多 GPU-CPU 混合异构平台下的光谱计算优化

Accelerating Spectral Calculation through Hybrid GPU-based Computing

领　　域：计算机技术工程

作者姓名：徐星宇

指导教师：孙济洲　教授

企业导师：纪　丽　研究员

天津大学计算机科学与技术学院

二零一五年十二月

独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作和取得的研究成果，除了文中特别加以标注和致谢之处外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得 **天津大学** 或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

学位论文作者签名：　　签字日期：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解 **天津大学** 有关保留、使用学位论文的规定。特授权 **天津大学** 可以将学位论文的全部或部分内容编入有关数据库进行检索，并采用影印、缩印或扫描等复制手段保存、汇编以供查阅和借阅。同意学校向国家有关部门或机构送交论文的复印件和磁盘。

（保密的学位论文在解密后适用本授权说明）

学位论文作者签名：　　导师签名：

签字日期：　　年　　月　　日　　签字日期：　　年　　月　　日

摘　要

太阳系外的所有宇宙天体的信息几乎都是通过光谱计算获得的，能够观察到的光谱包含了大量的重要信息，例如恒星的温度、年龄、金属丰度以及星系组成等。

目前天文领域有一些经典的光谱计算工具包，例如 XSPEC，ISIS，XSTART，APEC 等等，虽然这些工具包可以精确地进行光谱计算的求解，但程序结构仍停留在传统的串行模式上，目前并没有任何一个基于并行架构的光谱计算工具。光谱计算的核心部分是数值积分，随着 GPU 通用性的不断提高，计算性能的稳步增长，许多经典的数值积分算法已经发展出来 GPU 加速版本。但是目前现有的 GPU 加速版本的数值积分算法都是针对大区间的高维积分，并不适用于光谱的计算，光谱计算中数值积分的特点是大量的、一维的、积分区间非常小的积分计算。因此要在多 GPU-CPU 的混合异构平台上加速光谱计算，不但需要解决光谱计算的核心算法向 GPU 的迁移，而且必须对 GPU 与 CPU 进行合理的动态任务调度，充分发挥 GPU 和 CPU 各自的优势。

本文提出了一种多 CPU-GPU 混合异构并行方法来加速求解光谱计算。首先将计算密集型的积分部分放到了 GPU 上来计算，通过合理的任务粒度划分减少主机和设备之间频繁的数据拷贝来提高计算的性能。其次，提出了一种基于多个 CPU 与多个 GPU 之间的动态任务调度策略，该策略基于任务队列和共享内存的方法，该种方法相对于传统的客户机-服务器的体系结构可以很好地减少额外的通讯开销。最后，综合理论分析和实验验证了本文所提出的方法的有效性和准确性，实验显示，使用 24 个 CPU 核、3 个 GPU 设备，相对于传统的串行 APEC 实现方法，可以将整体的计算加速 300 倍，相对于纯 CPU 的 MPI 并行方法，整体的计算加速也有 22 倍。此外，本文提出的方法也可以适用于单个任务计算量小，但总体任务量巨大的应用，使用本文提出的方法，在求解非电离平衡的常微分方程的计算中，相对于纯 MPI 的并行方法提速了 15 倍。

关键字： 数值积分，负载平衡，GPU，混合异构，光谱计算

ABSTRACT

Essentially all information about astronomical objects outside the solar system comes through the study of electromagnetic radiation (light) as it reaches us. The observed spectrum contains a multitude of important information about star temperature, age, metal abundance and stellar composition etc.

Currently, quite a few tools had been developed to model, calculate and analysis the electromagnetic spectrum in astrophysics, and the widely used ones include the Interactive Spectral Interpretation System (ISIS), the XSPEC, the XSTAR, and the Astrophysical Plasma Emission Code (APEC) and so on. These kits are good ways to solve spectrum calculation, but structure of the program is still in the traditional serial mode. Until now, there is no any spectrum calculation tools which are based on parallel architecture. The core of the spectrum calculation is numerical integration. Meanwhile GPU-based high performance computers have gained popularity in scientific computing as a low cost and powerful parallel architecture in the last decades, and the use of GPUs has proven to significantly increase the performance in numerous applications, including solving large differential equations and high-dimensional numerical integrations. However, the spectral calculation has two distinct characteristics that common GPU-based numerical integration schemes seldom address. One is each single one-dimensional integral computing is very small and fast, but there are huge amounts of small integrations. The other is classical load balancing approaches for CPU-GPU hybrid architecture may be not efficient to schedule so many small tasks due to the extra overhead proportional to the frequency of scheduling

In this paper, we proposed a hybrid CPU-GPU parallel approach to accelerate spectral calculation. First, we offloaded the compute-intensive integral parts of the application to GPUs, and reduced the frequency of memory copy between device and host by combining many single integral operations within one ion into a coarse-grained task. Second, for a large number of small tasks in the spectral calculation, we developed a task scheduling scheme among multiple CPUs and GPUs via share memory that can avoid extra communication overhead in the traditional client-server architecture. Last, comprehensive theoretical analysis and experiments were conducted to verify the efficiency and accuracy of the approach, and the experiments showed that 24 CPU cores with 3 GPU devices can speed up the computation by a factor of 300 relative to the

original serial implementation, and a factor of 22 relative to the 24 CPU cores parallel version. Additionally, the approach was also adapted to NEI related application involving numerous ODEs and achieved a 15-fold speedup relative to the pure MPI implementation.

KEY WORDS: numerical integration, load balancing, GPU, hybrid architecture, spectral calculation

目 录

第一章 绪论

1.1 课题背景

随着计算机性能的不断提高，数值模拟已经在各个科研领域蓬勃发展起来，在天文领域亦是如此。大量观测数据的涌现不仅给构建理论模型带来了机遇，同时也带来了挑战，目前的观测可以清晰地分辨这些天体物理过程中单个谱线的信息，但简单的理论模型已经远远落后于空间望远镜的观测数据。所以对于天文领域而言，基本上太阳系外的天体的所有信息都来自于该天体到达地球的光谱所包含的信息[1]。我们观察到的光谱含有丰富的物理状态和信息，比如天体的温度、年龄、元素的丰度以及构成等等。当今的天文学家通常是利用观察到的光谱与光谱计算模型的计算结果进行对比来进行光谱的相关研究。为了更好地从光谱中把这些物理信息解析出来，设计高效的准确的光谱计算软件无疑是推进天文领域相关研究的有效方法。

数值模拟也叫计算机模拟，主要是依靠电子计算机，结合有限元或有限容积的概念，通过数值计算和图像显示的方法，达到对工程问题和物理问题乃至自然界各类问题研究的目的。天文数值模拟，顾名思义是应用在天文领域的数值模拟方法。在对天文观测现象，理论推理验证，天文现象发展预测的过程中首先要建立反映问题本质的数学模型。具体来说就是要建立反应问题中各量之间的数学方程及相关的定解条件。这是数值模拟的出发点。正确完善的数学模型是数值模拟得以进行的保障。数学模型建立后，需要解决的问题是寻求高效率、高准确度的计算方法。目前的数值模拟方法有很多，微分方程的离散化方法以及求解方法、贴体坐标的建立、边界条件的处理等等。在确定了计算方法后，就可根据问题本身进行程序的编写和计算。通过计算可以对结果进行进一步的完善。在计算工作完成后，大量数据可通过可视化显示出来。

在非电离平衡状态下存在一种非常重要的光谱——辐射合成连续（Radiative Recombination Continuum，以下简称 RRC[2]）光谱更是重中之重。它是由于电子与离子碰撞后重组，在此过程中产生了光子，光子的能量等于电子的动能加上与重新重组的电子的结合能量，由于电子的动能不是量化的，所以形成了一个连续谱线。有关光谱计算的具体过程我们会在第二章详细阐述。

1.2 国内外研究现状

国内外天文学家们针对于光谱计算的研究主要建立在完备的最新高能原子数据（AtomDB）基础之上。AtomDB 中涵盖了完整的天文原子数据，是天体物理计算的基础，它由美国 Randall Smith 团队创建而成，该数据库提供了电离平衡和非电离平衡的基础数据和计算模型。目前已经有不少软件用于天文领域光谱的模拟、计算和分析。被广泛应用的有 ISIS[3](Interactive Spectral Interpretation System)、XSPEC[4][5]、XSTAR[6]、APEC[7](Astrophysical Plasma Emission Code)。随着现代天文望远镜以及高性能计算系统的发展，不仅天文观测数据量暴涨，同时高分辨率的天体物理模拟也变为可能。因此，对于光谱计算的任务比例也随之增加。然而，以上的这些软件都来源于传统的体系结构，很多新的功能在过去的几十年里不断地集成到这些老体系结构中，由此产生了适应现代高性能计算的体系结构。所有这些常用的光谱计算软件都是建立在传统的 CPU 体系架构上的，而这一特点使得这些软件并不能有效地解决计算密集型的光谱计算任务。目前很多新兴的基础设施已经应用到了现代高性能计算领域，例如异构计算体系架构。随着基于 GPU 的高性能计算机在现代科学领域以低成本高并行的特性受到了越来越广泛的青睐，使用 GPU 来提高数值应用（例如求解大规模的数值微分方程[8]，高维的数值积分[8]等等）的性能也越来越被认可。

1.3 主要工作和贡献

光谱计算是天文领域非常重要的一项计算任务，从光谱中可获得大量的信息，例如离子丰度，温度，密度等等，然而现有的光谱计算的工具和方法使用的是纯 CPU 的体系架构，所有计算任务都是串行完成。而对于最基本的光谱计算空间，即使使用性能最好的 CPU，计算性能也远不能达到理想的效果，所以实现大规模的计算任务，成为了一项具有挑战性的工作。

本文主要完成了以下工作：

1）对现有的光谱计算工具和方法进行了调研，对比性能，找出提高性能的关键点。

2）提出了一个基于多 GPU-CPU 的异构混合并行架构来加速光谱计算。同时本架构还适用于其它由大量的计算量小的任务组成的计算密集型的应用。

3）本文提出了一种 GPU 与 CPU 之间的动态负载平衡策略，利用共享内存和任务队列来处理大量的相似的小计算量的任务调度。

4）通过实验验证和评价了本文提出的方法的有效性和准确性，同时证明了本方法的可迁移性。

5）对工作进行了总结，提出了改进方法，同时给出了未来工作的研究方向。

1.4 本文的内容和结构

本文深入研究了现有的光谱计算工具和方法，针对其特点以及光谱计算数据的特点，提出了一种基于 GPU-CPU 的混合异构并行架构，将光谱计算的核心算法由 CPU 迁移到了 GPU 上，并进行了合理的改进；同时提出了一种针对多 GPU 多 CPU 的动态调策略，平衡 CPU 和 GPU 之间的工作负载，以使 GPU 达到最大的利用率；此外，我们还对提出的方法进行了验证和评估，确保方法的效率、准确性，该方法还可迁移到其他类似的包含密集的小计算量任务的应用中。本文的主要内容如下：

第一章为绪论，主要介绍了光谱计算的课题背景，国内外的发展情况，同时阐述了光谱计算面临的亟待解决的问题，以及在本文提出的解决方案。

第二章主要介绍光谱计算的相关工作，给出了现有计算工具的特点、性能以及缺陷。介绍新兴技术的发展情况，对于光谱计算有哪些帮助，同时提出对光谱计算方法进行改进所面临的问题。

第三章详细介绍了基于 GPU-CPU 的光谱计算框架的设计，针对光谱计算特点提出了合理的架构设计，阐明光谱计算核心算法在 GPU 上的实现，同时给出支持 GPU-CPU 混合异构并行体系结构的动态负载平衡调度策略，充分利用 GPU 高并发的优势以及 CPU 的逻辑处理的长处。

第四章以数据和图表的形式实现和分析了本文提出的方法的性能、准确性。

第五章主要是讲本文中的研究方法迁移到非电离平衡方程的求解上，验证了本文方法的可扩展性。

第二章　光谱计算的相关工作

2.1 光谱计算的主要过程

2.1.1　光谱的物理概念

所有的原子都有一系列离散化的、量子化的能级，事实上这些能级的数量是不确定的。当原子在不同的能级之间进行跳跃的时候就会产生光谱，而此时的跳跃称之为跃迁。如公式(2-1)所示，其中E表示的是通过普朗克常量获得的不同能级之间的能量差，λ表示波长，ν是频率，c是光速，h表示普朗克常数，6.626068×10−34 J·s，$h\nu$表示光谱中每个粒子所包含的能量，称之为光子。

$$E = h\nu = \frac{hc}{\lambda} \tag{2-1}$$

每个原子吸收一系列特定波长的光。原子的光谱就好像是原子拥有的一个独特的指纹，可以轻松地将其与其他原子区别开来，如果可以阅读这些指纹信息，那么就可以从中获得大量关于该原子的信息。一个原子吸收光能就会跃迁到高能级，释放光能就会跃迁到低能级，如下图(2-1)所示，并不是所有的跃迁都是等可能的，每次跃迁不仅与波长相关，同时与跃迁发生的概率有关。

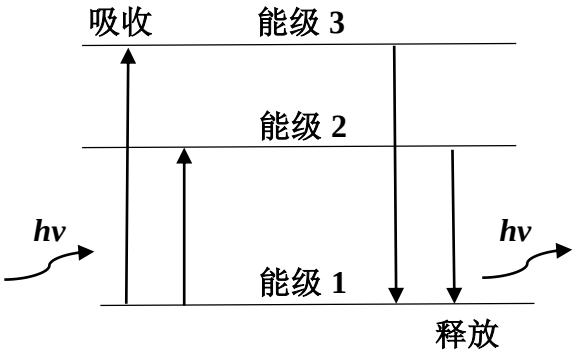


图 2-1 原子能级跃迁产生的能量吸收和释放

2.1.2　光谱计算的具体流程

光谱计算的相关工具很多，但基本都是在 CPU 上进行的串行计算，具体的计算工具本文 2.2 中会有详细的介绍，下面以 APEC 光谱计算软件为例，说明一

下纯 CPU 下光谱计算的具体流程：

1. 获取配置文件的参数
2. 初始化变量
3. 三层循环获得密度、温度和时间的值
   3.1. 获得当前元素以及原子参数
      3.1.1. 为当前元素的每个离子分配光谱数组空间
         3.1.1.1. 对于每个能级的离子进行 RRC 计算
4. 输出光谱信息

由以上的流程也可以看出，光谱计算的核心部分是 RRC 计算，且该计算处于多层嵌套循环中，GPU 善于做高强度的密集计算，但并不擅长复杂的逻辑计算，所以如何将 RRC 算法迁移到 GPU 上，是本文研究的重点和难点。但是由于在光谱计算的过程中，空间网格点（详见 2.1.3）之间相互没有依赖性，因此可以从此入手，首先实现 MPI 版本的并行优化。

