天津大学博士学位论文

# 非电离平衡天文数值模拟的性能优化

# Acceleration of Non-equilibrium Ionization Simulation


一级学科：计算机科学与技术

学科专业：计算机应用技术

研 究 生：肖 健

指导教师：孙济洲 教授

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

计算密集、耗时长是现代天文数值模拟的主要特点。提高模拟计算的性能，减少计算资源的消耗，在精度和性能之间取得一个最佳的平衡点，一直是天文数值模拟软件设计的关键目标。同时，建立完整的有效的非电离平衡（NEI）状态下的辐射流体数值模拟一直是天文数值模拟中的难题之一，然而传统的基于欧拉网格的非电离平衡求解过程在内存、计算和网络通讯上都带来了巨大的开销。

本文在总结以前研究工作的基础上，从非电离平衡模拟过程中制约性能的几个关键因素入手，针对多核异构体系和大规模并行环境，分别从工作流程、框架结构、数值求解三个层面对非电离平衡模拟进行了性能优化。

首先，本文分析并验证了传统算法的性能瓶颈，通过引入示踪粒子将底层的自适应网格与上层的非电离平衡计算解耦，然后基于 MapReduce 模型重新设计了非电离平衡的并行求解框架。同时针对新框架中伴随而来的大量粒子的快照生成、保存以及轨迹重建等问题，设计了串行 I/O、并行 I/O、直接 I/O 以及实时的流处理模式，使其能够适应不同的计算环境和具体要求。实验表明，框架结构层次上的优化克服了非电离平衡模拟在大规模并行时的性能瓶颈，在相同的实验环境下，仅用原来 1/4 的计算资源，就取得了 3 倍以上的性能提升。

其次，为了突破传统 CPU 结构在求解大量非电离平衡方程时的性能制约，本文继续提出了基于多 CPU-多 GPU 混合异构体系的非电离平衡求解器。算法设计上，通过使用基于共享内存和任务队列的任务调度策略，最大限度地发挥了 CPU 和 GPU 各自的优势，提高了整体的资源利用率，同时根据 CUDA 编程模型的特点，在算法的数据结构、任务粒度以及内存访问等方面进行了专门的优化。测试结果显示基于多核异构的求解器显著地提高了非电离平衡方程的求解效率，在 4 块 GPU 设备的情况下，加速比达到了 15 左右。

最后，本文利用可视分析和驾驭式计算技术来优化天文模拟的工作流程。基于自适应网格的层级结构，利用快速的低精度的组合分析来指导耗时的高精度的模拟计算。同时又根据天文数值模拟的特点，设计了参数分类和调整接口，进而帮助天文学家高效准确地把握并控制数值模拟过程。本文提出的可视化驾驭计算环境有效的加速了非电离平衡模拟生命周期中的模型建立、电离状态分析等过程，并在参数调整、数值误差控制等方面辅助用户决策。

文中所有实验都是基于实际的天文数值模拟，文中还对所有实验结果的精度进行了详细的对比分析。此外，本文对上述各类方法在其他问题的适应性上也进行了详细的分析和验证，相关实验显示本文的方法同样能够大幅度提升核合成、光谱计算等常见天文计算的性能，以及加速星风模型的探索和确立过程。

**关键词：** 天文模拟，非电离平衡，示踪粒子，MapReduce，GPU，驾驭式计算

# ABSTRACT

Astrophysical simulations are computation-intensive and time-consuming, so modern astrophysical codes aim at a good balance between accuracy and performance in massive parallel environments. Meanwhile, non-equilibrium ionization (NEI) is an important phenomenon related to various astrophysical processes, and its full modeling and simulation is still one of the biggest challenges faced by modern astrophysics. However the traditional time-splitting scheme tightly coupled the NEI solver with the underlying Eulerian mesh and introduced high overhead on computing, memory and communication. In this thesis, in order to accelerate NEI calculation on multi-core heterogeneous systems and large-scale parallel environments, several optimization schemes have been designed and implemented at three levels of simulation process, computing framework, and numerical algorithm.

Firstly, the performance bottlenecks of the traditional NEI scheme have been explained and validated. By introducing tracer particles the NEI solver is decoupled from underlying Eulerian mesh, and based on MapReduce model, a full parallel pipeline is proposed to support both post-processing and nonintrusive simulation-time data analysis. In order to quickly process large amounts of small particle snapshots continuously produced during the course of simulation, several I/O optimization schemes are also designed, including serial I/O mode, direct I/O mode, and real-time stream mode. Evaluations on up to 192 cores show that our approach can improve the end-to-end performance of a real world simulation by 3-fold above.

Secondly, in order to break through the performance limitation of the traditional homogeneous parallel processing systems based on CPUs, which is not efficient to tackle numerous compute-intensive tasks, a GPU-optimized NEI ODE solver is designed based on the CUDA programming model. Moreover a load balance strategy on hybrid multiple CPUs and GPUs architecture via share memory is also proposed to maximize performance and device utilization. Comparing with the 24 CPU cores (2.5GHz) parallel version, this implementation on 3 Tesla C2075 GPUs achieves a speed-up of up to 15.

Thirdly, in order to improve the traditional non-interactive mode of running simulations, a framework for adding interactive control functionalities during the course of time-consuming astrophysical simulations is presented, which is based on a hierarchical architecture where computational steering in the high-resolution run is performed under the guide of knowledge obtained in the gradually refined ensemble

analyses. Several visualization schemes for facilitating ensemble management, error analysis, parameter grouping and tuning are also integrated owing to the pluggable modular design. The visual steering approach effectively accelerates the process of simulation modeling and ionization state analysis, and facilitates decision making on parameter tuning and numerical error control etc.

Additionally, the thesis is a result of three-year collaboration with astronomers. All the proposed schemes are prototyped based on the FLASH multiphysics simulation framework and evaluated by real-world simulations, i.e., supernova remnant, nucleosynthesis, spectral calculation and stellar wind. The accuracy of all the proposed schemes is also validated and confirmed by domain experts. Moreover the adaptability of these methods to other reactive flow simulations is also explored and validated. The optimization approaches for NEI calculation issued in this thesis have high reference values for resolving other large-scale astrophysical simulation in future.

**KEY WORDS:** Astrophysical simulation, Non-equilibrium ionization, Tracer particle, MapReduce, GPU, Computational steering

# 目 录

# 第一章 绪 论

数值模拟主要用于解释观测现象、验证理论模型、以及预测发展演化趋势。随着计算机性能的提高，数值模拟已经在各个科研领域蓬勃发展起来，在天文领域中亦是如此。随着现代天文望远镜观测能力的提高，大量观测数据的涌现既给构建理论模型带来了机遇也带来了很大的挑战。现代的观测可以清晰地分辨这些天体物理过程中的单个谱线信息，简单的理论模型已经远远落后于空间望远镜的观测数据。数值模拟则是不可缺少的实验室。目前，数值模拟已经成为现代天体物理学和天文学的主要研究及分析手段之一。

## 1.1 非电离平衡模拟研究背景

### 1.1.1 多物理场天文模拟

计算量大、运行时间长既是现代天文数值模拟的特点，同时也是天文学家进行模拟实验时不得不面对的难题。随着超级计算技术的提高，现代天文模拟软件均朝着支持大规模并行异构计算环境的方向发展。这使得模拟程序本身的能力大大增强，能够在更大的尺度内或更精细的程度上，展示同时发生且相互影响的诸多物理过程，从而得以验证一些复杂的物理现象。但这种多物理场的模拟反过来又对计算能力提出了更高的要求。因此，提高模拟计算的性能，获得精度和性能的最佳平衡点，一直是设计天文数值模拟软件的关键目标之一。

为了方便建模和简化分析，一个系统内的物理过程可以看作是相对独立的，比如流场(fluid)由欧拉方程表示，而扩散现象由热传导方程(heat conduction)描述，但本质上这些过程是同时发生且相互影响的，这些相关的物理方程耦合在一起，用来描述多物理场组成的非线性系统[1]。在数值模拟中，一般采用时分(time-split)模式来简化计算和提高性能，即在一个演化周期（步长）内，将本来同时发生的多个物理过程简化为顺序发生的在一个个独立的过程，然后分别求解[2]。具体来说，描述整个非线性系统的方程组在形式被分解为由多个相对独立、且较容易求解的微分方程或方程组，各个方程组按照一定的顺序分别单独求解，并将自身的解传递到下一个方程组中。整个过程反复迭代，比如，流场中输出的温度用于热传导的输入，经过热传导更新后的温度在下一个周期又作为了流场的输入（详细

过程可参考图 2-2）。由于不同的物理过程的方程组的数学性质不同，从而解耦和求解方式都不尽相同，这导致了不同物理过程的求解在整体计算时间中的比重会有很大差别。比如，热传导的物理模型是一个包含整个模拟空间所有网格点（未知量）的微分方程组，求解时间随着网格精度和问题空间的增加而快速增长，是影响模拟计算性能的主要因素之一；而描述电子离子热交换(heat exchange)的偏微分方程组(PDE)仅涉及当前格点变量，相对于流体的欧拉方程和描述整个区域温度变化的热传导方程，计算量很小，其求解过程对整体计算时间的影响几乎可以忽略。

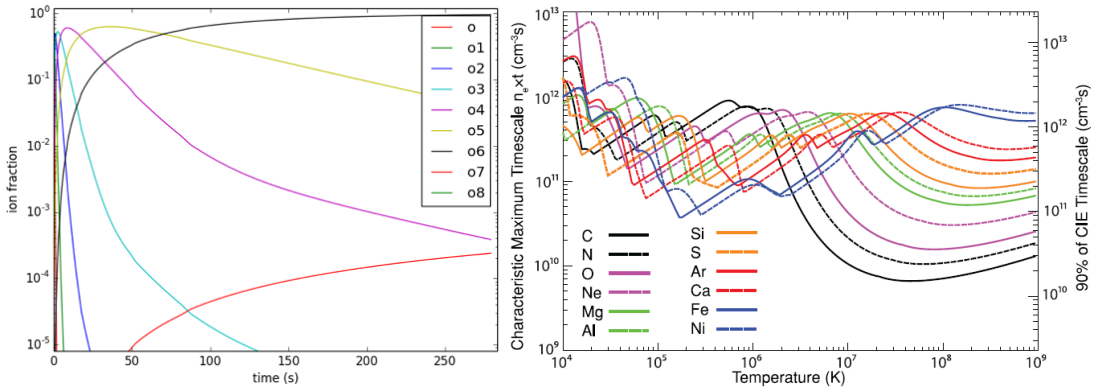
因此，现代天文模拟软件一方面要提供尽可能多的物理模型的求解器，以便天文学家组装合适的物理模块进行复杂现象的模拟；另一方面，在保证精度的同时，要兼顾具体求解器的性能及其耦合到系统后的整体性能。在传统的基于欧拉方法的多物理场的模拟中，涉及反应流(reactive flow)的模拟往往会带来巨大的性能开销。而非电离平衡(Non-equilibrium Ionization，以下简称 NEI)正是一个典型的具有反应作用的物理现象。

### 1.1.2 非电离平衡模拟

简单来说，非电离平衡即某种元素的离子处于一种非平衡状态，由于大量离子和自由电子的相互碰撞，导致离子一直在不同电离状态之间转换，各种离子的浓度（丰度）总是出于不断的变化中，期间伴随能量的吸收和释放。图 1-1(a)显示了一定量的处于非电离平衡状态的不同氧离子的丰度（经过了归一化处理）随时间变化情况；如果不考虑核合成，则任意时间点上，所有氧离子丰度之和为一个固定值，等于氧元素在宇宙中的丰度。非电离平衡状态下，离子丰度一直随时间变化，即使温度能够恒定下来，这种演化过程也要持续很长时间才能达到一个平稳态，图 1-1(b)是各个元素非电离平衡演化的时间尺度的分析结果，可以看出，对应不同的温度，这种不平衡的状态会持续几百年到几万年不等，所以在活动激烈的区域，几乎都伴随着非电离平衡过程。非电离平衡与许多天体物理过程都有关联，是一类非常重要的天体物理现象。无论是在太阳耀斑、超新星遗迹、星团风等小尺度的模拟中，还是在活动星系核、星际介质、星际飓风等大尺度模拟中，非电离平衡的因素都经常要被考虑进来[3]。

从孤立的角度看（不考虑其他物理过程，比如流体），非电离平衡的计算过程是将初始的离子丰度，代入到非电离平衡方程中，计算出经过一定时间之后的新的丰度值。较之于传统的单流体模拟，非电离平衡的模拟需要区分大量不同的离子，同一元素的不同离子又相互转换。而在实际中，非电离平衡总是要与流体等基础的物理过程耦合到一起，形成多物理场的模拟，求解过程也变得复杂。经

过与流场的欧拉方程的解耦，非电离平衡方程可以简化为多个形式上一致的常微分方程组(ODE)，相对于欧拉方程、热传导方程等较复杂的偏微分方程(PDE)，常微分方程的数值算法相对简单，计算过程也较快，但大量的常微分方程求解带来的开销却十分巨大，这也是本文要着重解决的难题之一。




(a) 氧元素的离子丰度　　(b) 常见元素达到电离平衡状态的时间[4]

图 1-1 非电离平衡现象及其时间尺度

### 1.1.3 天文模拟的典型流程

传统的天文数值模拟一般分为两个阶段，模拟和分析[5]。模拟是在线的(online)，一般从初始参数开始启动，一段时间后得到阶段性结果或最终结果。模拟的过程是连续，中间偶尔会被中断。模拟得到的中间或最终结果是目标天体在时空坐标上一个状态，只包含基本物理属性，比如：温度、密度、速度、离子丰度（非电离平衡的输出）、能量等，并不能直接用来跟观测现象或理论模型进行对比来得出科学结论。分析阶段则是离线的(offline)， 是模拟过程的延续，对模拟的结果进行深入加工；分析得到的结果，能够用来与真实的观测结果进行比较，进而验证模型和理论。对同一个模拟结果，还要从多角度进行分析，比如天体的密度分布、电离状态（根据离子丰度计算得出）的空间结构、光谱等等。也可以这样理解，模拟的输出包含了目标天体在指定时空区间内的细节，分析则是根据细节推导出可观测的宏观结构。

但模拟和分析并不是严格按照顺序都只进行一遍，由于所有模型都是经过简化的，只是真实物理过程的一个近似，甚至这种近似仅仅是某一个方面的近似。从这个角度看，模拟的过程更像是逐渐调优模型参数，使得最终的模拟结果接近预期。所以，并不能保证模型参数一开始就是正确的，只能是根据分析结果，逐步调整参数，再次运行并分析，这个“调整模型—运行模拟—分析比对”的过程要反复进行好几轮，才能得出期待的结果。

此外，分析本身也不是单一的过程。首先分析可以是多个方面的，多个分析

可以独立进行；其次分析可能是有顺序的，即存在依赖关系，一个分析可能需要前一个或几个分析的结果。对于非电离平衡模拟而言，模拟的输出结果就是空间上各个格点的离子丰度，而最直接的分析结果就是元素的电离状态，这是对同一元素的各个离子丰度的一个统计结果，再深入下去，可以计算离子的发射光谱，这个计算得出的能量谱可以与观测结果直接对比。文本的另一个研究内容则是对上述传统模拟流程的优化。

## 1.2 非电离平衡模拟的研究现状

目前在我们国家，天文领域基于大型计算机的数值模拟主要集中在宇宙学模拟中（如星系宇宙学模拟、大尺度结构模拟等）。而对于更广泛的其它天体物理过程的大型数值模拟还没有得到实现，尤其在非电离平衡模拟这个方向上，国内还没有一个完整的团队。国际上，这样的模拟已经逐渐成长起来，比如有着 10 多年历史的美国芝加哥大学的 FLASH[6]以及近几年出现在开源社区的 ENZO[7]。这些软件开放化，模块化的设计，可使不同用户通过采用不同的计算模块组合，并修改相应的初值条件，边界条件，计算结构条件（比如坐标格点选取，算法选取），物理条件（比如物态方程，引力场）等等来构造一个适用于特定物理问题的计算模型。

虽然国外的这些通用天文模拟软件功能很强大，足以支撑超大规模的计算，但就目前所知，针对大规模的非电离平衡模拟并没有成熟的框架或方案，要么只是独立的算法库，只适用于小规模的单机计算，要么是在大规模模拟中性能急剧下降，耗费大量的计算资源，很难达到实用要求[8]。下面就相关研究课题和现有解决方案进行一下分析和对比。

### 1.2.1 天文数值模拟框架

在天体物理和宇宙结构的模拟中，求解流体力学的欧拉方程的数值方法很多，一般来讲，这些方法可以分为两大类，基于网格的欧拉方法和基于粒子（无网格）的拉格朗日方法。最具代表性的拉格朗日方法如平滑粒子流体力学（SPH：Smoothed Particle Hydrodynamics）[9]，欧拉方法如自适应网格（AMR：Adaptive Mesh Refinement）[10]。这两类方法是天文数值模拟软件的主干算法，决定了整体代码的组织结构。如前所述，流体动力学之外的其他物理过程的模拟算法，都是以某种形式耦合到主干算法中。

**(1) FLASH Code**

FLASH Code[11]是一个开放式，模块化，并行化，多物理，大型数值计算模拟软件系统，主要基于 Fortran 语言实现。该软件开发的初衷是用以解决 Ia 型超新星的理论问题。 但是由于 FLASH 的核心目标—解决可压缩反应流体的动力学问题—在其他物理过程中同样有着关键的作用，因此其各个组件，特别是流体力学，磁流体力学，辐射转移等计算包以及格点计算等结构，被广泛应用于天体物理，高能密度物理，以及计算工程三大类跨学科研究领域。一项 2007 年的用户调查结果显示，应用 FLASH 的研究， 主要分布在高能天体物理、恒星结构与演化、宇宙学、星际介质物理、计算流体力学、算法研究等具体方向。

FLASH 框架基于自适应网格技术(AMR)，本身自带了一个基于欧拉方法的非电离平衡求解器[12]，正如前面所说，该方法会带来整体性能的大幅下降。此外，为简化计算，当前 FLASH 的处理方式要么做平衡假设，要么做辐射对流体动力学不重要的假设，并且在非电离平衡中只是实现几个元素的非电离平衡计算。然而，对于天体物理中感兴趣的课题常常是处于比较激烈的状态，比如激波，超新星爆发，吸积盘，星风，超风等等，而这样的系统几乎无一不是在非电离平衡状态之下。单单靠组合 FLASH 现有的模块并不能提供有效的解决方案。

**(2) ENZO**

ENZO[7]是一个新兴的基于结构化自适应网格(SAMR)的开源模拟框架，主要由 C/C++实现，并混有部分 Fortran-90 代码。其最初的设计是用于宇宙动力学问题的计算，目前已经发展成为一个通用的天体物理数值模拟软件，模拟范围涉及理想和非理想磁流体、宇宙大尺度结构、原始大气化学、辐射转移、星系形成、宇宙膨胀等等。相对于 FLASH 而言，ENZO 原生支持 N 体（N-Body）问题，子步长 (subcycling)也是其一大特色，即每一次大的演化周期内，同一物理过程的计算步长按照其所处的网格层数进行划分，下层网格的步长是其上一层网格步长的 1/n（n 为网格相邻层级的细分粒度）。相比统一步长的方法，支持子步长的设计有利于在准确性和性能之间获得一个更好的平衡。

ENZO 提供了包含化学反应和电离作用的气体求解模块，但涉及的元素种类较少，主要针对原始大气化学的模拟，仅涉及了原子氢、分子氢、氚等元素及其同位素，并没有提供相对通用的非电离平衡模拟的支持。但和 FLASH 一样，模块化的设计使其很容易扩展以适应新的问题。

(3) **其他模拟软件**

除了 FLASH 和 ENZO 这两款比较通用且使用最多的天文模拟框架外，还有

一些专门针对某类问题的天体物理模拟程序和框架。比如劳伦斯伯克利实验室（LBNL）的 Nyx[13]软件包同样基于自适应网格技术构建，主要针对 N 体问题、暗物质、宇宙大尺度结构的模拟。GADGET[14][15]是德国马普学会天体物理研究所推出的开源模拟软件包，与文中提到的其他模拟程序的不同之处是，该软件基于 SPH 技术构建，主要用于星系的碰撞融合、宇宙大尺度结构的演算。哈佛-史密松森天体物理中心（CFA）开发的 AREPO 代码库[16]，采用了独具创意的移动网格技术，该技术结合了基于网格的欧拉方法和无网格的拉格朗日方法的优点，能够捕捉到小尺度范围的扰动和湍流，从而能够为宇宙学模拟提供更加精准的结果。上述的模拟软件包均提供了扩展接口，但并没有针对非电离平衡的特殊要求提供支持。

### 1.2.2 反应流体求解的相关工作

从数学角度看，即从物理问题的方程式形式上，非电离平衡属于反应流体（reactive flow）的一种，这类问题的特点是流体中涉及多种不同的成分，典型代表问题包括燃烧（combustion）、核合成（nucleosynthesis）、电离（ionization）。经过数值解耦，反应流体的模型均可以表示为一个或多个常微分方程组（ODE），一般来讲，这些方程组都是刚性的，常规的显示（Explicit）解法无法胜任，需要计算量较大的隐式（Implicit）或半隐式的求解方法。此类问题的求解一般分为两个步骤，首先要构建反应网络（Reaction Network），即根据当时的物理参数计算常微分方程组的各个系数；然后是具体求解过程。

目前，燃烧模拟的计算规模一般较小，不需要大规模并行求解，在单台工作站或小规模的计算集群上配合 GPGPU（General-purpose computing on graphics processing units）等加速技术，即可以有效地完成大部分的模拟任务[17][18]。而非电离平衡和核合成的模拟计算，一般作为一个大型模拟的一部分，在比较大的空间尺度内进行，多是在高性能计算集群上运行。实验表明，一个典型的超新星爆炸模拟大概需要 100 万个 CPU 小时[19]。如果同时考虑在经典的欧拉方法中进行非电离平衡、核合成等模拟计算，内存消耗和计算量将随着反应网络（元素种类、离子种类、同位素种类）的增大而迅速增加，基本上是不现实的。所以，就目前已知的情况看，稍大规模的核合成计算（反应网络规模大于 13）均采用了后处理（Post-processing）的方式，即模拟过程中并不进行核合成的实时计算，而是定时保存中间结果，之后基于这些历史快照数据单独进行核合成的演算。

### 1.2.3 AtomDB 和 Libapecnei

AtomDB 是哈佛-史密松森天体物理中心（CFA）负责维护的天体物理原子数

据库[3]，该数据库为电离平衡（CIE:collisional ionization equilibrium）和非电离平衡计算提供了基础数据和模型，也被广泛用于发射光谱的计算。AtomDB 同时提供了一个开源的非电离平衡计算的代码库（Libapecnei），本文作者也有幸参与了其中光谱计算部分的优化工作。Libapecnei 是一个独立的专用函数库，可以灵活地嵌入到各种非电离平衡相关的大规模模拟中，但需要宿主程序的提供接口可以将非电离平衡计算耦合在其中，在这方面 Libapecnei 并没有提供专门的支持。而且该函数库也没有针对现代高性能平台广泛采用的多核异构结构进行专门的优化，所以目前 AtomDB 系列工具并不能充分利用硬件结构的优势，在性能方面还有很大的提升空间，这在一定程度上限制了其在大规模模拟中应用的范围。

### 1.2.4 欧拉方法与拉格朗日方法

非电离平衡很少独立进行模拟，通常情况下是多物理场模拟的一部分，需要在基础的流场模拟上进行。流体模拟有两类基本方法，欧拉方法和拉格朗日方法，这两类方法在计算准确度和复杂度上面，各有优缺点[20]。欧拉方法通过把空间划分成网格来进行离散化处理，具有守恒性质的物理量如质量、动量以及能量在各个网格点之间交换并求值，而且在这个过程中网格结构可能是变化的，这使得整个流场内的信息混合在一起，导致欧拉方法一个主要不足是无法保留各个流体元的演化历史。拉格朗日方法是一种无网格方法，其基本思想是将连续的流体采样为相互作用的蒙特卡罗粒子，各个质点（粒子）承载着各种物理量（质量、速度等），通过求解由这些质点组成的质点组的动力学方程进而得到整体的行为。相比基于网格的欧拉方法，拉格朗日方法能够跟踪每个质点的运动轨迹，但同时伴随而来的是动力学方程的积分结果的较大误差。

## 1.3 大规模非电离平衡模拟的性能问题

大规模的非电离平衡模拟必须建立在高性能计算平台之上，程序并行化要求高，程序间信息交互频繁，中间信息存储量大。所以，一直以来，建立完整的有效的非电离平衡状态之下的辐射流体数值模拟是天文数值模拟中的难题之一。相关实验表明，采用经典的欧拉模式，一个包括宇宙中最常见的 12 种元素的非电离平衡的计算会带来 **15** 倍左右的内存消耗和 **10** 倍左右的整体性能下降[8]，这对于本身就需要长时间运行的天文模拟来讲，是无法接受的（非电离平衡的传统解法的分析和性能测试结果详见第二章）。

此外，正如前面提到的，传统的天文数值模拟是典型的后台模式，尚不支持

实时的人机交互，一般来讲，整个模拟过程包含两个相对独立且不断重复的步骤：在线的数值计算和离线的结果分析，“建模—计算—分析”的过程循环需要一直持续直到获得预期结果。这对于长时间运行的天文模拟来说，会造成大量的计算资源及人力资源的浪费。

综合上述，现代大规模的天体物理并行计算面临着很多共性问题，非电离平衡的加入，使得这些问题更加突出：

1. 计算量大，中间结果数据多，耗时长，这方面在非电离平衡模拟中表现得尤为严重，非电离平衡的传统解法除了引入了巨大性能开销外，还限制了系统的水平延展性，无法充分发挥大规模并行计算的优势。
2. GPU-CPU 异构系统已成为大规模并行计算的主流模式，但由于遗留代码太多，绝大部分的天文数值模拟软件还停留在 MPI 进程级别的并行模式上，同其他物理模块不同的是，非电离平衡计算包含了数量众多且相对独立的 ODE 方程的建立和求解，仅仅依靠 MPI 并行模式并不能充分的发挥多核异构硬件平台的优势，也无法进一步提升性能。
3. 在整个模拟流程中，仍然缺少对中间结果进行可视化显示和分析的有效手段，无法及时发现中间结果中隐含的规律或错误；缺乏对计算过程、任务进行的有效监控和动态任务调度方法，不能及时调节并行计算参数，从而不能实现根据中间计算状态动态地进行程序调整和任务调节等目标。

## 1.4 研究目标和主要研究内容

本文的研究目标旨在针对传统欧拉方法下非电离平衡计算引入的巨大性能开销问题，在理论分析的基础上，从非电离平衡模拟过程中制约性能的几个关键因素入手，综合利用现有的计算加速理论和技术，优化已有算法或提出新的高效算法，提出一套支持非电离平衡模拟的完整生命周期的解决方案。

如图 1-2 所示，本文主要从两个层面的三个层次上对非电离平衡模拟的性能优化问题展开探索和研究，主要研究内容包括：

1. 通过理论手段，系统地分析基于欧拉网格的非电离平衡计算在内存、网络通讯以及计算量等方面带来的开销，同时指出了传统方法在选取迭代步长、平衡精度与性能等问题上的限制。
2. 研究天文数值模拟中的常用的时分（time-splitting）解耦方法，借助示踪粒子将非电离平衡计算与基于网格的黎曼（流体）求解器进行解耦，利用 MapReduce 模型构建并分析示踪粒子轨迹，在此基础上，形成一个能够适

用于大规模非电离平衡模拟的计算框架。

3. 研究设计在 CPU-GPU 混合异构系统下，快速构建并求解非电离平衡反应网络的方法；并针对非电离平衡刚性常微分方程组的特点，探索有效的 CPU-GPU 的负载平衡方案。
4. 研究利用可视分析技术和驾驭式计算辅助天文数值模拟的方法，借助自适应网格的层级结构，研究组合分析（Ensemble Analysis）和驾驭式计算对天文模拟过程的指导作用，将交互机制引入非电离平衡模拟的整个生命周期。
5. 研究和验证文中方法对相关物理问题的适应性以及方法的准确性；并扩展相关成果的应用范围，探索本文方法在核合成、光谱计算等其他天文模拟问题中的应用。

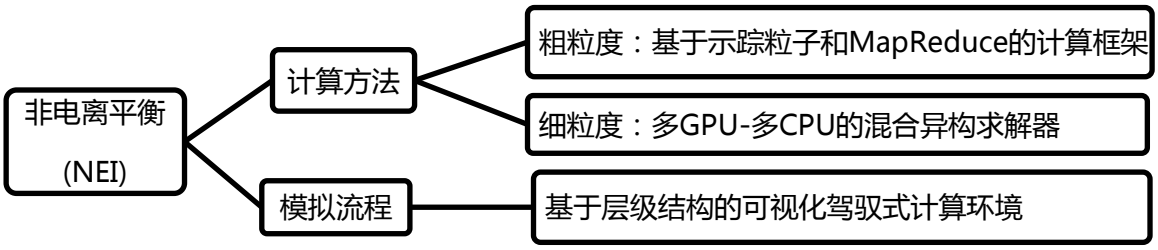



图 1-2 本论文的主要研究内容和组织结构

本文工作的创新点总结如下：

1. 提出了一个基于示踪粒子和 MapReduce 模型的非电离平衡计算框架，该框架消除了基于网格的传统解法引入的巨大性能开销，同时也针对大量示踪粒子的快照保存和轨迹生成问题，设计了多种优化策略；该框架还可方便地应用于加速其他反应流体的求解；
2. 针对非电离平衡模拟中的单体计算量小但总体任务密集的特点，设计了一个多 CPU-GPU 混合异构体系结构下的非电离平衡求解器，并根据其计算特点，引入了一种基于任务队列的调度机制，还将该方法扩展到了光谱计算中；
3. 基于自适应网格的层级结构，借助组合分析和驾驭式计算，为长时间运行的天文模拟程序引入了交互式控制机制，同时专门针对非电离平衡模拟设计了可视化和可视分析方案。

## 1.5 论文结构

本文研究内容紧密结合领域应用，属于天文模拟与高性能计算的交叉学科，相关研究工作分散在三个相对独立的优化层次上，彼此的关联度不高，不适合像

大多数论文那样统一展开论述，为方便阅读和参考，本文在第二章一开始，先从领域问题入手，详细说明问题的物理方程、现有的数值解法以及对现有方法性能瓶颈的分析和验证，为后面的章节做好铺垫；仅在第二章后半部分对相关理论和技术进行了简单概述，相关研究工作的详细论述及其与本文方法的对比则在后续章节中结合具体的优化策略给出。论文的余下部分按如下方式组织：

