 所示。数值天气预报模式是一种非线性化离散计算模式，其计算量巨大。我国对 GRAPE 数值天气预报系统的计算主要是在分布式共享主存(Distributed Shared Memory，DSM)结构的计算机系统上运行，也可以在 Cluster 集群环境下工作[2]。而对 GRAPES 在 GPU 环境下的并行化改造，本文是国内领先的。由于其软件框架与 WRF 的架构相似，因此其加速方案也采用以 CPU 为主、GPU 为辅的方式进行。

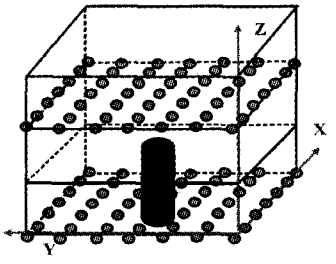

图 1 GRAPES 模式中并行计算模型

2.1 GRAPES 系统在 CPU 环境下的并行处理

目前对 GRAPES 的计算主要有这两种方式：一种是基于分布式存储的消息传递 MPI 并行计算，一种是基于共享存储的 Open MP 并行计算。而对于 GRAPES 模式在这两种计算方式下分别有如下并行化方案[2]：一是针对分布式内存计算机系统设计基于消息传递的区域分块(patch)并行方案，这也就是通常所说的多任务并行计算；一是针对共享内存计算机系统完成的瓦片(tile)并行计算，也就是多线程并行计算。

2.2 GRAPES 系统在 GPU 平台下的并行处理

GPU 以其强大的并行处理能力和极高的存储器带宽向通用计算领域扩展，且它的低成本、低功耗和高性能的特点在科学和工程计算领域受到了越来越多的关注。GPU 将成为高性能计算领域极具性价比的一个分支。

GRAPES 系统在 GPU 平台下的并行算法描述如下：

(1)设置 GRAPES 并行化的 CUDA 程序的 GPU 个数(它们使用 MPI 方式进行消息通信)、GPU 中线程块的个数、预报区域的大小等控制变量，设置并行区域的划分，并由此分配变量空间；

(2)设置一个 GPU 为主节点(用于分配和收集计算结果)，并将初始数据分配至各个 GPU 中；

(3)各个 GPU 计算那些不随时间变化的积分循环系数部分；

(4)各个 GPU 调用时间积分计算，包括模式动力框架和物理过程方案。在此过程中，各个 GPU 之间可能要进行必要的数据交换；

(5)各个 GPU 判断是否需要输出中间结果，如果需要，则将数据汇总至主节点 GPU 上输出；

(6)各个 GPU 判断积分是否已经结束，如果已经到达积分上限则将各自的结果收集至主节点并输出，并结束程序。否则返回第(4)步继续。

本文针对 GRAPE 模型中的 RRTM 长波辐射模块进行 GPU 的并行化改造和优化。

2.3 RRTM 模块的构成

RRTM 模块首先调用初始化例程 rrtminit()读取数据文件中的输入数据；然后调用 rrtmlwrad()例程计算长波辐射传输的过程。在 rrtmlwrad()例程中，展开一个水平维度(平面内)的双层嵌套循环并通过输入的三维数组确定垂直列的比例，然后再调用辐射模型子例程——rrtm()；

rrtm()子例程一次计算一个垂直列的辐射传输值。它主要包含以下几个基本步骤：

initrad()：计算臭氧混合比例分布；

mm5atm()：准备大气剖面；

setcoef()：计算这种大气剖面下辐射传输算法所需的各种量；

gasabs()：计算气态光学厚度；

rtrn()：计算晴空和云层的辐射传输值。

其中，gasabs()中的 16 个子例程 taugbn()(其中 $n=1,\cdots,16$)计算了全部 16 个长波波域谱带的气态光学厚度和普朗克(Planck)函数。RRTM 模块的函数包结构如图 2 所示。