2.1.3 光谱计算的核心算法

RRC 的计算是基于原子模型以及辐射机制。公式（2-2）描述了 RRC 光谱：

$$\frac{dP}{dE}=n_e n_{Z,j+1}4\left(\frac{E_\gamma-I_{Z,j,n}}{kT}\right)\sqrt{\frac{1}{2\pi m_e kT}}*\sigma_n^{rec}\left(E_\gamma-I_{Z,j,n}\right)exp\left(-\frac{E_\gamma-I_{Z,j,n}}{kT}\right)E_\gamma \quad (2\text{-}2)$$

其中P表示辐射功率，n_e表示电子密度，$n_{Z,j+1}$是离子（Z，j+1）的密度（Z 表示的是离子的原子序数，j+1 表示的是电离的状态），E_γ表示的是发射出的光子的能量，$I_{Z,j,n}$表示的是离子（Z，j）中的电子在 n 级时的约束能量，m_e表示的是电子的质量，$\sigma_n^{rec}\left(E_\gamma-I_{Z,j,n}\right)$表示的是电子能量$E_\gamma-I_{Z,j,n}$在能级 n 时的复合截面。为了在更长的波长区域获得高分辨率的谱线，实际方法是对所有离子的所有能级的有效能量区域进行光谱的积分计算：

$$\Lambda_{RRC}\left(E_{bin}\right)=\int_{E_0}^{E_1}\frac{dP}{dE}\left(E\right)dE \quad (2\text{-}3)$$

其中E_0，E_1分别表示的是能量积分边界的最小值和最大值。对于一维的积分计算的定义如公式（2-3）所示，许多经典的数值积分计算算法可以轻松地解决它。然而，对于真实世界的光谱计算，仅仅对一个大小适中的参数空间进行计算，计算的负载也很高。并且基于 2.1.2 中的 CPU 的程序，本研究对纯 CPU 下实现的光谱计算进行了性能评估以及瓶颈分析，发现 RRC 的计算占据了整个光谱计算的 98%的计算时间，而 RRC 中 90%以上的计算时间都是对公式（2-3）的反复求值。

图 2-1 表示的是光谱计算的典型空间结构，这是一个三维的参数空间：温度、

密度和时间。参数空间一般是由天文模拟的结果获得，或者仅仅是一个用于测试的简单的配置文件。对于参数空间中的每个格点，RRC 的积分计算都要执行三重嵌套循环。一般宇宙中最基本的元素总共包含了 496 个离子，每个离子包含几个到几千个不等的能级（理论上主要的能级以及子能级的数量是不确定的，但是实际上，剔除掉一些不是十分必要的能级，从而简化模型是必须的），每个能级中还包含大约10^5个能量区间（一般能级区间个数是因应用不同而有所不同，通常情况下是10^5的量级），所以每个格点中总的 RRC 积分计算大约是10^8的量级。

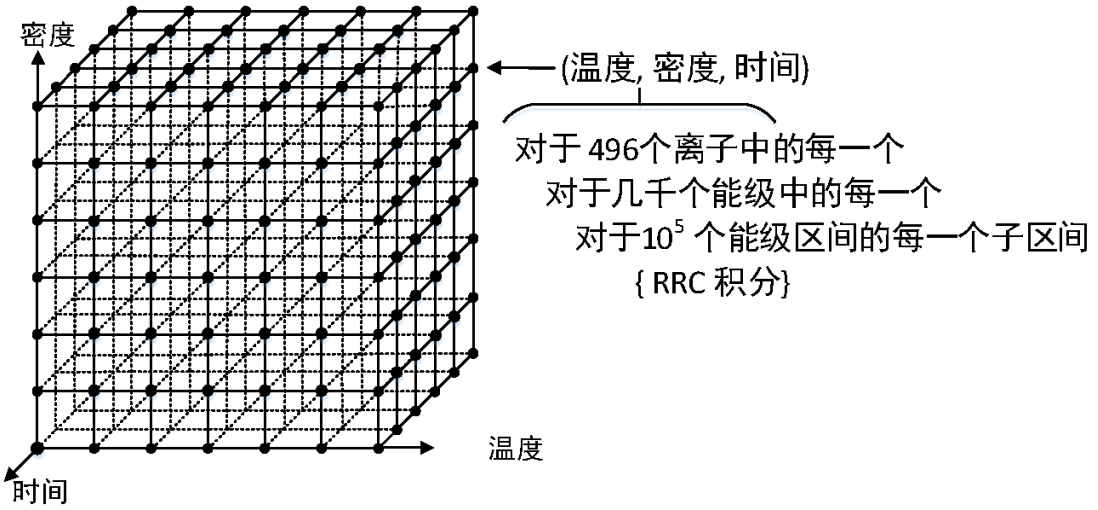


图 2-2 光谱计算的参数空间以及主程序的逻辑结构。每个黑色的圆点代表一组确定的输入参数（温度、密度、时间），对于每个网格点，在三层逻辑循环中 RRC 积分计算的数量可达到$2.0 * 10^8$

在主频 2.5GHz 的英特尔 Xeon 处理器 E5-2640 上进行的测试结果显示，平均计算一个格点的时间是 800s，使用 GNU gprof 显示积分操作占据了总计算时间的 90%以上，因此对于一个真实世界的中等规模的天文模拟问题来说，例如包含128^3个格点的计算空间，这将需要计算大约 50 万个 CPU 小时。显然这样一个数量巨大的小计算量的工作任务一样会造成无法承受的工作负载。

2.2 光谱计算的工具

目前天文领域有一些经典的光谱计算工具包，Arnaud 等人开发了一个叫做 XSPEC 的工具，同时在后续的工作中不断提升了它的计算性能[4]，XSPEC 的一个重要的特点就是它可以广泛地应用于很多物理模型中，特别是天文光谱拟合。最初 XSPEC 只适用于 EXOSAT 天文台的数据库，随着不断地发展它已经支持大多数的 X 射线天文光谱数据集。XSPEC 在业界已经应用了 20 多年，引用次数超

过几百次。

ISIS[3]是 XSPEC 的一个部分，它设计的最初目的就是用来进行高分辨下的 X 射线光栅光谱的分析，但同时它的另一个目标是通过最大化其可移植性来最大化软件的利用率。ISIS 通过一个工具包来查询数据库中的原子数据以及等离子体的发射模型，然后使用其核心操作来处理这些高分辨率的光谱数据。ISIS 中使用的工具可以简化低水平的操作，例如数据的输入、数据绘图和数据查找和检索，这样就可以使用户专注于高层次的问题分析。ISIS 的可编程性和可扩展性都很好，这意味着用户可以自行编写脚本来简化重复的分析任务，同时扩展了 ISIS 的命令语言。ISIS 和 XSPEC 都可通过 PModel 插件进行进程级别的并行，PModel 允许不同的天文物理模型的组件同时分布在处理器上进行计算，即使编程经验并不丰富的天文学家也可以使用自如。

XSTAR[6]是用来计算光电等离子气体的物理状态和发射谱线的，目前有两种方式可以使 XSTAR 实现并行化，一个是叫做 Parallel Virtual Machine（PVM）的封装器，它可以通过运行脚本设置独立参数，实现命令行级别的并行化，使得 XSTAR 实现应用程序并发执行[9]，通过 PVM 插件，ISIS 也能够同时运行多个 XSTAR 实例。

APEC 是另一种强大的被广泛应用的工具，APEC 能够同时计算同一个热等离子体在非电离平衡状态下的线谱和连续光谱[7]，它主要是利用天文等离子体放射数据库来计算热等离子体的光谱模型。天文等离子体放射数据库包含所需的所有数据，例如碰撞辐射率、重组交叉域、双电子性复合率、卫星线波长等，但是遗憾的是 APEC 并没有并行的实现版本。

截止目前，根据本研究的调研结果，目前并没有任何一个基于多 GPU 多 CPU 混合异构体系结构下的光谱计算的工具。

2.3 基于 GPU 的数值积分计算

数值积分拥有庞大的算法库，来进行有限区间的积分计算，许多算法不断发展，目前已经是标准数值库的一部分，例如 QUADPACK[10]、MKL[11]、CUBA[12]、GSL[13]等，对于高维积分来说提供可依赖的评估结果需要考虑 CPU 时间占用的比例，通常这需要有效的可并行算法的支持。然而，在自适应网格对应的数值积分计算中确定性的并行结构算法并不多，一些现有的并行算法也仅仅是简单地扩展原来的串行代码，使用多核平台来进行多线程的并行化，此种方法获得的计算加速比也只处于中等水平。在过去的几年中，GPU 的出现为实现高效并行的计算科学工程提供了可能，GPU 的大规模并行架构，使得 GPU 可以对大数据进行

有效的并行化处理。多 GPU、多 CPU 的混合架构确保了 GPU 通用计算的可伸缩性。OpenMP 和消息传递（MPI）便成为了 GPU 集群管理的工具。

随着 GPU 通用性的不断提高，计算性能的稳步增长，许多经典的算法已经发展出 GPU 加速版本。这使得在常见的廉价的硬件设备上显著提高多维自适应网格积分计算的性能成为了可能。Kamesh[14][15]等人重用 CHURE 算法，提出了一种基于多 GPU 的高效的确定的并行算法来求解自适应多维数值积分计算。由于全局内存的带宽限制，共享内存的有效使用使得 GPU 的性能显著提高。CHURE 串行算法使用了动态的大数据的堆结构进行频繁的数据访问。设计一个基于 GPU 的算法将一部分数据结构缓存在共享内存中，这样可以最小化全局内存的访问，而这恰恰是一个具有挑战性的任务。Kamesh 等人提出的算法刚好解决了此问题，此外他们还将此算法扩展到了多个 GPU 设备上。算法实现的硬件环境是 Intel Xeon CPU X5650 集群，每个节点 4 个 Tesla M2090 GPU 显卡，使用 OpenMP 和 MPI 进行通讯。在单个 GPU 设备上该算法可实现 240 倍的加速，而未做内存加速时只能实现 70 倍的加速。在拥有 6 个计算节点的集群上，与串行算法相比加速比可达 3300 倍。

Arakawa 在 1966 年利用雅可比矩阵实现了有限差分方案，从而求解二维不可压缩流的运动方程，该方法减少了非线性计算的不稳定性，同时为长期的数值积分提供了基础。Evren[16]等人针对 Arakawa 公式提出了一种有效的实现方法，该方法通过扩展单指令多数据流(Streaming SIMD Extension，简称 SSE)以及先验矢量扩展指令(Advanced Vector Extension，简称 AVX)，同时优化了内存访问的性能，性能评价显示矢量化实现相比非矢量化实现提高了将近两倍的计算速度。矢量处理本就广泛应用于超级计算机系统，但随着大规模处理器系统的变化而慢慢被放弃，但是随着 SIMD 的扩展，向量处理又重现发挥了自身的力量。如果没有内存和寄存器之间内存访问的瓶颈的话，向量化将是最有效的方法。Arakawa 公式并不包含任何像求根计算或者三角函数计算这类计算强度高的操作，所以最终限制该公式计算的是内存带宽，因为完成代数操作所需的时钟周期小于从内存中获取操作数的时钟周期数，但是使用向量化的代码就能很容易地解决该问题。

除此之外，传统的数值计算还可以通过使用自适应的辛普森算法进行 GPU 的并行优化，但是该种算法依赖于错误估计，因此这一点也成为了一个重要的计算开销；此外自适应辛普森算法需要进行递归调用，后一个计算的参数是上一步计算的计算结构，而这一点恰恰与 GPU 并行优化的计算特点背道而驰，尽管改写算法使之成为非递归的形式并不难，但是复杂的逻辑结构也会使算法不适合在 GPU 上进行并行优化。基于此 Daniel[17]的团队提出了一种新颖的自适应启发式积分方法，一般求解自适应积分计算的方法有两种，一种是后验方式评估积分误

差，然后再将子区间进行细化，另一种是利用先验方式获得积分区间的长度。Daniel 团队使用的就是第二种方法。对每个积分区间都使用梯形算法，如果积分结果超过阈值，那么将其丢弃，对积分区间继续进行细分，重复此过程，该种方法称之为 forward-backward。Daniel 团队提出的方案是一个接一个使用启发式的方法选择积分区间的间隔长度，任意的积分算法都可以应用在这些积分区间上，而不丢弃任何的积分结果。该种方法不仅比传统方法有效，而且可适用于加速基于 GPU 的大规模的并行积分计算。

以上的所有工作应用在大规模的多维积分计算上都有非常显著的性能提升，但是他们的设计并不适用于单个任务计算量小，任务数目巨大的积分计算，因为对于这样的密集型小计算任务在主机和 GPU 设备之间产生的通讯开销远远超过计算任务本身的开销。因此在使用 GPU 进行并行加速的过程中需要最小化 CPU 和 GPU 之间数据交换的频率，同时最大化 GPU 计算的使用率。

2.4 基于 GPU-CPU 的负载平衡调度

多 CPU 多 GPU 混合异构系统的诞生，给 CPU 和 GPU 之间的自动任务调度带来了许多困难和挑战。多 GPU 与多 CPU 构成的混合异构系统中工作负载的分配一般是将工作任务细分，然后利用异构系统的数据并行性将细化的任务分发到 GPU 和 CPU 上。但是人为地对 CPU 与 GPU 上的计算任务进行划分是一项不小的挑战，因为 GPU 的计算性能对计算任务的粒度是非常敏感的。目前现有的许多负载平衡调度算法都是静态的或者是动态的。静态的负载平衡调度会导致 CPU 和 GPU 之间频繁的数据同步，由于同步会产生开销，由此产生的总的开销必将降低整体的性能。

利用 GPU 求解大规模的常微分方程也是近年来的大趋势。在许多情况下，常微分方程的计算会占据大部分的模拟时间，于是 Yu[18]等人提出了一种新的显式的偏微分刚性方程求解器，它利用 GPU 的并行架构来加速多维空间上常微分方程。他们给出了两种常微分方程的求解器，一种是基于 CPU 的隐式求解器，另一种是基于 GPU 的显式求解器，利用动态的负载平衡调度器，将整个网格点中的复杂的常微分计算分配到基于 CPU 的隐式求解器或者是基于 GPU 的显示求解器上。对于网格点中的高度刚性常微分方程，CPU 上的隐式求解器更加高效；而对于中度刚性或者非刚性的常微分方程，基于 GPU 的显示求解器则更加高效。使用 CSP 等方法可以量化常微分方程的刚度，但是使用这些方法往往会涉及复杂的矩阵特征值分析，从而产生昂贵的计算开销。因此使用这类方法对网格点中常微分方程的刚度进行排序会引入很高的开销，为此该团队使用了

CHEMEQ2 算法对待求解的常微分方程的刚度进行排序，再根据排序结果将这些计算任务的分配到 CPU 或者 GPU 求解器上进行计算。Yu 团队提出的这个调度方法不仅可以扩展到其他类似的问题上，而且该方法能够灵活有效地分发任务，合理利用计算资源，使计算资源利用率达到最大。