第二章，从理论层面对现有的基于网格的非电离平衡计算方法进行了全面分析，并结合实验数据，阐述了传统算法在内存、网络通讯、计算量等方面的性能开销，附带探讨了非电离平衡计算中不可避免的多流体一致性的问题，最后对相关工作进行了简要的汇总和论述。

第三章，基于示踪粒子和 MapReduce 计算模型，构建了适应大规模并行的非电离平衡计算和分析的工作流框架，同时针对大量示踪粒子的快照保存和轨迹生成问题，分别提出了普通 I/O 模式、直接 I/O 模式、就近(In situ)模式以及其他优化策略，并对该方法的精度和性能进行了详细的分析测试；最后对该方法在核合成计算方面的应用进行了探讨。

第四章，在详细分析非电离平衡刚性常微分方程组的常用求解方法的基础上，针对非电离平衡模拟中的单体计算量小但总体任务密集的特点，提出了一种 CPU-GPU 混合体系结构下加速计算的方法，同时，设计了一个基于共享内存和任务队列的 CPU 和 GPU 的任务调度策略，并对实验结果进行了详细的分析；最后又通过光谱计算的实例再次验证了本章方法的可行性。

第五章，从非电离平衡模拟的典型工作流入手，针对天文数值模拟计算密集，中间结果巨大，运行时间长的特点，利用自适应网格的层级结构，提出了一个可视驾驭计算框架，并针对非电离平衡模拟，设计了专门的可视化策略。

论文最后对本文的研究工作进行了总结，分析了论文的不足并提出了今后进一步工作的方向。

# 第二章 非电离平衡模拟性能分析及相关工作

本章主要针对非电离平衡的建模和传统求解方法进行分析，得出现有方法引入巨大性能开销的原因，并对改进方向及相关工作进行论述。

## 2.1 非电离平衡方程

多物理场模拟中，非电离平衡（NEI）的基本方程组如下面的方程(2-1)—(2-5)所示[12][21]：

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0 \tag{2-1}$$

$$\frac{\partial \rho \mathbf{v}}{\partial t} + \nabla \cdot (\rho \mathbf{v}\mathbf{v}) + \nabla P = \rho g \tag{2-2}$$

$$\frac{\partial \rho E}{\partial t} + \nabla \cdot [(\rho E + P)v] = \rho v \cdot g[+S] \tag{2-3}$$

$$\frac{\partial n_i^Z}{\partial t} + \nabla \cdot n_i^Z v = R_i^Z \quad (i=1,\cdots,N_{spec}) \tag{2-4}$$

其中，方程 2-1—方程 2-3 是经典的气体动力学（或流体动力学）的欧拉方程，方程 2-4 是一组包含了各种离子的平流方程，其中的每个方程对应一种离子。这里，$\rho$表示流体密度，$t$代表时间，$v$是速度（矢量），$P$是压力，$E$是能量密度，即单位质量中的内能和动能之和，$g$是重力加速度，$S$代表能量项。密度、压力、内能可由方程 2-5 所示的状态方程联系起来。

$$P = (\gamma - 1)\rho\epsilon \tag{2-5}$$

其中$\gamma$是绝热系数，$\epsilon$为内能密度，$E = \epsilon + \frac{1}{2}|v|^2$

$n_i^Z$代表元素 $Z$的第$i$种离子的数量密度，$N_{spec}$表示元素$Z$的所有电离状态数，比如，碳元素（$C^6$）一共有 6 个电子和 7 个电离状态。$R$由下面的方程 2-6 来定义

$$R_i^Z = N_e\left[n_{i+1}^Z \alpha_{i+1}^Z + n_{i-1}^Z S_{i-1}^Z - n_i^Z\left(\alpha_i^Z + S_i^Z\right)\right] \tag{2-6}$$

其中，$N_e$代表电子的数量密度，$\alpha_i^Z = (N_e, T)$代表碰撞和双电子复合系数，是电子密度和温度的函数，$S_i^Z = (N_e, T)$则代表碰撞电离系数，同样是电子密度和温

度的函数，需要实时计算得到。对于单个离子而言，$\alpha_i^{\mathbf{Z}}$和$S_i^{\mathbf{Z}}$构成了这个离子的电离矩阵（Ionization Rate Matrix），或者更通用一点的说法叫做反应网络（Reactive Network）。

可以简单地把复合和电离看作是相反的物理过程，电离即离子吸收能量，失去一个电子，跃迁到高一级的电离态；复合则是离子得到一个电子，释放能量，回落到低一级的电离态。非电离平衡方程描述的是，在宏观范围内，离子数量随时间的变化。当离子数量不随时间变化时就是电离平衡态，但这并不表示电离作用不发生，而是电离作用和复合作用的统计效果相互抵消，即$\frac{\partial \mathrm{n}_i^{\mathbf{Z}}}{\partial \mathrm{t}} = 0$。而当电离作用与复合作用的速率不相等时，离子就处于非电离平衡态，当电离作用快于复合作用，离子处于过电离（Over Ionization），反之，则是低电离（Under Ionization）。在真实的宇宙环境中，绝大部分天文学家关心的现象都处于非电离平衡态。

## 2.2 基于网格的非电离平衡求解分析

### 2.2.1 自适应网格

天文模拟的计算规模一般都很大，在传统的欧拉方法中，为了追求性能和精度的最佳平衡点，一般都采用了自适应网格（AMR）技术[10]。不同于将整个计算区域划分成大小相等的网格的均匀网格技术（Uniform Grid），自适应网格在物理量变化激烈的区域网格也变得密集，在变化缓慢区域则相对稀疏，从而在不显著增加计算量的同时得到较高精度的解。具体来讲，自适应网格会根据预先设定的细化标准，根据当前的物理状态动态地调整网格结构，即在某些变化较为剧烈的区域，如激波面、接触间断面等，将网格细化直到符合预定标准，而当变化趋于平缓时，又会将细化后的网格合并为较稀疏的网格。

跟均匀网格相比，自适应网格最显著的优点是提高了计算效率，但同时也给数值算法增加了复杂度，尤其是在不同层级网格的邻接面，容易形成较大的计算误差。目前，在天文模拟中，采用较多的是结构化自适应网格技术。以 FLASH 中采用的基于块结构（block）的 AMR 框架—PARAMESH 为例（图 2-1）[22]，网格以块为基本划分单位，每个块由数目固定的格点组成（常见的配置为 $8^n$ 或 $16^n$ ，n 为网格维度）。网格以 2 为基数进行细分，即一个粗粒度（上层）的块可分为 $2^n$ 个细粒度（下层）的块，且每次只能细分或回退到相邻的层级。

值得注意的是这里的块结构既是基本的计算单元，同时也是 MPI 进程中的任务调度的最小单位。如图 2-1 所示，根据所用数值算法的要求，每个块的外围

都需要一定数目的卫格点（Guard Cell，并行计算中也称为 Ghost Point/Guard Point），。每轮迭代之后，除了更新格点自身携带的信息外，还要对卫格点进行更新，不在同一个进程内的块，通过 MPI 交换卫格点的信息。从图中可以看出，默认情况下一个块周围的卫格点的数目比块内的普通格点还要多，卫格点的数据结构在方便任务调度的同时，也带来了大量的内存开销，一个 MPI 进程中网格结构占用的内存可按照下面的公式估算[12]：

$$\mathrm{NUM\_VARS} \times (2 \times \mathrm{nguard} + \mathrm{nxb})^{\mathrm{ndims}} \times \mathrm{NUM\_BLOCKS} \quad \text{(Formula. 2.1)}$$

其中，NUM_VARS 为物理量数目，一般情况下在 10 到 20 之间，nguard 为一个块周围的卫格点的厚度（数目），nxb 代表一个块内部 x 方向上格点的数目（nxb=nyb=nzb），ndims 为网格维度（ndims=1,2,3），NUM_BLOCKS 为每个 MPI 进程中所能容纳的最大块数。

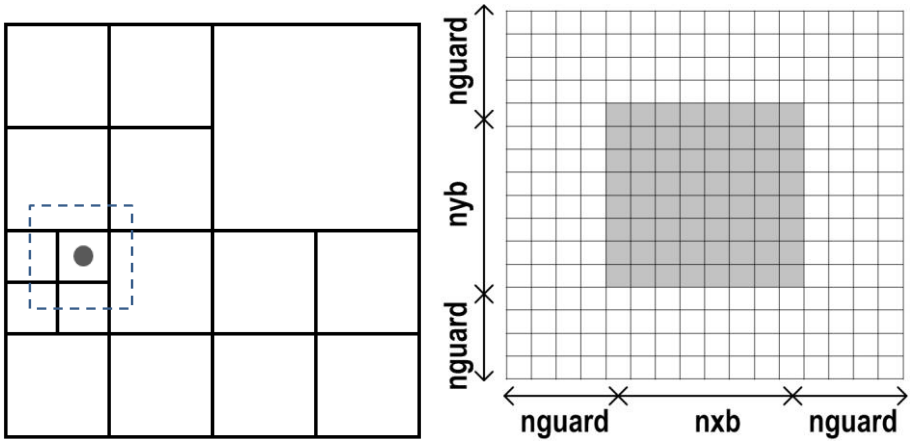



图 2-1 基于块结构的自适应网格示例。左图为一个 16 个块的自适应网格，实心圆点代表块内的一个格点，周围虚线代表该块的卫格点的范围；右图是块的内部结构，块的默认规格为 8×8，卫格点数目为每边 4 个。

### 2.2.2 偏微分方程组的解耦

非电离平衡的方程组（2-1—2-4）是由多个偏微分方程耦合而成的一个整体，考虑到统一求解这样一个非线性系统的复杂度、算法实现难度以及当前超级计算机的计算能力，适当的采用分治算法（Divide-and-Conquer）不失为一个现实的选择[23]。在数值算法中，用于偏微分方程的分治法有个专用名词，称为操作因子分解（Operator Splitting）。简单来说，就是把描述整个系统的偏微分方程组拆分成几个相对独立的小方程组或者单独的方程。一般而言，拆分后的每个方程（组）对应一个物理现象，且都有比较成熟的数值算法。

事实上，目前已知的所有天文模拟软件都采用了某种程度的操作因子分解算法。这里的某种程度视具体问题而定，例如 FLASH 的驱动程序提供了 Splitting

和 Unspliting 两类算法，用户可根据实际情况选择，但在总体框架内，FLASH 还是采用了基于时间的 Strang-split 算法[2]，将各个模块解耦并依次求解；而 Unspliting 算法仅在磁流体和气体动力学模块内支持，这意味着在 FLASH 中，如果采用 Unspliting 算法，除一般流体和磁流体外，不能再同时调用其他基于 Splitting 算法的物理模块。

Strang-split 方法基本原理如下所示：

以一个线性微分方程为例：

$$u' = Au + Bu, \qquad u(0) = u_0$$

此微分方程可以看作是由$u' = Au$和$u' = Bu$这两个不同的物理现象耦合而成的系统，不难得出上式的解析解为：$u(t) = e^{t(A+B)}u_0$

Strang-split 方法分解后

$$u_{(1,S)} = e^{\frac{1}{2}hA}e^{hB}e^{\frac{1}{2}hA}u_0$$

即在 Strang-split 的一次迭代内，在前半周期先计算 A 再计算 B，后半周期内先计算 B 再计算 A。

典型的时分迭代算法如图 2-2 所示，各个模块共享一个解空间，解空间上的每个点都是高维数据，存储着诸多物理量（密度、内能、速度等），解空间上的点与自适应网格的格点一一对应。在两个相邻（且相等）的演化步长内，各个求解器按照一定的时间和空间顺序（前半周期的空间顺序是 x→y→z， 后半周期的空间顺序是 z→y→x）进行计算，并依次更新解空间中对应物理量。

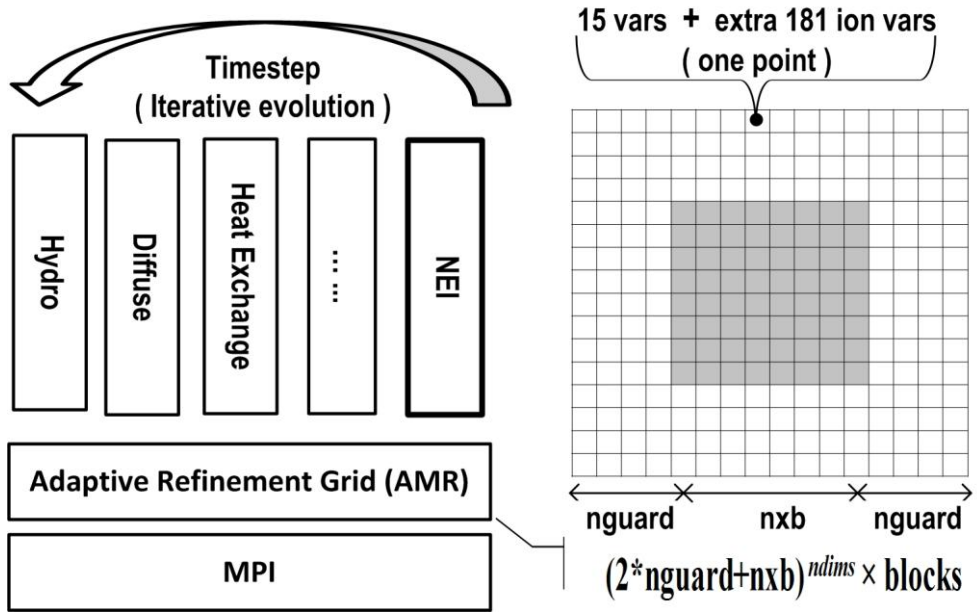



图 2-2　非电离平衡求解过程示意图。左图，欧拉网格上基于时分方法的天文数值模拟求解过程；右图，单位网格块的数据结构，每个格点（cell）都是高维数据。

### 2.2.3 非电离平衡求解器

按照上述的解耦方法，将非电离平衡部分从方程 2-4 中分离出来，形成了两个相对独立的方程组[12][21]。

$$\frac{\partial \rho X_i^Z}{\partial t} + \nabla \cdot \left(\rho X_i^Z \mathbf{v}\right) = 0 \quad \left(i=1,\cdots,N_{spec}\right) \tag{2-7}$$

$$\frac{\partial n_i^Z}{\partial t} = R_i^Z \quad \left(i=1,\cdots,N_{spec}\right) \tag{2-8}$$

其中，$X_i^Z$是元素 Z 的第 *i* 个离子的质量分量（Mass Fraction，等于此类离子的总质量除以所有元素所有离子的质量之和）。方程 2-7 是各个离子的平流（Advection）方程，方程 2-8 则是各个元素的非电离平衡的常微分方程组（ODE）。在模拟计算的每一个演化周期，平流方程可由流体求解器直接求解，如图 2-2 所示，流体计算之后，则对非电离平衡方程组进行积分。从整体看，非电离平衡方程是一个反应能量项（Reactive Source Term），在网格格点上，每个方程组都可以采用经典的 ODE 积分方法进行独立求解。值得一提的是，非电离平衡方程所表示是一个刚性的（Stiff）系统，常规的显示方法并不适用，需要采用隐式或半隐式方法，并需要对积分步长做适当的控制[24]。

- FLASH 中的解法

如图 2-2 所示，FLASH 提供的是一个整体的解决方案，在时分方法中依次调用各个模块的求解器，这些求解器通过底层网格共享同一个解空间。非电离平衡求解器利用了 Fortran 语言中久负盛名的 MA28 稀疏矩阵库，以及经典的 Bader-Deuflhard 隐式积分算法，这个组合已经被证明在求解稀疏刚性常微分方程组时是十分高效的[25][26]。

- AtomDB 中的解法

AtomDB 利用了反应系数矩阵的特征向量和特征值的方法来求解非电离平衡方程组[4][27]，其最大优势是拥有持续更新且高精准度的原子数据，这是建立高精度的非电离平衡模拟系统的前提。前面提到过，AtomDB 仅提供了对非电离平衡（方程 2-8）本身的求解，用户需要将其提供的 Libapecnei 引入到模拟程序中，这对于具有庞大代码量的天文模拟程序来说，并不是件轻而易举的工作，如果还考虑到保存必要的中间结果和从某一快照重启模拟的要求，将 AtomDB 集成到现有模拟程序的工作量还是很大的。

此外，无论是 FLASH 还是 AtomDB，在对非电离平衡方程的积分过程中，为简化计算，都将电子温度视为恒定的。

## 2.3 基于网格的性能实验与分析

基于上面的分析，非电离平衡自身(方程 2-8)的每一次求解包括两个步骤，首先根据当时的物理情况构建反应系数矩阵，然后是求解常微分方程组。值得注意的是，针对每一种元素，都要重复这两个步骤。根据在核合成计算中的经验，如果在每次迭代中对每个网格点都进行规模较大的刚性常微分方程的积分计算，这部分计算将会消耗掉整个模拟的大部分计算资源[25]。幸运的是，受到元素电离状态数（电子数）的限制，非电离平衡方程组都很小，每个方程组内的方程数量不超过 27 个（这里只考虑宇宙中最常见的 12 种元素）。实验表明，相对于流体动力学和电子热传导的偏微分方程组而言，单独求解方程 2-7 只会占用一部分时间（这个时间视具体物理条件而定，一般来说不超过总时间的 20%）。尽管单独用于求解非电离平衡的时间只是总时间的一小部分，但随着需要考虑的元素种类的增多，传统算法带来的其他开销却不容忽视。

如图 2-2(右)所示，如果不涉及非电离平衡计算，解空间上的每个格点仅需要保存 15 个左右双精度浮点型的变量（密度、速度、内能、动能等等），用于基本气体动力学的模拟，而且这些物理量也能够满足绝大多数不含非电离平衡的模拟的需要，比如热传导、辐射转移、自重力、宇宙膨胀等。一旦涉及了非电离平衡，每增加一个含有 $n$ 个质子的化学元素，就需要增加 $n$+1 个保存其离子丰度的物理量，同时增加 $n$+1 个平流方程(Eq.2.6)。如果考虑宇宙中最常见的 12 种元素（He、C、N、O、Ne、Mg、Si、S、Ar、Ca、Fe、Ni），则总共需要额外引入 181 个双精度浮点变量，外加 181 个平流方程。这些表示离子丰度的变量要在每个格点存储，这带来了近 15 倍的系统内存开销，而这些方程也要在每个格点求解，增加了数倍的计算负载。除了内存和计算量这两个显性的开销外，在自适应网格自动划分以及邻接格点数据交换时，这些新增的物理量带来的网络通讯开销同样不可忽视。

以 PARAMESH 为例，在一个典型的二维模拟中，每个 MPI 进程（一个进程对应一个 CPU 处理核心）默认处理的计算区域为 1000 个块，格点数为 16×16×1000，如果不涉及非电离平衡，这样的一个网格结构大约需要 65MB 的内存，如果包含电离平衡，内存将达到 880 MB。但如果是一个三维模拟，仅是保存解空间中必要的物理量，每个进程就要消耗 6G 内存，这时只能通过降低每个 CPU 核心能处理的子区域来分散如此大的内存压力，但这样势必会带来更严重的网络通讯开销。为了深入了解这种开销对整体性能的影响，本文设计了四组实验来验证上述的理论分析。这些实验以一个真实的二维模拟为基础，有关这个模拟的细节可参见第三章。这四组实验分别对应着基本流体模型（Hydro），流体加上电子

热传导（Diffuse），流体加上非电离平衡，以及流体加上电子热传导再加上非电离平衡。由于仅是测试性能，故每组实验都只模拟了 1000 步，图 2-3 对比了每组实验在 24、48、96、192 个核上的运行时间。

从图中数据看，有三个非常明显的结论：

1. 正如上述分析所指出的，传统的解法导致了非常严重的性能下降（大约 15 倍）
2. 热传导模块消耗了部分资源，但相对于没有非电离平衡的情况，在非电离平衡已经存在的情况下，热传导模块的加入带来了更严重的性能下降
3. 单纯的增加 CPU 个数，并不能有效的把整体运行时间降下来，包含了非电离平衡的实验在 192 个核下的运行时间比不包含非电离平衡的实验在 24 个核下的运行时间还要多出十几倍

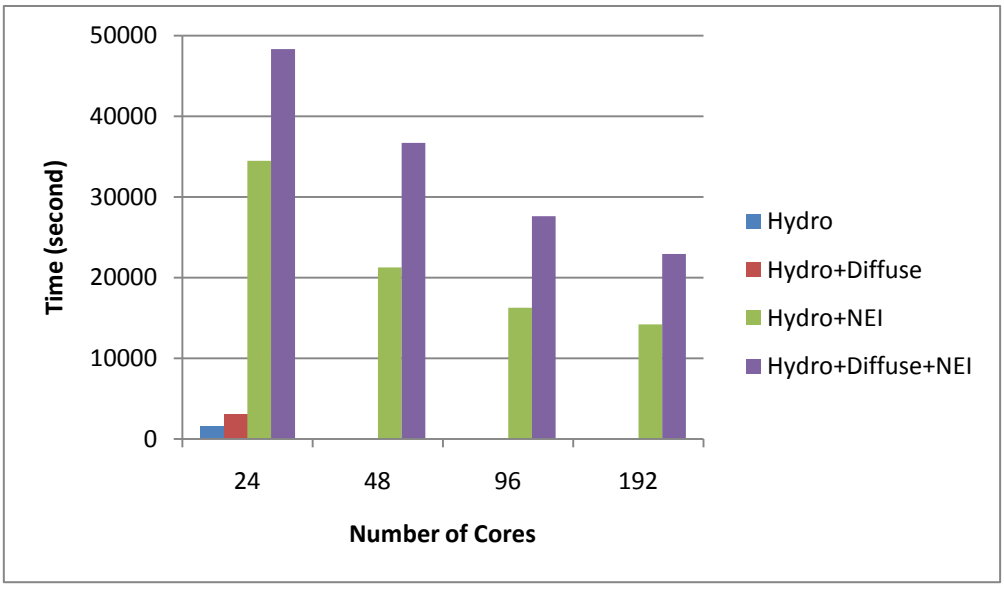



图2-3 基于网格的传统算法中，加入或不加入非电离平衡计算的性能对比。测试用例基于FLASH框架构建，每个测试的计算域相同，均运行了1000步，对不涉及非电离平衡的实验仅进行了24核的测试。

如此严重的性能下降，除了上述提到的明显原因外，还有一个原因就是天文数值模拟程序的基础代码很大，这些额外物理量和方程的引入，或多或少对很多看似不相关的代码也产生了影响。一般来讲，现实中的天文模拟都是多物理场的模拟，很少只涉及一个基本的流体动力学的模型。除了流体求解器和非电离平衡求解器外，其他物理模型的求解器（比如热传导、重力等）也不得不忍受这些跟其自身无关的开销，比如时分方法要求在每个求解器调用之后，都要同步更新网格块（block）周围的卫格点，这时大量的离子丰度数据也要被同步，即使下一个求解器仅需要基本的物理量。

这里特别要提起注意的是，上述实验仅仅测试了一个二维模拟的前1000步的运行时间，三维模拟的情形会更糟。而且在模拟演化过程中，随着自适应网格不断的细分，数据量将会越来越大，就需要投入更多的计算资源来应对这么高的负载，性能下降势必会更加严重。还有天文模拟的一个特点就是运行时间长，通常需要5万到50万步的演化，按照目前的测试结果，总的运行时间将变得不可接受。

## 2.4 多流体一致性问题

包括非电离平衡在内的反应流体的高精度模拟不仅给传统的数值算法在性能上带来了巨大压力，而且还带来了复杂的多流体一致性问题。由于多种元素的引入，在上述(2-1)—(2-4)所示的方程组中，还需要增加两个限制条件，用来描述整体一致性和局部一致性。

- 整体一致性（2-9）指任何一个格点在演化过程中的任何一步，所有不同种类的离子的质量分量之和必须是 1。
- 局部一致性（2-10）是把全局一致性应用到每种元素内，即此种元素的所有离子丰度分量之和必须等于该元素在宇宙中的丰度。(这里没有考虑核合成因素，即假定模拟过程中，各个元素的比例是固定不变的，参看图 1-1(a))

$$\sum_{Z}^{\{elements\}}\left(\sum_{i=1}^{N_{spec}} X_i^Z\right) = 1. \tag{2-9}$$

$$\sum_{i=1}^{N_{spec}} \frac{X_i^Z}{M^Z} = \mathrm{Abundance}^Z \tag{2-10}$$

其中，$\{elements\}$表示模拟中包含的所有元素，$M^Z$为元素 Z 的摩尔质量，$\mathrm{Abundance}^Z$表示元素 Z 在宇宙中的丰度，各类元素在宇宙中的丰度都是固定的，可看作常量。

整体一致性在流体动力学中经常被称作一致的多流体平流（CMA: consistent multi-fluid advection）[28]，本文中称为整体一致性是为了跟电离中的局部一致性进行对比。这里的局部一致性可以看成是 CMA 问题在非电离平衡中的加强版。这两个一致性约束保证了物质的量的守恒。在非电离平衡计算开始和结束均要进行一次一致性的检查，确保非电离平衡求解器的输入和输出都在可接受的范围内。

从理论层面上看，方程组(2-1)—(2-5)本来已经构成了一个闭合的系统，其解析解是能够自然满足这两个约束的，但由于欧拉方程的黎曼求解器本身具有很强的非线性[29]，所以数值解并不能保证很好的满足一致性约束。就目前已知情况，

针对这方面的专门研究很少，尤其是如何保证非电离平衡的局部一致性。一般的做法仅是设置一个误差上限，一旦误差过大，模拟将自动停止，用户需要调整相关参数（网格划分标准、流体演化的步长等）后重启模拟。从基于 FLASH 的测试结果看，当相对误差阈值设置为默认的 1.0e-4 时，局部一致性很容易得不到满足，尤其是在接触间断面或者物理状态变化剧烈的地方。本文也尝试了限制步长的做法，即使将步长调整得很小，也不能保证局部一致性。违反局部一致性的情况绝大部分出现在流体求解之后，非电离平衡计算之前，正如理论分析的那样，局部一致性问题主要来源于黎曼求解器的非线性特点，而非电离平衡方程的数值积分本身在绝大部分情况下却可以保证局部一致性。

流体一致性给基于网格的非电离平衡求解带来了很大障碍，因为无法从数值计算的理论层面保证一致性约束不被违反，只能通过改进黎曼求解器自身算法来减少误差。目前常见的做法是在流体求解过程中进行一个强制归一操作[12]，从而保证非电离平衡的输入数据满足一致性要求，但这样做的一个缺点是无法对最终的数值误差进行准确的估计，在很多次迭代之后，无法确保最终解的可用性。

## 2.5 优化方向

从上述的理论分析和实验结果可以看出，基于网格的非电离平衡算法之所以带来了如此大的性能开销，根本原因在于离子的数据结构跟底层网格耦合在了一起。这种耦合除了对性能产生的直接影响外，也不利于维护良好的软件结构。网格的数据结构是整个模拟程序的基础，是全局共享的，即使底层网格对上层模块提供了封装良好的接口，其他物理模块也不得不对网格的数据结构有所了解才能正常工作，这必然要影响到所有的代码，而且还增加了潜在的 Bug。

所以，非电离平衡模拟的优化首先要从非电离平衡与底层网格的解耦出发，在模拟程序的框架层次上，将非电离平衡相关的数据结构分离出来。通过框架结构上的调整，将非电离平衡计算封装为可插拔的模块，一方面直接减少了不必要的数据量、计算量和通讯量，另一方面为在某些具体方面展开更加深入的优化奠定了基础。

关注点被成功分离后，接下来的优化重点是非电离平衡计算本身。考虑到当前高性能计算机普遍采用的多核异构的体系结构，设计基于 CPU 和 GPU 的混合结构的非电离平衡求解器，才能充分发挥 CPU 和 GPU 各自的优势，从而获得进一步的性能提升。

最后，在计算性能提升的基础上，利用并行可视化、可视分析和驾驭式计算技术，加强对演化方向的人工控制，可以加快模拟向预期目标的收敛速度，进而

提高整个模拟的效率和效果。这部分涉及了对天文模拟传统工作流程的优化，通用性最强，对非电离平衡的特殊支持主要体现在非电离平衡专用的可视化设计上。

## 2.6 相关理论和技术

### 2.6.1 示踪粒子

非电离平衡与欧拉网格结构的分离，需要借助拉格朗日示踪粒子。欧拉方法能够得到某一时刻的整体网格状态，但并不能记录特定流体元的解是如何变化的；为克服传统欧拉方法的不足，示踪粒子用来在网格结构中记录朗格朗日式的演化。粒子可以看作是一个个质量恒定体积可变且不可分割的流体元，每个流体元的质量不一定相同，这取决于粒子和网格的映射算法（Particle-Mesh），流体元的质量一旦初始化，就不会在演化过程中改变。从微观角度看，一个粒子就是一个包含了固定数目的分子或原子的单元（液体或气体）。

示踪粒子是一种被动粒子，并不参与模拟演化的实际计算，也不会对模拟趋势产生任何影响。示踪粒子可以根据应用需要，携带本地瞬时的热动力学属性以及其他信息，最常见的属性包括密度、温度、速度等，从而提供系统在演化过程中任意时间点和空间点的状态，多用于某种感兴趣的物理量或物理变化的示踪剂。在欧拉网格上引入示踪粒子，需要选择合适的粒子网格映射（Particle-Mesh）算法和基于速度场的时间积分算法[20][30][31]。目前，这两类算法均有比较成熟的实现。针对于非电离平衡等反应流体的模拟，示踪粒子的优势在于能够记录完整的质点演化轨迹，这使得天文模拟能够在后处理阶段沿着轨迹上保留的物理信息，独立地进行反应方程的演算。

### 2.6.2 MapReduce

对大量示踪粒子的轨迹进行重建和分析可借助于 MapReduce 模型。MapReduce 是 Google 提出的一个软件架构，用于在分布式环境中大规模数据集的并行处理[32]。概念“Map（映射）”和“Reduce（归纳）”及其主要思想，都是从函数式编程语言借鉴过来的。简单地说，Map 函数，根据用户自定义的映射规则，把待处理的输入数据对映射成一组新的键值对，即<key, value>二元组。Reduce 函数则根据用户自定义的规约规则对 Map 产生的中间二元组数据一组一组地进行规约操作。