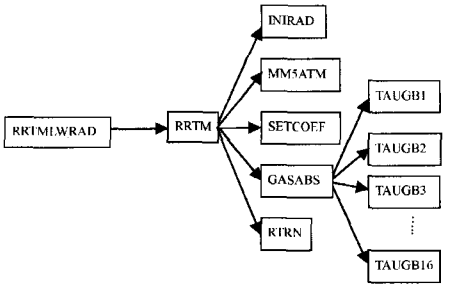

图 2 RRTM 模块的函数包结构

2.4 RRTM 模块并行化方案

由 2.3 节分析可知，rrtmlward( )例程的外层操作是一个水平坐标的双层嵌套循环，这个双层嵌套循环代表了水平方向的一个二维网格平面数据的计算，垂直方向则表示大气的层数。在辐射的计算过程中只依赖于同一个垂直方向上的数据，垂直方向每个气柱之间的计算是独立的。rrtm()子例程一次计算一个垂直气柱方向的辐射值。水平方向的每个格点

对应一组数据，水平方向上每个格点中的计算数据会相互作用，而垂直方向的计算是独立的，所以并行化的方案是让GPU沿着水平二维方向对垂直方向的数据进行处理。在GPU的体系结构中，每个GPU由多个SMP(流多处理器)组成，每个SMP又包含多个SP(流处理器)，因此在垂直方向上每个气柱被分配给一个流处理器，水平方向上每层气柱处理的数据正好分配给一个流处理器，这就实现了RRTM的细粒度并行。

在对RRTM模块经过CPU计算的测试中发现函数gasabs( )的计算时间占56%，因此整个并行化的过程中主要是对gasabs( )进行GPU代码的改造。在gasabs()例程中包含16个子例程taugbn()($n=1,\cdots,16$)，在进行CUDA代码设计时，将这16个子例程划分成16个核函数。

3 基于GPU的RRTM模块的CUDA算法实现

使用CUDA Fortran实现了现包含rrtminit()和rrtmlwrad()的例程。初始化子例程rrtminit_cuda()，调用了CPU版本的子例程rrtminit()，并且把计算数据传输到GPU设备的常数存储空间，调用rrtmlwrad()例程，开始rrtmlwrad()中5个子例程的计算，在计算过程中由于计算数据量较大和寄存器数量及存储器容量的限制，需要在GPU和CPU设备之间进行数据的传输以进行下一步运算，最后将结果传递给CPU进行归约计算并与CPU计算结果进行测试误差比较，直到误差接近0为止。为了避免CPU与GPU之间数据交换过于频繁，降低程序的并行执行效率，除了进行循环控制，迭代收敛控制以及MPI归约等任务时才将必要的数据传入内存并交由CPU计算外，其他的计算任务都交由GPU完成，避免了在CPU与GPU间频繁的数据传输。

3.1 数据层次结构及坐标变换

如第2节所述，GRAPES在运算时将整个大气看成一个三维网格，RRTM benchmark也是一个三维网格数据。在Fortran或者C语言中，可以使用二维数组或三维数组来表示平面坐标点或者立体坐标点，而在CUDA中只能使用一维的数组进行计算，因此需要对原CPU中的坐标进行变换。GRAPES中输入数据是按照左手坐标系(X,Z,Y)的方式定义三维网格，在程序中分别用循环变量i,k,j与之对应，索引变量i和j代表水平方向的经度和纬度的索引X和Y，变量k代表垂直方向的索引与三维网格中的垂直坐标Z对应。

考虑到GPU相对于CPU有其独立的内存空间——显存，并且待计算的数据必须从CPU传输至GPU，因此采用在不改变CPU数据结构的前提下，变换输入数据的坐标系。

坐标系(X,Z,Y)中，第一维二维坐标与第三维坐标方向是相互独立的。为了促进数据加载/保存的合并访问，将多维数组中独立的坐标方向作为第一维的索引，将坐标顺序调整为(X,Y,Z)。在CUDA中，针对多维数组中变化最快的那一维进行合并数据存取可以使填充后的合并访问减少空间和计算资源的浪费。由于垂直面Z相对于X,Y这两维是独立的，于是，将这两维合并为覆盖整个水平面的一维，即原来两维的数量分别从its到ite，jts到te，合并后的作为一个一维的数量nxy=(ite−its+1) * (jte−jts+1)，合并后更便于数据的传输，下标变换更直接。