快速傅里叶变换(Fast Fourier Transform 简称 FFT)是科学和工程领域应用最广泛的一种数值算法。对于大型的科学和工程计算如大型物理模拟、信号处理、数据压缩等 FFTs 计算占用了大量的计算执行时间，因为 FFT 的实现需要大量的计算资源和内存带宽。因此相比于多核 CPU，GPU 无疑是一个最佳的并行优化候选。但是随之而来的是 CPU 与 GPU 之间任务的调度问题。Shuo[19]等人基于混合异构系统实现了快速傅里叶变换计算的动态负载平衡调度，该方法通过将 GPU 和 CPU 上总的计算执行时间分割成几个基本的子步骤，分析各个子步骤的自身的性能模型和异构系统执行流，参数模型提供了不同工作负载下 GPU 上的执行时间的评估函数，对于每个硬件配置，校准的模型在两种配置下运行，一个是 CPU，一个是 GPU，根据不同的分配比率来确定模型参数的值。之后使用这些参数值，可以自动地进行评估不同分配比率下的总的执行时间，而非真正地执行每个功能函数。接着使用动态规划找到使用原语构建的不同问题的最优实现。所以使用本方法可以找到一个小区间的最优选择，在区间内部通过性能测试找到最优解。

Giuliano[20]的团队提出了一种基于异构多核系统的并行的自适应算法来求解多维积分计算，基于 CPU 和基于 GPU 的两种不同策略一起组成了 Giuliano 的算法：1）对多核 CPU 之间的线程进行负载平衡调度；2）GPU 核中运行有效的数值积分算法。使用该算法后，相比于同构多核 CPU 的情况性能提升 3 倍多。但不足的是，该团队提出的负载平衡调度仅仅是基于多 CPU 的线程调度。

Wang[21]等人提出了一种基于渐近分析(CO-Scheduling Asymptotic Profiling，简称 CAP)的协同调度策略，CAP 可以动态地划分任务，再将任务分发到 CPU 和 GPU 上，而该过程中只需要进行很少的几步同步操作，该策略采用的是性能分析技术来预测性能，再根据性能划分工作负载，在此过程中也优化了 GPU 的性能。与目前最好的协同调度策略相比，CAP 可以将整体性能提高 42.7%。

Alejandro[22]等人研发了一种动态负载平衡库，该库可以使可并行的代码在大规模的异构并行集群上部署并进行计算。计算任务在多 GPU 之间通过消息传递或者共享内存进行通讯，当所有任务的任务粒度非常接近时，该方法的效果可以发挥到最佳。

Bo[23]的团队提出了一种针对基于 GPU 和 CPU 异构混合体系集群下的大规模问题的有效的任务分配模型，该模型的核心思想很简单：对于高计算能力的处

理器尽量让其不要闲置，除非闲置是不可避免的，同时使用 CPU 和 GPU 可以最大化异构集群的计算能力，所以假设 GPU 中核的计算能力是一样的，CPU 中核的计算能力是一样的，那么分配计算任务主要分三个步骤：1）计算 CPU 工作时间波，实际上每个 CPU 处理器中每个 CPU 核的工作时间的波动很小；2）引入动态平衡因子 P；3）计算 P 的最优解。本文中提出的基于 GPU-CPU 的光谱计算的负载平衡策略与 Bo 的模型很相似。

值得指出的是 NVIDIA 提供的 Multi-Process Service（MPS）[24]也能够用于 CPU 和 GPU 之间的任务调度，MPS 是一个二进制兼容的客户端-服务器结构的运行时 CUDA API 实现，MPS 设计的目标就是利用 inter-MPI 级别的并行化，提高 GPU 整体的利用率。尽管 MPS 能够支持多 GPU，但是如果每个任务都很小，调度又很频繁的话这种用户-服务器架构会产生很多额外的开销，例如光谱计算就不适合 MPS API 来计算。

第三章 多 GPU-CPU 混合异构平台下的光谱计算的设计与实现

3.1 整体架构设计

如第二章相关工作中所描述的，光谱计算的核心计算是求解一维数值积分的过程，该过程的计算时间占整个光谱计算总计算时间的 90%以上，目前现有的光谱计算工具都只是基于 CPU 的串行计算或者基于命令行级别的并行计算。当计算规模扩大后，现有的光谱计算工具的计算性能远远不能达到理想的计算效率。同时，基于 GPU 的高性能计算机以其低开销性和高并发性的架构特点在科学计算领域获得越来越广泛的应用，使用 GPU 来进行数值计算的案例也越来越多，例如求解偏微分方程[8]、高维数值积分[8]等等都取得了显著的性能提升。所以使用 GPU 来加速光谱计算是一个不错的选择，然而基于光谱的数值积分计算相比于传统的利用 GPU 来求解的数值积分计算有以下两点不同：1）基于光谱的数值积分是一维的积分计算，每个积分区间都非常小，但是积分区间的数量巨大，而传统的基于 GPU 的数值积分计算大多都是求解高维的，积分区间大的，大规模数值积分计算。2）传统的基于 CPU-GPU 异构架构的负载平衡调度方法并不适用于目前的光谱计算，因为针对积分区间数量巨大的，积分区间小的数值积分计算来说，使用传统的调度算法会带来巨大的调度开销。

现有的基于 GPU 的数值积分算法对于求解高维的数值积分计算很有效，最主要的原因是传统 GPU 上的积分计算都是计算密集型的，相比于主机和 GPU 设备之间的通讯，GPU 上的计算时间占用了绝大部分，而主机和设备之间的通讯开销只占据非常小的一部分。而基于光谱的数值积分计算则完全不同，光谱中的数值积分计算不仅只是一维的积分计算，而且每次积分计算的积分区间非常小，因此尽管一次光谱计算在主机和 GPU 设备之间产生的通讯开销是非常小的，但是大量的光谱计算则会产生不可估量的通讯开销。实验表明，如果光谱计算的基本调度单元是一次 RRC 计算，那么使用传统的基于 GPU 的数值积分算法来加速光谱计算并不会提高性能，相反可能会降低计算性能，因为频繁地在主机和 GPU 设备之间进行数据的拷贝和传输，会产生昂贵的通讯开销，当通讯开销远远高于积分计算本身的开销的时候，那整体的计算性能必然会下降。所以将光谱计算中的数值积分计算直接按照传统的基于 GPU 的数值积分算法进行移植是不可行的。

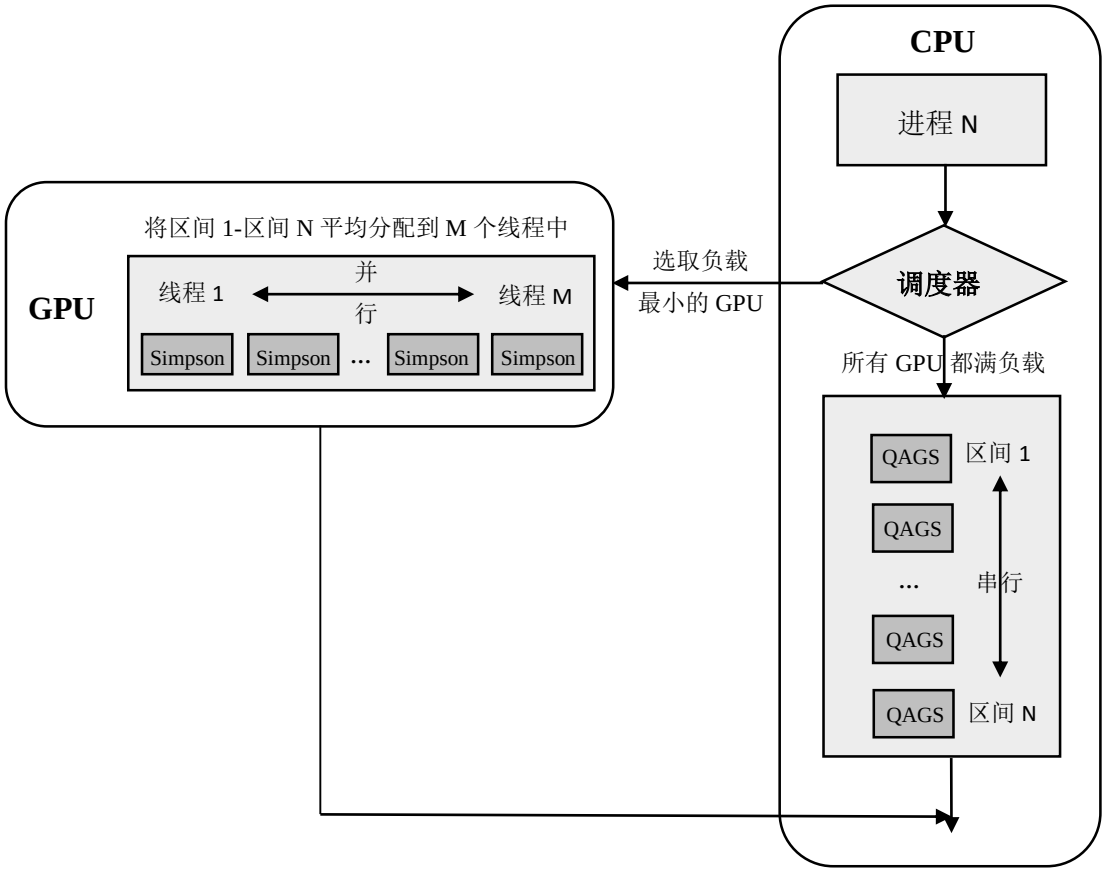


图 3-1 基于 GPU-CPU 的光谱计算的整体设计

由此我们给出了基于 GPU-CPU 的光谱计算的整体设计如图 3-1 所示，该设计主要由四个部分组成：1）基于 MPI 的并行封装器；2）基于 CPU 的数值积分求解器；3）基于 GPU 的数值积分求解器；4）GPU-CPU 之间的负载平衡动态任务调度器。在第二章中我们介绍过，光谱计算的空间结构是一个三维的网格点，各个网格点之间是没有信息交互的，因此在并行架构上可以使用 MPI 多进程来完成，进程之间没有消息传递大大减少了通讯开销。因此 CPU 上的主进程获得输入的参数后，会唤起 N-1 个 MPI 进程，主进程将整个参数空间划分成 N 份，再将各个子空间分配到 N 个进程中。在每物理节点内部会有一个动态任务调度器，他负责该节点中的所有 MPI 进程之间的任务调度。当 MPI 进程产生一个计算任务后，就会向调度器发送资源请求信息，调度器查看 GPU 设备的工作负载情况，当所有 GPU 设备都满负载的时候，分发给基于 CPU 的光谱计算求解器，CPU 求解器会利用 CPU 上现有的 QAGS 算法对积分计算进行串行的求解；如果有 GPU 设备可选，那么调度器会选取一个工作负载最小的 GPU 设备，将数值积分计算任务分发给基于 GPU 的光谱计算求解器，GPU 上的数值积分求解器使用的是 Simpson 算法进行并行的积分计算（3.3 中会有详细的算法介绍），计算完成之后再将结果回传到主机，进行下一次的任务计算。

3.2 多 GPU-CPU 的负载平衡调度设计

动态负载平衡调度器是本文的核心部分，本文提出的负载平衡策略基于一个简单有效的思想：尽量使 GPU 处于计算（忙碌）状态。本文中的主程序是建立在 MPI 并行架构之上的，为了简单稳定，并不存在一个统一的负载负责负载平衡调度的中心程序，取而代之的是，每个物理节点上都会有一个负载平衡的调度器，主程序只需要通过将整个参数空间进行划分，分配到不同的物理节点上即可。所以每个物理节点上的调度器只负责本节点内部的任务调度，各个调度器之间无任何的消息传递或干扰。

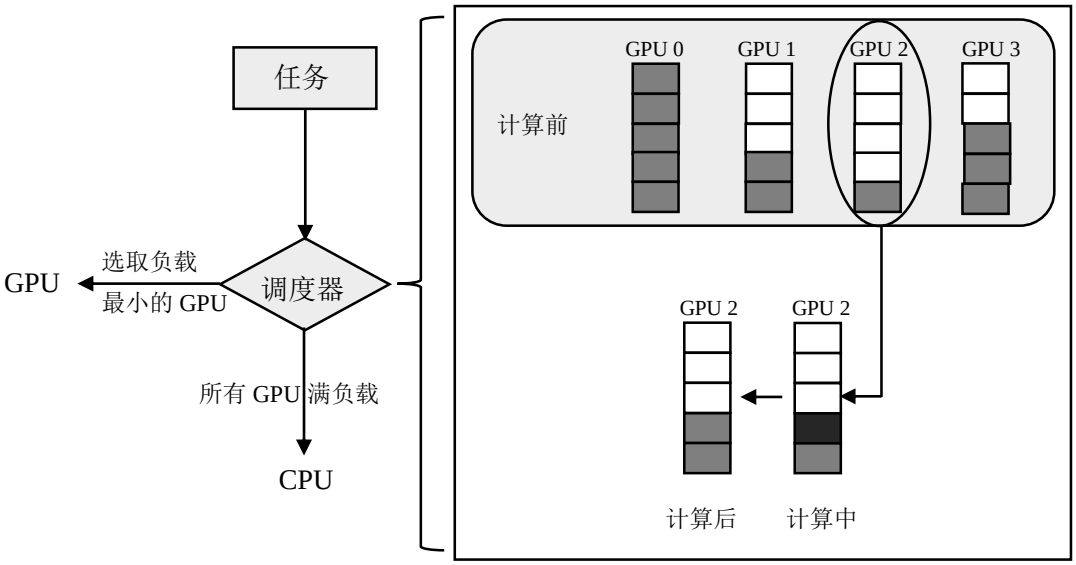


图 3-2 基于 GPU-CPU 的光谱计算的负载平衡调度器

负责动态负载平衡的基本数据结构是队列，每个 GPU 设备拥有一个私有的队列。任务队列的主要术语如下：

- 活动任务：运行在 GPU 上任务。
- 等待任务：即将运行在 GPU 上的任务。
- 负载：目前活动任务和等待任务的总和。
- 最大队列长度：任务负载的上界，处于满负载的 GPU 将不再接收任务。
- 历史任务数：任务队列到目前为止累积接收的任务数目和。

如图 3-2 所示，调度器负责维护所有任务队列，将任务分发到 CPU 和 GPU 上。首先，MPI 进程会请求调度器分发一个 GPU 资源给自己，用来运行积分计算。然后，调度器会从 GPU 队列中选择一个工作负载最小的 GPU 设备，如果当前不

止一个 GPU 设备工作负载处于最小值，那么拥有最小历史任务数的 GPU 设备将被调度器选择。如果调度器选中了一个 GPU 设备，那么 MPI 进程中当前的任务将在该选中的 GPU 设备上进行积分计算，并且此 GPU 设备在任务队列中的位置将一直处于占用状态，一直到此任务计算完成为止；如果调度器没有选中的 GPU 设备，那么说明所有的 GPU 设备的工作负载都达到了最大队列长度，那么该 MPI 进程将选择在 CPU 设备上进行计算，调用原始的 QAGS 积分算法[10]完成串行的积分计算。以图 3-2 为例，首先 MPI 进程产生了一个计算任务，此时它会请求调度器分配一个计算资源给自己，而调度器会先去访问 GPU 设备，查看所有 GPU 设备中工作负载最小的那一个，此时 GPU 0 处于满载状态，GPU 1 工作负载为 2，GPU 2 工作负载为 1，GPU 3 工作负载为 3，显然调度器会返回 GPU 2；随后 MPI 进程中的任务就分配给了 GPU 2 来进行计算，GPU 2 的工作负载增长一格，直到计算完成后，工作负载才会减少一格（当然如果在计算过程中又有新的任务分配给 GPU 2，那么他的工作负载会继续升格，一升一减是对应的）。