Map 过程针对每个输入数据独立进行，因此可以实现高度的并行化，而其产生的结果因为产生于各个节点上，天然具备并行的分布特点，同时为了 Reduce

过程的高效性，在输出时会根据二元组中的 key 值进行重新的排序、分发，以保证各个 Reduce 任务可以独立地完成于各自的节点上，避免不必要的节点间数据通信。“Map—Reduce”操作可以多次组合，进而形成一个有序的分布式工作流，这非常适合用来处理粒子轨迹这样的流式数据。

### 2.6.3 GPU 通用计算

较之于传统的 CPU，现代 GPU 都采用了高度并发的结构（图 2-4），同时在通用编程方面也有了完备的支持，计算行业正在从只使用 CPU 的“中央处理”向 CPU 与 GPU 并用的“协同处理”发展。目前，GPU 的编程环境主要有通用的工业标准 OpenCL[33]和 NVIDIA 显卡专用的 CUDA[34][35]。本文的工作内容都是基于 CUDA 实现的。相对于传统的单指令多数据（SIMD）的并行模型[36]，CUDA 提出了更加实用的单指令多线程模式（SIMT）[37]，在保证高度并发性的同时，为编程提供了更多的灵活性。CUDA 的相关测试结果显示，利用 GPU 实现的 FFT、BLAS、排序及线性方程组求解等科学计算，与单纯依靠 CPU 实现的算法相比，平均性能提高了近 20 倍[38]。

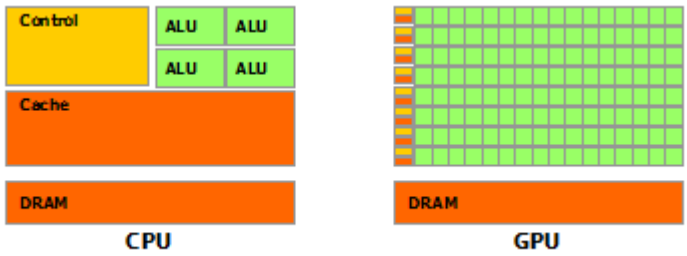



图 2-4　CPU 和 GPU 的体系结构对比[39]。

目前 GPU 上还没有比较通用的 ODE 求解器，在其他科学计算领域中，一般结合具体的问题，有针对性开发基于 GPU 的专用 ODE 求解器[17][18][40]。本文在充分分析非电离平衡方程及其计算特点的基础上，设计开发了多 GPU 和多 CPU 混合异构环境下的加速求解器。

### 2.6.4 驾驭式计算

驾驭式计算（Computation Steering）[41]赋予了用户在程序运行过程中调整参数的能力，并且通过将调整后的计算结果以实时或近似实时的方式反馈给用户，来帮助用户控制计算向着预定的目标演化。驾驭式计算本身并不是一个专门的技术，反而更像一种理念或者模式，需要借助其他具体的技术来实现。友好的人机交互界面，高维数据的展示需要可视化技术；参数空间与结果空间映射关系的探索以及对未来走势的预测，需要借助可视分析技术；大规模的科学数据的可视化

则需要综合利用图形学、并行计算的知识。

驾驭式计算一般都是和具体应用或领域紧密结合在一起的，目前天文模拟领域还没有一个比较成熟的支持驾驭计算的平台；因此，要有效地利用可视化驾驭计算技术来加速大规模的天文模拟，还需要针对天文计算的特点，在参数空间分类、天文数据可视化、以及快速反馈机制等方面进行深入研究。

### 2.6.5 组合分析

组合分析（Ensemble）[42]已经被广泛用于复杂模型的建模、参数空间探索以及参数敏感性的研究，其基本方法是基于对参数空间的有效采样子集，通过不断细化采样空间或者对大量已知解的拟合，建立起参数空间和解空间的联系，帮助用户深入理解模型，快速找到合适的参数组合。如果将组合分析与驾驭式计算结合起来，则能够为科学用户提供更加丰富的参考信息，能有效的加速模拟过程。

一般来说，对参数空间尽可能多的取样，组合分析的结果也越准确，所以组合分析更多的适用于运行时间较短的模拟。对于大规模的天文模拟，即使借助并行可视化技术，要实时反馈用户修改参数后的效果和展示组合分析的结果也绝非易事。文本结合了自适应网格的特点，提出了一种层级式的分析策略，用来优化组合分析和可视化驾驭计算在天文模拟中的应用。

## 2.7 本章小结

本章前半部分对现有的非电离平衡模拟算法进行了细致全面的理论分析，重点阐述了传统算法所带来的巨大性能开销的原因，并通过真实的模拟示例进行了验证。后半部分结合非电离平衡计算以及现代天体物理模拟面临的问题，提出了优化的层次和方向，并对相关研究工作和技术进行了简要的介绍和总结。

# 第三章 基于示踪粒子和 MapReduce 模型的计算框架

优化非电离平衡模拟的首要目标是在保证模拟精度的前提下将非电离平衡计算所带来的工作负载降至最低，即尽量避免不必要的性能开销，根据第二章的分析，可以避免或减少的开销主要包括：

1. 网格的块结构中的卫格点（Guard Cell）中保存的离子数据（质量分量）是基于欧拉方法带来的额外开销，但内部网格点用于存储离子数据的内存是非电离平衡必须的数据，无论采取什么办法，这部分占用的内存都无法避免
2. 各种离子所对应的平流方程的计算，这部分计算被流体的黎曼求解器分担，是非电离平衡在流体动力学部分引入的额外负载
3. 占据大部分内存的离子数据不可避免给 MPI 进程带来了网络通讯开销，如果前面两项开销能够减少，这部分开销也能够相应的减少
4. 离子数据同样也要保存到中间结果中，给磁盘 I/O 和存储增加了数倍的负担，如果中间结果保存比较频繁，也会带来一部分性能下降
5. 由于与底层网格的紧耦合，对其他物理过程的求解带来的影响

消除或降低这些开销的关键在于把非电离平衡部分从网格结构中独立出来，把原来基于网格的公有数据结构转移到内部的私有数据空间，最终形成一个可按需加载的模块，所有相关计算都封装其中。照此思路，具有拉格朗日特性的示踪粒子正好可以作为非电离平衡与模拟主体之间的桥梁，将彼此彻底解耦，从而能够有效的消除上述大部分的开销。

但正如任何特性都不是免费的那样，使用示踪粒子同样会面临很多新问题，好在这些问题都能找到比较满意的解决方案。本章下面的内容将就如何借助示踪粒子优化非电离平衡传统解法进行全面的论述和验证。

## 3.1 利用示踪粒子的解决方法及问题

跟基于网格的模拟一样，示踪粒子也需要定期的保存快照，只不过这些快照不单单用于中间结果分析和断点重启，其主要的目的是重建粒子的演化轨迹。一般的应用中，示踪粒子的快照间隔跟迭代步长相比可以很大，因为天文模拟的时间都很长，一般需要 10 万次数量级的迭代。如果每一步都保存一个全局的粒子

快照，对性能的影响还是相当大的，同时也给磁盘 I/O 和存储系统带来了额外的压力，大量快照文件的管理也绝非易事。所以，粒子快照操作的间隔一般根据应用的具体需求，在满足模拟精度的前提下，尽量减少快照的生成。

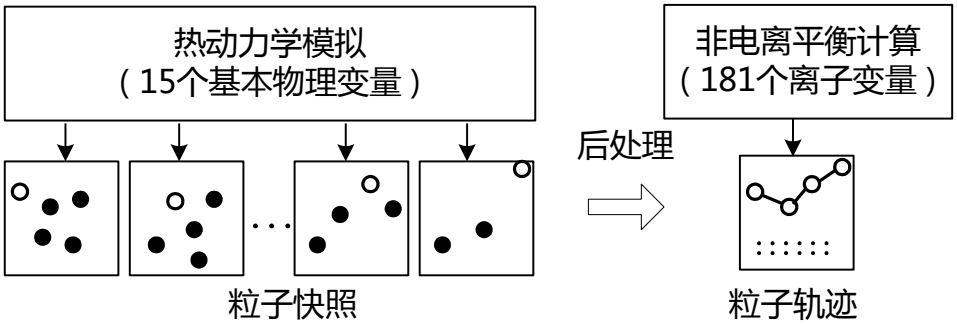



图 3-1　利用示踪粒子求解非电离平衡的示意图。随着模拟的进行，粒子可能流出物理边界，从而导致粒子数目的减少。

根据示踪粒子记录的热动力轨迹进行非电离平衡计算是可能的，这样一来，底层的网格结构就没有必要保存数倍于常规物理量的离子信息，在传统方法中的内存开销、网络通讯开销以及求解大量的离子平流方程的开销就都可以避免了。利用示踪粒子进行非电离平衡计算的基本原理如图 3-1 所示，但在非电离平衡模拟中使用示踪粒子，还要处理好下面的两个问题：

1. 为保证模拟精度，粒子的数目要足够多以及初始分布要合理，这是所有使用示踪粒子的共性问题；
2. 每次迭代结束后，都需要生成所有粒子的快照，以便后期重建粒子的演化轨迹，这样才能完成最终的非电离平衡计算。

因此，示踪粒子快照的频繁保存，大量的快照文件的管理，以及基于快照文件重构粒子轨迹都是下一步要解决的问题。同时还要注意，示踪粒子本身也会引入性能开销，所有这些都需要一个平衡性能和精度的折中方案。而 MapReduce 计算模型正好适用于解决粒子流式处理的很多问题。

## 3.2 应用 MapReduce 计算模型重构和分析粒子轨迹

近年来，MapReduce 计算模型已经被成功用于大规模的数据流处理和粒子轨迹分析。与 Google 论文中实现的用于大数据分析的 MapReduce 最大不同之处在于，高性能计算领域更喜欢流式处理模式。Ekanayake 等用 Java 语言实现了一个流式的 MapReduce 计算框架[43]，该框架不借助传统的硬盘来缓存各个阶段的中间结果，而是将数据以流的形式直接从数据的生成者（Map）发送到消费者

（Reduce），这样做消除了文件 I/O 方面的开销。在分子动力学领域，Tu 等人首次设计了一个分析 TB 级分子活动轨迹的 MapReduce 框架“Hi-Mach”[44]。Hi-Mach 显著的提高了分子结构分析的性能，它的最大特点是构建在 MPI 并行库之上，十分适用于大规模的高性能计算环境。

美国能源部 Sandia 国家实验室 Plimpton 教授的研究团队一直致力于基于 MPI 实现高性能的 MapReduce 框架。他们先后贡献了 MapReduce-MPI (MR-MPI)[45]和 PHISH(Parallel Harness for Informatic Stream Hashing)[46]两个开源框架。MR-MPI 非常适用于大规模的图论算法和分析，而 PHISH 侧重对数据分析工作流的支持，其数据流可以来自文件，也可以是上游程序产生的实时数据，PHISH 首先将工作流映射成的多级的 MapReduce 结构（包括数据处理的先后顺序，每个处理阶段要启动的任务数等），然后让数据依次流过任务拓扑结构上的各处理节点。正是受到了 PHISH 等优秀框架的启发，本文同样利用 MapReduce 模型来加速非电离平衡模拟中示踪粒子的处理。

图 3-2 描述了利用 MapReduce 模型来重构示踪粒子轨迹和进行非电离平衡计算的原理。该图的内容可以用一句话来概括，用 Map 将粒子的快照视图（横向）转换为粒子的轨迹视图（纵向），然后用 Reduce 进行非电离平衡计算。

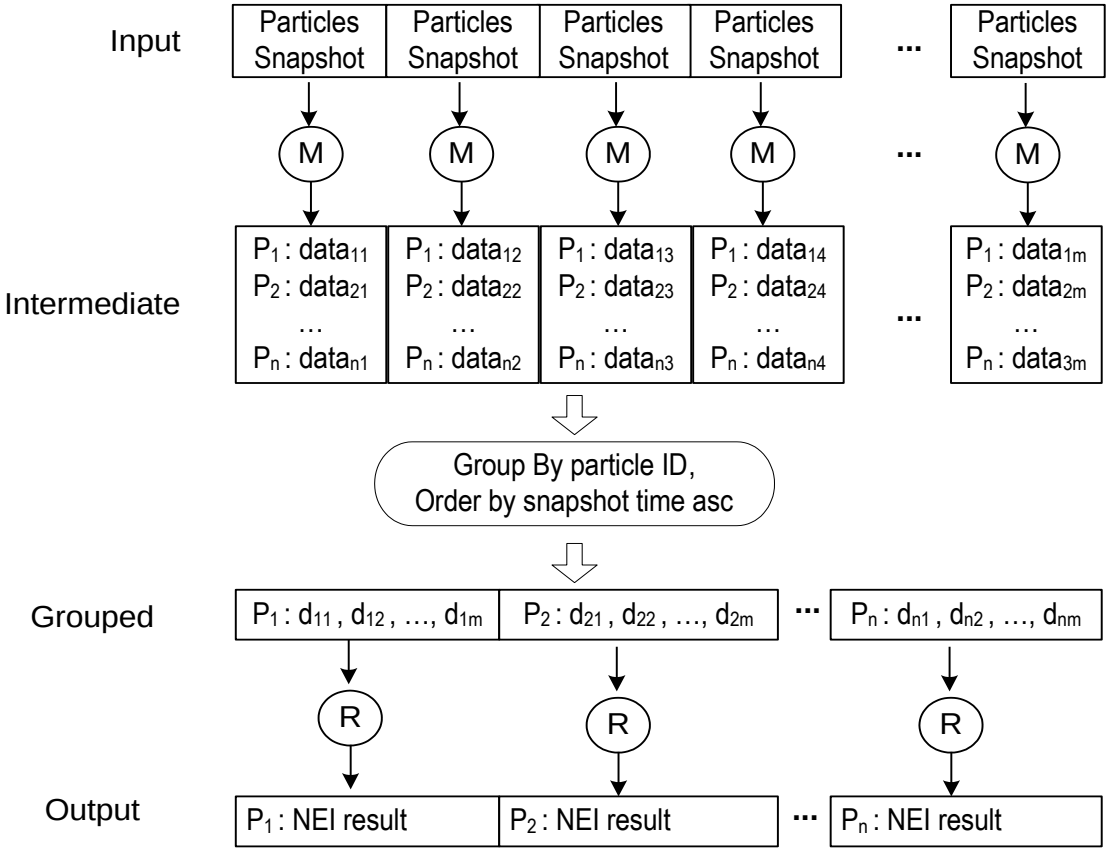



图 3-2 MapReduce 用于示踪粒子轨迹重建和分析的原理图，n 表示粒子总数，m 表示粒子快照总数，P 表示粒子，d 表示粒子携带的数据。

## 3.3 利用空间划分方法优化粒子快照生成

MapReduce 能很好的解决粒子轨迹重建的问题，但在此之前，即便对于一个需要几万次迭代的中等规模的模拟，频繁的生成大量粒子的快照也会给存储和磁盘 I/O 带来了很大的负担，势必影响性能。在经典的基于时分（time partitioning）模式的系统中，模拟过程中要进行其他操作的话，比如生成快照，保存断点等，计算本身必须要暂停去处理完这些任务，然后才能继续。近些年随着多核体系结构在高性能计算平台的加速普及，利用空间划分（space partitioning）的模式来缓解网络和文件系统竞争的方法的有效性已经在多个科研和实际的项目中得到了验证。

美国佐治亚理工和橡树岭国家实验室的李旻等研究人员设计开发了一套基于多核结构进行功能并行（functional partitioning）的运行时环境[47]，该方法的核心思想是部署一组辅助应用（auxiliary applications）运行在专门指定的核上，这些应用监听同在一个物理节点上的主程序并以主程序透明的方式完成一些辅助性工作，比如输出中间结果到分布式存储上。该方法的实现借助于用户空间文件系统（FUSE）连接主程序和辅程序，在一个拥有 160 个核的集群上，每 8 核指定一个专用核，并借助固态硬盘技术（SSD），在 FLASH 的基准测试中，完成了 40%以上的磁盘 I/O 性能提升。Dorier 博士的研究团队则更近一步，在其设计的 Damaris I/O 中间件中，利用共享内存技术，将计算与文件操作并行，避免了多核之间的同步[48][49]。该方法省去了数据拷贝的开销，特别适合于大规模模拟中的就近分析（in-situ analyses）和可视化。但 Damaris 的实现方式较为复杂，如果集成到像 FLASH 这样拥有大量基础代码的通用框架中，还需要做大量额外的开发和调试工作。美国 Argonne 国家实验室 Vishwanath 博士等设计了 GLEAN 框架[50][51]，该框架利用应用本身的数据语义和网络拓扑结构，能够有效的优化 I/O 操作并支持在模拟运行的同时进行分析，基于 FLASH 的实验表明，该框架能够大幅提高中间结果数据输出的性能。

跟上述应用情况不同的是，在示踪粒子快照输出和轨迹重建中，非电离平衡的计算是沿着不断移动的粒子进行的，并不是网格中位置相对固定的格点。因此，在借鉴上述工作的基础上，本文通过改进模拟程序的 I/O 接口，将粒子快速转移到附近的分析节点上，同样实现了在模拟过程中进行非电离平衡计算。这种方式的最大特点是实现简单，可灵活定制，易于集成。

一般情况下，当处理器个数不是很多的时候，使用并行文件系统能够提供获得良好的 I/O 性能[52][53]，然而，Fisher 等人的研究工作表明当 FLASH 模拟程序扩展到 1024 个处理器以上时，天文模拟中的常见文件格式 HDF5 以及 PnetCDF

等均不能保证并行访问效率[54]。因此，除了提供上述的实时分析模式外，文本针对后处理的情况，还提供了直接 I/O，聚合 I/O 来加速粒子快照和轨迹生成。

## 3.4 基于示踪粒子和 MapReduce 的计算框架的设计

综合考虑计算资源及存储资源的优化利用和天文学家的工作习惯，该框架支持传统的后处理（post-processing）模式，也可以支持实时或就近处理模式。这两类模式各有优缺点，可根据需要进行选择。

后处理模式，顾名思义，电离平衡在主模拟之后进行，整个过程分为两个阶段，第一阶段是模拟阶段，该阶段把非电离平衡计算从原来的模拟程序中去掉，设置示踪粒子的初始分布，然后启动模拟，在模拟运行中的每一个演化周期（迭代步长）的结尾，输出所有示踪粒子的快照。第二阶段是基于这些快照利用 MapReduce 模型来生成每个粒子的热力学运动轨迹，然后沿着各个轨迹独立的进行非电离平衡计算，最后汇总结果，进行分析。后处理模式是天文学家进行模拟实验的传统工作模式，不仅仅限于非电离平衡，一个数值模拟任务，一般要经过多次“物理建模—模拟计算—分析处理”的反复循环，才能得到预期的结果。后处理模式的优点在于所有的中间快照都可以保存下来，方便进行多次分析和重复计算，而且有些模拟具有一定的通用意义，比如宇宙湍流，但也会消耗大量的计算资源，这些模拟数据就可以公开发表，供其他科学人员进行多种用途的处理分析[54]。后处理的缺点是在模拟过程中，过多的磁盘 I/O 会带来一定的性能下降，尤其在大规模分布式模拟中，同时存储这些中间数据也会占用大量硬盘空间[55]。为了适应不同规模的模拟以及缓解频繁的磁盘 I/O 带来的性能瓶颈，本文在后处理模式中提供了三种粒子快照的输出方式。

实时模式，可以理解为在模拟运行的同时，进行分析处理。在本文的环境中，是指在主模拟程序运行同时完成对示踪粒子的非电离平衡计算，所以这里的实时模式，更准确的叫法为“与模拟同时”（Simulation Time）。正如第二章的性能分析所述，即使是实时模式，非电离平衡的计算也不能在主程序内完成，否则使用示踪粒子也就失去了意义。原因在于，如果只是简单的将非电离平衡计算从网格内迁移到在网格上面移动的粒子上，虽然避免了求解额外的平流方程，但单个粒子上同样要建立一个包含全部离子丰度的数据结构，由于携带着大量数据的粒子是不断移动的，这并不能缓解原方法中网络传输的开销，而且，在粒子与网格的映射算法（Particle-Mesh）中，还要加入非电离平衡的计算，一来容易造成程序逻辑结构的混乱，破坏了良好的模块化设计，二来示踪粒子本身对整体性能的影响很小，非电离平衡的计算量要比粒子本身的移动及映射计算大得多，而粒子在

进行相关计算的同时，整个模拟程序必须等待，所以如果将非电离平衡计算直接加到示踪粒子上，同样会极大的影响整个程序的性能。因此，本文的方法是将示踪粒子的快照信息快速地从主模拟程序中转移到附近的一个专用集群上，然后再开始非电离平衡的计算，在相关文献中，也可称为就近模式（in situ）[49][50]。实时模型的优点是能够避免与性能相对较慢的 I/O 系统打交道，大幅度地减少需要保存在硬盘上的数据量，尤其是在大规模并行模拟中，频繁的输入输出是主要的性能瓶颈[52]。另外一个好处是，可以提供对驾驭式计算的支持，用户可以根据实时分析的结果更好的调控模拟程序的运行参数和演化方向。这种方法的缺点也很明显，由于没有保存中间结果信息，所以实时的计算过程事后不能被重复，同样，事后也没有办法再进行其他的分析计算。

### 3.4.1 框架的体系结构设计

考虑到计算框架的通用性，该框架严格遵循了模块化的设计原则。如图 3-3 至图 3-5 所示，后处理模式和实时模式共享一套相同的基础体系结构，区别仅体现在部分模块的内部设计实现上，这些模块可以根据配置参数进行加载，从而决定整个系统在哪种模式下运行。整个框架有几个基本的组成模块，分别是过滤模块、聚合模块、map 模块、以及多个 reduce 模块。整个系统的设计借鉴了 PHISH 框架[46]，采用流式的结构将各个可插拔的模块连接起来，构成了一个网状工作流，数据从一个模块流到另一个模块。按照模块所在的物理位置，这些模块可以分为两大类，内部模块和外部模块。内部模块位于主模拟程序所在的集群，作为模拟程序的插件，与模拟程序实时通讯，在运行时从模拟程序中收集和整理出示踪粒子的信息；外部模块并不跟模拟程序产生直接的通讯，主要是处理来自内部模块的粒子信息，可以与模拟程序同在一个节点，也可以部署在单独的节点上。

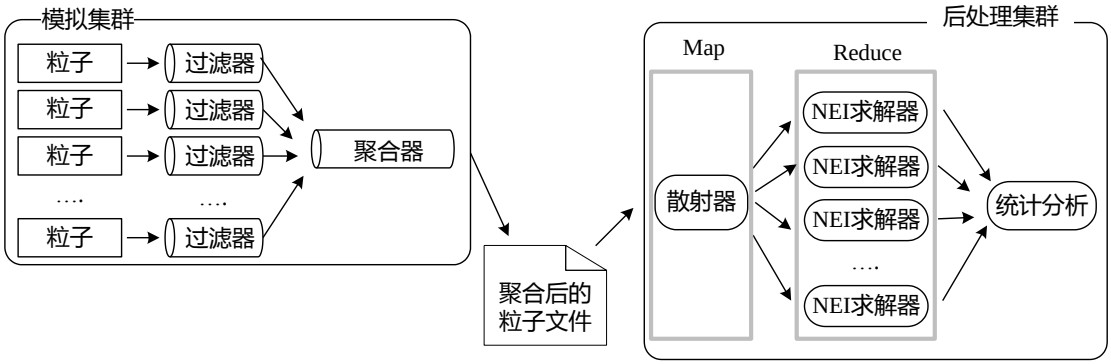



图 3-3 基于示踪粒子和 Map-Reduce 的计算框架，后处理模式（串行 I/O）的体系结构。

框架主要模块的设计思路和功能说明如下：

**过滤器（Filter）**

过滤掉不必要的数据，主要用于减少数据量。欧拉网格上的示踪粒子本身有自己的数据结构，同时，还要记录非电离平衡计算需要的信息，但其中粒子本身的一些固有属性并不是非电离平衡计算需要的，为了减少要存储和传输的数据量，过滤器只传输重建粒子热力学轨迹所必须的数据。一般情况下，一个示踪粒子的数据结构中包括 8 个基本属性，当前位置信息（x, y, z），速度矢量（x, y, z 三个方向），一个块标识记录粒子所在的网格块，一个粒子 ID（该粒子的唯一标识）。非电离平衡的计算需要再携带三个属性：温度、密度、步长，所有这些属性都是双精度的浮点数。如果对于一个中等规模的模拟来说，100 万个粒子，10 万个演化周期，将产生 8TB 的快照文件。实际上，11 个属性中，非电离平衡计算所需的信息仅有 4 个，粒子标识、温度、密度和步长。通过过滤掉其他 8 个不必要的属性，可以减少 72.7%的数据传输量和存储空间。

这里要注意的是，非电离平衡计算本身只是整个模拟的一部分，而其他信息也许会在后续的统计分析中用到，比如要显示电离状态的空间分布状况，就需要用到粒子的位置信息，如果要加上能量，就还需要速度信息。但一般情况下，此类分析都是阶段性的，不像非电离平衡计算那样，需要连续的粒子轨迹。可以通过定期保存一份带有完整信息的中间结果（checkpoint）来解决这个问题，这样统计分析可以在任意的检查点上进行，能够满足绝大部分的需求。如果要考虑实时的统计分析，框架的实时模式也提供了相应的支持（详见 3.4.4 节）。

**聚合器（Aggregator）**

聚合器只能在后处理模式下工作，主要有两部分功能，一是加强过滤器的工作，进一步减少数据体积，二是避免产生大量的小文件。天文模拟数据输出的通常模式是一个文件保存一个快照，这些中间结果文件都是自描述的，即文件包含自身的元数据，并不需要其他的配置文件或者额外信息来描述文件内容。如果按照这种默认机制，将会产生大量的粒子快照文件，同时每个文件都包含几乎相同的元数据。聚合器接收来自过滤器的原始粒子快照，合并相同的元数据和其他重复信息，源源不断地将这些已经去重的快照合并到若干个较大的文件中，以减少文件的数量和总容量。步长是非电离平衡计算所需的四个属性中的一个，但同一个演化周期内的所有粒子的步长都是相等的，为保证各个快照文件中相同的内容只会被记录一次，一个快照中只会保留一个步长，被所有粒子共享。通过上述措施，聚合后总数据量的大小比过滤器出来的结果又减少了近 1/4。值得注意的是，聚合器在合并快照文件时，这些小文件的内容要严格按照演化时间的升序排序，以节约粒子轨迹的生成时间。

**散射模块（Scatter/Map）**

该模块对应 MapReduce 中的 Map 操作，将各个粒子轨迹上的数据严格按照顺序发送给对应的 NEI 求解器，即 Reduce 操作。为简化计算，这里的哈希算法仅仅是粒子 ID 对 Reducer 数量的模数。它能够保证同一个轨迹上的信息单元都被同一个求解器处理。在后处理模式中，粒子数据来自聚合后的快照文件，在实时模式中，数据则直接来自上游模块。为了避免在系统内存中重建出完整却超长的粒子轨迹，Scatter 要保证同一个粒子信息是严格按照模拟时间的升序（轨迹的自然顺序）来发送的。

**NEI 求解器（Reduce）**

该模块对应 MapReduce 中的 Reduce 操作，对同一个粒子轨迹中包含的各个元素严格按照时间顺序计算其电离状态。由前面分析得知，非电离平衡的物理模型表示为一组刚性常微分方程组(ODE)，求解常微分方程组算法的输入有两部分，一是初始状态，二是代表反应率的系数。当前的离子状态用来计算方程组的系数，而同一轨迹上的上一个状态的计算结果便是当前的初始状态。刚性常微分方程组有很多经典的求解算法，这里采用了 Bader-Deuflhard 隐式积分算法[56]。经过 NEI 求解器处理后的结果，可以直接发给下游的模块继续进行其他计算，也可以作为中间结果定期保存到硬盘上，便于后续的分析，这比直接保存原始的粒子快照能节约磁盘空间。

正如在第二章性能分析中提到的，无论是以欧拉网格为载体，还是以拉格朗日粒子为载体，保存大量离子电离状态的数据结构都是必须的。只不过基于网格的方法在网络通讯、平流方程计算、以及保存邻居格点等方面带来了过多的额外开销。以粒子为载体，虽然避免了这些额外开销，但保存电离状态的这部分内存是无法节省的。按照常见的 181 个离子计算，对于一个 100 万个粒子的中型模拟来说，所有粒子的数据结构大约需要 1.5G 的内存。

**统计分析模块（Reduce）**

可根据非电离平衡计算的输出结果，进行一系列的统计分析，比如，电离状态的空间分布等。涉及空间分布的计算都需要位置信息，而前面提到的过滤器，为了节省存储空间和传输数据量，将粒子本身携带的空间坐标信息都去掉了。所以后续的分析一般需要借助定期保存的检查点文件（checkpoint）来获取除离子丰度以外的其他物理信息。对统计分析流程的支持详见下面的 3.3.4 节。

**配置模块**

配置模块的主要作用就是增加系统的灵活性和扩展性。整个框架是可以根据需要灵活定制的，所有这些模块都是可以组装的，模块间的组织结构以及控制参数都保存在了配置文件中。可配置的内容包括信息的流向、粒子的数据结构、过

滤器要过滤的信息、共享信息、聚合文件中包含的快照数目、以及 I/O 模式等等。

### 3.4.2 后处理模式—串行 I/O 与并行 I/O

由于后处理模式需要保存全部的粒子快照信息，尽管通过过滤与聚合机制，要传输和保存的数据体积已经减少了 80%，但对于大规模的分布式模拟，如此频繁的 I/O 依然是影响性能的主要因素之一。所以，后处理模式针对不同模拟规模的需要，提供了三种 I/O 方式，分别是针对小规模模拟的串行 I/O、中等规模模拟的并行 I/O，以及可适用于大型模拟的直接 I/O。

串行 I/O 方式如图 3-3 所示，即在每个演化周期的结尾，所有进程上的粒子信息都汇聚到主进程，主进程按照顺序合并整理后，再统一写入到文件中。串行方式实现上最简单，仅依赖操作系统本身提供的 I/O 库，但性能也是最差的。因为收发信息以及写文件都是在整个程序的演化周期内进行，写文件完成之后，整个模拟才能继续运行。如果进程很多的话，这会造成大量数据集中发送到主进程，且主进程在处理并写入这些数据时，其他进程只能等待。所以，串行方式仅仅适用于小规模的模拟。