CUDA Fortran版本代码中，每一个线程计算的三维空间由idxy代表水平面网格点X、Y的坐标，具体的坐标变换为idxy=(blockIdx%x−1) * blockDim%x+threadIdx%x，lay代表垂直维的坐标Z，lay=(blockIdx%y−1) * blockDim%y+threadIdx%y。idxy和lay就固定了三维坐标中的一个网格点。

3.2 核函数的执行配置

GPU基于SIMT(Single Instruction，Multiple Thread)特点，可以同时执行成千上万个并发线程。由于GPU计算依赖于CPU的调度，需要在GPU和CPU之间进行数据传输，造成一定的存储时延，因此GPU适合于有足够的活动线程以隐藏存储延迟的计算的情况。为了能让尽可能多的进程并发执行，在并行度的划分上采用了两种方案。

在GRAPES的RRTM模块的计算中，给定输入的数据是一个(X,Y,Z)=(73，60，28)的三维网格(mesh)(根据RRTM benchmark设定)，在计算当中将mesh采用几种粒度并行，一种是列并行，每个线程计算一个独立的列数据，则水平维度上共有73 * 60=4380个线程。另一种是细粒度并行，在水平维度划分的基础上再按层划分并行度，共有73 * 60 * 28=122640个线程计算三维网格中的一个点。

函数计算存在数据迭代和依赖，存在数据依赖部分计算阻塞了一些线程，这些线程对应着三维空间中的一个点而不仅仅是垂直列的一维坐标。考虑这些因素，采用启动多个核函数，包含多个不同的执行配置。

在数据独立性允许的前提下，可以启动一个核函数，其中每个线程处理三维空间中的一单个点；而当数据存在依赖时，启动的核函数中每个线程计算一列数据。使用多个核函数还可以减少寄存器使用压力，从而减少从寄存器向局部存储器转移变量的可能。不同的核函数在执行配置中的线程块的大小可根据情况作出调整。

4 算法的进一步优化

在对RRTM模块进行CUDA平台下的并行化设计后，其加速后的执行时间并没有提高很多，其加速比如表1所列。因此除了将代码简单移植到CUDA平台上，还需要针对CPU/GPU异构平台的特点进行优化。

表1 GPU加速结果和CPU对比

Module	CPU(us)	GPU(us)
RRTM	805	609

4.1 线程分支的两个问题

线程分支是RRTM GPU加速中影响性能的比较关键的因素。控制流指令(if、switch、do 、for、while)可能引起一个warp内的线程跳转到不同的分支，这将严重影响指令吞吐量。一旦发生分支，那么不同的执行路径就必须被串行地执行，导致这个warp中指令总数增多。当所有分支的指令都执行结束之后，这些线程才会重回到同一条执行路径上。只有在控制流与线程ID(treadIdx/warpsize)有关时，各warp在block中的分支才是确定的，应该修改控制条件，尽量避免在warp内发生分支。

如图3所示，它是从taugb10()的CUDA Fortran版本代码中提取的部分代码，其中每一个线程计算三维网格空间中的由idxy和lay决定的一单个点，idxy=(blockIdx%x+1) * blockDim%x+threadIdx%x和lay=(blockIdx%y−1) *