需要说明的是这里所说的最大队列长度依赖于 GPU 设备的计算能力和计算任务本身的特点，最大队列长度与计算性能的关系会在第四章中进行详细的说明，实际上调度器通过自动测试就可以找到最大队列长度。最开始调度器会通过增加最大队列长度的值寻找最合适的最大队列长度，一开始整体的计算性能会先随着最大队列长度的增加而提高，随着最大队列长度的值的不断增加，会从某一点开始，计算性能开始下降，那么此性能转折点的值就是最大队列长度的最优值。此外，任务调度以及 GPU 内部的并发模式是因 GPU 设备的不同而有所不同的，例如，应用级别的上下文切换在 Fermi 架构中是必须的，因此队列中的任务在此架构中是按照他们的提交顺序串行执行的，也就是说 Fermi 架构中每次只会有一个任务处于活动状态，而其他进入任务队列的任务必须排队等待；但 Kepler 架构的 GPU 设备情况却大不相同，Hyper-Q 技术可以允许最多 32 个来自多个 MPI 进程的计算任务在 Kepler GPU 上同时运行，这一特性可以使 GPU 的利用率显著提高。因此对于 Kepler 架构的 GPU 设备，活动任务数就不止一个了。

详细的调度算法在算法 1 中进行了描述。$\boldsymbol{L}$是积分区间的下界，$\boldsymbol{U}$是积分区间的上界，$\boldsymbol{N}$表示的是积分区间的个数，$\boldsymbol{f_{rrc}}$表示的是被积函数，$\boldsymbol{device}$表示的被选中执行积分计算的 GPU 设备。各个进程之间使用的是共享内存的方式来保存任务队列内的值，每次对任务队列中的值进行修改的时候必须使用原子操作。每次将 GPU 设备进行占用前需要调用 SCHE-ALLOC()函数，该函数的主要作用是从 GPU 设备中选出负载最小的那一个，如果多个设备负载处于最小，那么历史任务数目最小的设备将被选中，同时负载队列中相应位置的值做原子加操作，该位置的历史任务数也需要做加一操作，最后 SCHE-ALLOC()函数会返回该设备的

编号。但是如果当前所有的设备都处于满载状态，那么函数返回-1。计算完成后，进程会再调用 SCHE-FREE()函数，释放 GPU 设备，会对负载队列中对应位置的值做原子减操作。

算法 1 Scheduling

1: $for\ all\ process_i\ do$
2: $device = SCHE - ALLOC()$;
3: $if\ device \geq 0\ then$
4: $GPU - Integration(L, U, N, f_{rrc}, device)$;
5: $SCHE - FREE(device)$;
6: $else$
7: $CPU - Integration(L, U, N, f_{rrc}, err_{abs}, err_{rel})$;
8: $end\ if$
9: $end\ for$

算法 Subroutines in **Scheduling**

1: $l_i\ is\ the\ load\ of\ device\ i$;
2: $h_i\ is\ the\ history\ tasks\ count\ of\ device\ i$;
3: $Both\ l_i\ and\ h_i\ are\ global\ variable$;

4: $function\ SCHE - ALLOC()$
5: $l_i, h_i \leftarrow shmat()$;
6: $l_{min} \leftarrow l_0$;
7: $h_{min} \leftarrow h_0$;
8: $for\ all\ i < device_{num}\ do$
9: $l_{min} \leftarrow min(l_i, l_{min})$;
10: $if\ l_i == l_{min}\ then$
11: $deivce \leftarrow min(h_i, h_{min})$;
12: $end\ if$
13: $end\ for$
14: $if\ l_{min} < l_{max}\ then$
15: $atomic\ operation${
16: $l_{device} + +$;
17: $h_{device} + +$;
18: }

```
19:         return device;
20:     else
21:         return −1;
22:     end if
23: end function

24: function SCHE − FREE(device)
25:     l_device ← shmat();
26:     atomic operation{
27:         l_device −−;
28:     }
29: end function
```

3.3 基于 MPI 的数值积分求解器的设计

针对于光谱计算，本研究首先进行了基于 CPU 的 MPI 版本的并行优化方案。2.1.3 中提到了光谱计算的典型空间结构，它是一个三维的参数空间：温度、密度和时间。参数空间一般是由天文模拟的结果获得，或者仅仅是一个用于测试的简单的配置文件。对于参数空间中的每个格点，它表示了特定的温度、密度和时间，空间网格中的各个点之间是没有信息交换的，串行的计算方式主要是对于每个网格点进行的。因此可以根据光谱计算的这一空间特点，将纯 CPU 上的串行计算优化成基于 MPI 的并行计算。

下图 3-3 描述了利用 MPI 实现的并行优化方案。首先 MPI 初始化环境，然后从配置文件中读取配置参数，将光谱计算的环境变量初始化后，根据 MPI 进程数以及空间网格点的数量进行任务划分；当每个 MPI 进程获得了计算任务后，开始串行版本中 RRC 的计算，此处依旧使用的是串行版本中的 QAGS[10]算法。对于每个计算进程，当一个网格点的计算结束后，将计算结果保存到输出文件中。由于每个进程一次只能计算一个网格点，因此需要循环若干次才能将所有计算完成。当所有计算任务都完成后，MPI 进程销毁，输出计算结果。

3.4 基于 GPU 的数值积分求解器的设计

由 GPU 的编程模型可知，GPU 并不能脱离 CPU 独立地完成工作，GPU 只

是一个协处理器，必须与 CPU 一起，CPU 负责总体的逻辑控制，而 GPU 负责并行计算，这样才可完成计算任务[25]。运行在 GPU 上的 CUDA 并行计算函数称为内核函数（kernel），内核函数并不能独立完成计算，它只是整个 CUDA 程序中并行执行计算的那部分。CPU 负载初始化各种变量以及程序开头和结尾的串行工作，其中包括内核函数启动之前所需要的数据，以及 GPU 设备的初始化工作。CPU 启动了内核函数后，内核函数才开始并行地执行计算任务。基于 3.3 节中 MPI 版本的并行优化，图 3-4 描述了基于 GPU 的数值积分求解器的结构以及执行流程，由于光谱计算中数值积分的逻辑复杂度与任务粒度的划分密切相关，所以对于不同的任务粒度，CPU 和 GPU 上的工作复杂度也不同，此时就需要在主存和显存中都预留一定的内存空间，以存放复杂计算过程中的中间结构。

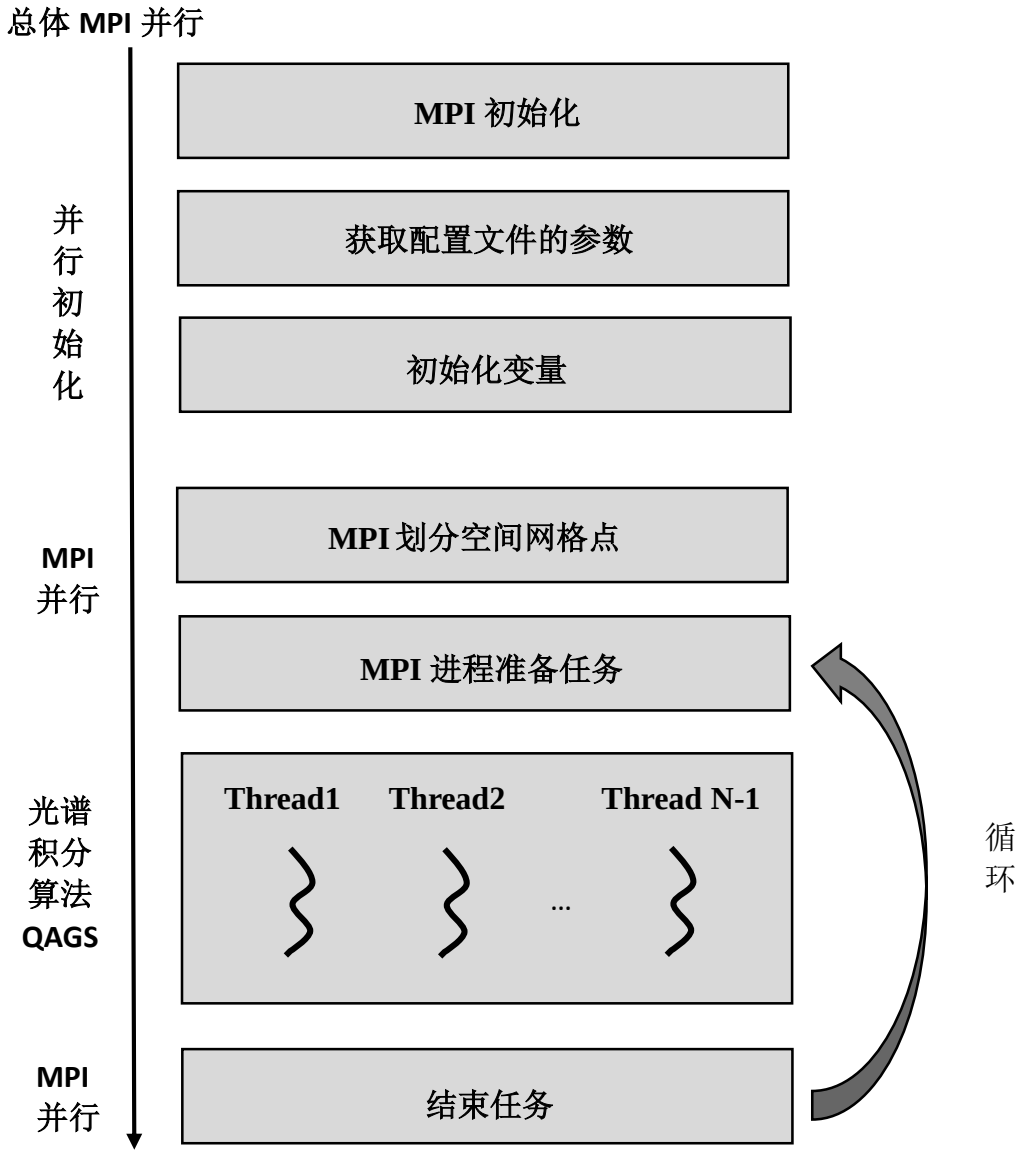


图 3-3 MPI 求解数值积分的算法结构图

CUDA 并行计算架构有多个不同类型的内存空间[26]，按照访问速度排列如下：

1. 寄存器（register）是 GPU 片上的高速缓存， 执行单元可以以极低的延迟访问寄存器。寄存器的基本单元是寄存器文件，每个寄存器文件大小为32bit。对于每个线程，局部存储器也是私有的。如果寄存器被消耗完。数据将被存储在局部存储器中。如果每个线程使用了过多的寄存器，或声明了大型结构体或数据，或者编译器无法确定数据的大小，线程的私有数据就有可能被分配到 local memory 中，一个线程的输入和中间变量将被保存在寄存器或者是局部存储器中。局部存储器中的数据被保存在显存中，而不是片上的寄存器或者缓存中，因此对 local memory 的访问速度很慢。

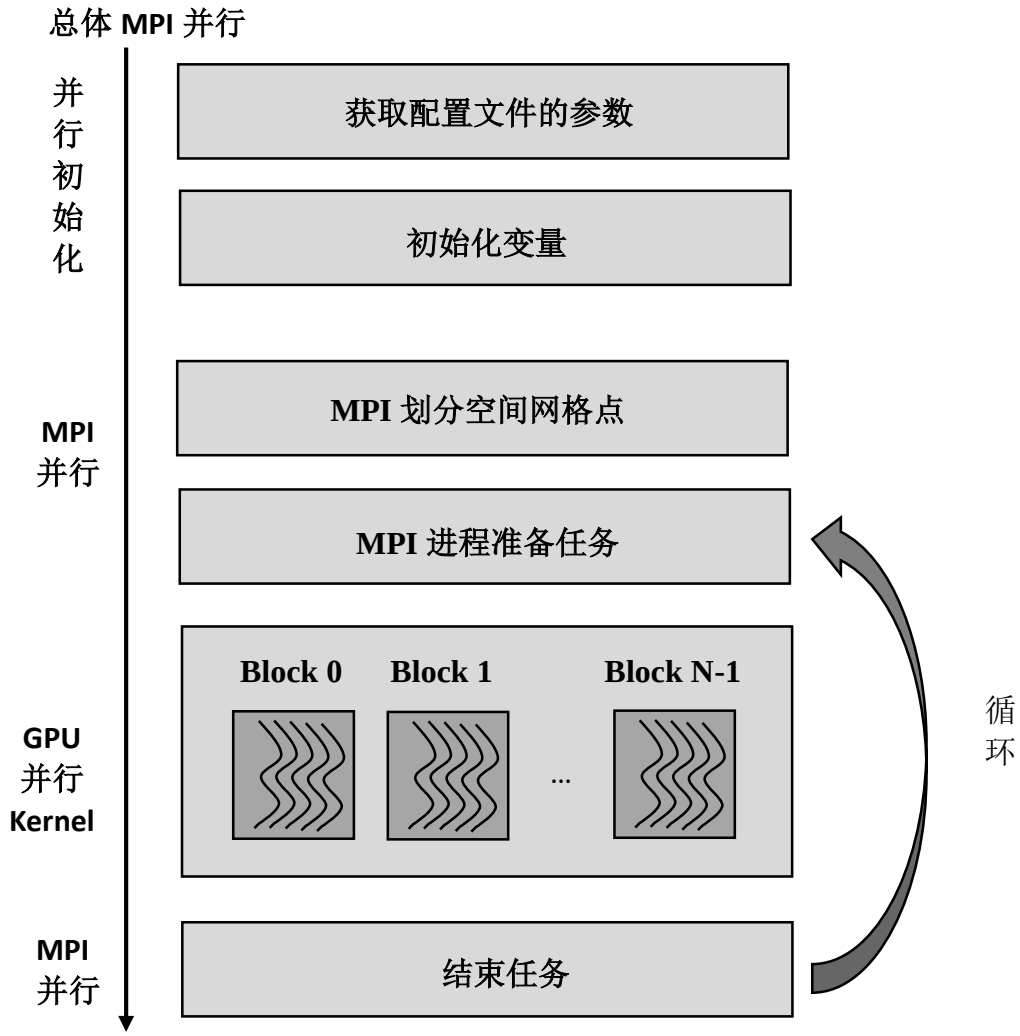


图 3-4 GPU 求解数值积分的算法结构图

2. 共享内存（share memory），高速，由于共享内存是片上内存，因此相比于本地内存和全局内存，共享内存具有高带宽，低延迟的特性。为了获得

高带宽，共享内存被划分为相等的存储模块，称之为 banks，这些存储模块可以同时访问，每次读内存或者写内存，如果需要 n 个地址，那么这 n 个地址就会分布在这 n 个存储模块上。当两个地址访问同一个存储模块的时候，就会出现内存访问冲突，这个时候就只能串行地进行内存访问了。共享内存是一块可以被同一 block 中的所有线程访问的可读存储器。使用关键字 share 添加到变量的声明中，这将使这个变量驻留在共享内存中。CUDA C 编译器对共享内存中的变量和普通变量将分别采取不同的处理方式。对于在 GPU 上启动的每个线程块，CUDA C 编译器都将创建该变量的一个副本，线程块中的每一个线程都共享这块内存，但这个线程却无法看到也不能修改其他线程块的变量的副本，这就实现了一种非常好的方式，使得一个线程块中的多个线程能够在计算上进行通信和协作，而且，共享内存缓冲区驻留在物理 GPU 上，而不是驻留在 GPU 之外的系统内存中。在 Fermi 架构中，共享内存的容量是 64k，CUDA3.5 之后共享内存的容量大于等于 64k。