并行 I/O，是多个进程同时将不同的部分写入到同一个文件中。要发挥并行 I/O 的最佳性能，同时需要底层的并行文件系统以及高层的并行 I/O 库的支持，此外高性能的硬件存储也是必不可少的。一般的高性能计算平台，都提供此类配置，比较常用的并行文件系统和文件格式的组合是 Lustre + HDF5。并行 I/O 方式能够在大量并发文件访问的情况下保持稳定的吞吐率，但相关实验表明，当并发进程数超过 1024 个，并行 I/O 方式也会带来严重的性能下降[54]。

### 3.4.3 后处理模式—直接 I/O

当代的高性能计算环境都是支持多用户并发访问的，为方便用户使用以及管理，大都采用了共享存储的方式。即用户空间的数据保存在安装了并行文件系统的高速磁盘阵列上，所有用户进程都可以访问这个存储，采用操作系统的安全机制来进行数据隔离。串行 I/O 和并行 I/O 的一部分甚至是大部分瓶颈也是缘于这个共享的存储，除了本用户的程序，来自其他用户的外部程序，都在竞争这个共享的 I/O 资源。

对于处理器个数成千上万的大规模模拟，为避免并发访问共享存储带来的性能损失，最原始的直接 I/O 不失为一个有效的办法。直接 I/O 是将数据写到本地硬盘，这样既不需要网络传输，也没有协同多个进程的开销，唯一的难题是如何管理分散在各个物理节点的大量小文件。可以通过后台守护进程的方式，将这些小文件合并后搬移到指定的位置进行统一管理，也可以直接利用 MapReduce 框

架，在各个节点上直接读取文件，把数据发送到分析节点上的 map 进程。如图 3-4 所示，这里采用了直接发送数据的方式。

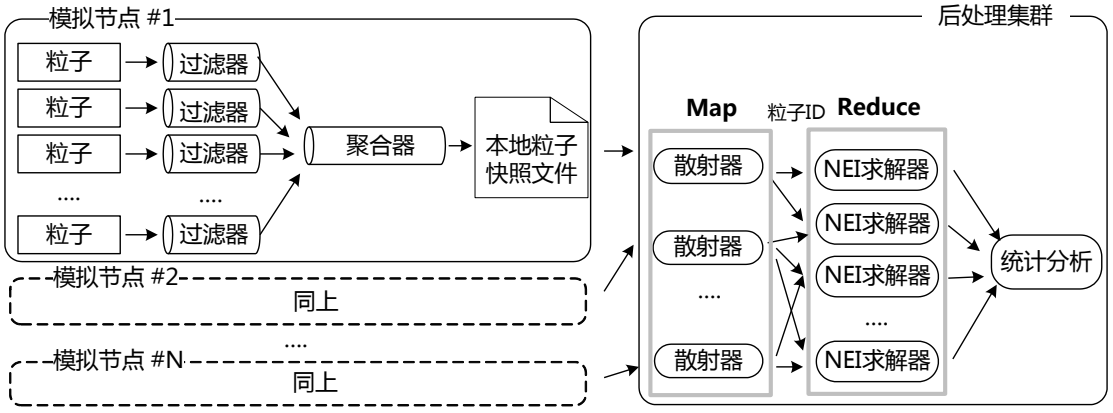



图 3-4　后处理模式—以物理节点为单位的直接 I/O，粒子快照直接输出到本地硬盘，同一个节点上，只配置一个负责数据输出的进程。

虽然直接 I/O 能够解决大规模模拟中的粒子快照生成的瓶颈，但即使在本地，多个进程同时输出文件，同样会有竞争和资源浪费。又考虑到多核结构已经十分普及，尤其在高性能计算平台上，一个物理节点拥有多颗多核的 CPU 已成主流配置。所以，借鉴 3.3 节中提到的空间划分的方法，在同一个物理节点，只配置了一个负责输出的进程，负责输出本节点内所有进程的数据，再通过共享内存技术，优化节点内的进程通讯。

### 3.4.4 实时模式—工作流

在模拟的同时进行非电离平衡计算及相关分析，可以及时了解模拟的演化情况，判断演化的方向是不是符合预期，进而对模型的参数进行有针对性的调整，控制模拟朝着预期的方向演化，这是驾驭式计算的基本思想。要使得框架能够在未来的版本中支持驾驭式计算，框架首先要在基础结构上，支持在模拟的同时进行分析，这也是实时模式的意义所在。

框架的实时模式是基于流式结构在后处理模式上进化而来，如图 3-5 所示，实时模式中省略了保存粒子快照的中间环节。粒子的轨迹在内存中隐式地生成，包含着粒子状态的数据直接发送到 NEI 求解器中进行求解。虽然实时模式中的两个运行环境，模拟环境和分析环境通过粒子的数据流联系在一起，但彼此是相互独立的，粒子信息一旦从主程序中输出，主程序就返回了，继续下一个周期的演化。而分析环境中的 NEI 求解器只是被动的接收任务（粒子的信息流），任务来了之后，启动计算，并将求解的结果发送到下游的处理单元中。整个过程都是基于数据流的，中间没有任何形式的硬盘文件存在。

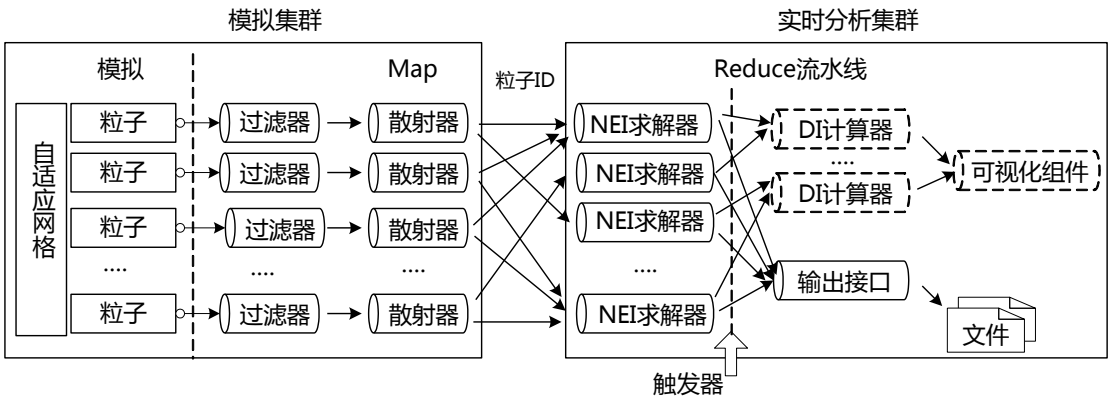



图 3-5 实时模式（simulation-time or in situ mode）的并行工作流，各个组件通过并行数据流连接在一起，中间结果可以被有选择的保存下来，以备重复利用。

与后处理模式相比，实时模式增加了一些特有的模块，说明如下：

**电离状态（DI）计算器**

非电离平衡特有的分析模块。经过 NEI 求解器之后，得到的是各个离子的丰度（abundance）信息，这些丰度信息需要跟电离平衡下的恒定状态进行对比，才能得到对应元素的电离状态，用于解释观测现象和验证模型。电离状态一般用电离偏离值（DI：Difference of Ionization）来表示。计算出的 DI 值，还要与粒子的空间信息结合，才能提供有意义的结果。这里给出的 DI Calculator，仅是一个分析模块的代表，该框架采用的可插拔的模块化结构，使得用户可以根据需要，增加任意多个分析模块，比如计算离子的发射光谱的模块，详见第四章第 5 节。

**可视化组件**

分析结果的可视化，就非电离平衡而言，电离状态的空间结构是普遍关心的内容。但这是一个高度定制的组件，不同的分析结果可能需要特殊的可视化技术才能得到最佳效果。近年来，科学可视化、可视分析已经成为一门独立的相当活跃的研究领域，本文在第五章将可视化驾驭计算技术首次应用到了非电离平衡的模拟中。

**输出接口**

输出非电离平衡计算的结果。这个设计可以看作是后处理模式的延续，在实时模式中，原始的粒子快照文件是全部丢弃的，为了继承后处理模式中数据持久性的优点，同时又要避免原始数据占用大量的硬盘空间，所以有选择性的保存处理后的数据是一个不错的折中办法。

在实时模式中允许输出中间结果到硬盘，还有更深层次的考虑。毕竟，实时模式是一个基于流式数据的框架，这种框架一个明显的缺点是，因为数据只在内存中缓存，所以数据流经的各个计算节点的处理速度是要匹配的，否则，一旦堆积的待处理数据超过了内存限制，很可能会造成整个工作流的崩溃，而数据都是

实时计算出来的，模拟程序只能从头再来，也许在这之前，整个程序已经稳定运行了很长时间，从头再来将浪费大量的计算资源和人力资源。所以，适合实时模式的所有组件的计算量要保持大致相等，这可以通过为不同的模块调配不同的计算资源来保证。但有些分析计算确实比较耗时，不适合被装配到实时工作流中，而这些保存下来的中间结果就可以用于这类离线的耗时处理。

中间结果的输出还可以作为检查点，用来重启模拟。这个框架的设计目标之一就是能够与通用的模拟软件配合，支持大规模长时间的非电离平衡数值模拟；目前流行的通用模拟框架都支持断点重启的功能[6][7][12]，所以此处的设计，也是为了保证该框架和与其连接的主程序在断点重启上的兼容性。这个设计还有一个优点，作为检查点的中间结果的输出操作，可以在较大的时间间隔内进行，比起频繁地保存原始的快照文件所带来的磁盘 I/O 压力，这对系统性能的影响很小。

**触发器**

主要用于控制可视化和统计分析的频率，以及数据流的同步操作。除了非电离平衡计算本身(NEI Solver)外，其他的分析和显示只是针对当前结果的某种计算，并没有像粒子轨迹那样的前后依赖关系，所以每次分析和可视化都是独立的。此外，一般来说，这类全局的统计和可视化操作耗时相对较长，而且，连续进行这类操作也没有实际意义，因为相对于计算步长绝大部分的天体演化都是非常缓慢的，即使跨越多个步长的两个结果也几乎看不出差别。所以，为了节约计算资源，这部分功能设计为按需调用。触发器可以按照预定的间隔来调用其后续的分析和可视化模块，而且，在调用之前，要保证后续操作所需的数据已经准备好，并确保相关模块得到的是一个一致的数据视图。触发器的参数通过本节中提到的配置模块来管理。

## 3.5 基于示踪粒子和 MapReduce 的计算框架的实现

本文具体针对的是天文数值模拟中如何高效的求解非电离平衡问题，同时兼顾类似的反应流体模拟。在实际中，现代天文模拟往往都是多物理场的模拟，非电离平衡的求解也总是与其他物理问题的求解一同进行。天文模拟的实现涉及高性能计算、天体物理、数值计算、可视化等多个领域的知识，多年下来已经积累了大量的可重用代码和几个较为通用的框架[6][7][13][14]。所以如何将文中的方法以非侵入（nonintrusive）的方式应用到已有的模拟程序是实现阶段的主要目标。这样，天文学家就能以非常小的编码代价在他们熟悉的模拟环境中引用本文的方法。

### 3.5.1 基于 FLASH Code 引入示踪粒子功能

目前，该框架是基于 FLASH code [6]实现的，但这主要是为了测试的需要，框架本身对 FLASH 的依赖很少。FLASH 的最大优点在于物理与网格的分离，这意味着使用者只需要知晓实现物理过程的相关机制，而计算相关的部分，如何差分、如何划分解域网格以及如何实现并行运算等等复杂问题，都留给了计算机领域的专家来解决。这也使得 FLASH 可以方便的引入新型的自适应网格算法来提高计算精度和性能。FLASH 的代码主体是用 Fortran90 以及 C 语言编写的，其中还有少量的 Python 脚本，当前的最新版本是 4.2.2 版。FLASH 是目前应用最广泛的一个开源的天文数值模拟框架，自 2.5 版本起，就提供了基于网格的非电离平衡模块；但直到 3.3 版之后，才将拉格朗日粒子引入到了其基于欧拉网格的架构中。FLASH 也提供了粒子快照的保存功能，采用了一个快照一个文件的方式，但就目前所知，FLASH 的最新实现版本并没有针对非电离平衡问题，对如此频繁的粒子快照的输出做特别的优化[57]。

FLASH 并没有借助编程语言来实现面向对象的功能，而是采用了基于目录层次结构的继承机制，在编译准备阶段，下层目录的代码会覆盖上层目录中同名的代码。这样，如果要修改默认的实现，可直接在优先级高的目录中，将新算法写到一个与旧算法同名的文件中即可。按照这种思路，本文为 FLASH 中的粒子模块的输出接口提供了一个新的实现，用于与框架的连接，该实现将粒子的信息直接发到框架的过滤器模块（Filter），经过过滤和聚合，最后将粒子快照保存为 HDF5 格式。

### 3.5.2 基于 MPI 实现 MapReduce 模型

由于几乎所有的大型天文数值模拟都是在专用的高性能计算平台上完成的，Map 和 Reduce 之间也无需使用文件进行信息交换。这里采用了 MPICH2[58]来实现文中提出的 MapReduce 模型，除了原始的粒子快照，所有的数据都以流的形式在 MPI 进程间传递。在运行时，框架的每一个组件的每一个实例都被表示为一个 MPI 进程。通常一个进程要处理多条粒子轨迹，粒子通过某种散列算法被发送到对应的 MPI 进程，这里的散列算法是对粒子编号和 Reduce 进程个数执行模运算，以确保同一个轨迹中的粒子信息都被发送到同一个进程。

为了保证对原有模拟程序的影响最小，还要做好主程序的 MPI 运行环境与框架的 MPI 运行环境的隔离。本框架的使用，不能影响到 FLASH 程序中原有的各个 MPI 通讯域（MPI_Comm），否则将会造成整个程序的混乱。所以，本框架采用了独立的 MPI 上下文，借助 MPI 提供的 client/server 模式，FLASH 的主程序可以向该框架内部的任何一个 MPI 进程发送消息。此外，由于粒子轨迹的生成

必须按照严格的时间顺序，实现上利用了在 MPI 的消息同步机制，从而避免了轨迹中出现乱序。

## 3.6 实验和评价

为了全面的测试本章提出的基于示踪粒子和 MapReduce 进行非电离平衡模拟的方法的性能及精度，此处选取了一个实际的天文模拟（W49B）作为测试的基本用例，并且在后面的章节中，也会反复用到这个模拟。选择这个测试用例的原因有两个，一是 W49B 是近年来天体物理领域研究的热门天体之一，观测结果显示其中的 Fe 元素有非常明显的过电离现象，模拟 W49B 也就有了实际意义；二是本文课题的合作单位—紫金山天文台对 W49B 的辐射电离机制已经有深入的研究，从而能对整个框架的实用性以及测试结果进行比较全面和科学的评价。

### 3.6.1 测试用例

测试用例的目标天体是代号为 W49B 超新星遗迹，关于 W49B 的天文现象以及模拟结果的物理含义可参考文献[59]，这里仅对相关概念和物理模型进行说明。

- 超新星爆发：是某些恒星在演化接近末期时经历的一种剧烈爆炸。
- 超新星遗迹：超新星爆发会将大部分甚至几乎所有物质以十分之一光速的速度向外抛射出，并向周围的星际物质辐射激波。这种激波会导致形成一个膨胀的气体和尘埃构成的壳状结构，被称为超新星遗迹。

W49B 是目前发现的最年轻的超新星遗迹，虽然被称为遗迹，但其在很长一段时间内依然发生着物理和化学变化。W49B 的观测结果和物理模型如下图所示。

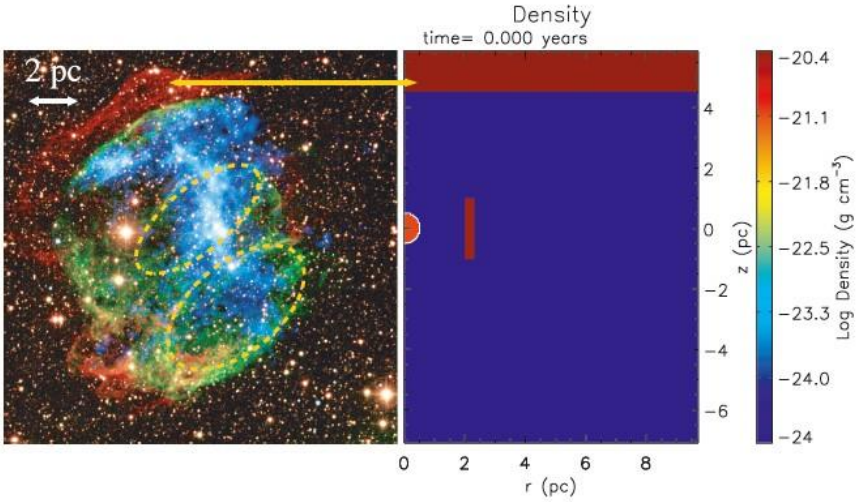



图 3-6　W49B 的观测合成图像（左）和简化后的初始物理模型（右）[59]。

左图是钱德拉 X 射线太空望远镜(Chandra X-ray Observatory)拍摄的 W49B 的照片（处理后），图中红色部分表示一层致密的分子云，绿色部分也是一层致密的分子云（密度比红色部分稍低），绿色分子云呈现出柱状结构（图中已用黄色的虚线勾画出来），并且内部中心为密度稍小的爆炸遗迹，其他部分可以看作是密度非常低的星级介质。由于左图在整合的过程中被翻转成了“上东右北”，实际方向与常规地图的方向标注并不相符。

右图是左图简化后的二维流体动力学模型对应的初始状态，即遗迹被高速抛出之前的状态。横纵坐标表示的都是距离，单位是秒差距（英文缩写为 *pc*，1*pc*= 3.26164 光年= 3.08568e18 厘米），颜色值则表示密度。简化后的模型是一个轴对称的圆柱体，在这个简化的模型中，左图中的致密的红色分子云被简化成了均匀分布的高密度分子墙（右图中的红色区域），而密度稍小的绿色分子云被简化成了一个均匀的高密度分子环。初始状态下，遗迹的大部分物质分布在中心的一个半径为 0.5*pc* 的球形区域内，图中的绝大部分空间是稀薄的星际介质。利用柱面坐标的对称性，右图实际上展示的是这个圆柱体的半个轴切面，一个二维的矩形（9×12*pc*）。

为简化起见，此处的物理模型中，只涉及了三个物理过程：方程组(2-1)—(2-3)描述的气体动力演化过程，方程 2-4 描述的非电离平衡现象，以及方程 3-1 描述的电子热传导过程。加入电子热传导的原因有两个，一是热传导在实际的 W49B 演化过程中确实起了很重要的作用，二是为了说明非电离平衡对其他物理过程模拟性能的影响。电子热传导的方程如下：

$$\rho C_{\mathrm{v,ele}} \frac{\partial \mathrm{T_{ele}}}{\partial t} = \nabla \cdot K_{\mathrm{ele}} \nabla T_{\mathrm{ele}} \tag{3-1}$$

其中，$\rho$表示流体密度，$t$代表时间，$C_{\mathrm{v,ele}}$是电子的比热（specific heat），$K_{\mathrm{ele}}$是电子的热传导系数，需要根据实时的物理状态进行计算得出。电子热传导是扩散（diffusion）方程的一个退化形式，是典型的抛物线型偏微分方程，需要用到隐式算法。其核心是对一个稀疏的线性系统（AX=B）求解，线性系统的大小与网格分辨率成正比。求解热传导的详细过程可参考文献[12][60]。

**测试环境：**

一台拥有 10 个节点的 NUMA 集群，其中，一个节点用来任务提交和系统管理工作，一个预留做专用的实时 NEI 求解和分析，余下的 8 个节点用于运行加入了示踪粒子的主模拟程序。

每个物理节点的配置如下

处理器 ：4 颗 Intel Xeon X5650，2.6GHz，6 cores/CPU

内存 ：48GB

**测试方法：**

实验分为 5 组，分别在 12，24，48，96，192 颗处理器核心上运行，欧拉网格的初始大小为 16×16×20k，示踪粒子均匀地分布在 0.5×0.5*pc* 的区域（遗迹中心所在区域，其他区域放置粒子与否对结果无影响），数量为 100 万。由于实验最关注的是性能，又鉴于实际的 W49B 模拟比较耗时（在 8×8×5k 的初始分辨率的情况下，仅仅包含了 Fe 元素共 30 个离子，72 颗处理器核心连续运算了 20 天，共 40 万步，模拟了 2400 年间的演化过程），故每组实验都只运行了 1000 步，运行时间从 0.5 到 25 个小时不等。这五组实验分别为：

1. 不涉及非电离平衡的基本热动力学模拟，标记为“AMR without NEI”
2. 使用传统方法进行的非电离平衡模拟，标记为“AMR with NEI”
3. 后处理模式串行 I/O 方法，标记为“Particle serial I/O”
4. 后处理模式直接 I/O 方法，标记为“Particle direct I/O”
5. 实时流处理模式，标记为“Particle in situ”

### 3.6.2 性能测试结果及分析

下图展示了每个实验的耗时对比。本文提出的计算框架的三个模式拥有相似的性能表现，都是在 48 个核的情况下，耗时最少，之后随着处理器的增加，性能开始下降并出现了加剧的趋势。类似的性能下降现象在基础实验用例（AMR without NEI）中更为明显，这反映了并行计算领域中的一个基本的规则，对于给定的负载，仅仅通过增加计算资源（处理器）并不能获得成比例的性能提升，相反，并行化带来的开销可能会抵消甚至超过其带来的加速效果。由于实验环境的限制，测试用例的规模只能被限定在这样一个普通的二维模拟的范围。

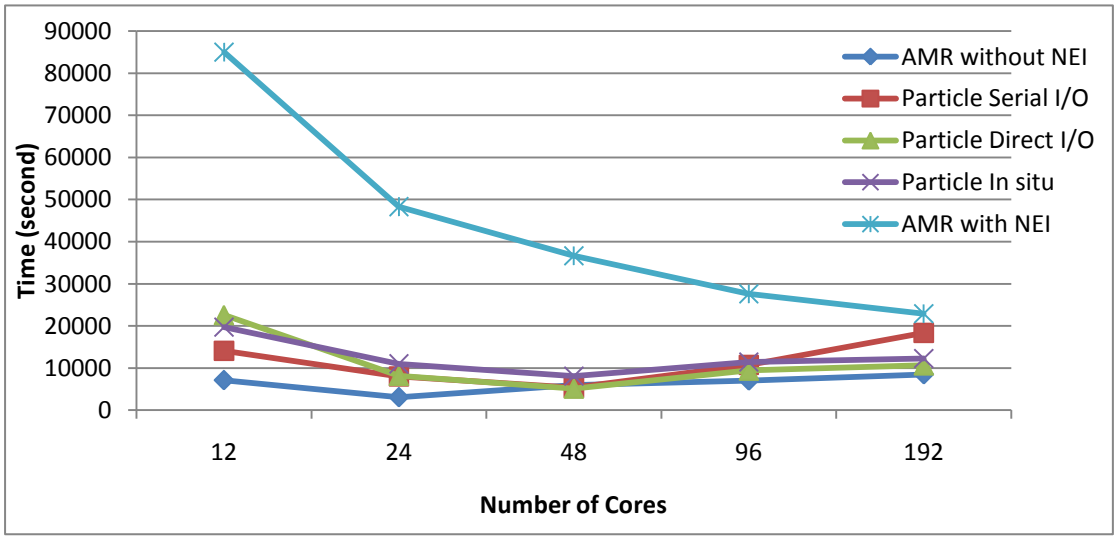



图 3-7　性能测试结果，纵轴表示运行时间，横轴表示处理器数。

从上图中可以明显看出，正如前面对传统算法的分析所说，由于非电离平衡与底层欧拉网格的紧耦合，传统算法（AMR with NEI）至少造成了一个数量级以上的性能下降。虽然处理器数量从 12 增加到 192 的过程中，总时间从 85k 秒下降到了 22k 秒，但这仍然是没有加入 NEI 的基础实验在 24 个核时的 6 倍。仅仅因为加入了非电离平衡计算，在 8 倍计算资源的情况下，却才达到了原来 1/6 的性能。从性能曲线的变化趋势中可以看出，随着处理器的增多，加速比逐渐趋于平滑，这不难说明，仅仅通过增加处理器数量，并不能有效提高传统算法的性价比。实验结果也充分验证了理论分析，传统算法并没有充分考虑非电离平衡模拟中的实际问题，仅仅是在功能上实现了非电离平衡计算，而忽略了实用性要求。与传统算法在 192 个处理器时取得的最好成绩相比，直接 I/O 模式仅用了一半的计算资源，却完成了 4.5 倍的加速；而对于实时模式，最大加速比约为 3。但这里需要声明的是，实时模式里已经包含了对非电离平衡的计算，而后处理模式仅仅是将粒子快照保存。

这里值得着重强调的是，运行时间长是天文数值模拟的一个显著特点，例如，按照此实验的参数推算，W49B 的完整模拟在 192 个核上至少需要运行 28 天。本文方法仅用 1/4 的计算资源获得了 3 倍的性能提升，在实际运行中，必将节约大量的人工成本和计算资源。实验数据仅仅测试了最初 1000 步的模拟情况，随着时间推移，自适应网格将不断细化，按照 W49B 的完整模拟数据，模拟结束时的网格精度是初始时的 10 倍左右，工作负载和内存也将逐渐增长到这个比例。这就说明在真实的模拟中，端到端的性能提升要大于此处实验所显示的 3 倍，如果使用同等的计算资源（以图 3-7 中的 48 核为例），加速比接近 6。

### 3.6.3 粒子引入的开销分析

从性能曲线图上可以看出，向欧拉网格中引入示踪粒子所带来的开销相比于传统算法要低得多。示踪粒子的主要开销包括以下四部分：

(1) 粒子快照文件的输出

(2) 粒子在欧拉网格上的移动

(3) 确定下一步的步长时，需要计算粒子移动范围的限制

(4) 依据粒子分布的网格自动调整

图 3-8 详细展示了模拟程序的各个主要功能的耗时对比，所有的实验均在 96 个处理器上完成。结合性能曲线图，对于后三种开销，无论在理论层面，还是在实践层面，本文框架提出的三种模式的表现十分接近，这在很大程度上要归功于 FLASH 框架本身对拉格朗日粒子算法的完美优化[31]。

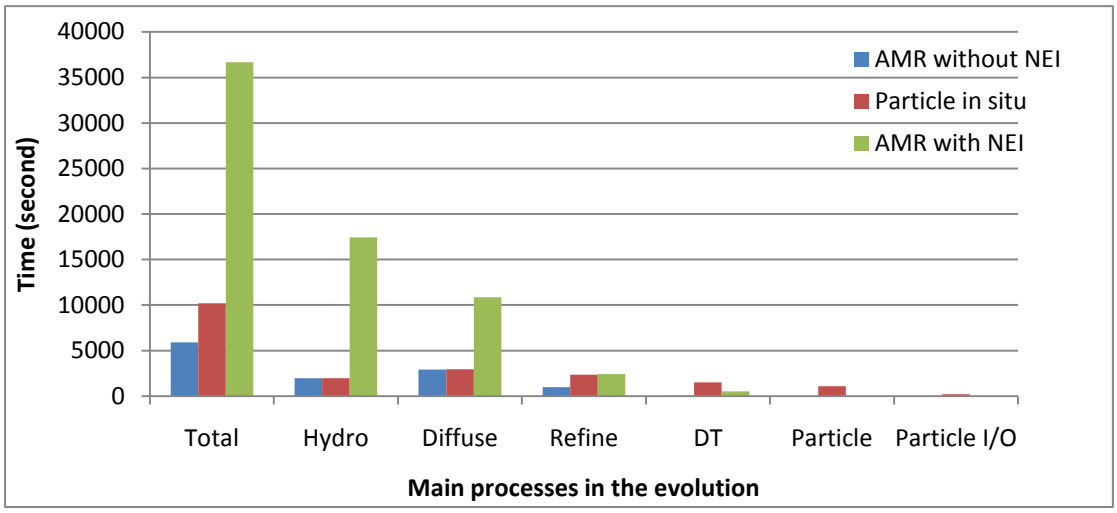



图 3-8　开销统计结果，横轴表示模拟中的主要过程，纵轴表示耗费的时间。Hydro：流体；Diffuse：热传导；Refine：网格自适应；DT：步长计算；Particle：粒子移动；Particle I/O：粒子快照输出。

对于要着重关注的大量粒子快照生成的开销，从性能曲线图上可以清楚看到，随着处理器数量的增多，进程之间在通讯和同步方面的开销也逐渐增加，不可避免的限制了串行 I/O 模式的水平延展性（scalability）。而在直接 I/O 模式和实时模式下，几乎可以忽略粒子快照输出对性能产生的影响。另外，从非电离平衡的传统算法与基本模拟（AMR without NEI）的比较中可以看出，非电离平衡带来的开销使得流体求解器与热传导求解器的运行时间大幅增长，总的运行时间增加了 600%之多；而本文提出的方法中，上述四种主要开销的总和也没有超过原始运行时间的 50%。

**后处理模式的额外开销**

在后处理模式中，保存大量的粒子快照所需的硬盘空间是最主要的开销。通过过滤和聚合机制，确实能够节省大量的硬盘空间。实验表明，对于演化了 10k 步的 100 万个粒子，文中的方法仅产生了 230G 的文件，以此类推到演化周期为 100k 左右的一般情况，大约需要 2.3T 的存储空间。

需要特别指出的是，按照本文的方法，非电离平衡计算是与主模拟程序是分开的，但在上述的性能和开销分析中，还没有包含非电离平衡的独立计算部分。按照实验数据，对于一条长度为 1000 的粒子轨迹，一个进程需要大约 0.2 秒完成全部的非电离平衡计算，以此类推，处理 100 万个同等大小的粒子轨迹，大约需要 $0.2 \times 10^6$秒。不同于传统方法，由于各个 NEI 求解器是沿着独立的粒子轨迹进行计算的，进程间不需要交换粒子信息，所以随着处理器数量的增加，本文的计算方法可以获得几近理想的线性加速比。在 192 颗处理器上完成 100 万个长度为 1000 的粒子轨迹的处理，实际耗时为 1160 秒。加上性能测试中的最优结果

（5120 秒，由直接 I/O 模式在 48 个核的环境下完成），后处理模式能够得到的最大加速比为 3，这与实时模式几乎一样。

### 3.6.4 关于准确度的讨论

传统方法的非电离平衡计算是在网格点上进行的，文中方法将非电离平衡计算迁移到了示踪粒子上。无论是哪种方式，鉴于物理模型总是真实世界的简化，这里的准确度只能是相对而言的，本节只讨论两种方法的差别，而不是哪一种更能反应真实情况。