blockDim%y+threadIdx%y 是分别来自水平维和垂直维的坐标。若线程束中的不同线程通过 if 语句陷入不同的分支，每一个线程测试一个分支条件，那些不满足分支条件的线程则被切换出去，但是这样必然招致更多的计算开销。因此很有必要使一个线程束中的所有线程具有一致的分支判断，以提高执行的效率。

```
    if(idxy <= nxy) then
    if(lay <= laytrop(idxy)) then
      ind0 =((jp(idxy,lay)-1) * 5+(jt(idxy,lay)-1)) * nspa_d
(10) + 1
      ind1 =(jp(idxy,lay) * 5+(jt1(idxy,lay)-1)) * nspa_d(10) + 1
        do ig = 1,ng10
          taug(idxy,ngs9+ig,lay) = colh2o(idxy,lay) * &
          (fac00(idxy,lay) * absa10_d(ind0,ig) + &
            fac10(idxy,lay) * absa10_d(ind0+1,ig) + &
            fac01(idxy,lay) * absa10_d(ind1,ig) + &
            fac11(idxy,lay) * absa10_d(ind1+1,ig))
            pfrac(idxy,ngs9+ig,lay) = fracrefac10_d(ig)
        enddo
      else if(lay <= nlayers) then
        ind0 =((jp(idxy,lay)-13) * 5+(jt(idxy,lay)-1)) *
              nspb_d(10) + 1
        ind1 =((jp(idxy,lay)-12) * 5+(jt1(idxy,lay)-1)) *
              nspb_d(10)+ 1
          do ig = 1,ng10
            taug(idxy,ngs9+ig,lay) = colh2o(idxy,lay) * &
            (fac00(idxy,lay) * absb10_d(ind0,ig) + &
            fac10(idxy,lay) * absb10_d(ind0+1,ig) + &
            fac01(idxy,lay) * absb10_d(ind1,ig) + &
            fac11(idxy,lay) * absb10_d(ind1+1,ig))
            pfrac(idxy,ngs9+ig,lay) = fracrefbc10_d(ig)
          enddo
    endif
endif
```

图 3　taugb10()例程的部分 CUDA Fortran 代码

在优化的过程中我们利用问题域固有的物理分层来处理分支问题。由于数组已经被填充为一个水平面，使得垂直维度上的线程数量是 warp 的整数倍，因此一个 warp 中的所有线程有相同的 lay 索引。由于垂直方向的物理分层，因此 laytrop(idxy)就和 warp 中的线程 ID 保持一致。可以将控制条件改写成 lay<=laytrop(idxy)，这样控制流就只与线程的 ID 相关，达到了避免线程分支的目的。

4.2　线程块的维度进行划分与编译优化

在对线程做维度进行划分时，应考虑选择最大化可用计算资源的利用率的线程块的维度划分。因此，线程块的数量至少应与设备中的多处理器数量相同。为了避免在线程同步或访问设备存储器过程中流处理器出现空闲，同时有效地隐藏流水线延迟，线程块的数量至少应为设备中多处理器数量的 2 倍，以 Tesla C1060 为例，则至少应包含 64 个线程块(block)，且应设置为 64 的倍数。

而且每个线程块所分配的共享存储器也至少应为每个多处理器可用的共享存储器总量的一半。在线程块的数量足够多后，每个线程块的线程数量应为 warp(每个 32 个线程一个 warp)块大小的倍数，以避免因 warp 块填充不足而造成计算资源浪费，最好使线程数量是 64 的倍数。

因此，在 RRTM 模块的并行化算法中，对线程块的大小进行一些测试，其运行环境采用了 centOS(内核版本 2.6.18-164.el5 x86_64)操作系统，使用 Intel Xeon5500 系列 CPU，36GB 内存；GPU 为 4 块 NIVIDA Tesla C1060。由于整个 GRAPES 系统主要以 Fortran90 和 C 语言进行编写，因此系统需要安装相应的编译器和预编译器。本文采用 pgi 10.4 作为 Fortran 语言的编译器，另外还需要 MPI 的支持，本文的实验环境安装了 mpich 2-1.2.1p1。除此之外，CUDA 编程环境是必不可少的，这里安装的是 cuda 3.1 版本。为了 GRAPES 系统的后期处理和显示，可以选择性地安装 netcdf 或者 GrADS 等后处理软件[8]。

表 2　RRTM 模块运行性能比较

待测变量	运行时间(us)				
	−O3 选项	−fast 选项	GPU 运行 128	GPU 运行 256	GPU 运行 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