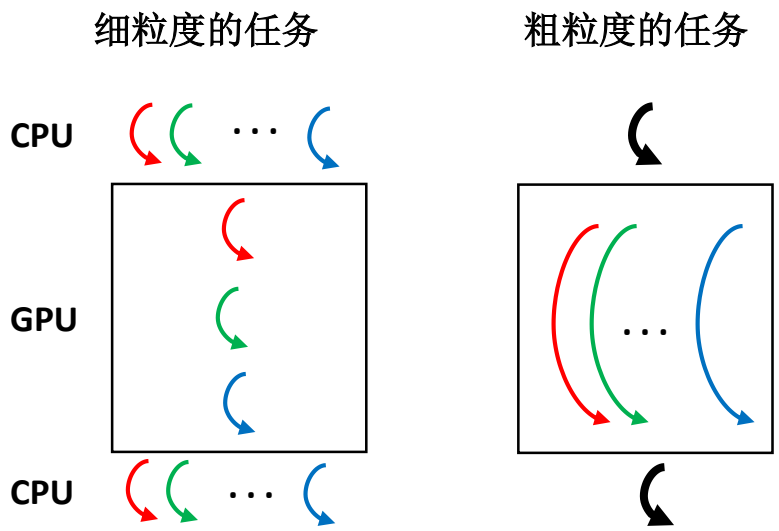
3. 常量内存（constant memory），低速，把变量的访问限制为只读。在接受了这种限制之后，与全局内存中读数据相比，从常量内存中读取相同的数据可以节约内存的带宽，主要有两个原因：
   - 对常量内存的单次读操作可以广播到其他的“邻近”线程，这将节约 15 次读取操作。
   - 常量内存的数据缓存起来，因此对相同地址的连续读取操作将不会产生额外的内存通信量。

   “邻近”是指半个 warp 中的线程。当处理常量内存时。NVIDIA 硬件将把单次内存读取操作广播到每个半线程束。每个半线程束中包含了 16 个线程，即线程束中数量的一半。如果在半线程束中的每一个线程访问相同的常量内存地址。那么 GPU 只会发生一次读操作事件并在随后将数据广播到每个线程。如果从常量内存中读取大量的数据，那么这种方式产生的内存流量只是全局内存时的 1/16，然而，当使用常量内存时也可能产生负面影响。如果半线程束的所有 16 个线程需要访问常量内存中不同的数据，那么这个 16 次读取操作会被串行化，从而需要 16 倍的时间来发出请求。但如果从全局内存中读取，那么这些请求会同时发出。在这种情况下，从常量内存读取就慢于从全局内存中读取。
4. 本地内存（local memory）驻留在设备内存中，所以本地内存的访问具有高延迟、低带宽的特性。本地内存并不是一类独立的物理内存模型，而是全局内存的一个抽象，可以理解为仅限于线程内部访问的全局内存，但可

借助缓存提高性能。当寄存器被使用完毕后，数据将被存储在本地内存中；当每个线程中使用过多的寄存器，或声明了大型的结构体或者数组以及无法确定大小的数组，线程的私有数据就会被分配到本地内存中；线程的输入和中间的变量等也都保存在本地内存中。

5. 全局内存（global memory），低速，容量大，可以和主存进行数据交换。位于显存（占据了大部分的显存），GPU，CPU，都可以进行读取访问。整个网格中的任意线程都能读写全局存储器的任意位置。在目前的架构中，全局存储器没有缓存。

基于 GPU 的算法的两种通用模式如下图 3-5 中的(a)(b)所示：



(a) 重点移植模型　　(b) 整体移植模型

图 3-5 光谱计算在 GPU 上的两种加速方法

a) 重点移植：为了充分发挥 GPU 在数值积分计算方面的优势，仅将比较耗时的数值积分计算转移到 GPU 上。而算法中的主要控制逻辑仍然由 CPU 完成。目前 GPU 上已经有了不少可供选择的数值积分工具库，QUADPACK[10]、MKL[11]、CUBA[12]、GSL[13]等。这些方法的优势在于实现简单，只须将现有函数库中的积分计算抽取出来后委托给 GPU 即可，而不必了解 GPU 的积分计算的编程细节，这些工具库适合于求解高维的、积分区间很大的积分计算，而对于光谱计算恰好不适用。在方式(a)中，多个 CPU 进程可同时向 GPU 提交任务，即多个进程共享一个 GPU，但任务在 GPU 中处理的顺序则由 GPU 的硬件结构决定，程序本身无法控制。NVIDIA 的 Kepler 架构借助 Hyper-Q 技术最大可支持 32 个进程同时连接到 GPU，从而能够获得较高的利用率；而较早一些的 Fermi 架构就要求必须进行上下文的交换，即多个进程提交的

任务在 GPU 中只能串行的执行。

b) 整体移植：为了充分发挥 GPU 在高并发方面的优势，可以直接将完整数值积分的求解算法移植到 GPU 上。此时 GPU 可以看成是一个向量处理器，可以同时对多个数值积分计算进行求解。这两种方式中，任务的粒度是不同的。在方式(a)中，一个任务仅仅是一次积分计算，此运算是 GPU 多个线程协同完成的；而在方式(b)中，一个任务包含多个独立的积分求解，GPU 的每个线程都是一个独立的数值积分求解器，运行过程中不需要协作。还有在方式(b)中，多个 CPU 也可以共享一个 GPU。

3.5 基于 GPU 的数值积分算法的实现

为了处理大量的小积分区间的 RRC 积分计算，并且减少主机和 GPU 设备之间频繁的数据交换，本文中定义了一种粗粒度的任务，每个粗粒度的任务中包含大约几千万次 RRC 积分计算。正如第一章中介绍的，每个网格点中包含了 496 个离子，由于每个离子有不同数量的能级，因此以一个能级或者一个离子作为一个任务的粒度都是可行的（每个离子可以包含数百个能级）。如果任务粒度是能级级别的，那么通常一个能级中包含 5 万个积分区间，而这样多的积分计算在 GPU 上进行仍然算是一种细粒度的并行，因为相比于 GPU 与 CPU 之间的数据 IO，GPU 上 5 万次的数值积分计算仍然只占很小的一部分运行时间。如果任务粒度是一个离子，那么对于一个空间网格点来说，会有 496 个这样的任务，实验数据也表明这样的任务粒度在异构并行系统下刚好可以获得非常可观的性能提升。值得强调的是任务粒度的大小不仅与 GPU 设备的性能有关，同时也与应用本身的特点密切相关，对于光谱计算，最佳的任务粒度是离子级别的，因为如果再将任务粒度扩大，例如元素级别的（每个元素包含若干颗离子），那么 GPU 核中的数值积分算法的逻辑结构会变得很复杂，复杂的逻辑结构本就不适合运行在 GPU 内部，所以不仅不会提高整个计算的性能，反而可能使性能有所下降。

算法 2 GPU-Integration($L, U, N, f_{rrc}, device$)

1: $n \leftarrow N/Thread\_num$;

2: $size \leftarrow (L-U)/N$;

3: $j \leftarrow threadIdx.x + blockIdx.x * blockDim.x$;

4: $while\ i < n+1\ do$

5: $\quad r_i \leftarrow L + (n*j+i) * size$;

6: $end\ while$

7: $\boldsymbol{for\ all\ i < n\ do}$

8: $\boldsymbol{left \leftarrow r_i}$;

9: $\boldsymbol{right \leftarrow r_{i+1}}$;

10: $\boldsymbol{emi_{n*j+i} \leftarrow Simpson(f_{rrc}, left, right)}$;

11: $\boldsymbol{end\ for}$

$\triangleright$ $\boldsymbol{emi}$ $\boldsymbol{:\ a\ N\ size\ array\ containing\ emission\ power\ of\ all\ energy\ bins}$;

算法 2 是 RRC 积分的伪代码，L 表示积分区间的下界，U 表示积分区间的上界，N 表示积分区间的个数，f_{rrc}表示的是被积函数，$device$ 表示的是执行计算任务的 GPU 设备，首先 GPU 设备会根据线程数量和积分区间的总个数N，获得每个线程上需要计算的积分计算的计算量；然后根据积分上界U和积分下界L 获得每个积分区间的大小$\boldsymbol{size}$；在根据 GPU 内部的块索引和块内的线程索引的编号，获得当前的线程编号；第四步就可获得每个积分区间的具体的左边界和右边界了；利用左右边界值即可调用数值积分算法进行积分计算了，对于每个离子来说，每个能级在每次积分过程中的发射率结果会在存在 GPU 中，直到该离子的所有能级中的积分计算全部完成为止，最后才将此结果由 GPU 设备传递到 CPU 主机上。我们首先使用的是数值积分算法是最普通的梯形算法如公式（3-1）所示，

$$\int_a^b f(x)dx \approx (b-a)*\frac{f(a)+f(b)}{2} \tag{3-1}$$

$$\int_a^b f(x)dx \approx \frac{b-a}{6}\left[f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right] \tag{3-2}$$

梯形算法简单、易懂，且容易实现，非常适合于最初的算法验证工作，为了保证计算的准确性，复合辛普森（Simpson）算法[27]被选作 GPU 上的最终的数值积分算法。辛普森算法如公式（3-2）所示，辛普森算法是三点式的牛顿-科特斯（Newton-Cotes）算法，该算法是由数学家汤姆斯•辛普森（Thomas Simpson）提出的，牛顿-科特斯公式的求积余项公式表明，求积节点 n 越大，对应的求积公式精度越高，但由于牛顿-科特斯公式在 n>8 时数值不稳定，因此不能用增加求积节点数的方法来提高计算精度。实用中常将求积区间[a，b]分成若干个小区间，然后在每个小区间上采用数值稳定的牛顿-科特斯公式求小区间上的定积分，最后把所有小区间上的计算结果相加作为原定积分的近似值。采用这种方法构造的求积公式就称为复合求积公式。复合求积公式具有计算简单且可以任意逼近所求定积分值的特点，这是牛顿-科特斯公式一般做不到的。

3.6 多 GPU-CPU 混合异构平台下的光谱计算的整体实现

本文中的实现都是以 APEC[7]为原型的，之所以选择 APEC 是因为：1）APEC 是所有光谱计算工具中最典型的 RRC 计算工具包。2）APEC 使用广泛，并且能够与原子数据库 AtomDB 相结合，这样可以获得真实有效的实验数据，实验结果的可行度会更高。3）APEC 结构小巧，无论是使用还是优化都更容易。整个方法的编程语言为 C 和 CUDA C，使用 MPI 进行底层的并行封装，使用 MPI 封装不仅方便不同节点之间的通讯，同时可以使原始的 APEC 能够在多 CPU-GPU 的并行体系架构上进行计算，同时方便进程与 GPU 进行通讯。为了减少代码的改动，因此，CPU 上实现的数值积分算法依旧是 APEC 原有的经典 QAGS 程序[10]。基于 GPU 的加速数值积分求解器拥有一个通用的接口与主程序相连，这样基于 CPU 的数值积分算法和基于 GPU 的加速数值积分算法都可以在需要的时候连接到主程序中。在目前的实现中，我们在 GPU 上不仅实现了 Simpson 积分算法，为了保证本文方法的可迁移性以及可扩展性，本文还使用了 Romberg 积分算法，Romberg 算法相比于 Simpson 算法计算复杂度相同，但是计算结果的准确度却大大提高了。

为了保证可插拔和模块化的设计理念，我们的实现只是在原始的 APEC 代码上进行少量的改动，对于使用者来说，只需要做少许的改动便可应用本文提出的优化方法。此外，本文中的实现方法是可以自动检测系统中 GPU 设备的数量的，即使实验环境并没有 GPU 设备也没有关系，那么所有的任务会在 CPU 上完成，而并不需要更改任何的代码。为了避免在用户-服务器体系架构中产生额外的开销，本地的任务调度器通过共享内存与其他 MPI 进程以及 GPU 进行通讯。共享内存方法中主要使用了两个数组，一个数组表示的是每个 GPU 设备上当前的负载数量，另一个数组保存了每个 GPU 设备历史任务数量。当一个 GPU 设备被调度器选中后，负载数组中对应该 GPU 设备位置的数据需要做原子性的加一操作；同样，当计算任务完成后，选中的 GPU 设备的计算资源被释放，那么负载数组中对应该 GPU 设备的位置的数据需要做原子性的减一操作，而在历史任务数数值中对应该 GPU 设备的位置的数据需要做原子性的加一操作。

对于任务粒度的选择，光谱计算本身可进行不同粒度的任务划分，以单个元素为一个计算任务，或以一个离子作为一个计算任务，或者以一个能级作为一个计算任务。任务粒度的选择本身就与计算任务的特性有关系，因此需要通过实验来确定最佳的任务粒度。考虑到 GPU 计算的特点，需要尽量减少主机和 GPU 设备之间的数据 IO，让计算任务在 GPU 中执行时间越久越好，但同时要结合 kernel 内部的计算算法，避免过多的逻辑运算。基于以上的考虑,本文设计了分别以离子

为单位的计算粒度和以能级为单位的计算粒度，来评估算法的性能。

第四章　实验结果及性能分析

本研究的实验环境包含两个英特尔 Xeon CPU E5-2640(2.5GHz)，其中，每个 CPU 含有 12 个核；以及 4 个英伟达 Tesla C2075 GPU 显卡。Tesla C2075 显卡是基于 Fermi 架构的显卡，每个卡拥有 6GB GDDR5 的板上内存。448 个流处理器核（1.5GHz），每个 GPU 可提供高达 515 Gigaflops 双精度计算峰值。这些 GPU 设备与主机之间通过 PCI Express 2.0 实现互联。

测试用例的参数空间由 24 个网格点组成，所有的测试都运行在 24 个 MPI 进程上，这 24 个 MPI 进程共用一个物理节点。测试中使用更多的网格点并没有太大意义，因为当所有的网格点集中在某个小区间的时候，每个网格点上的计算基本是一样的，因此即使每个进程上的网格点的计算量增加，对验证算法的性能并没有实质性的作用。另外每个网格点会包含 496 个子计算任务，这样的任务量，已经完全可以验证本文中提出的负载平衡方法的有效性。