理论上，基于示踪粒子的方法能不能最大限度的重现基于纯网格的方法的结果，主要取决于网格与粒子的映射算法和粒子前进的积分算法。由于这部分内容不是本文研究的重点，所以，在当前的实现版本中，直接利用了 FLASH code 提供的粒子模块[30]。FLASH 中提供了四种粒子移动的积分算法和两种粒子与网格的映射算法，文本采用了推荐的显示两步龙格库塔积分（Two-Stage Runge-Kutta）与二次差值映射（Quadratic Mesh Mapping）的组合。

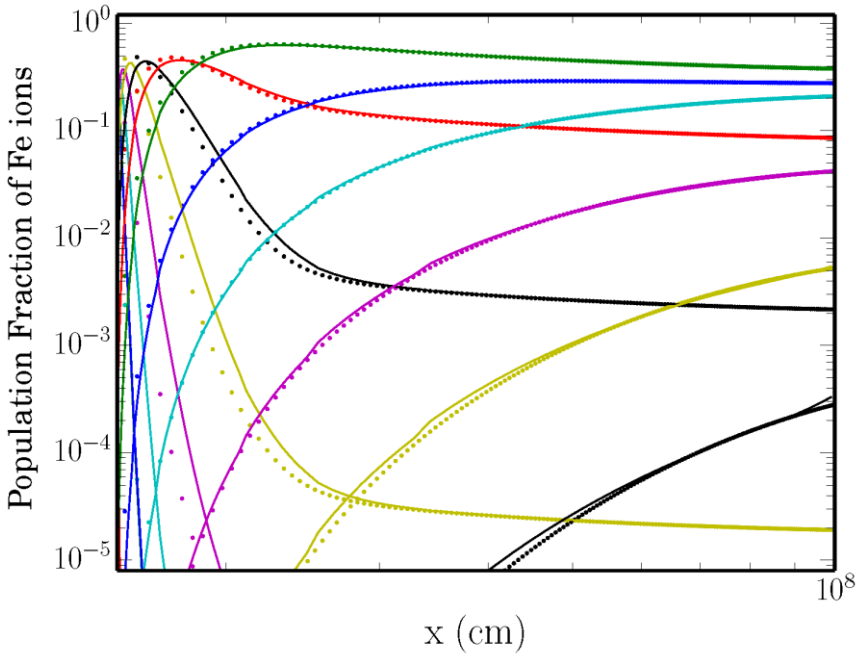



图 3-9 基于传统网格与示踪粒子进行的非电离平衡模拟结果对比。实线表示基于网格的计算结果，虚线表示基于粒子的计算结果。

上图对比了基于相同初始条件下两种方法的计算得出的铁元素(Fe)的离子丰度（经过了归一化处理），较大的差别出现在接触间断面或者是物理量变化剧烈的地方，但大部分情况下，两个结果的曲线都吻合很好。

FLASH 中的粒子移动算法都是经典的基于局部速度矢量场的积分方法，但

也有最新的研究成果指出，对于复杂的流体现象，比如湍流，基于本地速度场的方法并不能正确反映出底层网格的质量流，而基于蒙特卡洛的粒子方法可能更胜一筹，但这种基于概率的算法，需要用大量的粒子数来抵消该算法本身固有的数值扩散（numerical diffusion）作用[20]。

### 3.6.5 实时模式下的可视分析

基于粒子流和 MapReduce 的框架除了在性能上的贡献外，还为传统的天文模拟流程增加了实时分析的能力，用户可以在模拟程序进行的同时及时掌握演化趋势。本节演示如何将非电离平衡的计算结果与网格的空间结构相结合，从而对 W49B 中 Fe 元素的电离状态进行分析。元素的电离平衡偏移指标 DI（differential ionization）定义如下[59]：

$$\mathrm{DI} = \langle \mathrm{l} \rangle_{\mathrm{simulation}} - \langle \mathrm{l} \rangle_{\mathrm{equilibration}} \tag{3-1}$$

其中，$\langle \mathrm{l} \rangle = \frac{\sum(\mathrm{f_i} \times \mathrm{l_i})}{\sum \mathrm{f_i}}$表示某一元素以离子丰度为权重的平均电离水平，$\mathrm{l_i}$代表第 $i$ 个离子的电离级别，$\mathrm{f_i}$代表该离子在元素中的数量分量（population fraction）。等号右边公式的两个下标分别代表实际的模拟值和电离平衡状态下的理论值。当 DI>0 时为过电离（over ionization），DI<0 时为低电离（under ionization）,DI=0 是电离平衡状态。

按照框架的模块化设计，很容易将一个 DI 计算器和可视化组件插入到如图 3-5 所示的并行工作流中。每个 DI 计算器负责一部分粒子，将结果发送到可视化组件，可视化组件负责将 DI 值与空间位置对应起来，图 3-10 显示了实时渲染出来的 Fe 元素电离状态的空间分布。

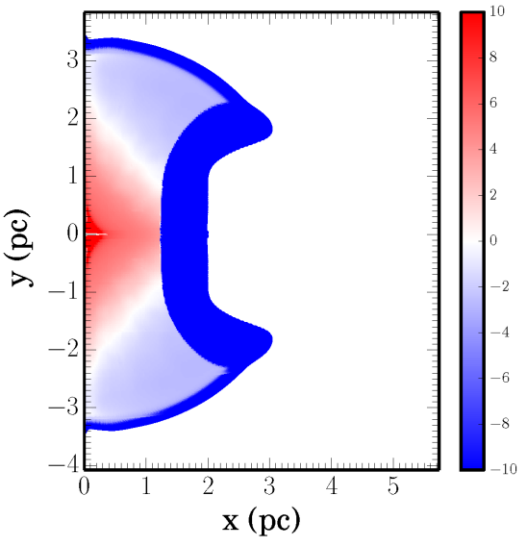



图 3-10　基于示踪粒子计算的电离状态的空间结构（演化了 631 年的 W49B 中的 Fe 元素的电离状态分布，红色代表过电离态，蓝色代表低电离态）。

### 3.6.6 多流体一致性问题与性能的权衡

第二章指出了像非电离平衡这样的反应流体模拟中，不可避免的会遇到多流体一致性问题。采用基于粒子的方法之所以能在一定程度上有效的绕开这个问题，根本原因在于作为基本流体元的粒子省略了元素相关的平流方程的求解，而同时把元素相关的其他计算封装到了粒子内部。这相当于回避了经典的黎曼求解器由于数值耗散问题造成的多流体质量不守恒[12][28]，但这并不意味着在基于粒子的非电离平衡模拟中多流体一致性问题就完全消失了。

在本文提出的非电离平衡的方案中，从流体动力学层面上看，基于示踪粒子的算法成功的避免了多流体一致性问题造成的模拟无法继续的问题，但在每个粒子内部的非电离平衡的计算中，同样要考虑这个一致性问题。现有的常微分方程数值算法并不能保证满足方程 2-9 和 2-10 规定的一致性约束。如果积分步长过大，依然会带来较大的数值误差，层层累积的误差将导致模拟结果的不可信。目前采取的办法是限制步长。在模拟的时间尺度固定的前提下，步长与计算量成反比，所以一般来说，步长越大，总的计算量越少，模拟速度越快，但精度越差，反之减小步长，就会带来计算量成倍的增长。所以限制步长，就需要在性能和精度之间找到一个合适的平衡点。本文设计了一系列小实验来辅助步长的选择，根据实验结果，非电离平衡的步长（积分区间）最好不要超过$1.0 \times 10^4$秒，但这同时也就限制了主模拟程序的步长也不能超过这个阈值。

但实际模拟的绝大部分情况下，流体计算所需的最小步长都比这个阈值要大得多，这导致了要维持非电离平衡所需的精度，流体、热传导等其他部分不得不小步前进。一个解决办法就是在非电离平衡的计算中，支持子步长模式（sub cycle），即“流体一大步，电离数小步”。这样，流体和其他部分可以采用相对大的步长来计算以节省资源提高速度，而在非电离平衡中，需要把其他模块通用的大步长切分成大小适中的子区间，然后在每个子区间内，依次计算。通过这种方式，基本上能够把该框架的性能优势最大限度的发挥出来。

## 3.7 框架对其他问题的适应性

该框架良好的模块化设计和可插拔式的结构，使其能够比较容易的应用到其他反应流体的模拟中。在实际应用中，要发挥框架的最大效用，一般还需要针对具体的问题，进行特殊的优化并配合合适的折中策略。下面以超新星爆炸中常见的核合成（nucleosynthesis[25]）模拟为例，来说明框架的适应性。

非电离平衡中的元素丰度是恒定，变化的只有离子丰度，而核合成是原子与

原子的结合，从而使元素的丰度处于不断变化中。为便于描述，这里只是采用了举例的方式来说明核合成的反应网络的形式，并不深入探究其复杂的推导过程。

$$\dot{Y}(O^{16}) = +Y(He^{4})Y(C^{12})\,R + \cdots \tag{3-2}$$

其中，Y 表示对应同位素的摩尔丰度，上标表示对应元素的质子数，$\dot{Y}$表示 Y 对时间的导数，R 为总的反应速率，标识核反应进行的快慢，省略号表示后面还有别的类似项，这个方程式的表面意思是：

氧-16 丰度变化率 = 氦-4 和碳-12 合成氧的速率 + 其他元素合成氧的速率。这里的 R 跟非电离平衡中的$R_i^Z$含义类似，都需要根据当时的物理条件通过实时计算得到。

将该框架用于核合成的计算，需要进行三个准备步骤：

(1) 将核反应网络的求解器封装到一个规约器（reducer）中

(2) 设计一个能过滤掉不发生核反应的粒子的过滤器(Filter)

(3) 通过小规模的测试，估计出实时模式所需的计算资源

由于针对大量同位素的反应方程的建立过于复杂，而目前也没有良好结构的代码用于参考，所以，本文仅针对 13 个同位素的核合成模拟进行了测试。实验的目的主要是测试方法的适应性，重点不是性能。测试结果表明采用文本方法的核合成计算约获得了 30%的性能提升。这个结果跟同位素的数量较少有直接关系，因为如此小规模的核反应网络对性能的影响本身就不大。

在现实中，对于一个典型的带有热核反应的超新星模拟[19]，其计算规模和拉格朗日粒子数都要超过上例 W49B 一个数量级左右。描述核合成的常微分方程组也比非电离平衡复杂许多，反应网络的规模也较大（最多达到 489 个同位素，即 489 个方程，非电离平衡中，最大的方程组仅包含 29 个方程），求解起来也会更耗时[25][26]。根据该框架在非电离平衡模拟中的表现，以及其良好的适应性，作者有理由相信其在大规模的核合成反应中也会有不错的表现。

## 3.8 不足与改进

该框架目前在实现上还存在两处明显的不足，需要在以后的工作中不断完善。

### 3.8.1 物理上的不足之处

物理上，并不是自洽的（self-consistent）。这表现在由电离和复合作用产生的能量并没有反馈到能量方程（方程 2-3 中方括号表示的项）中。在基于欧拉网格的传统算法中，由于所有的求解器都共享同一个网格结构，这一点不难做到，

非电离平衡计算中产生的能量直接写入到底层网格的共享数据结构中，可以被其他模块直接取到，从而及时调整相应的能量项，保证能量守恒。但基于示踪粒子的方法，将非电离平衡的计算与底层网格彻底隔离开来，数据的交换是单向的（网格→粒子→快照），这里没有提供一个有效的回路能够将非电离平衡阶段产生的能量反馈至网格。

即便在实时模式中，要完成这个操作也是困难重重。在图 3-5 所示的框架中，如果将分析节点上的能量项反馈到模拟节点上的网格中，有两个难题：

(1) 程序同步的代价。能量回馈的过程中，主模拟程序必须要暂停，等待分析结果返回到对应的粒子上，然后再通过粒子与网格的映射机制，将粒子接收到的能量项反向映射到网格中，然后主程序才能继续。这样势必造成计算资源的浪费。

(2) 有效的反馈回路的建立。MapReduce 模型本身的数据流就是单向的，加之示踪粒子又是不断移动的。在基于 MPI 的实现中，能量项要返回到对应的粒子上，必须要知道该粒子所在的进程号，才能把信息发回去。这就需要维护粒子到进程的映射信息，无形中又增加了计算量。即使有了这样的映射结构，能量项也只能是发送给对应的进程，进程还要对其所有的粒子进行遍历才能定位到具体的粒子。如果采用群发机制，则势必大大增加网络通讯的负担，基本上不可行。

**实际中的注意事项和解决办法**

对于非电离平衡来说，在某些情况下，电离作用产生的能量是可以忽略的，比如在光学薄（optically thin），这时使用本文提出的方法是安全的[21]。如果基于粒子轨迹的计算产生的能量项不能忽略，则不能简单的引用该框架，否则由于能量守恒不能保证，主程序每一步的计算结果都会有偏差，这个偏差也会传播给粒子，这会造成每一步的计算结果都不准确，越到后期偏差越大，从而无法反映出模型的真实情况。

一个可行的折中办法是，对于这类伴有物理或化学反应的计算，可引入一种修正机制，将能量相关部分单独建模，使用某种近似的冷却(cooling)或加热(heating)模型来表示能量项，在每一个演化周期的末尾，把这个估计值反馈到能量方程中。这个过程全部是基于欧拉网格完成的，跟粒子没有直接关系，修正后的结果会经过网格间接的反应到粒子所携带的物理属性上，从而保证每一步的粒子快照都是正确的。比如，在 W49B 的模拟中，天文学家就设计了一个冷却函数来表示辐射电离过程中的引发的能量损失。在核合成中，核反应产生的能量是绝对不能忽略的，这需要采用一个合理的核燃烧模型来得到能量项的近似值[19]。

### 3.8.2 性能上的改进空间

无论是非电离平衡，还是核合成，由于粒子数目巨大且运行时间长，虽然针对一个粒子的计算微乎其微，但累积消耗的时间也是非常大的。目前，在基于粒子的计算任务的分发上，该框架采用了将粒子轨迹平均分配到每个 reducer 的方法。该方法简单易行且表面上公平，但如果针对具体问题，却不一定是合理的任务调度方案。

一般物理或化学反应都要满足特定的条件，最常见的比如温度和密度要在一定范围内，这就很有可能造成在某些轨迹上真正发生反应的点很少甚至没有，也就不会有对应的计算，所以框架缺省的 scatter 算法就无法提供良好的负载平衡。造成负载不均的还有一种情况，就是在演化过程中，粒子可能会飞出模拟区域，从而后续的计算也就不需要了。对于前一种情况，可能需要针对具体问题，采用适当的过滤机制，去掉无关的粒子；而后一种情况，则需要针对当前有效的粒子数，来重新分配任务。

无论哪种情况，具体优化的实现都不轻松，尤其是在实时模式下，因为根本没有办法预先判断一条轨迹从头到尾都不需要计算，这个调度只能是动态的。但这里的任务是以轨迹为单位的，每条轨迹的计算是要连续的，每个任务也都是要持续到整个模拟结束的，所以调度的时候还要进行任务迁移，任务分配算法还要维护一个额外的庞大任务列表，用来确定粒子要发往的目的进程。一个可行的办法是采用粒度更大的任务单位，利用现代高性能计算平台多核结构的特点，将原来多个进程分别处理的任务打包成一个物理节点的任务，节点内部采用共享内存和轻量级的线程来统一处理这些任务。

## 3.9 本章小结

本章主要针对基于欧拉网格的非电离平衡模拟在算法设计上的缺点及其所带来的严重性能下降问题，通过示踪粒子将底层网格与非电离平衡计算彻底解耦，并借助 MapReduce 模型来支持后续的粒子轨迹的演算，消除了现有方法在内存、计算和网络通讯上的额外开销，达到了优化程序结构和提升性能的目的。实验表明，在相同的实验环境下，该方法仅使用了原有 1/4 的计算资源，就完成了至少 3 倍的加速。

本章的方法不仅仅完成了性能上的提升，还为非电离平衡模拟的整个生命周期提供了一个可扩展的流式框架，为后续的优化工作奠定了基础，同时对其他反应流场模拟的优化也具有很好的参考价值。

# 第四章 基于 CPU-GPU 混合异构系统的非电离平衡求解器

上一章提出的基于示踪粒子和 MapReduce 模型的框架将非电离平衡计算从基于欧拉网格的主程序中分离了出来，这种粗粒度的并行，在性能上取得了很大的提高，同时也为单独优化非电离平衡的常微分方程组求解提供了条件。本章将在更细的层面上，针对当前高性能计算平台普遍采用的 CPU-GPU 混合结构，研究求解非电离平衡的刚性常微分方程组的优化机制。

## 4.1 非电离平衡方程的特点和求解过程

常微分方程的求解一般基于数值差分方法，大致分为显示、隐式和半隐式三种形式。显示算法仅根据当前状态直接计算未来的状态，典型代表为向前欧拉算法[61]，而隐式算法则需要求解一个包含当前状态和未来状态的方程，典型代表为向后欧拉算法[62]。显而易见，隐式算法比显示算法需要额外的计算量，在实现上也比显示算法要复杂得多。针对刚性方程，显示算法需要非常小的步长才能保证数值稳定性，这造成了迭代次数急剧上升甚至无法在实际中使用。相比之下，隐式算法则可以采用较大的步长，计算次数会大大减少，最后总的计算时间可能要远小于显示算法。因此在实践中需要根据问题类型来选择合适的算法。现实中隐式方法要求解的方程往往都是非线性的，可以利用牛顿法将非线性方程进行线性化来简化计算，这称为半隐式算法[24]，半隐式算法是求解刚性常微分方程时被广泛采用的方法。

本节结合非电离平衡方程的特点，指出传统的基于 CPU 的解法在性能上的局限性，并给出基于 GPU 进行并行优化的思路。

### 4.1.1 非电离平衡方程的特点

根据第二章的分析，与气体动力学的黎曼求解器解耦后的非电离平衡模型可以表示为如下形式（方程 2-6 与方程 2-8 两式合并）：

$$\frac{\partial \mathrm{n}_i^{\mathbf{Z}}}{\partial t} = \mathrm{N_e}\left[\mathrm{n}_{i+1}^{\mathbf{Z}}\alpha_{i+1}^{\mathbf{Z}} + \mathrm{n}_{i-1}^{\mathbf{Z}}\mathrm{S}_{i-1}^{\mathbf{Z}} - \mathrm{n}_i^{\mathbf{Z}}\left(\alpha_i^{\mathbf{Z}} + \mathrm{S}_i^{\mathbf{Z}}\right)\right] \left(\mathrm{i}=1,\cdots,N_{spec}\right) \tag{4-1}$$

其中，$\mathrm{n}_i^{\mathbf{Z}}$代表元素 $Z$的第 $i$ 种离子的数量密度（未知量），$t$ 表示时间，$N_{spec}$表示元素 $Z$ 的所有电离状态数，$\mathrm{N_e}$代表电子的数量密度，$\alpha_i^{\mathbf{Z}} = (\mathrm{N_e}, \mathrm{T})$代表碰撞和双

电子复合系数， $S_i^{\mathbf{Z}} = (\mathrm{N_e}, \mathrm{T})$则代表碰撞电离系数。非电离平衡方程式的特点总结如下：

1. 每种元素都对应一个形如方程 4-1 的常微分方程组（ODE），元素的电离状态数（质子数+1）决定了方程组中的方程个数；
2. 方程组等式右边（RHS：Right Hand Side）的系数$\alpha_i^{\mathbf{Z}}$和$S_i^{\mathbf{Z}}$由电子的数量密度和温度决定，需要实时计算，即除方程求解本身外，方程组的建立同样需要消耗计算资源；
3. RHS 的系数$\alpha_i^{\mathbf{Z}}$和$S_i^{\mathbf{Z}}$对电子温度和密度的变化比较敏感，在同样的物理条件下，不同离子对应的系数可能相差好几个数量级，故该常微分方程组是刚性的；
4. 不管元素有多少个电离状态，每种离子的浓度（质量分量）仅取决于其自身及其相邻的两种离子，所以方程组的系数矩阵是稀疏的。

为保证算法的效率和精度，非电离平衡求解器必须要兼顾上述的四个特点，有针对性的进行设计：

1. 元素质子数限制了其对应的 NEI 方程组的大小，如果仅考虑宇宙中最常见的元素，则每组非电离平衡方程中不超过 30 个未知变量；
2. 从严格的物理意义上考虑，RHS 系数的确立同样需要一定的计算，也是影响性能的主要因素之一，在实际应用中，可采用某种的近似算法来获得精度和性能上的一个合适的折中；
3. 刚性 ODE 方程组一般要考虑使用隐式算法，相对于显示算法，隐式算法虽然计算量大，但能保证数值计算的稳定性；
4. 考虑到 RHS 的稀疏性，可考虑利用稀疏矩阵技术来进一步提高性能。

### 4.1.2 ODE 隐式算法的一般结构

隐式算法是解决刚性常微分方程的首选，其一般的表达形式如下所示[24]：

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \tag{4-2}$$

$$\mathbf{y_{n+1}} = \mathbf{y_n} + h\mathbf{f}(\mathbf{y_{n+1}}) \tag{4-3}$$

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

其中，方程 4-2 为原方程，方程 4-3 为原方程的隐式差分形式，方程 4-4 则是基于牛顿法将方程 4-3 进行线性化之后得到的半隐式形式， $y$ 为未知量，$\frac{\partial \mathrm{f}}{\partial \mathbf{y}}$为方程等式右边（RHS）的偏导数矩阵（雅可比矩阵），$h$ 为步长。

在实际计算中，一般采用方程 4-4 所示半隐式形式，当 $h$ 较大时，算法内部还需要经过多次迭代。为了进一步提高性能，工程中使用的算法还需要有步长的自适应机制和数值误差控制机制，整个计算的主要包括以下三个部分：

1. 驱动部分(driver)：主程序，循环调用 stepper，直到结束。
2. 单步积分(stepper)：实现步长自适应和误差控制逻辑，并调用 algorithm 来完成具体的积分计算，由于有步长自适应和误差控制，这个过程可能要重复多次，直到得到可接受的计算结果。
3. 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，RHS 的值，RHS 雅可比矩阵操作，以及将上述两个计算结果代入特定算法的差分公式。

### 4.1.3 基于 CPU 的解决方法和面临问题

目前，刚性常微分方程组的求解有许多经典的算法，其中的很多算法已经包含在基于 Fortran 或 C 的数值函数库中，比如 GNU Scientific Library（GSL）[63]，劳伦斯利弗莫尔实验室（LLNL）的基于 C 的 CVODES[64]以及基于 Fortran 的 ODEPack[65]，Intel 公司开发的 Math Kernel Library（MKL）[66]，同时剑桥大学出版的专著《Numerical Recipes》[24]中对许多重要的 ODE 求解方法都有详细的论述。上述的函数库都支持显示算法和隐式算法。

Timmes 早在 1999 年，就对求解核合成反应方程的性能问题做了比较全面的研究，针对 3 个经典的半隐式算法和 8 个广泛使用的线性代数函数库，在 24 个不同规模的方程组下，分别在性能、内存、精度等几个方面进行了测试，得出的结论是，在大部分情况下基于 Fortran 的 MA28 稀疏矩阵库和 Bader-Deuflhard 隐式积分算法是一个比较好的选择[56]。核反应和非电离平衡在方程形式上是相似的，因此，在 FLASH 框架自带的电离模块中，利用了 Timmer 推荐的组合[25]；而 AtomDB 则采用了一种基于特征向量和特征值的隐式解法[67]。

实际上，目前基于 CPU 的各类算法在性能和数值稳定性上都已经比较成熟，能够在相对较短的时间内求解大规模的 ODE 方程组。同时可以肯定的是，对于非电离平衡方程的个体而言，上述的各种求解器都会有不错的表现。但由于非电离平衡方程组的规模并不大（<30），也就不涉及大规模的矩阵运算，所以影响性能的关键因素并不是单个方程组的规模，而是求解的次数。

在非电离平衡演化过程中的每一步，需要针对空间中的每个点的每个元素进行 ODE 求解， 考虑到宇宙中的最常见元素，则每个点上有 12 个 ODE 方程组，每个方程组中的变量数从 3 到 29 不等。基于 CPU 的算法是一个典型的二重循环：

*外层：对于网格中的每一个点（或者每一个粒子的快照）*

*内层：对于每一个元素，构建并求解对应的刚性 ODE*

并行的方法仅是对空间进行分割，每个处理器负责一块子区域。根据实验结果，即使采用 Timmes 给出的最优组合[25]，非电离平衡方程求解本身也会占用整个模

拟 20%左右的计算资源。

从并行角度看，大量同一形式的常微分方程求解是一个典型的单指令多数据流的形式。虽然现代 CPU 都已采用了多核技术，但对于大量简单的重复劳动却并不擅长，求解时间将随着求解次数呈线性增长。而较之于传统的 CPU 结构，GPU 则更适合于多数据流的密集型计算。此外，隐式算法中不可避免的涉及了大量的矩阵运算，这对传统 CPU 结构也是一个巨大的挑战，而矩阵运算也是 GPU 的主要优势之一。

## 4.2 GPU 加速方案的分析

### 4.2.1 非电离平衡 GPU 加速方式的选择

根据 4.1 节的分析，非电离平衡隐式算法的主要耗时部分包括 RHS 的求值、雅克比矩阵的求值以及矩阵运算。结合 GPU 的体系结构特点(参考图 2-4)，不难得出基于 GPU 的算法可以有两种通用模式，如图 4-1 中的(a)和(b)所示。

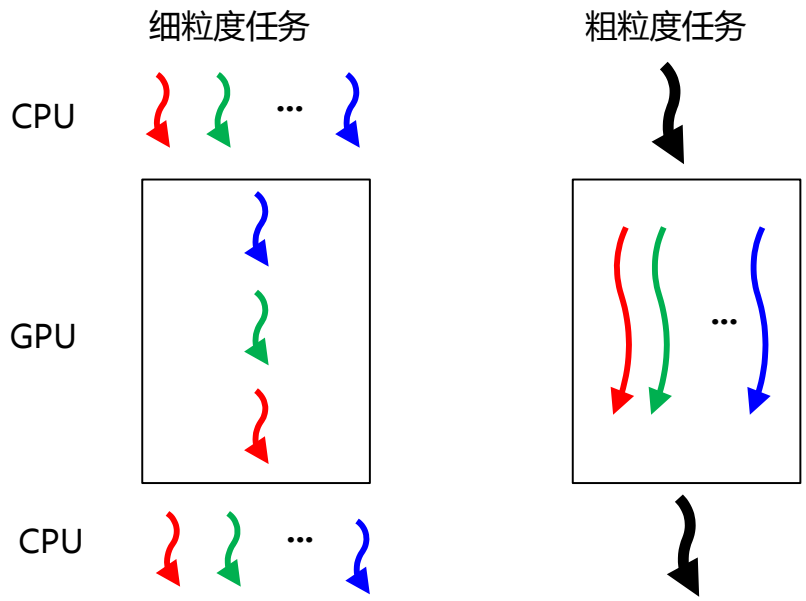
**方式(a)、重点移植**

为了充分发挥 GPU 在矩阵计算方面的优势，仅将比较耗时的矩阵相关计算转移到 GPU 上，包括雅可比矩阵反转、LU 分解等操作。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线性代数工具库，比如 MAGMA[68]、cuBLAS[39]、MTL4[69]、ViennaCL[70]、VexCL[69]等。该方法的优势在于实现简单，只须将现有函数库中的矩阵运算抽取出来后委托给 GPU 即可，而不必了解 GPU 的矩阵运算的编程细节。

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。

**方式(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 ODE 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，可以同时对多组 ODE 方程求解。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矩阵运算，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 ODE 的求解，GPU 的每个线程都是一个独立的 ODE 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。




(a) 重点移植模式　　(b) 整体移植模式

图 4-1 非电离平衡在 GPU 上的两种加速方式。

**两种方式的选择**

方式(a)除了实现简单外，其最大优势在于能够充分利用 GPU 在处理大规模矩阵时的计算优势。但就非电离平衡而言，矩阵的秩一般不会超过 30，所以此优势并不明显。但在核合成反应中，一般有数百个未知数，矩阵计算的优势就能得到充分的体现。另外，该方式中 CPU 和 GPU 之间的交互相对频繁，且 GPU 在进程上下文切换时也需要延时。由于 CPU 和 GPU 之间数据带宽的限制，主机与设备之间内存拷贝的总量和频率都是影响整体性能的关键因素。因此，对非电离平衡而言，方式(a)的整体优势并不明显。

当矩阵规模比较小时，可以考虑算法的整体移植，即采用方式(b)。但方式(b)有两个主要缺点，首先，GPU 在处理控制结构方面并不擅长，而隐式算法又相对比较复杂，分支控制语句较多，这将大大限制 GPU 并发的效率；其次，GPU 的每个流多处理器（Streaming Multiprocessor）能够利用的高速内存有限，按照最新的 Kepler 架构，一个流多处理器可以包含 192 个处理器(core)，这些处理器共享 64k 的高速内存（共享内存）。因此，方式(b)实现起来比较复杂，也无法胜任较大规模的方程。

综合上述分析，又考虑到非电离平衡的方程规模不超过 30，方式(a)的优势无法充分发挥，且主机与设备之间频繁的内存拷贝会导致较长的延时，所以本文优先采用了方式(b)，但同时也针对非电离平衡的情况探索了如何提高方式(a)的效率。

### 4.2.2 混合结构下的负载平衡策略分析

要充分发挥 CPU-GPU 混合异构体系的整体优势，必须考虑 CPU 和 GPU 之间的任务调度机制。目前比较流行的做法是，将大任务分解成的小的原子过程，建立 CPU 和 GPU 的性能模型（硬件相关），根据性能模型估算出 GPU 和 CPU 的任务分配比例，此类性能模型都是动态的，在运行过程中，会根据以往的调度结果进行自我调整，不断优化[71]。

Alejandro 等人针对多个异构的 GPU 开发了一个动态任务调度函数库，该方法可支持消息传递和共享内存两种模式，可依据 GPU 的计算能力按比例的分配任务，特别适用于配置了多个计算能力不同的 GPU 的系统[72]。在任务划分比较均匀的情况下，该方法能获得最佳性能，其局限性是目前仅支持 GPU 之间的负载平衡。

国防科技大学的刘杰研究员在考虑了实际中计算空闲时间的波动情况，即多个处理器即使在相同的任务量下完成时间也总会有小幅波动，而 GPU 的时间波动相对于 CPU 则要小得多，然后提出了一种基于负载平衡因子的任务分配算法，该算法的核心思想是如果空闲无法避免，则尽量不要闲置具有较高计算能力的处理器[73]。 该算法在大规模蒙特卡洛计算的实验中，总体资源利用率达到了 98.72%。