在测试过程中将 block 大小设定成 128，256 和 512，表 2 是测试比较结果。在测试中，对 RRTM 模块的 CPU 代码使用了-O3 和-fast 优化选项。其中，-O3 选项将加大存储的开销和延长编译时间，不过它可以加快运行速度；而-fast 选项则是使输入数据向量化，它会使代码执行得更快，但是可能带来一定的精度损失。

表 2 和图 4 给出了 RRTM 模块中各个子例程在 CPU 和 GPU 不同分块中的性能比较。可以更直观地看到 RRTM 模块各例程占总执行时间的比例。rtrn()和 gasabs()例程在 RRTM 模块(此处即指 rrtmlwrad()例程)中总共的执行时间大约为 86%。在 GPU 版本的执行时间约占模块总时间的 80%，略少于 CPU 代码的执行时间，主要原因是从主机端到 GPU 设备端的数据传输消耗了一部分时间，因而真正用于计算的时间比例就相对有所下降。

同时，在两种具有不同编译优化选项的 CPU 代码中，gasabs()都是最耗时的部分，约占总时间的 50%以上，而 rtrn()则在 30%左右；但是在 GPU 版本中则不然，GPU 加速后的 RRTM 模块中最耗时的子例程是 rtrn()(约占总时间的 52%)而不是 gasabs()(约占总时间的 28%)。

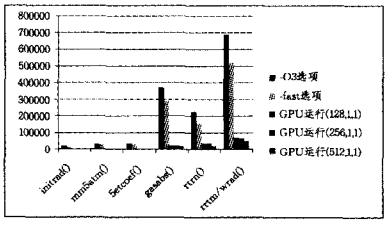

图 4　RRTM 模块各例程的运行性能比较

造成 rtrn()在 GPU 版本代码中消耗更多的时间的原因是 GPU 的利用率不高。由于物理网格内数据的依赖性，rtrn()模块的核函数只有有限的线程覆盖一个网格的水平面。执行该核函数总共 4380 个线程，远小于 Tesla C1060 最高占用率的一半并发线程数——大约 15000 个。相反，在 gasabs()子例程中的若干 taugbn()核函数在垂直坐标上没有数据依赖性，

因此可以拥有足够多的线程覆盖整个网格。

表3给出了RRTM模块主要输出结果，其中主要涉及冷却率rthraten、向上长波辐射olr和表面向下长波通量(flux)值glw在GPU版本与CPU版本中计算的误差比较。这里我们以-O3选项的计算结果作为参考数组。从表3中可以看出，olr和glw的误差相对较小，而rthraten的误差相对大一点，这是因为在冷却率计算时，在相邻的垂直层次中采用了不同的向上和向下净通量。利用GPU进行加速计算后的输出数据与-fast选项相差并不大，在允许的误差范围内[9]，因此GPU加速的方法被证明也是可行的。

表3 rthraten、olr、glw误差比较

变量	-fast选项	GPU运行情况
rthraten	1.36E-03	5.12E-04
olr	2.35E-07	5.84E-07
glw	2.54E-09	8.13E-07

根据表2列出的GPU并行化的程序在不同线程块维度值下的执行时间对比，可以得到(128,1,1)和(256,1,1)划分下的各子例程，其执行时间区别不大，除initrad()子例程外，其他例程在线程块维度为(512,1,1)的执行时间都比其他两种维度下的执行时间有明显减少，说明线程维度值的设计对算法的并行性能有一定影响。

出现这种情况的主要原因是在不同线程维度值下，一个线程块的大小是不同的，因此GPU上的存储器访问调度时机也是不同的，线程维度值太小，以线程块为单位的数据被更频繁地调入GPU存储器(如全局存储器、常量存储器、纹理存储器等)中，如果线程块内的数据访问的数据偏移量比线程块大时，就导致存储器访问延迟没有被隐藏，从而影响了整体的执行时间。因此，增大线程块的大小(线程块的维度值)可以在一定程度上隐藏访存延迟，增加算法并行化性能。当然，也不是线程块定义得越大越好，因为同一线程块内的所有线程是共享同一块常量缓存、同一块纹理缓存和同一块共享存储器，当这些线程申请的存储空间大于硬件配置的大小时，可能引起核函数启动失败。经过以上分析，RRTM模块的线程块维度定义为(512,1,1)时可以有效提升算法性能，并最终使RRTM模块的加速比达到13.7倍。