实验中，所有的 GPU 设备上运行的数值积分算法都是复合的辛普森算法，除了表 4-1 和图 4-4，使用 1 个 GPU 进行的测试用 1 GPU 表示，同样，使用两个 GPU 设备进行的测试用 2 GPUs 表示，三个 GPU 和四个 GPU 同上。完全使用 CPU 进行的测试，也就是串行的测试和 MPI 测试，在我们实验结果图中并没有与 GPU 测试的结果直接对比给出，因为性能差别较大，CPU 曲线几乎与坐标横轴重叠，24 个核的 MPI 并行版本的速度与原串行版本的速度相比只提高了 13.5 倍。

4.1 加速比

图 4-1 给出了两种不同的任务粒度下的加速比，粗粒度的任务映射到光谱计算中对应于一个元素的一个离子，标记为 Ion；对应的，细粒度的任务映射到光谱计算中对应的是一个离子的一个能级，标记为 Level。我们将 CPU/GPU 混合异构系统下的计算任务的总的执行时间与原始的串行 APEC 方法的计算时间对比，获得加速比的值。图 4-1 中显示了，随着 GPU 适量的增加，整体的计算的加速比在提高，并且此时 Level 级别任务粒度的实现的加速比已经很好了，当使用 3 个 GPU 设备时，整体的计算比就可达到 150 倍。同时，随着 GPU 设备数的增加，以 Ion 为粒度的 GPU 版本的加速比的可观性更好，同样使用 3 个 GPU 设备时，整个计算的加速比可达 300 倍以上。两种不同任务粒度，速度几乎相差一

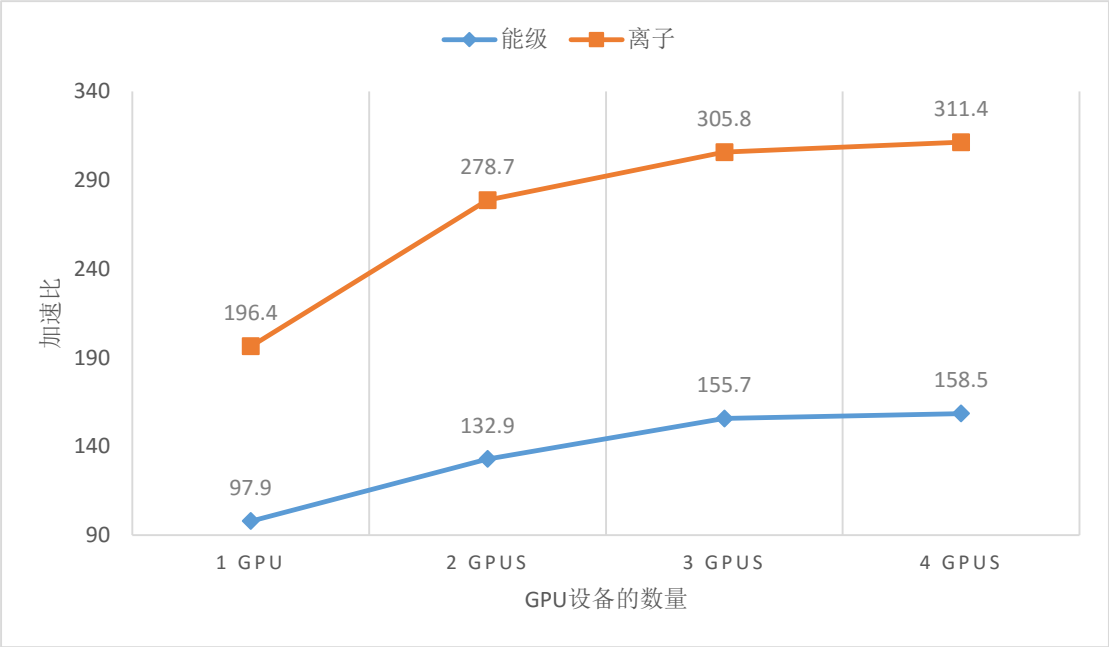


图 4-1 不同任务粒度的加速比

矩形标记的橘黄色线表示的是以一个元素的离子为任务粒度的映射关系，即粗粒度；
菱形标记的蓝色线表示以一个离子的一个能级作为任务粒度的映射关系，即细粒度；

倍，其实 Level 级别的任务粒度的加速比之所以没有 Ion 级别任务粒度的高，是因为相对于 Ion 级别的任务粒度，在 Level 级别任务粒度的测试中 GPU 设备与主机之间会产生频繁的内存数据拷贝；而在 Ion 级别的任务粒度的测试中，是先将一个离子的每个能级的计算结果暂时保存在 GPU 中，每次计算最后都会有一个累加计算的操作，当一个离子的所有能级的计算都完成后，才将此累加结果回传给 CPU，此过程中大大减少了数据输入输出的开销。因此对于大量的小任务来说，选择一种合适的任务粒度是非常重要的，任务粒度的选择可以通过不同任务粒度的对比测试，对比测试可以很好地找出最合适的任务粒度，从而使 GPU 的利用率达到最高。

除了任务调度的粒度以外，最大队列长度是另一个影响总的计算性能和负载平衡结果的重要的因素。图 4-2 给出了对于 24 个网格点的计算任务，当最大队列长度与 GPU 设备数进行不同数量的组合时，每个组合的总计算时间。在此实验中，需要进行调度的总的任务数是 24（points）*496（ions），总共超过一万条任务。正如图 4-2 所示，随着最大队列长度的增加，总的计算时间在不断减少，对于所有测试用例，当最大队列长度等于 10~12 的时候，计算性能达到了峰值，随后，再增加最大队列长度，计算性能开始缓慢地下降。出现这种现象的原因可以由图 4-3 解释，随着最大队列长度的增加，越来越多的任务被调度到 GPU 上计算，由于 GPU 的计算能力远远高于 CPU，因此起初整体的计算性能会提高很多。但是随着 GPU 上的任务负载的不断提高，GPU 计算的能力还是有限的，当

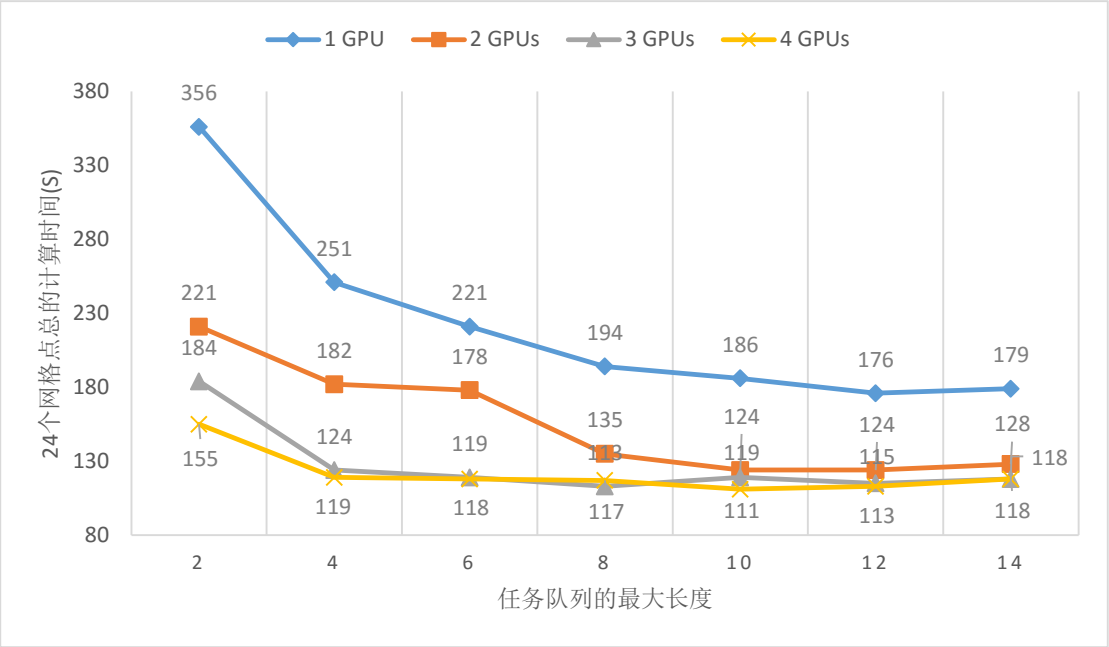


图 4-2 不同任务队列最大长度下 24 个网格点总的计算时间

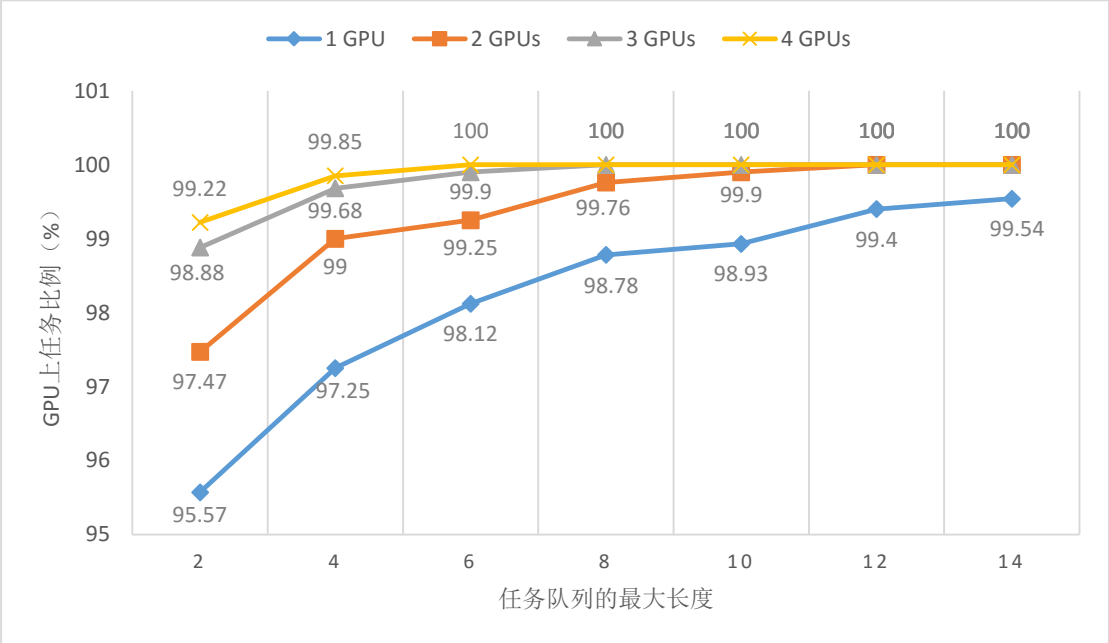


图 4-3 不同任务队列长度下 GPU 上分配的任务比例

任务比例 = GPU 上完成的任务数/总的任务数

任务负载超过 GPU 的最大计算能力时，会有越来越多的任务处于等待状态，而与此同时 CPU 却一直处于闲置状态，CPU 必须等待 GPU 上的所有任务完成才能进行下一次的循环调度，这样一来整体的计算性能必然会下降。值得指出的是，随着最大队列长度的增加会出现这样一种现象：当最大队列长度大于 6 的时候，2 个 GPU 设备和 3 个 GPU 设备上任务总的计算时间相差的越来越小；3 个 GPU

设备和 4 个 GPU 设备总的计算时间在整个过程中都基本相同。这种现象的出现表明，对于光谱计算来说，在目前的实验环境下，用 2 个 GPU 设备来处理来自 24 个 CPU 核的任务就已经足够了。所以，在此情况下，想通过增加 GPU 的数量来提高计算的性能是不可行的。这也说明，在很多情况下，并不是一味地增加计算资源就可以提高性能，具体问题具体分析，对于不同的问题，任务粒度的选择、计算资源的利用方式、CPU 和 GPU 的数量配比等，都是决定计算性能的关键因素。

4.2 负载平衡

图 4-3 表示，在以上的使用复合辛普森算法的实验中，即使最大队列长度是 2，也有超过 95%的计算任务被分配到了 GPU 上来完成，这样的结果依赖于问题域和硬件环境。在光谱计算中，大多数情况下，复合辛普森算法可以提供足够的准确性，它将一个积分区间分为 64 个相等的小区间，然后再做积分计算，在我们的实验中该计算强度对于 GPU 来说算是中等的。因此，大多数的任务都被分配到了 GPU 上，所以对于 GPU 来说还有足够的能力承受强度更大的计算。由于一些其他的应用可能需要更高的准确性，为了进一步验证本文中提出的负载平衡调度方法的有效性和适应性，我们又进行了更高计算精度的测试，这些测试使用的是 Romberg 数值积分算法[27]。与复合辛普森算法相比，Romberg 算法能够获得更高的计算准确性，同时又不会增加额外的计算复杂度。

Romberg 算法的计算准确性是由数值 k 决定的，如公式 4-1 所示，k 表示的是二分的次数，随着 k 值的增加，准确性也会越高，单个任务总的计算量也会随着 k 值的增加以 2 的指数的量级不断地增长。图 4-4 给出了不同 k 值的情况下 GPU 设备 0 上的负载分布，这里本文只给出了一个 GPU 设备上的负载情况，横坐标表示的是 GPU 设备上的任务数为[0，6]，纵坐标表示的任务负载的时间百分比，此处做了归一化，不同颜色的柱子表示 k 的不同取值，分别为 7，9，11，13。在图 4-4 中，每个柱子表示在一次测试完整执行的过程中某个 GPU 上的负载的百度分情况，在次测试中最大队列长队固定为 6。例如，图表中 GPU 设备 0 上的任务负载数目为 6 的时候，黄色的柱子表示单个任务的计算复杂度为2^{13}个 RRC 计算的时候，在整体的计算过程中有 44.04%的时间，GPU 设备 0 的负载数目为 6。

$$T_m^{(k)} = \frac{4^m}{4^m - 1} T_{m-1}^{(k+1)} - \frac{1}{4^m - 1} T_{m-1}^{(k)} \tag{4-1}$$