上述的任务调度策略主要是针对容易并行的计算问题，对 CPU 和 GPU 各自的擅长领域考虑较少。对于非电离平衡中的刚性 ODE 求解而言，矩阵规模小且隐式算法逻辑复杂，并不利用 GPU 作业。Yu Shi 等在化学动力学的燃烧模拟中提出了一种基于方程刚性程度的调度算法，通过预先判断待解方程的刚性程度，将中等刚性的方程分配到基于 GPU 的显示算法上处理，而将高度刚性的方程交给基于 CPU 的隐式算法处理[18]。该方法考虑到了 GPU 和 CPU 在分别在数据处理和逻辑处理方面优势，也简化了 GPU 上算法的实现过程，显示算法相对隐式算法容易实现，更适合 GPU 处理。

此外，为了提高 GPU 的利用率，NVIDIA 也提供了一个支持多进程的服务环境(MPS：Multi-Process Service)。MPS 能够将 GPU 的资源进行分割，不同进程提交的任务可以并发的执行，但其基于 Client-Server 模式的架构，在频繁的任务调度时，会带来不少的额外开销。

相对于上述的研究工作，非电离平衡计算任务的一个显著特点是单个任务都很小，但总的任务量十分巨大，任务调度会相当频繁。这要求在设计任务调度策略时要选择合适的任务粒度，尽量减少调度次数，最小化 CPU 与 GPU 之间的通讯开销。

### 4.2.3 基于 GPU 的 ODE 求解器

目前 GPU 上还没有相对通用的 ODE 求解器，但已经出现了多个线性代数函数库，比如基于 C 的 MAGMA[68]，基于 C++的 MTL4[69] 、ViennaCL[70]和 VexCL[69]，以及 CUDA 自带的 cuBLAS[39]等。利用 C++语言的模版和元编程技术[74]，应用程序可以将这些函数库嵌入到 CPU 上的通用 ODE 求解器中，从而快速实现向 GPU 的移植。ODEINT[75]就是这类通用求解器的一个典型代表，ODEINT 提供了封装良好的 ODE 算法骨架，将数值算法本身与求解的方程彻底分离，应用程序不必关心算法逻辑，只需实现待解方程本身的计算即可，包括 RHS 求值、矩阵运算等，而这部分计算完全可以利用 GPU 上现成的函数库[39]。从形式上看，此种实现方式与方式(a)一致（图 4-1(a)），比较适合大规模 ODE 的求解，对于大量小规模的非电离平衡方程的加速效果并不明显。

在其他科学计算领域中的 ODE 求解，由于具体问题的特殊性，一般根据需求有针对性进行基于 GPU 的优化。比如 Yu Shi 等在多维燃烧模拟中，考虑到不同 ODE 刚性程度的比例特点，为了最大限度的利用 GPU 的并行性，就采用了快速的显示求解器，将占大多数的中等刚性程度的方程交给 GPU 来解决[18]。Fred 等在基于 GPU 的心脏细胞模拟优化中使用了相对简单的五点向后欧拉算法[76][77]。 此外，还有一些应用研究集中在如何利用 GPU 解决超大规模的偏微分方程问题[78]，这对于与非电离平衡密切相关的一些物理过程的算法优化有重要的借鉴意义，比如电子热传导（详见 3.6 节，方程 3-1）。

借鉴上述工作的成果，本文采用了将 CPU 上的成熟 ODE 解法整体移植到 GPU 上的方式，并根据 GPU 编程模型的特点，对 RHS 系数生成、电离状态的数据结构等方面进行了单独优化。

## 4.3 基于 CPU-GPU 混合结构的求解器设计

求解器的设计目标是在多 CPU 和多 GPU 的环境下最大限度的提高资源利用率，在计算规模一定的情况下，资源利用率的提高也就意味着性能的提高。为配合第三章提出的示踪粒子和 MapReduce 的框架，非电离平衡的求解器可以嵌入到 MPI 进程中。如图 4-2 所示，基于多 CPU 和多 GPU 异构体系的求解器主要由三部分组成，本地任务调度器、以及分别基于 CPU 和 GPU 的求解模块。

每个物理节点上有一个本地的任务调度器，多个 MPI 进程共享本地的一个或多个 GPU 设备。进程接到任务后，先去查询任务调度器，如果所有的 GPU 都是满负荷状态，任务将由该进程所在的 CPU 处理器来完成；否则，任务调度器根

据一定的规则选择一个当前负载最轻的 GPU 来完成任务。为了充分发挥 GPU 高度并发的优势，一个任务中包含多个非电离平衡方程（任务粒度）。在 CPU 处理器的单一进程内，任务中的每个方程只能串行的求解，CPU 上的求解器基于 CVODE 函数库[64]实现，主要过程是对 CVODE 例程的循环调用；在 GPU 上，一个线程将负责一个或多个方程，多线程并发的执行。

按照第三章提出的基于示踪粒子和 MapReduce 的粗粒度并行结构，map 中采用的哈希算法(scatter)相当于整个系统中的全局任务分配器，粒子被尽可能均匀的分散到多个物理节点上的所有 reducer（MPI 进程）中。

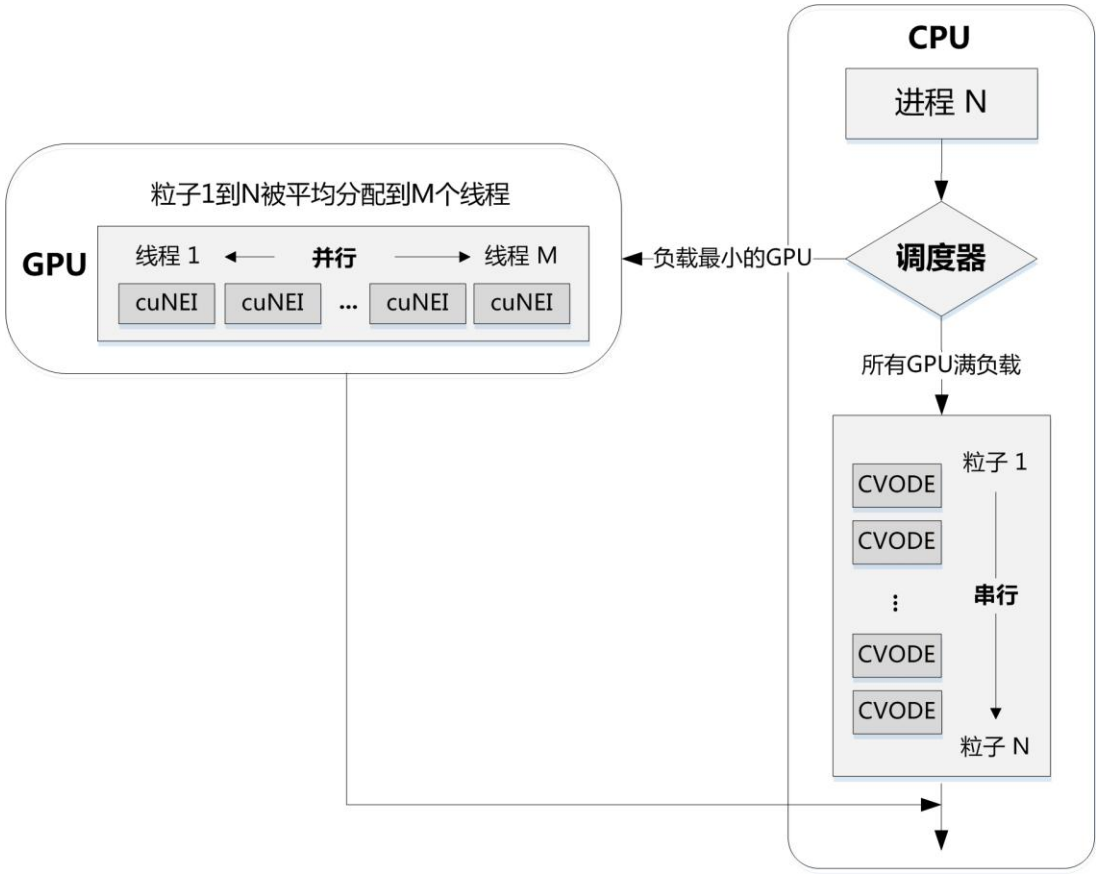



图 4-2　基于多 CPU 和多 GPU 的非电离平衡求解流程。

### 4.3.1 多 CPU 和多 GPU 结构下的负载平衡设计

本文提出的混合异构体系下任务调度策略的基本思想很简单，即优先使用计算能力更强的 GPU 资源，只有在 GPU 满负荷时，才将任务调度给 CPU 执行。由于 GPU 和 CPU 都有内部的排队和调度机制，所以这里定义了一个度量值来标识 GPU 的繁忙程度，该值的上限即表示满负荷状态。

一般来说，一个物理节点上所有 CPU 的核数大于所有 GPU 的个数，所以，多个 MPI 进程可能会同时向 GPU 提交任务。当一个进程向 GPU 提交任务的时候，GPU 可能还在处理上一个任务，并可能还有其他任务再等待处理，因此本

文采用了任务队列作为任务调度的基本数据结构，每个 GPU 都跟一个专用的任务队列相关联，任务队列的相关名词定义如下：

- 活动任务：正在 GPU 上运行的任务
- 等待任务：已处于任务队列中，等待被执行
- 当前负载：所有活动任务数和等待任务数之和
- 最大负载：任务队列的所允许的最大长度
- 任务总数：该任务队列所接收到的所有任务数之和

调度器负责维护所有的任务队列，并将任务分配给 GPU 或者 CPU。任务调度的具体过程如图 4-3 所示：

1. CPU 进程首先向 Scheduler 请求将当前任务加入到某个 GPU 的队列中
2. Scheduler 选择合适的 GPU
   2.1 如果所有的 GPU 都没有达到额定的最大负载，则选取当前负载最小的 GPU；如果有两个以上的 GPU 有相同的最小负载，则优先选取任务总数最小的 GPU；返回被选择的设备标识，继续步骤 3
   2.2 如果所有的 GPU 都工作在最大负载上，则跳转到步骤 4
3. 调用 GPU 求解器，同时将对应的任务队列的当前负载加一；等待 GPU 返回，返回后将任务队列的当前负载减一
4. 调用 CPU 求解器，等待计算结束
5. 接收计算结果，并完成后续处理，比如写 FITS 文件等
6. 准备下一个任务

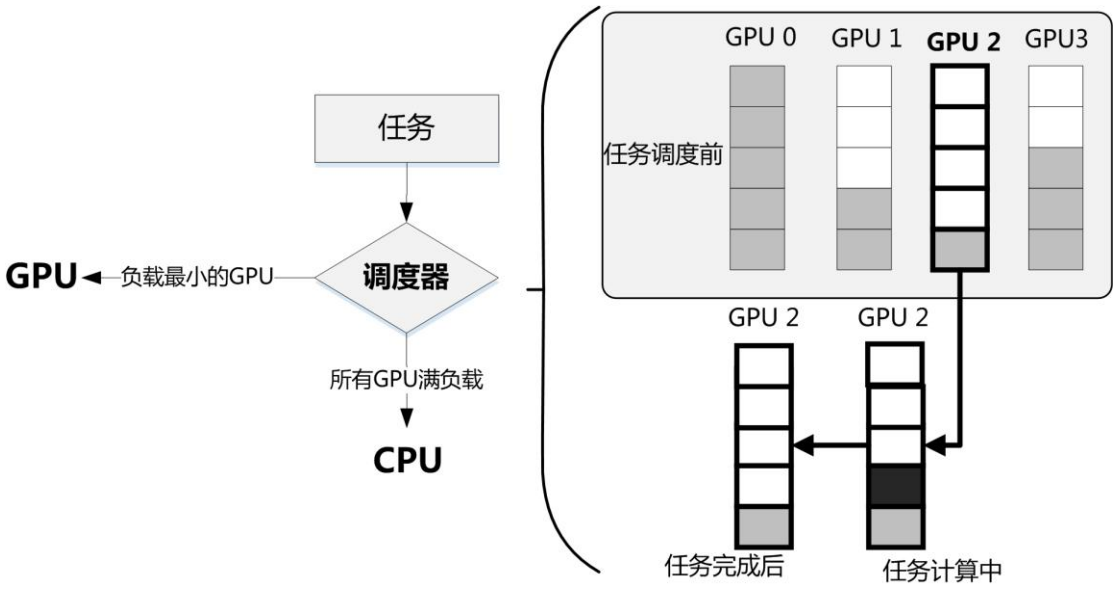



图 4-3　多 CPU 和多 GPU 结构下的任务调度过程。

值得注意的是，这里的最大负载依赖于具体设备的计算能力以及任务本身。另外 GPU 内部的线程调度和并发模式也跟具体设备有关，比如上文提到面对多

进程共享一个 GPU 时，Kepler 架构最大可支持 32 个进程同时连接到 GPU，从而能够获得更高的利用率，而 Fermi 架构则必须要进行进程上下文切换（串行）。

不同于无时间依赖的任务，此类任务之间是相对独立的，也极易实现并行，非电离平衡计算有其特殊性，单次计算的计算量很小，且计算必须是连续性，即上一次计算的结果是下一次计算的输入。考虑到任务在 CPU 和 GPU 之间的调度，则每次的计算结果都需要传回到主存中，如果每次 ODE 求解都要向 CPU 回传结果的话，势必会造成大部分的时间消耗在主机与设备之间的内存拷贝上。这要求合理设置任务粒度，但任务粒度的优化与具体设备相关，需要通过实验确定。

### 4.3.2 GPU 非电离平衡求解器的设计

按照 CUDA 的编程模型，GPU 只是协处理器，并不能独立的工作，CPU 与 GPU 需要协同工作，各司其职[34]，CPU 负责总体控制，GPU 专注并行。运行在 GPU 上的 CUDA 并行计算函数称为内核（kernel），这个 kernel 函数并不是一个独行的程序，而是整个 CUDA 程序中可以被并行执行的部分。CPU 负责串行处理的部分，包括 kernel 启动前的数据准备和 GPU 设备的初始化工作，并启动内核函数。图 4-4 描述了基于 GPU 的非电离平衡求解器的执行流程和结构，由于非电离平衡计算的连续性，CPU 的工作也变得复杂，需要在主存和显存中各维护一套数据结构，用于存放中间结果。

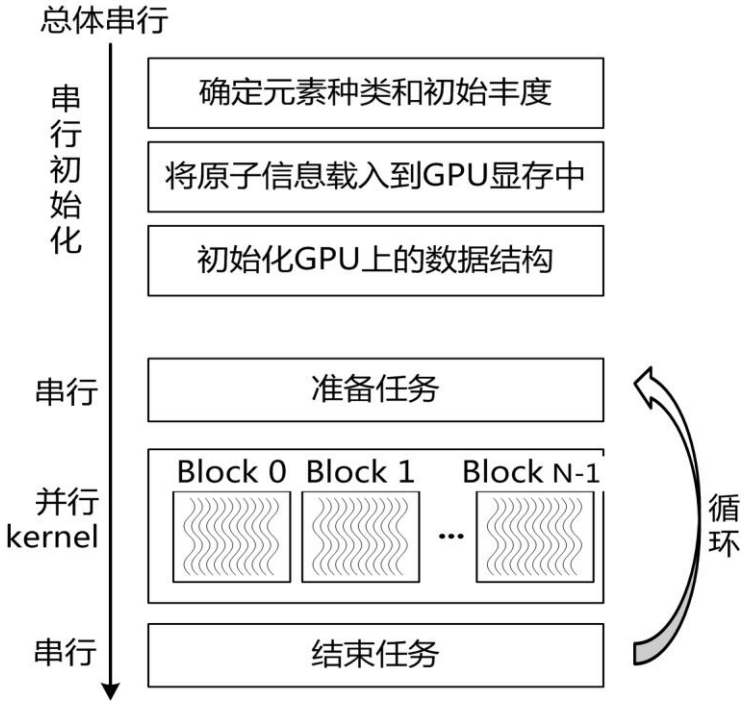



图 4-4　GPU 求解非电离平衡的算法结构和流程。

CPU 前期的准备主要包括：

1. 根据配置文件确定元素种类，并初始化元素的各种属性，其中元素的初始丰度跟具体物理模型有关，一般通过调用用户自定义的函数生成
2. 加载原子数据，并将原子数据载入到 GPU 的全局内存中
3. 初始化 GPU 中的数据结构，

与 CPU 擅长逻辑控制和通用类型数据运算不同，GPU 擅长的是大规模并发计算，尤其适用于计算密集、逻辑分支简单的程序。要充分发挥 GPU 的优势，除算法本身外，还要考虑以下两个方面：

1. GPU 与 CPU 的数据交换是通过主板上的 PCI 接口进行的，相对于内存的访问速度要低很多，如果频繁的进行主存与显存的数据交换，则会大大降低 GPU 的利用率，所以要保证尽量一次性分配给 GPU 足够多的任务并减少与 CPU 的通讯。
2. 需要针对 GPU 的内存模型，对程序的数据结构、变量访问以及存储模式相关等操作进行专门的优化。

CUDA 设备有多个不同类型的内存空间[39]，按照访问速度排列如下：

1. 寄存器（register），高速，仅线程内部访问，CUDA 编译器会优先使用寄存器，但寄存器的数量有限，变量中超出寄存器大小的部分将会放到本地内存。
2. 共享内存（share memory），高速，同一 Block 内的线程均可访问，在 Fermi 架构中，共享内存和一级缓存的总容量只有 64K，在 CUDA3.5 之后共享内存的容量可大于等于 64K。
3. 常量内存（constant memory），低速，只读，但相对于可读写的全局内存，对于多个线程同时访问同一个地址的情况，常量内存借助广播功能，能显著提高访问速度，但其容量只有 64K，不适合大数据结构。
4. 本地内存（local memory），低速。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。
5. 全局内存（global memory），低速，容量大，可以和主存进行数据交换。

结合 GPU 的内存模型，GPU 上的非电离平衡算法的数据结构与硬件的映射关系如图 4-5 所示，

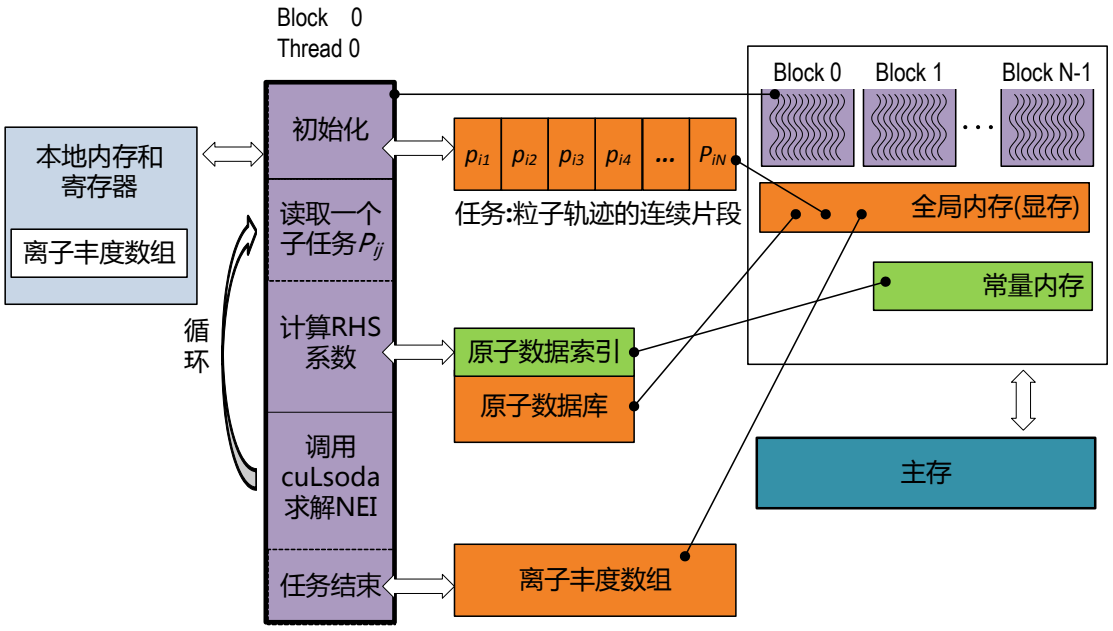



图 4-5　cuNEI 内部结构及其与 GPU 硬件的映射关系。

具体的优化方案和设计理由如下：

- **任务定义**

为减少 GPU 与 CPU 的数据交换，让单个任务内的计算密度更高，这里将任务定义成一定数量粒子轨迹中的连续片段，可以将任务看成是一个由离散的粒子和离散的时间组成的长方体。

任务定义：

$$\boldsymbol{Task} \equiv \text{particle\_num} \times \boldsymbol{subtask}$$

一个任务单位包含了particle_num个子任务，particle_num为粒子个数，每个子任务映射成 GPU 的一个线程（kernel），并发执行

子任务定义：

$$\boldsymbol{Subtask} \equiv \text{step\_num} \times \text{element\_num} \times \text{ODE}$$

其中，step_num表示要计算的步数，即轨迹片段中包含的快照数（时间点），该值在所有任务中都相同；element_num是元素个数，一个元素对应一组包含其所有离子的 ODE。

子任务参数定义：

$$\boldsymbol{Subtask\_param} \equiv \{p_{id}, [(h, \text{temp}, \text{dens}), (\text{h}, \text{temp}, \text{dens}), \ldots, (\text{h}, \text{temp}, \text{dens})], [\text{abandance array}]\}$$

$p_{id}$为粒子标识，$h$为步长，temp为温度，dens为密度，一个子任务包含step_num个$(h, \text{temp}, \text{dens})$三元组，abandance array为所有元素的离子丰度矩阵，是方程 4-1 的初值。

任务的输入参数和计算结果要在系统内存和 GPU 全局内存之间进行交换，各个子任务（kernel）在计算的过程中，中间结果都缓存在寄存器或本地内存中，CPU 和 GPU 之间并不需要进行数据交换。所以在计算量一定的情况下，step_num 的值越大，CPU 和 GPU 之间的数据拷贝次数就越少，GPU 用于计算的时间比重也就越大，性能越好。但考虑到数据流的实时性和内存大小的限制，并不可能将完整的粒子轨迹一次性传递给 GPU。

- **系数计算**

计算方程 4-1 中的系数也是整个求解过程的重要一环。系数的准确性直接决定结果的准确性。方程的系数由物理条件决定，但这些系数并无法根据理论公式直接计算，只能基于标准的基础数据集推导得出，基础集合是离散的，而物理参数空间是连续的，只能是通过数值算法得到最接近的值。所以系数的计算要分为两个过程，首先检索标准的原子数据库，找出差值的基准点，然后利用差值公式求出系数的近似值。

作为基准的原子数据库的选择跟模拟的具体要求相关，基础数据越多，则计算结果也就越准确，但同时也会占用大量内存；反之，基础的采样点越稀疏，内存消耗和计算量越少，但计算结果也越粗糙。FLASH 框架中采用的原子库接近 1MB，而 AtomDB 的原子库则在 400MB~1GB 之间。这些基础数据的特点是只读，一次读取，多次使用，非常适合使用缓存机制来提高性能，这在 CPU 中不成问题，但在 GPU 中却是难题。由于 GPU 的处理器访问全局内存的延时很长，而专门针对只读数据进行了访问优化的常量内存被限制在 64KB，无法容纳全部的原子数据。所以文本采用了一个折中的办法，对原子数据建立索引，然后将索引放到常量内存中。以 FLASH 自带的 1M 的原子数据为例，索引的大小在 1k 左右，完全可以放到常量内存中。

常量内存对性能的优化效果根据问题而不同，此处将原子数据索引放到常量内存中在一定程度上提高了对原子数据的检索速度。经过实际测试得出，此处使用常量内存的性能比完全使用全局内存的性能要高 3%~5%左右。虽然性能提升效果并不明显，但在大规模的模拟中，还是能节约不少计算资源。

- **其他优化**

按照数值算法，一个任务的所有子任务的步长列表都是相同的，是可以共享的，一个任务中步长数据的大小是step_num × 8个字节，完全可以放到共享内存或者常量内存中。

### 4.3.3 混合结构下求解器的实现

本文提出的基于 CPU-GPU 混合异构系统的非电离平衡求解器基于 C 和

CUDA C 实现，原子数据库直接采用了 FLASH 框架提供的系数文件[79]，CPU 下的 NEI 方程求解则利用了 CVODE 函数库。

- 基于共享内存实现的共享数据结构

  为了减少任务调度的开销，调度器被实现为一个普通的例程（routine），该例程利用共享内存维护着本节点内所有 GPU 的任务队列。这样，每个 CPU 进程通过调用该例程获得候选的 GPU 设备，然后调用相应的 kernel 函数将任务传递给 GPU，并等待返回结果。为确保数据的一致性，对任务队列的访问都是原子操作。

- 基于 LSODA 实现的 GPU NEI 求解算法

  基于 Fortran、C 和 C++有很多成熟的刚性 ODE 求解器，但很少有 GPU 版本的实现。4.2 节的分析结果得出 NEI 的 GPU 加速适合整体移植策略，并不适合使用抽象层次较高的 ODEINT 等函数库来求解，所以在综合考虑现有的求解器的特点以及向 GPU 移植难度的基础上，本文以久负盛名的 ODEPACK 中的 LSODA 例程[65]为基础，借助 Fortran 语言和 C 语言的自动翻译器 f2c[80]，并参考了相关的开源代码，经过一定的修改，快速实现了 LSODA 的 CUDA C 版本，并将其和系数计算、任务输入输出等功能封装到了一个统一的接口 cuNEI（参见图 4-2、图 4-4 和图 4-5）。

算法 1 和算法 2 中的伪代码描述了基于共享内存和任务队列实现的调度过程。

**算法 1. 调度函数, 每个 MPI 进程在任务准备就绪后调用该函数提交任务**

```
1: FUNCTION schedule() {
2:     device_id = scheduler_alloc();
3:     IF (device >= 0) {
4:         gpu_nei();   /* 与 GPU 通讯，并调用 cuNEI()完成计算 */
5:         scheduler_free(device_id);
6:     } ELSE {
7:         cpu_nei();   /* 调用 CVODE 例程完成计算 */
8:     }
9: }
```

**算法 2. 相关的子程序集合**

```
1: q_min ;   /* 记录最小负载的任务队列 */
2: FUNCTION scheduler_alloc() {
3:     q_min.load = MAX_LOAD + 1;   /* MAX_LOAD: 所允许的最大负载 */
4:     q_min.device_id = -1;        /* 默认情况下，不选择任何设备 */
```

```
5:    FOR (each device i ) {
6:       q = get_taskqueue_from_shm(i);
7:       IF (    (q.load < qmin.load)    OR
                 ( (q.load = qmin.load) AND (q.hist_load < qmin.hist_load) )    ) {
8:             qmin= q;
9:        }
10:    }
11:    IF (qmin.device_id >= 0 ) {
12:       ATOMIC { qmin.load++;    qmin.hist_load++}
13:    }
13:    RETURN qmin.device_id;
14: }

15: FUNCTION scheduler_free(device_id) {
16:   ATOMIC {
17:      q = get_taskqueue_from_shm(i);
18:      qmin.load--;
19:   }
20: }
```

## 4.4 实验和评价

**测试环境：**

- GPU: 4 块 NVIDA Tesla C2075 显卡，Fermi 架构， 6GB GDDR5，每个 GPU 共有 448 颗流处理器（1.15GHz）
- CPU: 4 颗 Intel Xeon E5-2640 ，2.5GHz，6 核/CPU，总计 24 个核心
- PCI-E: 2.0
- 软件环境：CUDA 6.0， MPICH2

**实验设计：**

为了便于和第三章的内容进行对比，本章实验的最大计算量相当于处理 100 万个粒子的连续 1000 个快照文件，即 100 万个粒子演化 1000 步。单个任务定义为 4480 粒子×10 步 NEI 计算（12 个元素），24 个 MPI 进程，每个进程的任务

数均为 1000，共计 24000 个任务。实验分别从并行度与加速比、性能与任务粒度、性能与队列长度、任务调度与队列长度、以及结果准确性等五个方面对算法进行了验证。

**并行度、任务粒度与性能：**

本组实验考察 GPU 线程数、任务粒度与性能的关系，如图 4-6 所示，纵轴表示混合求解器（4 块 GPU+24 个 CPU 处理核心）对纯 CPU 求解器（24 个处理核心并行）的加速比（最大任务队列长度固定为 8），横轴表示每个 GPU 启动的线程数，对于单个 CPU 处理核心来说，等价于循环次数。四条曲线代表了不同的任务粒度设置(step_num分别为 1，2，5，10)。

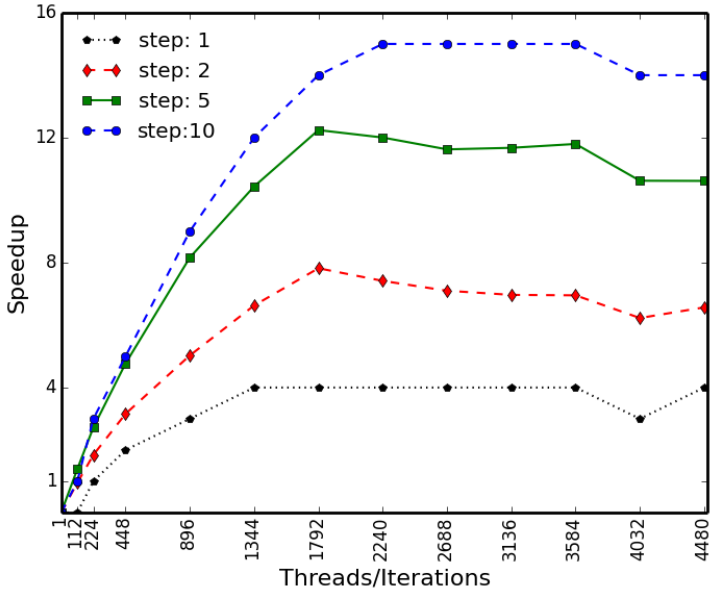



图 4-6　不同 GPU 线程数和任务粒度下的加速比。

从实验结果可以得出以下两个结论：

- 当线程数较少（线程数小于 GPU 的流处理器数）时，一部分流式处理器处于闲置状态，GPU 完成任务的时间是恒定的（基本上为一个线程时的时间），并不随着线程数的增加而增加。此时，由于 GPU 的能力没有充分发挥，CPU 的性能要明显高于 GPU（刚性 ODE 求解器中的逻辑分支判断较多，CPU 有明显优势）。并发线程数数超过 224 时，加速比开始大于 1；线程数达到 1792（448×4）时，加速比基本上达到了峰值，以后一直保持。
- 任务粒度对性能的影响非常明显，在总计算量一定的情况下，随着单个任务中包含的计算量的增加，加速比呈快速升高趋势。任务总量一定时，任务粒度与任务分配频率（CPU 与 GPU 之间的数据传输次数）成反比。这说明在非电离平衡求解中，相对于单纯的计算，内存拷贝操作带来的