4.3 使用常量存储器进行优化

根据之前的优化，我们在线程块定义为(512,1,1)时获得了13.7倍的加速比。将模块中的常量信息使用以下的方式申明，在CUDA Fortran编译器编译后，这些变量将存储于GPU的常量存储器中，例如：

```
! 模块输入数据维度值相关的常量，
! 有 constant 关键字标识表示申明存储到常量存储器
integer,constant::kts,kte,ktep1,nlayers
integer,constant::nx,ny,nxy,nxyPad
! setcoef()子例程用到的常量
real,constant::preflog_c(59),tref_c(59)
```

常量存储器是只读的，并且是带缓存的，在被缓存时，其访问延迟只有一个时钟周期，而不带缓存时，访存效率与访问全局存储器相同，大约400～600个时钟周期。因此，将一些只读变量存储在GPU的常量存储器中，利用常量缓存可以有效减少访问全局存储器的次数，从而提高计算速度。以(512,1,1)的线程块划分，增加常量存储器进行优化后，我们得到以下结果，如表4所列。

表4 使用常量存储器后的优化结果

运行时间(us)	
不使用常量存储器	使用常量存储器
50256	48375

通过时间对比，使用常量存储器后的执行时间比之前减少了(50256-48375)/50256=3.7%，最终达到的综合加速比为690524/48375=14.3。

结束语 本文对GRAPES全球/区域同化预报系统的原理、结构、并行化方案进行了研究，并提出针对RRTM模块进行GPU并行化的基本思路；在性能分析的基础上，针对GPU体系结构的特点，从代码优化、存储器优化、编译选项等方面对程序性能进行优化，并取得了14X倍的加速比。

由于RRTM模块依赖于查找表，性能的提升也与输入数据有相关性，尤其是决定变量存储的时候(存储在寄存器或常量存储器等)。本文用到的全局存储器和常量存储器是一种方法，但是对于特定的输入数据集却不一定是最佳的选择。另外对查找表，还涉及合并内存访问的问题，如果采用纹理存储器存储这些查找表比存储在有限的常量存储器中更合适。目前的CUDA Fortran还不支持纹理存储器，所以本文中没有采用这样的方式。经过测试表明，长波辐射RRTM模块在GPU上并行计算过程正确、稳定而且有效，并为GRAPES系统未来的在GPU平台上的并行化发展奠定了一定的基础。

参考文献

[1] GRAPES新一代全球/区域多尺度统一数值预报模式总体设计研究[J]. 科学通报，2008,53(20):2396-2407

[2] 新一代数值预报模式GRAPES的并行计算方案设计与实现[J]. 计算机研究与发展，2007,44(3):510-515

[3] Simek V,Dvorak R,Zboril F,et al. Towards accelerated computation of atmospheric equations using CUDA[C]// Proceeding of 11th International Conference on Computer Modelling and Simulation,UKSIM '09. Cambridge,2009:449-45

[4] Huang B, Mielikainen J, Oh H, et al. 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

[5] Michalakes J,Vachharajani M. GPU Acceleration of Numerical Weather Prediction [C] // IEEE International Symposium on Parallel and Distributed Processing. IPDPS 2008. Miami, FL, USA,2008:1-7

[6] Mlawer E J,Taubman S J,Brown P D,et al. Radiative transfer for inhomogeneous atmospheres: RRTM,a validated correlated-k model for the longwave[J]. Journal of Geophysical Research, 1997,102(D14):16663-16682

[7] Michalakes J, Dudhia J, Gill D, et al. The Weather Research and Forecast model: Software Architecture and Performance[C]// Proceedings of the 11th ECMWF Workshop on the Use of High Performance Computing In Meteorology. 2004

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

[9] 清华大学计算机系. 并行计算[EB/OL]. http://wlzy.aynuedu.cn/jsj/wlkc/bxjs/text/ch01/se04/1_4.htm