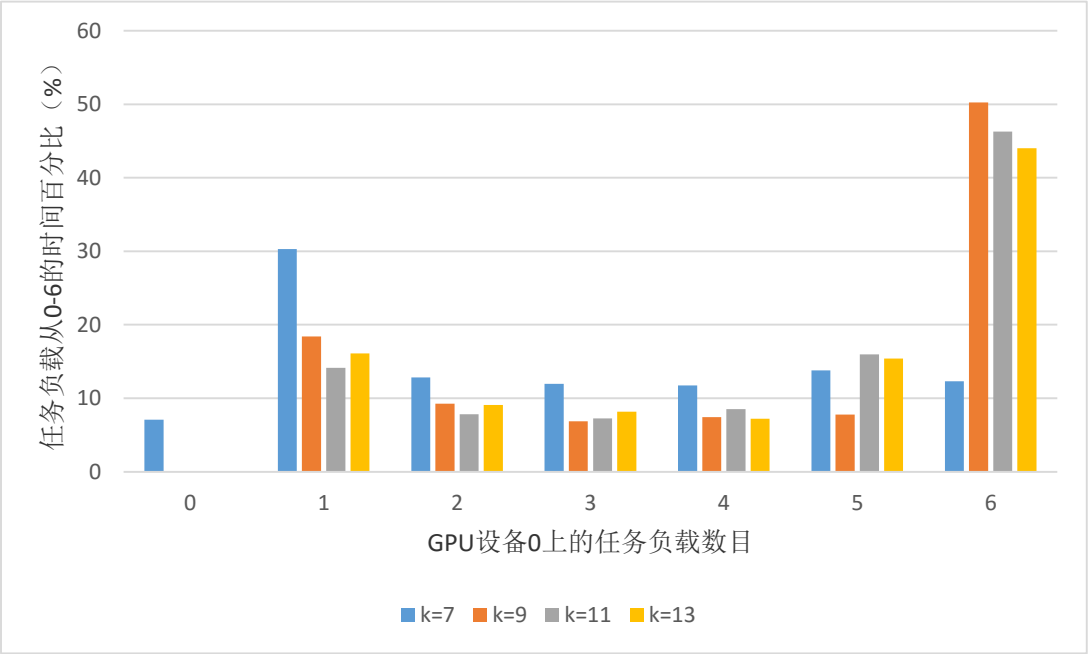


图 4-4 在不同的计算复杂度下，端对端的执行过程中 GPU 上的任务负载情况；
此实验中 GPU 设备数量为 2，最大任务队列数目为 6

表 4-1 显示了 GPU 与 CPU 之间的任务分布的统计结果。给出了在不同 k 值的条件下，GPU 上的任务数，任务数占总任务的百分比，以及负载数大于等于 3 的百分比。根据表 4-1 所示，当单个任务的计算复杂度是2^7的时候，GPU 会承担超过 98%的计算负载，并且在总的计算时间内，有 37%的时间，GPU 上的负载数高于 3。但是随着单个任务计算复杂度的提高，GPU 计算单个任务的时间显著增加，从而增加 GPU 上的负载强度，因此越来越多的任务被分配到了 CPU 上完成。当单个任务的计算复杂度达到2^{13}的时候，GPU 上的任务比例只有 40.9%，而 CPU 上的任务计算比例有 60%。这组实验数据显示，单个任务所包含的计算量会影响整个计算的负载平衡。因此，在 CPU 和 GPU 混合异构计算体系下，为了使端对端模式下的计算实现最大的性能，在一个特定的计算应用中，选择一个合适的任务划分策略是相当重要的。

表 4-1 不同计算复杂度下 GPU 设备上的任务比例

每个任务的计算量	GPU 设备上的任务数	GPU 设备上的任务比例	GPU 设备负载数≥ **3**
$\mathbf{2^7}$	6674	98.26%	37.85%
$\mathbf{2^9}$	6344	93.40%	65.46%
$\mathbf{2^{11}}$	4518	66.52%	70.76%
$\mathbf{2^{13}}$	2779	40.92%	66.64%

4.3 准确性

以上实验保证了本文提出的研究方法的有效性以及负载平衡调度的可行性，为了验证本研究方法的准确性，我们进行了一组对比，在计算参数空间相同的前提下，将 CPU-GPU 混合异构体系下运行的 APEC 方法所计算的结果与原始的串行的 APEC 方法的计算结果相对比，图 4-5 显示了使用原始的串行 APEC 方法所计算的在一定波长区间内的归一化通量值，图 4-6 显示的是使用异构并行的 APEC 方法计算所获得的一定波长区间内的归一化通量值。这两幅图用肉眼看是非常相似，几乎是一模一样的。因此为了更加准确地反映计算结果的准确性，我们又给出了量化的误差结果，图 4-7 表示的是这两种方法之间的数值误差的分布曲线图。根据这个曲线图可以看出，相对误差值分布在[-0.0003%，0.0033%]区间内，有 99%的相对误差分布在[0%，0.0005%]的区间内。误差分析结果验证了本文所提出的方法不经可以有效地进行光谱计算，而且该方法的计算结果没有任何明显的精度损失。

4.4 实验结论

以上三个实验分别从计算的性能提升情况，负载平衡调度算法的可行性，以及计算的准确度三个方面对研究方法进行了验证和分析。首先利用混合异构的系统架构，我们分别给出了以离子为一个计算级别的任务粒度和以能级为一个计算级别的任务粒度分别相对于传统的串行 APEC 方法的加速比，两者的加速比分别是 300 和 150，此外当 GPU 上的最大任务量取不同数值的时候，计算性能也有很大差别，这两组实验充分说明任务粒度的选择以及最高任务量的确定对于计算的加速比有非常重要的影响。同时，通过 GPU 上不同的最大任务量以及 Romberg 算法我们得出，单个任务的计算量越高，GPU 内部计算越耗时，此时分到 GPU 上的任务量也随之减少，因此 CPU 和 GPU 之间负载平衡的控制与单个任务的计算量密切相关。光谱计算整体的计算性能是验证本文方法的一个方面，而另一个方面就是保证计算的准确性，这也是验证本文方法可行的一个非常重要的方面。我们利用光谱计算的结果给出了传统串行的 APEC 方法和异构并行的 APEC 方法归一化的通量值，同时给出了量化的误差分析曲线，该曲线表明，99%的相对误差分布在[0%，0.0005%]的区间内，此数据也充分说明本文方法的准确性。以上三个方面充分说明了本文提出的 GPU 加速算法和负载平衡调度算法的正确性和可行性。

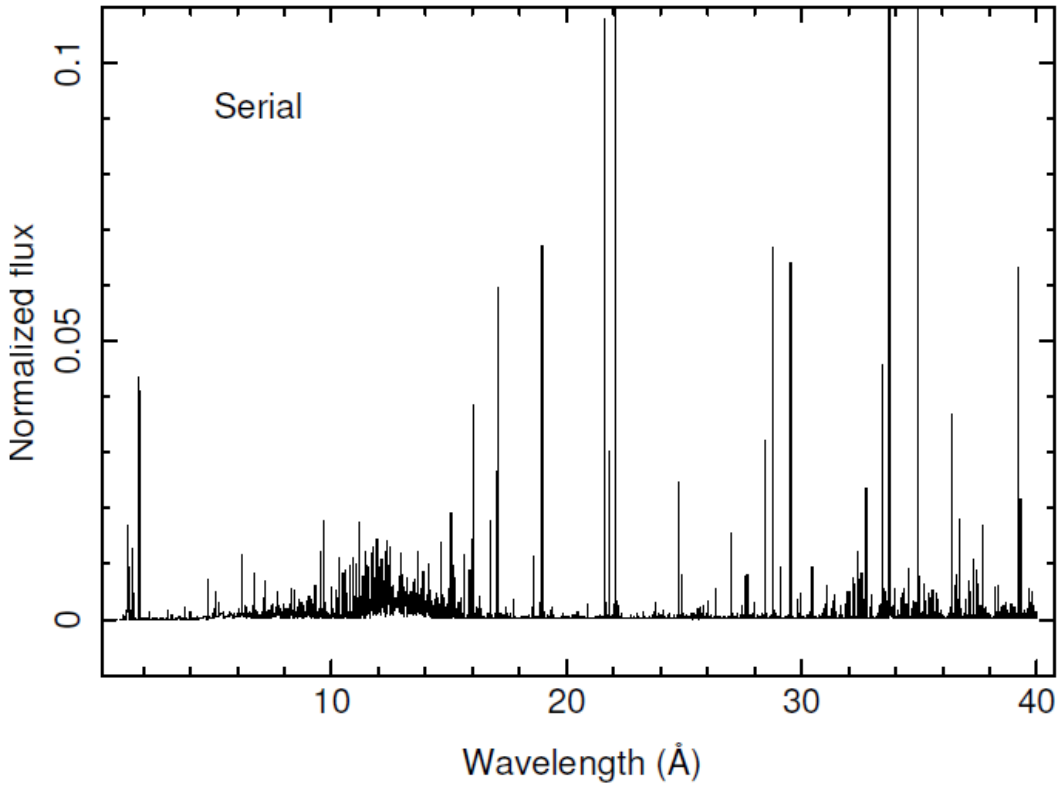


图 4-5 串行方法 APEC 计算的辐射率

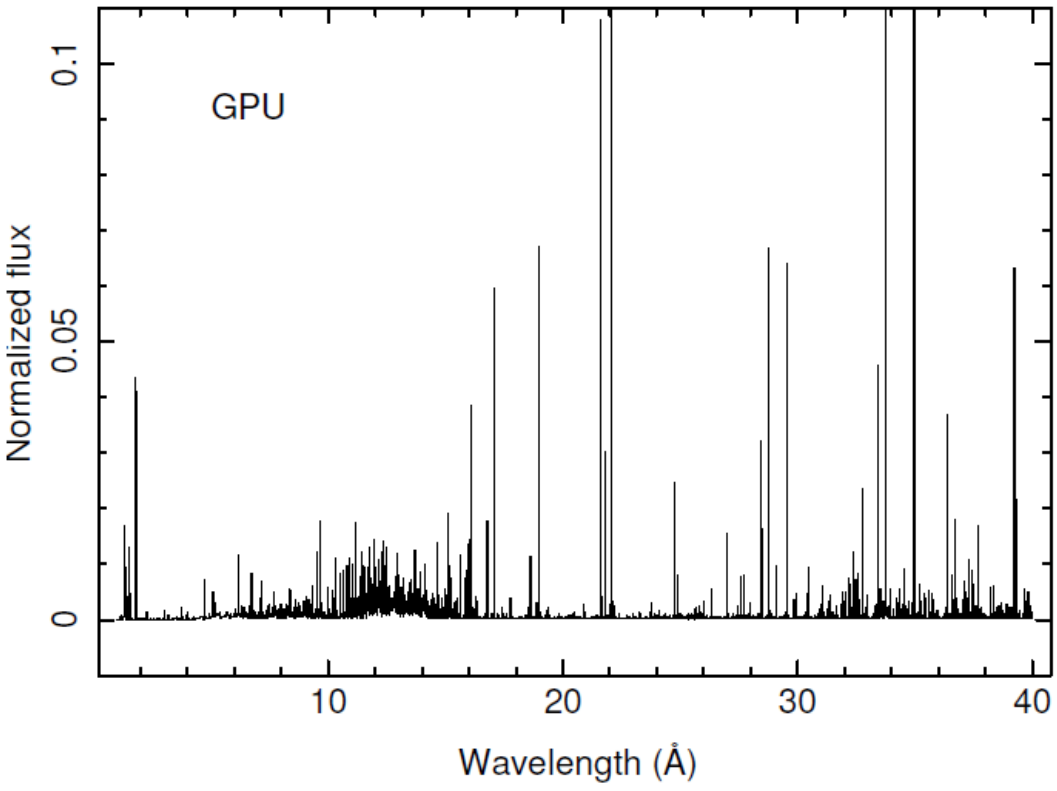


图 4-6 异构并行的 APEC 方法计算的辐射率

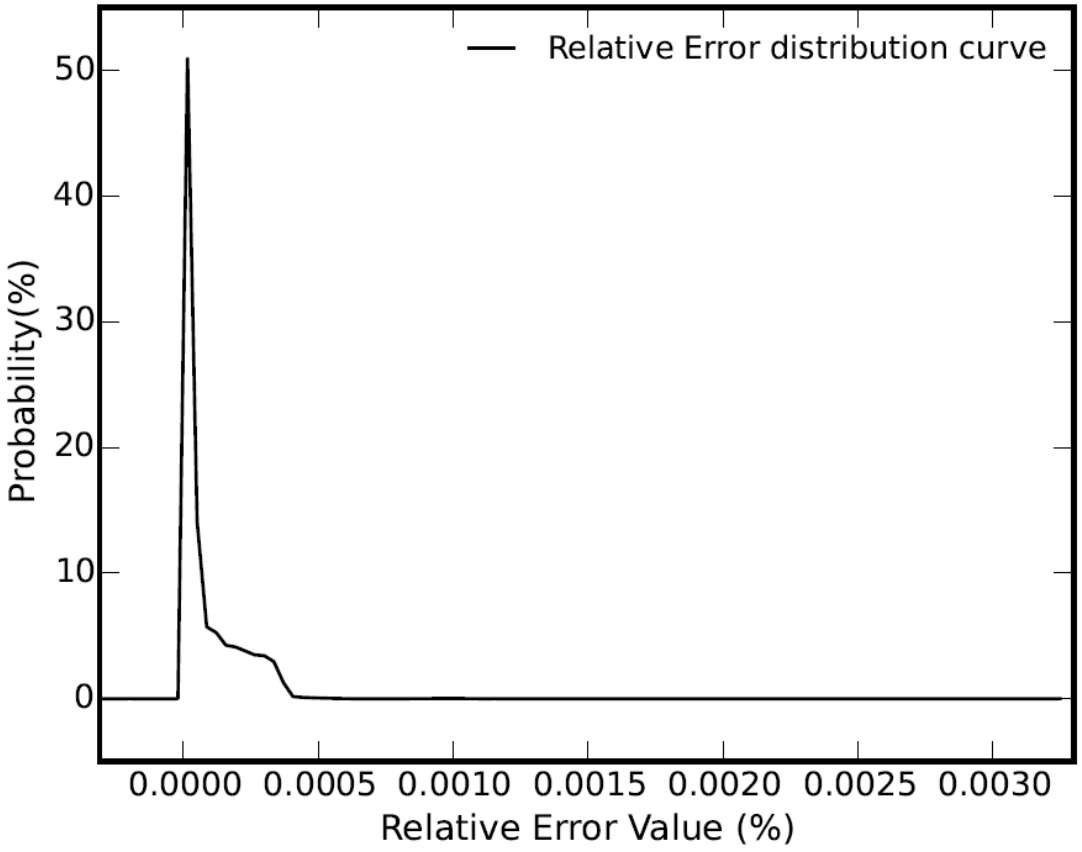

Relative Error distribution curve
Probability(%)
50
40
30
20
10
0
0.0000
0.0005
0.0010
0.0015
0.0020
0.0025
0.0030
Relative Error Value (%)

图 4-7 误差值分布曲线

第五章　扩展应用

本文提出的CPU-GPU混合异构并行方法使光谱的计算在计算性能方面提升很高，同时也保证了 CPU 与 GPU 之间的负载平衡，以及光谱计算的准确性。由此可见本研究提出的混合异构并行方法适用于包含大量小的相似任务的计算密集型的应用，由此我们希望通过非电离平衡（NEI）问题的相关应用来验证这一适用性。

5.1 非电离平衡方程的特点

非电离平衡是一种重要的天文物理现象，他与许多物理过程相关，基本的非电离平衡模型由以下给出的常微分方程 (ordinary differential equation(ODE)) 所示：

$$\frac{\partial n_i^Z}{\partial t} = N_e\left[n_{i+1}^Z\alpha_{i+1}^Z + n_{i-1}^Z S_{i-1}^Z - n_i^Z\left(\alpha_i^Z + S_i^Z\right)\right]\left(i = 1,\ldots,N_{spec}\right) \tag{5-1}$$