延时较大。不过随着任务粒度的不断增加，整体性能的提升也越来越小。当step_num为 10 时，能够完成 15 倍左右的加速。

### 性能与队列长度

图 4-7 统计了混合求解器使用不同数量的 GPU 在不同最大队列长度下（任务粒度固定，step_num=10）的运行时间。该组实验中每个实验的计算量均相当于 100 万个粒子演化了 1000 步（在相同的计算量下，24 颗 CPU 核心所需要的时间约为 9280 秒）。随着任务队列长度的增加，总体时间在不断减少，但逐渐趋于平缓；当最大任务队列长度大于 8 时，时间曲线基本不再下降。这说明 GPU 已经满负荷运转或者是任务到达的速率已经达到峰值，这种情况下单位时间内的任务量小于等于 GPU 的处理能力。这组实验还说明了对本文中的实验环境而言，在单个物理节点下，24 颗 CPU 处理器配合 4 块 GPU 卡的整体性能最好。

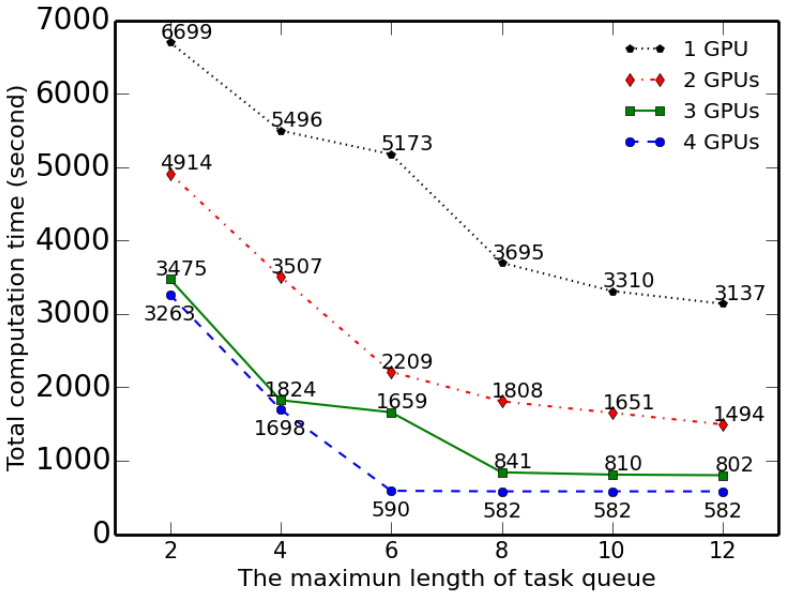



图 4-7　总计算时间与最大任务队列长度的关系。

### 任务调度与队列长度

图 4-8 描述了在不同的最大任务队列长度下的任务调度情况，随着任务队列长度的增加，GPU 处理的任务数也在增加。图 4-7 与图 4-8 可以结合起来看，针对当前的实验环境和问题规模而言，绝大部分的任务都被调度到了 GPU 上。但当 GPU 数量小于等于 3 时，GPU 上负载过重，造成了 CPU 等待任务完成的时间比较长，性能上也达到了极限。当 GPU 数量增加到 4 时，所有 GPU 的综合处理能力已经超过了 CPU 准备任务的速度，CPU 可以不再参与 NEI 的计算，性能达到最好。

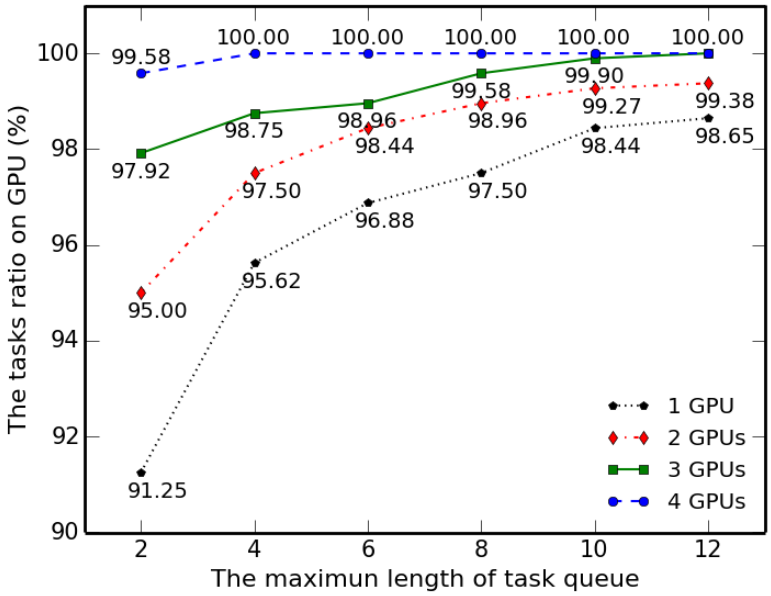



图 4-8　不同最大任务队列长度下 GPU 的负载情况。

## 结果误差分析

图 4-9 显示了本文提出的 cuNEI 求解器与原始 CPU 求解器的结果误差的分布情况。此处的误差是在相同的初始条件下非电离平衡演化了 1000 步后的结果对比，误差分布曲线表明 99%相对误差都小于 0.0000015，这个误差远小于非电离平衡一致性所要求的 0.0001（参见 2.4 节），这说明基于 GPU 的方法在准确度方面能够达到要求。

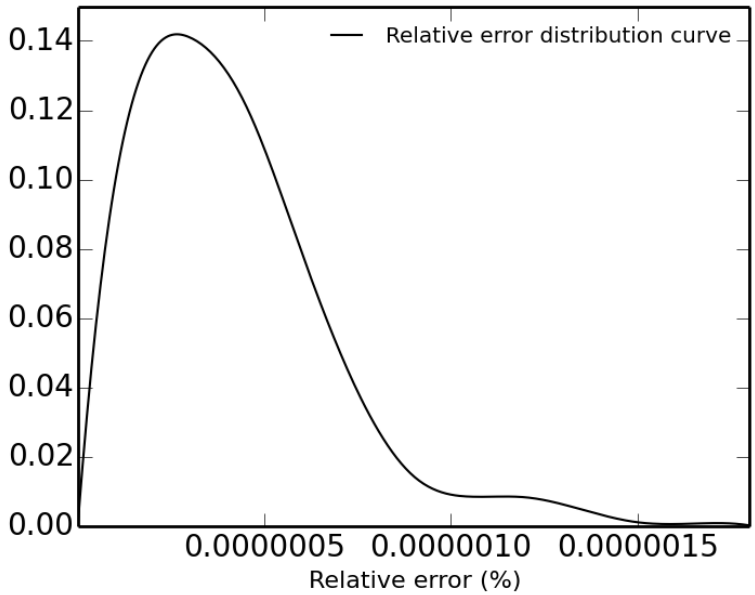



图 4-9　基于 GPU 的 NEI 计算的误差分析。

## 4.5 异构求解器在光谱计算中的应用

本章所提出的基于多 CPU 和多 GPU 的任务调度方法，适用于单个任务比较小，而且总体任务量又非常多的情况。除了非电离平衡外，本文还将光谱计算的核心算法移植到了混合异构求解器的任务调度框架下，从而进一步验证了其扩展性和适应性。

### 4.5.1 光谱计算面临的问题

目前，太阳系以外的天体的所有信息都是通过分析其到达地球的电磁辐射（光谱）得到的。光谱中包含了丰富的内容，从中可以推断出星体的温度、年龄、金属丰度等。在物理上，光谱与电离紧密相关，光谱计算需要知道电离状态。电离是微分方程，而光谱是积分方程，对于参数空间中的同一个点，计算光谱所需的积分个数要远远大于非电离平衡中 ODE 的个数。而目前常用的光谱计算工具中，仅实现了进程级别的并行[81]，在参数空间比较大的情况下，需要消耗大量的计算时间。下面以热离子的辐射复合连续谱（RRC：Radiative Recombination Continuum）[84]为例，来说明本节提出的方法在光谱计算中的应用。按照实验数据，在基于 CPU 的程序中，RRC 中 90%以上的计算时间是对方程 4-6 的反复求值。

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

$$\Lambda_{\mathrm{RRC}}(E_{bin}) = \int_{E_0}^{E_1}\frac{dP}{dE}(E)dE \qquad (4\text{-}6)$$

其中，*P* 为辐射功率，*E0* 和 *E1* 分别为能量积分区间的上下限，方程式中的其他变量与本文内容相关性不大，故省略说明，具体含义可参考文献[84][85]。

要得到一个全波段的高分辨率光谱，必须针对所有离子的所有能级在足够多数量的能量区间(bin)上对 Eq.4.6 进行数值积分。实际应用中，足够多的能量区间一般设置为 5 万左右，常用的离子数目为 496 个，而理论上离子的能级是无穷多的，为了简化计算，需要采用过滤机制只抽取某些重要的能级，从而参与实际计算的能级数量从几个到几千不等。按照这个推算，一个网格点上的 RRC 积分数量在 $2.0*10^8$ 量级。按照本文的实验数据，在 Intel Xeon E5-2640 (2.5GHz)处理器上的一个进程完成一个点上的计算耗时约 600~800 秒（不同物理条件下的计算公式不同，导致各个任务的计算时间有小幅波动）。照此推算，处理一个 $128^3$ 的三维模拟的结果，则最少需要 40 万个 CPU 小时。同非电离平衡一样，如此巨大的负载是由数量众多的重复性小任务累积而成的。

### 4.5.2 光谱计算的优化方案

相对于 ODE 方程的求解，数值积分算法要简单得多，控制逻辑也相对较少，非常有利于 GPU 并行。目前数值积分在 GPU 上的优化研究主要集中在大规模的多维积分在 GPU 上的并行化问题[86]，而光谱中的积分更适合于整体移植（参见图 4-1(b)），即每个线程都对应一次小规模但完整的积分运算。事实上，在光谱计算的优化中，由于单位任务的积分数量巨大，一次性分配给单个线程的积分数量也很大，线程（kernel）内部的多个积分运算只能是串行完成。

本文的优化方案借助了一个被广泛采用的光谱计算工具包 APEC（Astrophysical Plasma Emission Code）[85]来实现，优化后的代码简记为 cuAPEC，原始 APEC 在其中的主要作用是提供光谱计算的流程模版，以及准备必要的数据、结果输出。优化的主要工作包括：

1. 为 APEC 开发了一个 MPI 包装器，实现了进程级别的并行
2. 将 APEC 中原来的积分函数抽取出来，作为 CPU 上的求解器
3. 将经典的数值积分算法 Simpson 算法[24]移植到了 GPU 上，作为 GPU 上的积分求解器
4. 调整任务粒度，优化数据结构，减少主机与设备之间的通讯频次

**算法 3. RRC 积分的 GPU kernel 伪代码**

```
1: global_idx : 线程的全局编号
2: /*分段的积分区间*/
3: bin_size = (global_upper_limit – global_lower_limit)/total_bin_num;
4: /*该线程负责的积分区间数量*/
5: local_bin_num = total_bin_num / total_threads_num;
6: global_bin_offset = local_bin_num * global_idx;
7: WHILE local_bin_idx < local_bin_num {
8:     global_bin_idx = global_bin_offset + local_bin_idx;
9:     l_limit = global_lower_limit + global_bin_idx * bin_size;
10:    u_limit = l_limit + bin_size;
11:    emission[global_bin_idx] = Simpson(f_rrc, l_limit, u_limit);
12:    local_bin_idx = local_bin_idx + 1;
}
```

### 4.5.3 光谱计算的优化结果

**测试环境**

- GPU: 4 块 NVIDA Tesla C2075 显卡，Fermi 架构， 6GB GDDR5，共 448 颗流处理器（1.15GHz）
- CPU: 4 颗 Intel Xeon E5-2640（2.5GHz），6 核/CPU，共 24 个处理核心

实验使用了 APEC 光谱计算包的计算结果作为测试的基准数据，参数空间为 24 个网格点。cuAPEC 启动了 24 个 MPI 进程，每个进程负责一个格点的计算任务。每个格点中至少包含仅 496 个任务，这个量级对于测试任务调度效果和整体性能已经足够。实验分别从性能与任务粒度、性能与队列长度、任务调度与队列长度、以及结果准确性等四个方面对算法进行了验证。

**性能与任务粒度**

图 4-10 展示了在不同 GPU 数目和不同任务粒度的情况下 cuAPEC 相对于原始 APEC 的加速比。从图中可以看出，随着 GPU 资源的增多，加速比也在不断增大，但 4 块卡较 3 块卡的性能提升很小，且 3 块较 2 块卡的增幅仅为 10%左右，这说明对于光谱计算，24 个 CPU 核完成的积分以外的其他处理的速度与 2 块 GPU 卡完成积分计算的速度相当，简单地增加 GPU 对整体性能并不会有明显的提升。

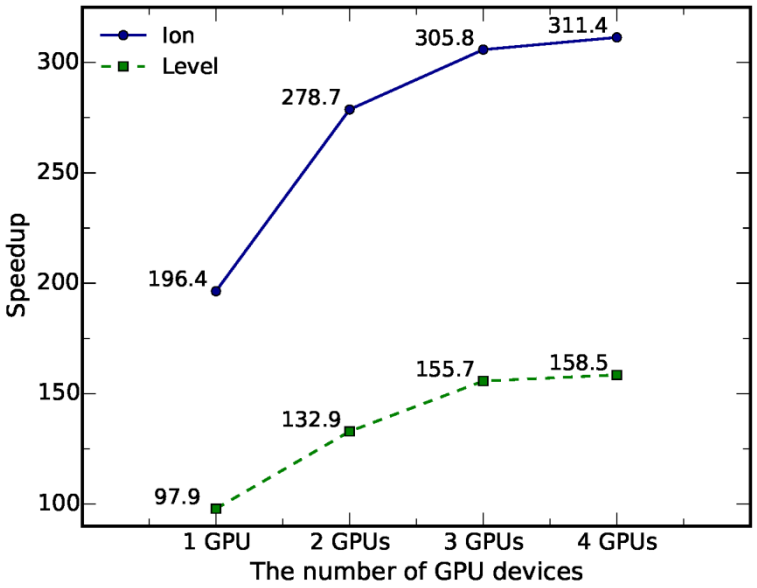



图 4-10 基于 GPU 的光谱计算的加速比。

由于 CPU 与 GPU 之间数据传输的延时较大，所以在一定的任务量下，任务粒度对性能也有很大的影响。这里对比了以离子（ion）和能级（level）分别作为任务粒度时的加速情况，一个格点内包含 496 个离子，一个离子内包含几个到几千个能级，离子作为任务粒度时的加速比基本上是能级时的 2 倍。但如果继续

增大任务粒度，则 GPU 线程中的逻辑控制操作会大幅增加，严重影响并发效率，同时也不利于对 kernel 函数进行封装。

相对于串行的 APEC 程序，24 个 MPI 进程加上 3 块 GPU 设备完成了 300 倍的性能提升，相对于简单的 MPI 并行的 APEC 版本，cuAPEC 能够完成约 22 倍的加速（由于显示比例相差较大，图 4-10 中未标出）。

### 性能与队列长度

如图 4-11 所示，实验中考察了队列长度从 2 逐渐增加到 14 的情况，对于不同数目的 GPU 来说，最大队列长度等于 12 时是性能的转折点。实际上，当队列长度大于 8 时，性能提升已接近极限，大于 12 时，性能反而有少许下降。这说明 GPU 的计算能力已经达到了极限，任务排队时间变长，CPU 等待 GPU 完成任务的时间也变长，从而影响了整体性能。这个图也从侧面反映了对于文中的光谱计算问题和实验环境，3 块 GPU 设备已经足够。

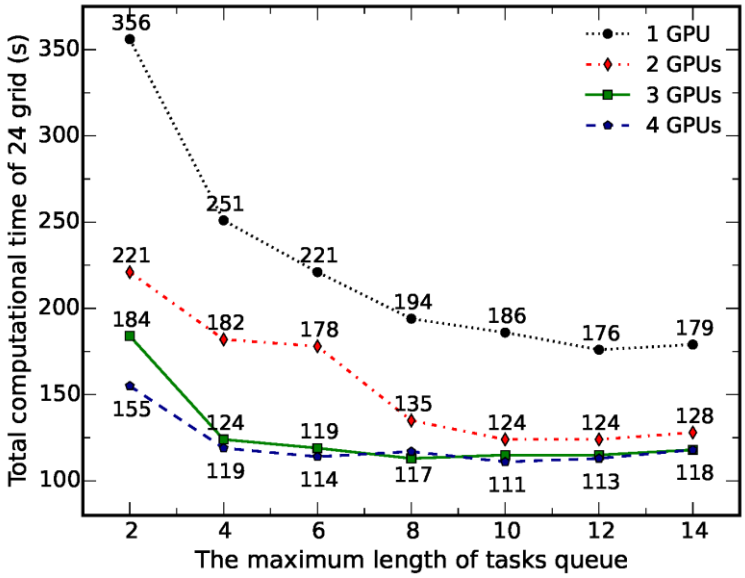



图 4-11　总计算时间与最大任务队列长度的关系。

### 任务调度与队列长度

图 4-12 统计了 CPU-GPU 之间的任务调度比例与最大队列长度的关系，当任务队列大于 8 时，所有的任务都被调度到了 GPU 上执行，这也支持了图 4-11 的结论，GPU 上负载过重，造成了等待任务完成的时间比较长，性能达到了极限。从总体上，绝大部分的任务都被调度到了 GPU 上，CPU 在整个计算过程中主要做辅助工作，比如读取原子数据库、准备任务参数、将结果保存到文件等。

任务调度的结果跟具体应用也有很大的关系，取决于单位任务中包含的计算量。计算量越大，CPU 分配到的任务比例也就越大。在另一组采用了精度更高

计算量更大的龙贝格积分算法[24]中，CPU 上的任务比例占到了 30%。

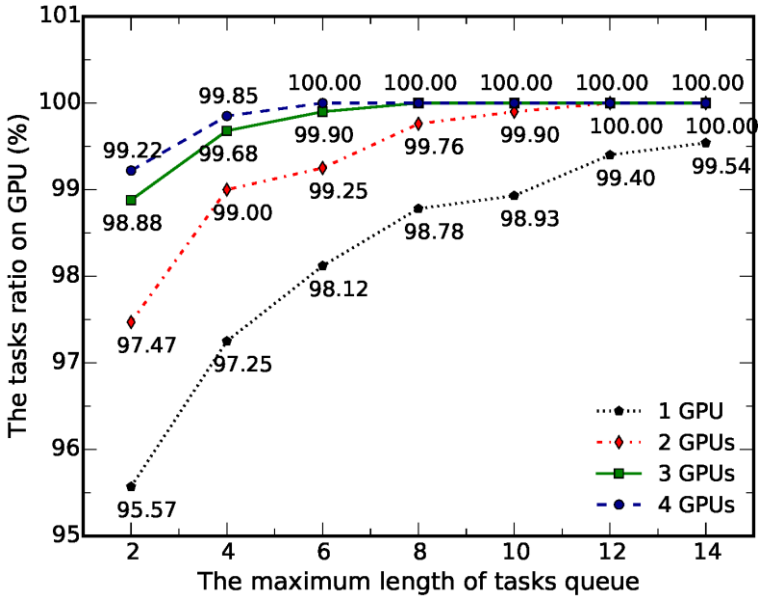



图 4-12　不同最大任务队列长度下 GPU 的负载情况。

**结果准确性分析**

光谱计算的最终结果是各个波段上的能量通量，图 4-13 是 cuAPEC 相对于原始 APEC 程序的计算结果的误差分析。从误差分布曲线可以看出，相对误差在 0.003%之内，且 99%的误差都小于 0.0005%。统计数据表明 cuAPEC 的计算结果和原始的 CPU 程序相差无几，而且计算结果的准确性已经得到了领域专家的认可，对于绝大部分的光谱计算，cuAPEC 是能够胜任的。

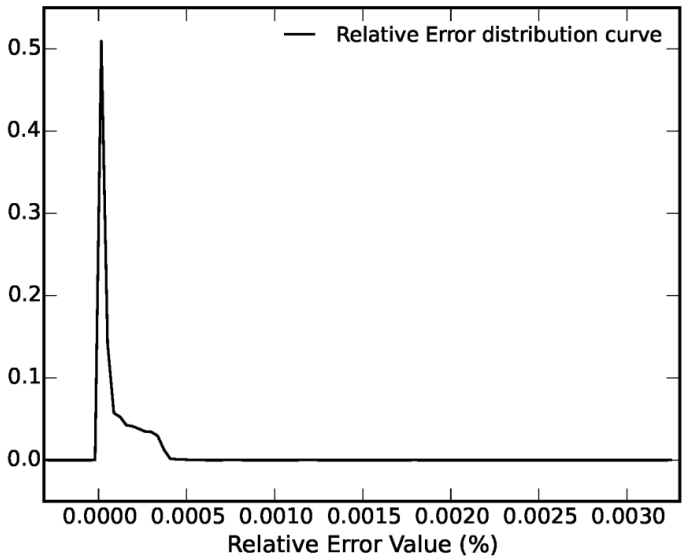



图 4-13　基于 GPU 的光谱计算的误差分析。

**光谱优化中未解决的问题**

目前，对 APEC 的优化并没有考虑时间的依赖性，所有的离子都被认为处于电离平衡状态（Collisional ionization Equilibrium），所以各个任务是完全独立的。如果要考虑非电离平衡的因素，就需要在任务准备和调度阶段要严格按照时间顺序进行，而且还要将 4.3 节提出的基于 GPU 的非电离平衡求解器（cuNEI）集成进来。

### 4.5.4 对其他问题适用性

本章中提出的基于CPU-GPU混合异构系统的任务分配策略以及ODE求解器，除了非电离平衡、光谱计算外，该可以用于加速核合成的计算。3.7 节对核合成的过程有简单的描述，核合成 ODE 方程中的未知数将近 500，而且仅是建立 ODE 的计算过程就很复杂，比较适合采用 4.2 节中提出的重点移植方式（图 4-1(a)）进行优化。目前，天文模拟领域并没有一套高性能的通用核合成框架，这部分工作可以作为文本后续研究的重点。

## 4.6 不足与改进

**并行化及任务调度的不足**

目前的并行化方案是基于 MPI 和 GPU 的两级并行，基本思路是优先利用计算能力强的 GPU，当 GPU 满负荷时，才将后面到来的任务分配给 CPU。之所以在 CPU 部分采用了 MPI 多进程，主要是为了简化与主流天文模拟框架的集成（绝大多数的模拟框架都是基于 MPI 的）。GPU 的线程（kernel）都是异步执行的，当 CPU 将任务提交给 GPU 后，控制权立即返回到 CPU，但目前的方案中，CPU 只是在等待 GPU 回传计算结果，在 GPU 计算的这段时间内，CPU 是空闲的。

改进的方案是基于现代 CPU 的多核结构，利用轻量级的多线程代替原来的多进程，在每个物理节点上只设置一个 MPI 进程，负责调度多个 CPU 线程和 GPU 设备。同时要充分利用 CUDA multiple streams 机制（不同流中的操作可以同时执行）[34][39]，将数据传输与计算重叠，最大限度的提高 GPU 的利用率。

此外，调度算法在针对不同硬件环境下的通用性上还略有不足，表现为最大任务队列长度与具体应用和硬件都有关，在正式启动模拟之前，需要通过小规模的测试，得出特定软硬件环境下针对本问题的最佳任务队列长度设置。这个预处理工作可以通过自动化的脚本来完成，但也需要人工判断。后续的改进工作可以根据性能参数建立评估模型，然后利用 NVML（NVIDIA Management Library）

获得 GPU 运行时的统计数据，进而实现完全自动化的任务调度。改进的调度方案同样需要考虑调度本身的开销，初步的实验结果表明，基于 NVML 进行任务分配的开销比本文基于共享内存的方式超出 10%左右。后续的改进工作将在未来继续。

**精度上的考虑**

目前的实现方案中，非电离平衡方程的系数是基于 40 年前的数据经过数值差值得到的[79]，并没有使用最新的更加精确的原子数据。一部分原因是以前的数据量比较小，相应地读取和差值操作的速度也就比较快，实现起来也相对容易，同时精度上能够满足大部分场景的需要。当对计算准确度要求较高时，可考虑采用 AtomDB 提供的原子数据库[3]，AtomDB 的数据全面且准确，但体积比较大，在 GPU 上需要采用一定的优化策略，比如 3.5 节中提到在常量内存或共享内存中维护数据的索引。

**基于稀疏矩阵的优化策略**

虽然 NEI 方程是稀疏的，但由于已经实现的优化策略是将 NEI 方程的求解整体移植到 GPU 的 kernel 中，这并不利于使用稀疏矩阵技术。CUDA 本身提供了稀疏矩阵的函数库，同时开源社区也有一些优秀的线性代数工具包，比如 MAGMA[68]，对多核异构体系提供了很好的支持。这些工具包比较适合重点移植方式（图 4-1(a)），比如将 CVODE 函数库中的矩阵操作全部委托给 MAGMA。由于单个元素的非电离平衡方程的规模都很小，一般不超过 30 个变量，计算本身瞬间即可完成，如果采用上述模式，这些工具包调用的开销以及之前的准备工作往往抵消了性能上的提升。如果采用将矩阵操作移植到 GPU 的优化方式，就需要充分利用稀疏矩阵的优势，可以考虑下面的方案。

目前的方案是一个元素对应一个非电离平衡方程，方程中的未知量等于元素的原子数加 1，同一个格点内的不同元素的 NEI 方程是串行求解的。可以将多个元素的 NEI 方程合并成一个大的方程组，常见 12 个元素组成的大方程组共有 181 个未知量，形式如下：

$$\begin{pmatrix} \dot{\mathbf{He}} \\ \dot{\mathbf{C}} \\ \dot{\mathbf{N}} \\ \vdots \\ \dot{\mathbf{Fe}} \\ \dot{\mathbf{Ni}} \end{pmatrix} = \begin{bmatrix} \mathbf{He} & 0 & 0 & \cdots & 0 & 0 \\ 0 & \mathbf{C} & 0 & \cdots & 0 & 0 \\ 0 & 0 & \mathbf{N} & 0 & \vdots & \vdots \\ \vdots & \vdots & 0 & \ddots & 0 & 0 \\ 0 & 0 & 0 & \cdots & \mathbf{Fe} & 0 \\ 0 & 0 & \cdots & 0 & 0 & \mathbf{Ni} \end{bmatrix} \tag{4-7}$$

这里的元素符号代表该元素对应的 NEI 方程的 RHS，元素的导数形式（如$\dot{\mathrm{He}}$）则代表该元素的各个离子丰度对时间的变化率所组成的向量，其余位置都是 0，

按照这种形式组织的方程，就可以直接利用 CUDA 提供的稀疏矩阵函数库(cuSPARSE)[39]，充分发挥 GPU 在矩阵运算方面的优势。

**移植到其他多核体系结构**

本章的工作内容都是基于 CUDA 实现，也可以转化为 OpenCL 的实现，这样就能够适应更多的硬件平台。进一步的优化还可以考虑 Intel 公司的众核加速器（MIC）和 AMD 公司的加速处理器（APU）。

基于 CUDA 架构的一个明显限制在于 CPU 与 GPU 之间的数据传输是通过 PCIE 总线进行的，主机与设备之间频繁的数据交换将在很大程度上影响整体性能。APU 的特点在于将中央处理器和图形处理器集成在一个芯片内，即两种不同的架构可以在同一芯片内协同运行，避免了 CUDA 架构中数据拷贝的开销。APU 是对 CPU 和 GPU 的深度集成，可以根据任务类别将计算自动的分配到不同的计算单元中。APU 支持 OpenCL 工业标准，有利于编写移植性更好的程序。由于实验条件限制，文本并没有尝试基于 APU 的加速方案，这部分内容计划在未来工作中完成。

## 4.7 本章小结

本章是上一章优化工作的继续和深入，在基于 MapReduce 的并行架构下，针对多 CPU-多 GPU 的混合异构平台，着重对非电离平衡方程求解本身进行了优化。本章设计并开发了基于 GPU 的 NEI 求解器，在经典 NEI 数值算法的基础上，针对 NEI 问题本身与 CUDA 的架构特点，采用了较为直接的整体移植模式。整体移植对原有算法的改动较少，主要依靠 GPU 的高并发来提升性能，移植后的算法在数据结构、内存访问等方面进行了专门的优化，包括采用常量内存作为原子数据的缓存，使用可变的任务粒度减少主存与显存之间的数据交换次数等。同时本章还针对非电离平衡计算单体任务小但总体数量巨大的特点，提出了一个基于任务队列和共享内存的 CPU 和 GPU 之间的任务调度策略，与其他调度策略相比，该方法通过优先利用 GPU 资源来提高整体性能，且本身开销很小。实验数据表明相对于 CPU 的解法，GPU 方法取得了 15 倍的性能提升。本章最后还验证了该方法在光谱计算方面的适用性。

# 第五章 基于层级结构的可视化驾驭式计算环境

前面两章的内容主要从计算本身的角度来优化非电离平衡模拟的性能。天文数值模型属于大型的计算任务，通过最大限度的优化算法来压榨处理器，固然能带来性能的大幅提升，但并不能保证这期间不做无用功，能以最小的代价达到预期的结果。本章将从整个模拟工作流程的层面入手，在前两章算法层面基础上，通过引入高效的人机交互机制，探索可能的改进空间。故较之于前两章，本章的方法适用范围更广，除了对非电离平衡模拟提供的特殊支持外，也可用于其他天文数值模拟。

## 5.1 天文数值模拟过程的人机交互分析

传统的天文数值模拟存在两个相对独立的环节：线上运行(online simulation)、线下分析(offline analysis)，这两个环节并不是一次完成的，初次运行结束后，根据分析的结果调整模型参数并再次运行，经过多次这样的迭代，才能得到预期的结果。即便是FLASH code团队本身，也是采用把一个模拟分成多个阶段来运行，一个阶段一天到几天左右，如果阶段性结果没有问题，再继续运行；如果发现问题，可能退回到之前的某个阶段，调整参数后重新启动；最坏的情况是抛弃所有阶段性成果，从头开始[5]。也就是说，错误发现得越早，更正的成本就越小。