其中n_i^Z表示的是元素Z的第i个离子的数量密度，t表示的是时间，N_{spec}表示的是元素Z的所有电离状态数的总和，N_e表示的是电子的数量密度，$\alpha_i^Z = (N_e, T)$表示的是碰撞和双电子复合系数，$S_i^Z = (N_e, T)$表示的是碰撞电离系数。非电离平衡求解过程可参考之前的工作[28]，非电离平衡方程计算的特点如下：

- 参数空间的每一点都会对应若干常微分方程组，每组方程的数量与该元素对应的电离状态个数相同。
- 方程中的系数α_i^Z和S_i^Z的值由电子的数量密度和温度所决定，同时此值需要实时地计算。因此除了方程组本身的求解外，常微分方程组的建立也是需要消耗一些计算资源的。
- 方程中的系数α_i^Z和S_i^Z对电子的温度和密度的变化是非常敏感的，所以在相同的物理条件下，同一元素的不同离子对应的系数也许会相差好几个数量级，因此此处的常微分方程组都是刚性的。
- 由于每种离子的浓度即质量分数仅由其自身以及相连的两种离子决定，因此对应的常微分方程组的系数矩阵都是稀疏的。

通过以上特点可知，由于元素中的质子数目，限定了对应的非电离平衡方程组的大小，所以如果单纯地考虑宇宙中最基本的元素，那么对于每组非电离平衡方程

组来说，方程组中的未知数的数量不会超过 30 个；由于方程组右边的系数的确定也是需要进行一定的计算的，因此这也成为了影响性能的主要元素之一，对于实际的应用中的计算，可以根据实际情况选择适当的算法，使得计算的精度和性能上能获得一个相对适中的结果；

5.2 非电离平衡方程的一般解法

求解常微分方程组的方法有一些，而隐式的求解算法则是求解刚性常微分方程的第一选择，其一般的表达形式如下所示：

$$\dot{y} = f\left(y\right) \tag{5-2}$$

$$y_{n+1} = y_n + hf\left(y_{n+1}\right) \tag{5-3}$$

$$y_{n+1} = y_n + h\left[1 - h\frac{\partial f}{\partial y}\right]^{-1} \cdot f\left(y_n\right) \tag{5-4}$$

其中，公式 (5-2) 为常微分方程初始形式，公式 (5-3) 为该常微分方程的隐式差分表示形式，公式 (5-4) 则是利用牛顿法，将公式 (5-3) 进行了线性化之后获得的半隐式的表示形式，y值表示的是未知量，$\frac{\partial f}{\partial y}$表示的是方程组等式右边的偏导数矩阵（即雅可比矩阵），h表示的计算的步长。

在现实的计算中，我们一般采取的是公式 (5-4) 中所表示的半隐式形式，当步长h较大的时候，算法的内部一般还会经历多次的迭代。工程计算中计算的性能是一个非常重要的指标，因此一般为了提高计算的性能，使用的算法中还会进行步长的自适应调节机制以及数值误差控制机制等，基本的算法大致如下：

- 驱动部分(driver)：算法的主程序，执行的过程中会循环调用单步积分，直到结束。
- 单步积分(stepper)：主要任务是实现步长的自适应调节以及误差控制的逻辑工作，与此同时还会调用核心算法来完成具体的积分计算，由于存在步长的自适应调节以及误差控制机制，所以这个过程可能会重复多次，直到当前的计算结果是可以接受的。
- 算法核心(algorithm)：针对给定步长根据算法的具体差分形式进行计算，计算内容主要有三部分，常微分方程组等式右边的值，常微分方程组等式右边的雅可比矩阵，以及将上述两个计算结果代入特定算法的差分公式。

5.3 利用混合异构并行方法求解非电离平衡方程

即使是使用现代的求解方法来计算常微分方程组，计算离子丰度的多维模拟一样会产生非常昂贵的计算和内存开销。本章使用的所有测试用例都包含一百万个网格点，每个点包含 1000 个时间步长，运行时环境与之前的光谱计算是一样的。为了更加有效地使用本文提出的混合异构并行方法，利用 GPU 加速非电离平衡求解器是基于经典的 LSODA[29]常微分方程求解器，每十次计算被划分为一个计算任务，这样可以有效地减少主机和 GPU 设备之间频繁的数据拷贝。

表 5-1 不同 GPU 设备数目下 NEI 的加速比

	1 GPU	2 GPUs	3 GPUs	4 GPUs
加速比	2.8	5.9	10.8	15.1
时间（s）	3137	1494	810	582

表 5-1 显示了不同 GPU 设备数目下，NEI 的加速比，该加速比是通过将 CPU/GPU 混合异构并行方法与 24 个 CPU 核的 MPI 版本的并行方法对比获得的。当最大队列长度为 8，GPU 数量为 4 的时候，获得了最好的加速效果，此时加速比为 15。以上的实验表明本文提出的混合异构并行方法对于其他计算密集型的任务具有良好的适应性，但是获得最好加速比的配置与具体的应用程序密切相关，需要具体问题具体分析。

第六章 总结和展望

6.1 总结

本文中，我们提出了一种CPU-GPU混合异构并行方法来加速求解光谱计算。首先将计算密集型的积分部分放到了GPU上来计算，通过减少主机和设备之间频繁的数据拷贝来提高计算的性能，减少数据拷贝的方法主要是将一个离子中的多个小的积分任务合并成为一个大的积分任务，集中在GPU内部完成计算，再将结果值回传给CPU主机。其次，提出了多CPU-GPU混合异构下的任务调度策略，本文中我们使用的是共享内存的方式，该种方式相对于传统的客户机-服务器（如英伟达提出的MPS[24]方法）的体系结构可以很好地减少额外的通讯开销。最后，综合理论分析和实验验证了本文所提出的方法的有效性和准确性，实验显示，使用24个CPU核3个GPU设备，相对于传统的串行APEC实现方法，可以将整体的计算加速300倍，相对于纯CPU的MPI并行方法，整体的计算加速也有22倍。此外，本文提出的方法也可以适用于非电离平衡相关的其他应用，例如求解常微分方程，使用本文提出的方法，常微分方程的计算相对于纯MPI的并行方法提速了15倍。

在目前的实现中也存在着一些不足之处。目前的方法只支持同步模式的任务调度，也就是说CPU提交一个任务后，如果该任务被分配到了GPU上来进行计算，那么CPU将一直处于阻塞状态，直到该任务计算完成，结果从GPU回传到了CPU上。当GPU上的单个计算任务的计算时间并不是很长的时候，该方法并不会产生很明显的资源浪费，但是当GPU上的单个计算任务计算时间很长的时候，就会出现一种情况，CPU经常处于等待状态，此时就浪费了CPU的计算资源，从而降低了整体的计算性能。所以，当单个任务属于GPU耗时的任务时，需要增加对异步模式的支持，使系统的通讯和计算相分离，从而减少CPU的等待时间，提高整体的计算性能。

6.2 展望

对于光谱计算本文提出了异构系统下的优化方法，同时给出了混合异构体系

下的调度策略，同时针对于目前方法的不足，我们对于未来的工作提出了进一步的计划。首先需要对目前的负载平衡调度策略进行进一步的完善，增加异步执行的机制，使得用户可针对不同的计算应用使用不同的调度模式。其次是需要进一步提高该方法的适用性，以更加适合其他复杂的天文应用，例如求解电离平衡方程以及核合成等相关的天文物理问题。

参考文献

[1] R. D. Cowan, The theory of atomic structure and spectra [M]. University of California Press, vol. 3, 1981:2-12,.

[2] R. Smith. (2008, Aug.) Calculating radiative recombination continuum from a hot plasma. [EB/OL]. Available: http://atomdb.org/Physics/rrc.pdf

[3] J. Houck and L. Denicola, ISIS: An interactive spectral interpretation system for high resolution x-ray spectroscopy [J], Astronomical Data Analysis Software and Systems IX, vol.216, 2000:591~594.

[4] K. Arnaud, XSPEC: The first ten years [J], Astronomical Data Analysis Software and Systems V, vol. 101, Qubec City, Canada, 1996:17~20.

[5] M. S. Noble and M. A. Nowak, Beyond XSPEC: Toward highly configurable astrophysical analysis [J], Publications of the Astronomical Society of the Pacific, Vol.120, No.869, 2008:821~837.

[6] M. Bautista and T. Kallman, The XSTAR atomic database [J], The Astrophysical Journal Supplement Series, Vol. 134, No.1, 2001:139.

[7] R. K. Smith, N. S. Brickhouse, D. A. Liedahl, et al, Collisional plasma models with APEC/APED: emission-line diagnostics of hydrogen-like and helium-like ions [J], The Astrophysical Journal Letters, Vol. 556, No.2, 2001:91.

[8] A. Al-Omari, J. Arnold, T. Taha, et al, Solving large nonlinear systems of first-order ordinary differential equations with hierarchical structure using multi-gpgpus and an adaptive runge kutta ode solver [J], Access, IEEE, vol.1, 2013:770~777.

[9] M. S. Noble, L. Ji, A. Young, et al, Parallelizing the xstar photoionization code [C], in Astronomical Data Analysis Software and Systems XVIII, Qubec City, Canada, 2009:301~304.

[10] R. Piessens, D. Doncker-Kapenga, C. U ¨ berhuber, et al, QUADPACK, a subroutine package for automatic integration [J], Springer Series in Computational Mathematics, vol. 1, 1983:301.

[11] Intel math kernel library. [EB/OL]. Available: http://software.intel.com/en-us/intel-mkl

[12] T. Hahn, CUBA a library for multidimensional numerical integration [J], Computer Physics Communications, Vol. 168, No. 2, 2005:78~95.

[13] B. Gough, GNU scientific library reference manual. Network Theory Ltd., 2009.

[14] K. Arumugam, A. Godunov, D. Ranjan, et al, An efficient deterministic parallel algorithm for adaptive multidimensional numerical integration on gpus [C], 42nd International Conference on Parallel Processing, Lyon, France. 2013:486~491.

[15] K. Arumugam, A. Godunov, D. Ranjan, et al, A memory efficient algorithm for adaptive multidimensional integration with multiple gpus [C], 20th International Conference on High Performance Computing, Bengaluru, India. 2013:169~175.

[16] E. Yurtesen, M. Ropo, M. Aspnas, et al, SSE vectorized and gpu implementations of arakawa's formula for numerical integration of equations of fluid motion [C], 19th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, Ayia Napa, Cyprus. 2011:341~348.

[17] D. Thuerck, S. Widmer, A. Kuijper, et al, Efficient heuristic adaptive quadrature on gpus: Design and evaluation [J], in Parallel Processing and Applied Mathematics, Warsaw, Poland. 2013: 652~662.

[18] Y. Shi, W. H. Green, H.-W.Wong, et al, Accelerating multi-dimensional combustion simulations using gpu and hybrid explicit/implicit ode integration [J], Combustion and Flame, Vol. 159, No.7, 2012: 2388~2397.

[19] S. Chen, A hybrid gpu/cpu FFT library for large FFT problems [J], Master's thesis, University of Delaware, Newark, USA, 2013. Available: http://udspace.udel.edu/handle/19716/12800

[20] G. Laccetti, M. Lapegna, V. Mele, et al, A study on adaptive algorithms for numerical quadrature on heterogeneous gpu and multicore based systems [J], Parallel Processing and Applied Mathematics, Warsaw, Poland, Sep. 2013:704~713.

[21] Z. Wang, L. Zheng, Q. Chen, et al, CPU + GPU scheduling with asymptotic profiling [J], Parallel Computing, Vol. 40, No. 2, 2014:107~115.

[22] A. Acosta, V. Blanco, and F. Almeida, Dynamic load balancing on heterogeneous multi-gpu systems [J], Computers & Electrical Engineering, Vol. 39, No. 8, 2013: 2591~2602.

[23] B. Yang, K. Lu, J. Liu, et al, Hybrid embarrassingly parallel algorithm for heterogeneous cpu/gpu clusters [C], 7th International Conference on Computing and Convergence Technology, Seoul, Korea. 2012:373~378.

[24] NVIDIA Corporation. Sharing a gpu between mpi processes: multi-process service (mps) overview [EB/OL], 2013. Available:

http://docs.nvidia.com/deploy/mps/index.html

[25]张舒，褚艳丽，赵开勇．GPU 之高性能运算之 CUDA[M]．北京：中国水利水电出版社，2009：23~47．

[26]NVIDIA Corporation, CUDA C Programming Guide [EB/OL], 2015-09-01. http://docs.nvidia.com/cuda/cuda-c-programming-guide/#axzz3r5X4dhwN

[27]W. H. Press, Numerical recipes 3rd edition: The art of scientific computing [M]. Cambridge, UK: Cambridge university press, 2007.

[28]J. Xiao, X. Xu, J. Sun, et al, Acceleration of solving non-equilibrium ionization via tracer particles and mapreduce on eulerian mesh [C], International Conference on Algorithms and Architectures for Parallel Processing, Dalian, China. 2014:29~42.

[29]L. Petzold and A. Hindmarsh, LSODA–livermore solver for ordinary differential equations, with automatic method switching for stiff and non-stiff problems [J], Computing and Mathematics Research Division, I-316 Lawrence Livermore National Laboratory, vol. 94550, 1997.

发表论文和参加科研情况说明

发表的论文：

[1] Jian Xiao, Xingyu Xu, Jizhou Sun, Xin Zhou, and Li Ji. “Acceleration of Solving Non-Equilibrium Ionization via Tracer Particles and MapReduce on Eulerian Mesh”. International Conference on Algorithms and Architectures for Parallel Processing, ICA3PP 2014.

[2] Jian Xiao, Xingyu Xu, Ce Yu, Jiawan Zhang, Shuinai Zhang, Li Ji, Jizhou Sun.. “Accelerating Spectral Calculation through Hybrid GPU-based Computing”. International Conference on Parallel Processing, ICPP 2015.

软件著作权：

[1] 徐星宇，肖健，孙济洲，多 GPU 多 CPU 混合异构体系下的光谱计算软件。软件著作权登记号 2015SR117745. 2015

参与的科研项目：

致　　谢

本论文的工作是在我的导师孙济洲老师以及肖健老师的悉心指导下完成的，孙济洲老师以及肖健老师严谨的治学态度和科学的工作方法给了我极大的帮助和影响。在此衷心感谢三年来孙老师以及肖老师对我的关心和指导。

加入天文信息联合实验室以来，于策和肖健老师对我的科研工作和日常工作给予了莫大的帮助和鼓励，感谢他们在我迷茫时开导我，受挫时鼓励我，进步时肯定我，在此向于老师和肖老师表达最真诚的敬意与感谢。

在实验室工作中感谢毕业的学长学姐的帮助和指导，感谢学弟学妹在日常生活中的帮助和关心。

在论文撰写过程中感谢学长李佳骏，学姐王春玉给予的帮助，感谢学妹王洁给予的生活上的帮助和关心，感谢舍友的支持和理解。

另外也感谢家人和朋友，是他们的付出使我能够在学校专心完成我的学业。