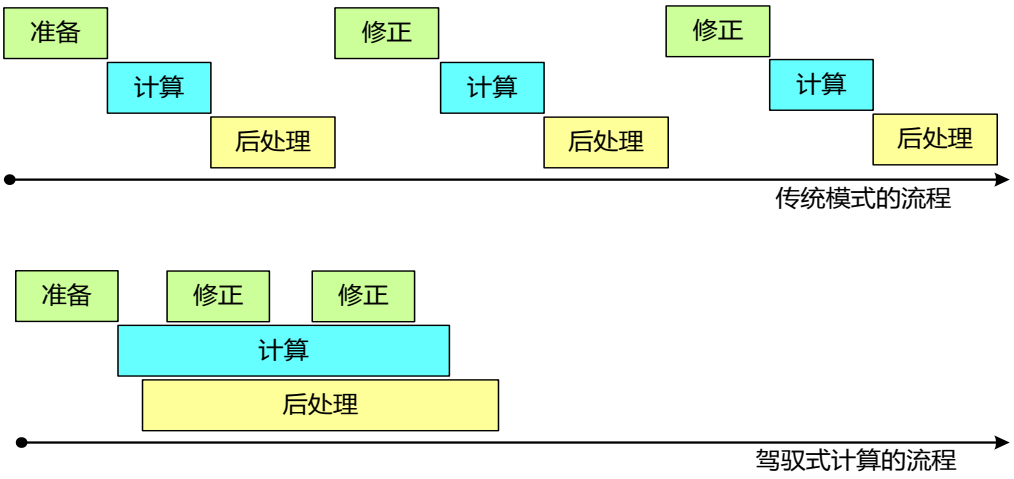



图 5-1 传统方法中的串行模式与驾驭式计算方法中的重叠并行模式对比[90]。

上图形象的展示了传统方法与驾驭式计算方法的特点。驾驭式计算的核心思

想就是融合这两个过程，赋予使用者在程序运行时控制程序演化方向的能力，从而加速模拟过程的收敛，同时节约人工和计算成本。要实现这一目标，以下两个功能尤为重要：

（1） 将进行时的模拟的结果快速的呈现给用户；

（2） 帮助用户决定下一步的参数调整方向。

要实现这两个功能，既需要通用的高效的可视化技术、可视分析手段，又需要针对具体问题提供合适的展现方式，最后还需要在系统层面上提供一个高度可扩展的架构以集成这些技术。

### 5.1.1 驾驭式计算的研究和应用现状

早在 20 世纪末，Mulder 等[91]便从适用范围、体系结构以及用户接口等方面对当时流行的驾驭式计算技术和框架进行了一次全面的对比和分析。近年来，Wright[92]通过多个实例分析并展望了驾驭计算结合可视化技术在现代计算科学中的应用。虽然目前还没有专门针对天文模拟的驾驭式计算框架，但这些通用或专用的驾驭式计算工具正是本文的设计基础。

CUMULVS[93]是橡树岭国家实验室开发的一个协同软件架构，可谓是驾驭式计算的先驱，旨在为分布式应用的多个使用者提供一个支持交互可视化、远程控制的运行环境。CUMULVS 的显著特点是良好的容错性和支持多用户协同操作，虽然自 2000 年后研发基本出于停滞状态，但目前仍有若干的应用借助其运行。

SCIRun[94]与 CUMULVS 拥有同样悠久的历史，是传统驾驭式软件的代表，同时也一直保持与时俱进的作风[95]。SCIRun 针对科学问题的建模、模拟、可视化提供一个统一的问题解决环境，目前广泛用于生物电学、生物医学领域的模拟。SCIRun 的核心是一个可灵活装配的模拟网络，针对不同的实验，各个模块组合成一个高层次的工作流，每个模块的所有可调参数对使用者都是开放的。

近年来的驾驭式计算研究领域加强了对 e-science 的支持。RealityGrid [96]是一个专门为现代计算密集型学科提供分布式协同控制功能的项目，其最大的优势是支持开放网格服务架构（OGSI），这样使用者可以根据网格注册和发现协议随时连接到任何一个或多个提供驾驭接口的模拟程序上。RealityGrid 目前主要用于材料科学。同样能较好的支持网格环境的还有 COVS[97]，COVS 更侧重于支持虚拟组织中多用户协同进行的模拟和可视化，常用于 N 体问题的模拟。

由于大规模模拟本身的复杂性，加上不同问题的模拟又有其自身的特殊性，驾驭式计算也很难提供全面且通用的支持，所以还有一些研究是侧重支持某些方面的。比如 iFluids[92][98]侧重于土木建筑工程的通风相关的可视化建模和模拟，ESPN[99][100]重点支持遗留的模拟系统与可视化组件的连接，FSSteering[101]能够支

持在 CFD 模拟中进行交互式的网格划分，Steereo 框架[102]则仅支持硬编码方式的参数调整，而不涉及任何可视化等其他功能。

从上面的分析可以看出，目前的驾驭式计算一般都是跟具体领域或应用紧密结合在一起的，天文模拟领域还没有一个比较成熟的支持驾驭计算的产品；相对通用的框架又要求程序必须基于该框架构建，这对于具有大量基础代码的天文数值模拟而言，代码移植的成本太高而难以在实际中使用。

### 5.1.2 天文模拟中的驾驭式计算对可视化的要求

驾驭式计算的过程离不开可视化和可视分析的支持。运行期的参数调整的基本依据就是已经完成的中间结果，利用合适的可视化和可视分析技术能帮助用户快速的做出正确的决策。计算密集，数据规模大，运行时间长既是天文数值模拟的特点，也是应用可视化技术面临的挑战。

面对大量的分布式数据，可视分析本身也需要消耗大量的计算资源，如果在传统的后处理模式，也许还可以接受，但如果是模拟过程中实时地显示模拟结果，很多情况下是不现实的。因为此时要显示的数据还在模拟程序的内存中，模拟本身不可能暂停去进行渲染工作，而且大部分的高性能计算环境都是针对计算本身的，并没有对图像渲染提供专门的支持。目前针对这个问题的解决方案是采用 I/O 加速技术将需要的数据快速移出或者在模拟程序之上增加就近分析（in-situ analysis）的功能。GLEAN[51]和 Damaris/viz[49]分别是两者的代表。GLEAN 框架综合利用模拟程序的数据结构以及计算环境的拓扑结构，能够在内存和网络中高效的迁移数据，从而将有用的数据实时的传递给可视化和分析程序。GLEAN 能与 FLASH code 等多个常见的天文模拟框架配合。Damaris/viz 框架在每个物理节点上预留特定的处理器来负责节点内部数据的可视化计算，通过将模拟数据放入共享内存来避免大量的数据拷贝工作，从而可以在不中断主程序运行的情况下，实现了 in-situ 可视化。但利用 Damaris 要按照一定的要求对模拟程序本身进行改造，这对于本来就已经很复杂的天文数值模拟也是一个不小的挑战。

### 5.1.3 组合分析在天文模拟中的应用问题

要有效地指导用户调控模拟的演化趋势，除了能够对模拟结果实时显示外，还需要能够结合调控历史以及历史结果进行对比分析，这就需要组合分析和可视分析等相关技术。组合分析（ensemble）是通过对比同一模型在不同参数组合下的运行结果，来获得对模型的深层次认识，比如不稳定性、敏感性等，还用于辅助探索模型的参数空间。组合分析需要多次运行同一个模拟，从数学角度看参数空间都是无限的，每次运行不过是从参数空间中选取一个点，但是并不是每次参

数调整都是有明确依据或者能得到有参考价值的结果，如果模拟运行时间又比较长的话，组合分析的效率和效果都很难保证。

代理模型（Surrogate Model）[103][104]是用来提高组合分析性能的一种实用技术。这种方法利用稀疏网格理论[105]基于一个较小的参数空间子集（通过某种采样策略得到）构建一个对应的模拟结果空间子集，这个参数空间子集与结果空间子集的映射一般被成为原模拟的代理模型。这样对应任何一组输入参数，就可以在不必实际运行模拟的情况下，在初始的参数空间子集的基础上采用某种数学插值方法在结果空间中得到一个近似的结果。如果插值结果有参考价值，再实际运行这组参数对应的模拟，然后还可以将这组参数和对应的真实结果加入到初始采样空间，扩大采样范围，提高后续的插值精度。如此反复，直到得到预期结果。代理模型在使用前，需要有计划的选择参数采样点，然后遍历采样后得到的参数子集并逐一运行模拟才能得到初始的结果数据集。即使这样，对于很多长时间运行的天文模拟而言，计算时间上的消耗也是难以接受的。

除了性能，组合分析中另一个重要方面就是可视化设计。Denis 等人通过对内燃机等复杂工程的模拟过程进行任务分解和阶段细化，提出了一个综合的交互式驾驭计算工作流，整个系统基于参数空间、结果空间、特征空间的回归模型，通过集成最新的可视分析技术，帮助用户快速把握参数空间与模拟结果的深层次联系，从而加速内燃机的设计过程[106][107]。该方法特别适用于大量的能在短时间内完成的模拟，还无法应用到长时间演化的天文模拟。北京大学可视计算团队在大气模拟中使用朗格朗日距离矩阵来分析多个模拟中目标粒子的移动轨迹，同时借助 MapReduce 模型来适应大规模的模拟环境[42]，使得粒子移动特征的计算可以与模拟本身同步进行。该方法中利用 MapReduce 加速粒子特征的计算部分，跟本文第三章的设计思路比较相似，对如何在天文模拟中进行的实时组合分析有重要的借鉴意义。

综合上述，组合分析应用到天文模拟，除了性能上的考虑外，还需要一个集成度高和扩展性强的运行环境，既能够支持天文模拟的整个流程，又能针对具体的模拟问题进行灵活的定制。

### 5.1.4 参数分区在天文模拟中的意义和研究现状

组合分析的另外一个应用是辅助用户高效的探索参数空间。像天文模拟这样复杂的计算，输入参数与最后结果的关系并不直观，不同的参数组合甚至某一参数的微小变化，都有可能对模拟结果有较大的影响，所以探索参数和结果之间的关系对于优化模型有重要的意义。虽然参数和结果之间的映射规律很难把握，但往往某些参数对结果的某一方面影响较为显著，而对结果的其他方面则影响不大，

掌握这种参数分组特征对调控计算的方向有很强的指导作用，能够缩小参数调整的范围，增强人工调控的准确性，快速得到一个合理的初始化配置。

Steven 等人研发了 ParaGlide 框架[108]，该框架根据具体问题定义结果的特征空间，特征空间中的变量都是从结果中衍生出来的，然后将输入参数与结果表现之间的映射转换成输入参数与特征空间的映射。在运行期，根据特征值，反过来对参数空间进行聚类，从而把输入参数和结果特征的关系能够较为直观的表示出来。该方法在医学图像分割和燃料电池的设计中均表现良好。

Hrvoje 等[109]采用了将多次模拟结果叠加显示的方式，来分析不同参数组合对模拟结果的影响，该方法在使用上需要针对具体问题设计模拟结果特征的聚合（Aggregation）表现方式。Hrvoje 对于洪水模拟问题给出了一套可视分析方案，类似的思想也适用于交通方面的模拟。但对于模拟目标、模拟尺度和模拟形式都各异的天文模拟来说，则很难设计一套通用的聚合可视方案。

A. Johanne 等[110]提出了一种针对生物医学图像处理算法的参数调优的可视化方案，该方法将采样后的参数组织成聚类的层级结构，用户在聚类树的层次上对输出结果进行评价，逐层细化直到找到高质量结果对应的参数组合。W. Berger[111]借鉴统计学习的方法支持参数空间和多目标值的映射的连续分析，对用户指定区域的多目标值进行预测，并使用了点散图和并行坐标系来显示模型固有的不稳定性，该方法被成功应用于汽车发动机的设计过程。

如上所述，参数空间探索在很多领域已经积累了大量的研究成果和应用经验，为天文模拟中的参数分区提供了很好的借鉴。但天文模拟涉及了多物理场的相互作用，参数与结果之间的映射关系更为复杂，且天文模拟运行时间长，基于参数空间的大量采样（一个采样需要一次完整的运行）进行对比分析并不现实，所以，如果能在不显著影响其准确性的情况下加速组合分析的过程，则能将组合分析和驾驭式计算结合起来，为天文模拟整体流程的优化提供支持。

### 5.1.5 采用层级结构的原因

综合本节上面的内容，要在天文模拟中同时利用驾驭式计算和组合分析的优势，最重要的一点就是性能问题。天文模拟计算量大、运行时间长，反复的运行模拟会耗费大量的计算资源。层级式方法是将组合分析和驾驭式计算结合起来的一种折中方式，首先以较少的资源进行粗略的组合分析，尽量以最小的代价得到一个大致合理的参数范围，然后用正常的资源进行实际的模拟，模拟过程中，基于前期组合分析的结果以及中间结果的实时可视化，进行参数的精细调整。

层级的概念就体现在用较少的资源上。用较少的资源不能保证计算精度，这小部分的计算代价只是为后来的高精度模拟提供一个较为合理的配置，这样就避

免了模拟后期发现错误重头再来所带来的更大的代价。在高精度计算的过程中，采用驾驭式计算技术，对模拟过程和结果进行更精确的控制。数值方法的离散特性是对层级结构的一个完美映射，计算结果的精度在相当大的程序上取决于底层网格的细化程度。计算区域的网格越密，计算量越大，精度也越高，反之亦然。

Butaru 等[104]在研究利用代理模型和稀疏网格快速得到近似结果时，就提出了基于线下线上两个阶段进行迭代模拟的方法，线下阶段包括基于采样参数空间的计算，线上阶段运行实际的模拟，线下阶段为线上阶段提供快速的近似表示，而线上阶段的结果会反馈到线下阶段，提高近似的准确性。此处的两个阶段的精度是一样的，尚不构成层级的结构，不适用于长时间运行的天文模拟。

Walker[112]提出了另一种两阶段式的运行模拟的方法，其思想与层级结构相近，但具体实现却很独特。该方法采用一种称为模拟踪迹（trail）的数据结构记录下来用户在不同精度层级下对模拟进行交互式调控的历史，然后根据用这些历史的调控命令自动指导稍后的高精度的模拟。这个方法能够在基于平滑粒子的模拟中控制模拟的精度，同样对天文模拟中示踪粒子的精度控制有很好的借鉴意义。

Atanas 等[98]将 iFluid 应用于土木工程的模拟中，借助自适应网格的层级结构采用了逐层细化的方式运行。具体方法是在粗粒度网格下，充分利用计算量小但对用户动作反馈快的特点，进行大量的交互式探索，确定合理的参数配置，而配置一旦确定，就开始进行非交互式的高精度模拟。这里的粗粒度是一个相对量，粒度可以逐渐细化，直到能够得出正确的参数配置。这个思想也非常适合在天文模拟中使用。

因此，在借鉴上述研究成果的基础上，本文借助欧拉网格的层级式结构将组合分析和可视化驾驭计算的优点结合在一起，来对天文数值模拟的整个生命周期进行优化。

## 5.2 可视化驾驭计算框架的设计

按照前面的分析，本文针对天文模拟的特点提出了一个基于层级结构的驾驭式计算工作流程。如图 5-2 所示，该流程也是采用了两个阶段的思想：

第一个阶段是组合分析阶段，此阶段在不同的网格划分级别上进行，利用低分辨率的模拟所需资源少运行时间短的优势，可以进行多次快速的对比分析，对物理模型有一个初步的了解，例如大致的模拟演化过程、模型的不稳定性、参数敏感性、重点要关注的区域以及数值误差等等，同时为后来的高精度运行得出一个较为合理的初始化参数配置。

第二阶段是驾驭式计算阶段，根据组合分析阶段得到的初始化配置，在较高

的精度上运行模拟。系统为此阶段提供了多角度的专用视图把当前模拟状况及时地反馈给用户，用户也可以通过参数接口交互式地控制模拟进程。

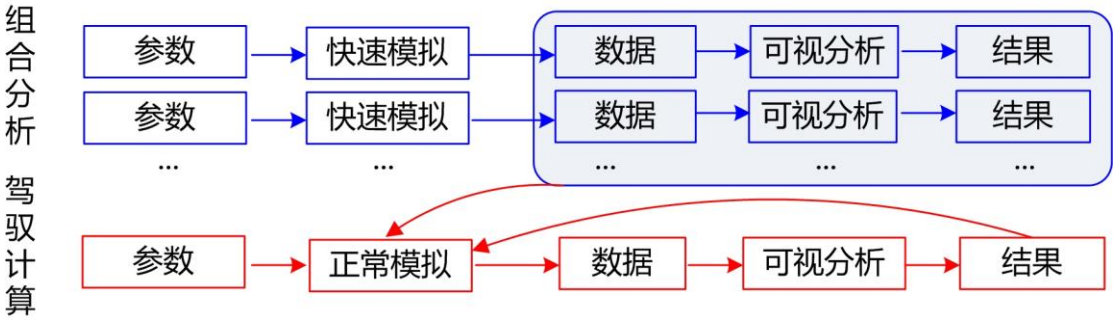
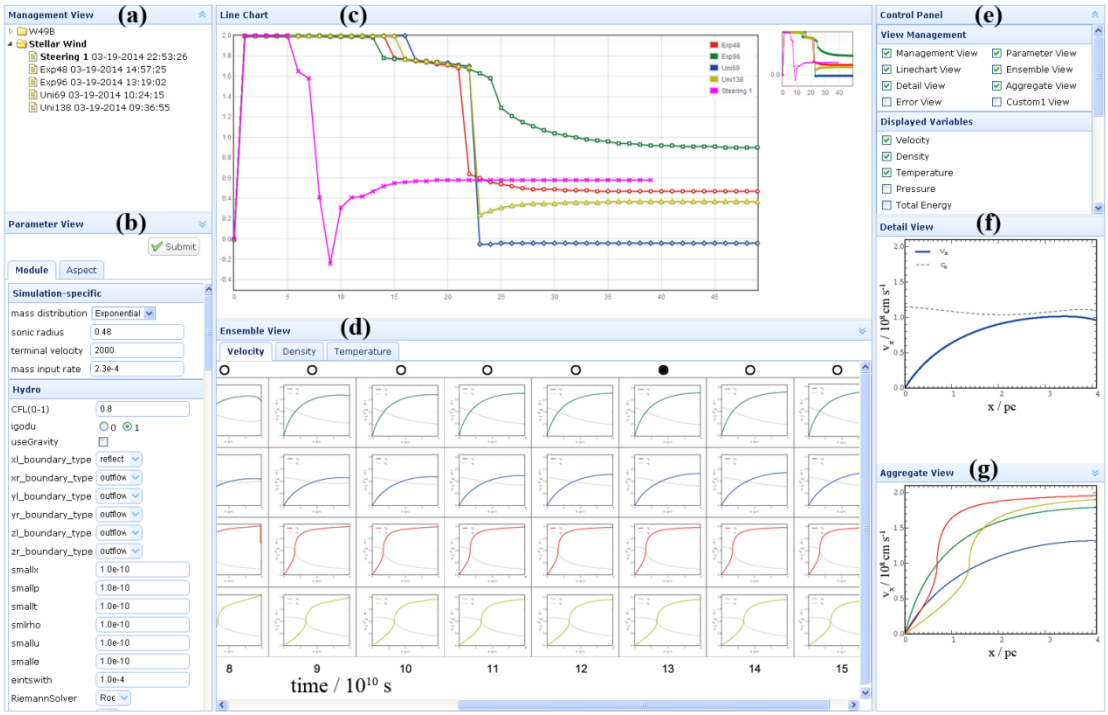



图 5-2　基于组合分析和可视化驾驭计算的工作流程。

组合分析阶段的主要目的是以较小的代价获得对模型的大致了解，除了采用较低的精度外，也可以利用其他方法，比如降低维度，也能节省大量的计算资源，从而快速的得到结果。

为现实上述的工作流程，本文提出了一个综合的可视化驾驭计算环境。图 5-3 显示了该集成环境的前端界面。该框架的主体由多个独立的组件构成，这些组件分别负责模拟实例的管理、参数分类和调整、对比分析、高维模拟结果显示、以及计算误差可视化等等。




(a) 模拟管理器；(b) 参数分组和修改接口；(c) 显示演化特征的趋势图；(d) 组合分析视图；(e) 视图控制面板；(f) 指定变量的详细视图；(g) 同时显示多个变量的聚合视图。

图 5-3　针对天文模拟的可视化驾驭计算环境的运行界面。

模拟管理器（图 5-3(a)）的设计与实现参考了 FLASH code 的一个辅助工具 Smaash[113]，Smaash 针对数据密集型的模拟，将多个模拟的中间数据和结果数据管理起来，方便用户分析。模拟管理器从日志文件和配置文件中自动的收集并整理每次模拟运行的信息，并按照其运行历史，组织成树状结构。用户可以选择感兴趣的那些模拟进行后续分析。视图控制面板（图 5-3(e)）用于视图和物理变量的显示管理，比如，选中的物理变量将被显示、选中的视图会出现在用户界面上。下面将分别介绍该环境中的可视化和可视分析的组件。

### 5.2.1 参数分区和调控

模型的初始化配置和运行时参数一般会被放到文本文件中，如果在运行中需要修改参数，用户不得不先暂停模拟，修改文件，然后再次启动，这是目前最普遍的天文模拟的运行方式。而且，天文模拟的参数非常之多，一般配置文件中只列出经常使用的参数信息，而大多数的参数都以缺省的形式起作用。由于天文模拟的涉及范围很广，对于很多涉及物理方面的参数，同一参数值在不同模拟下相差会很大，比如作为网格划分依据的梯度变化率、电子的比热等等。这些参数的配置错误往往比较隐蔽不易被发现，但对模拟结果的影响却很大。参数组件的设计目的就是为了让天文学家更加直观和准确的进行参数修改，提高参数调控的效率和效果。

天文模拟由于规模大，可调参数也非常之多，但天文模拟的诸多参数本身也有规律可循，完全采用探索式的参数管理方式并不十分有效[104][107]。参数调控组件的设计主要有以下两点考虑，第一，让用户能够清晰的了解每个参数的特点，比如参数的数学类型、与其他参数的依赖关系、约束条件等，防止用户在调解参数的时候出现违背物理约束和数值约束的错误；第二，通过提供合适的参数分组标准，由程序自动地解析配置文件，完成参数信息的提取和分组，提高参数调整的针对性。

图 5-3(b)显示了这个用户友好的参数接口。目前，该组件提供了针对四种参数类型的小部件和两个视图来支持运行期的参数修改。四种部件分别对应着四种数学性质的参数，常量型、二值性、离散型、连续型。参数之间的依赖由程序自动控制，比如一个开关量的改变，也会引起其他开关量的改变。如果用户对使用的模拟软件不是很熟悉的话，这类关联操作很容易会被忽略，从而导致错误的结果。两个视图对应于不同的参数分区标准。对于天文学家来讲，一般对物理模型相关的参数比较了解，但对于调解性能或计算本身相关的参数可能并不擅长。参数的横向分类视图中，参数按照用途被划分为四组，包括环境（资源）参数、数值（算法）参数、模拟（物理）参数以及通用参数。在纵向视图中，参数主要按

照物理模块来组织，控制同一个方程组的参数被放到一起。横向视图侧重优化模拟的某个方面，比如性能、网格粒度、数值误差等。纵向视图更适合精确的控制某个具体的物理过程，例如热传导、非电离平衡、冷却等。

虽然这种静态的参数分区可能无法充分挖掘出参数空间与结果空间的联系，但也有助于降低天文模拟的复杂性，对于大规模的多物理场的模拟，使得天文学家能够集中精力处理某一方面的问题。同时，参数组件还将原来人工方式的“暂停—修改—运行”的参数修改模式升级为“热部署”模式。

### 5.2.2 模拟演化趋势的特征线

图 5-3(c)所代表的特征线图从原始的模拟历史树形图演化而来[114]，如果模拟能够从某个中间结果重启，则模拟运行和重启的历史就能够以树的形式表现出来。下图可以展示出一个模拟在反复的优化过程中所经过的路径。但该图仅能表示路径，并不能显示在某一条路径上，用户所关注的信息的变化。

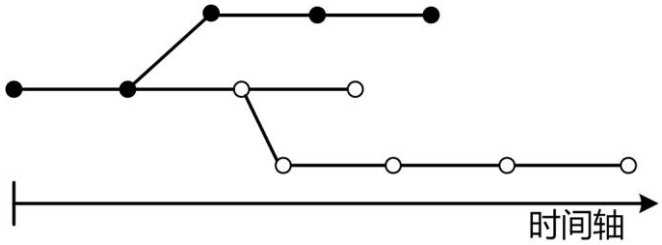



图 5-4　模拟运行历史的树状图[92]。

在本文设计的特征线图中，用户所关注的信息用特征来表示。特征是从模拟结果中提取出来的，可以直接对应模拟结果中一个的物理变量，或者是模拟结果的一个函数。很多情况下直接使用模拟结果并不能很好的反应出问题的本质，跟表象层面上的模拟结果相比，从结果中提出的特征则可以表达更深层的物理规律。例如，在非电离平衡的模拟中，模拟的直接结果为离子丰度，但大量的离子丰度并不能直观的反映出电离状态，只有转化为电离偏差(DI，参见 3.6.5 节)，才能定量地描述电离状态，并与观测现象做比较。

“参数—结果—特征”映射关系的可视化也是组合分析中常用的办法之一。这个特征映射关系在本文中称为特征函数，从数学角度来将，特征线图就是以特征函数的值为权重的曲线图。特征线图的横轴表示演化时间，纵轴代表特征函数的值。模拟的不同演化路线用不同的颜色标记。特征函数跟具体的模拟问题相关，使用者可以自定义。为了便于分析，特征线图支持放大和缩小操作。从特征线图上，可以很清楚的看到用户所关心的物理特征是如何演化的，同一个特征在各个模拟分支上的变化情况，以及对于特定参数区间的特征的变化范围等等。这种对比分析可以为运行期的参数调整提供定量的参考。

### 5.2.3 组合分析及其可视化

特征线图能很好的表现出用户感兴趣的区域的演化趋势，但并不擅长描述整个模拟区域的情况。鉴于线图本身的限制，虽然特征在一般情况下是全局范围的，但映射到特征线上，只能是一个采样区域（更多的时候是一个特定的网格点或者是一个粒子）对应一条线。大部分情况下，用户同时要关注的位置可能不止一个，这样每个采样点都需要对应一个特征线图。为了能让使用者能了解整个模拟空间上的情况，同样基于模拟历史的树状图（图 5-4），本文集成了一系列的视图组件，实现从多个角度对全局结果进行展示和对比。

如图 5-3(d)所示，该视图描述了同一模型多次运行中的结果画面。横轴表示演化时间，纵轴是同一物理量在不同运行实例中的结果。该视图上部的标签对应常用的物理量，视图控制面板（图 5-3(e)）负责管理物理量的显示和隐藏，例如，该图中目前可以查看速度、温度、密度等三个物理量的结果。为了方便对比分析，另外设计了大尺寸的详细视图（图 5-3(f)）和聚合视图（图 5-3(g)）。聚合视图也是组合分析中常用的一种可视化手段[109]，该方法将多次运行的结果在同一个视图中显示出来，用不同的颜色加以区分，十分方便不同运行实例之间的比较，尤其是对模型在不同参数组合下的相似特征的挖掘很有帮助。使用聚合视图的示例参见后面关于星风模拟的实验。

同样是用于对比分析，该系列视图跟特征线图的最大区别在于，特征线图擅长跟踪和表现特定区域的演化趋势，而组合视图是模拟结果的常规表示，从组合视图上能清楚的观察到模拟的整体演化趋势。一般情况下，组合视图是对模拟结果的直接可视化，这也是天文学家最习惯的表现方式。在模拟过程中，将组合视图和特征线图结合起来使用，能最大限度的发挥各自的优势。

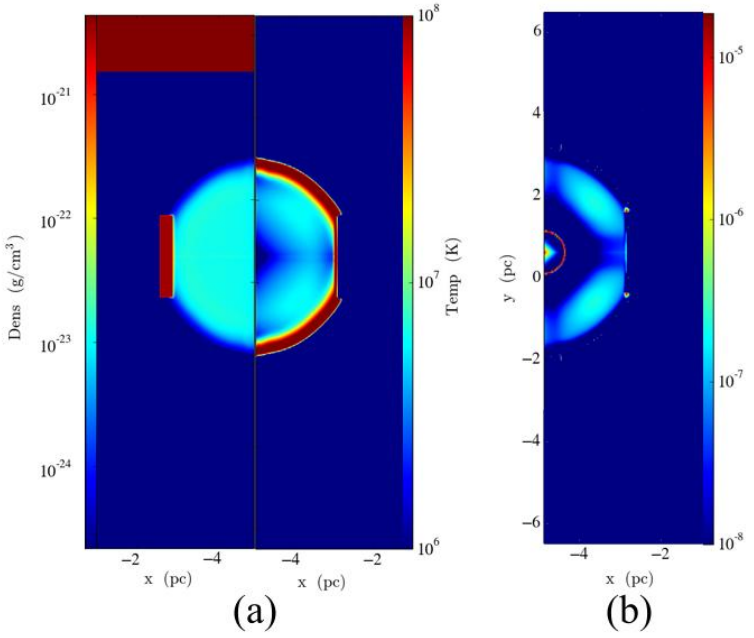
### 5.2.4 数值误差可视化

通常情况下，天文学家采用对物理场直接渲染的方式来观察模拟结果，对演化过程和结果的判断更多的依赖于他们对物理模型的理解和经验。但相关研究指出，自适应网格技术在某些情况下会引入较大的数值误差，而且大部分的误差都是与具体问题相关的[115]。较大的误差容易出现在网格密度变化迅速的区域，但同时这类区域也常常是用户所关注区域的边界。再加上误差的积累和传播，如果此时有较大的误差存在，可能会导致部分模拟结果不可信。基于这个原因，本文设计了一个数值误差可视化组件，帮助用户了解误差的分布情况以及误差出现的模式，比如当误差值较大时，及时折回到以前的某个中间点，修改参数后重启，而避免继续运行造成的资源浪费。

误差视图由三个组件构成，误差估值、误差过滤、以及误差显示。误差估值

组件根据物理模型及其求解器的特点，推导出误差或误差上限，并将误差的真实值转化成便于观察的形式；误差过滤器用于控制误差显示的阈值，低于某个阈值的误差将不会被显示，从而突出主要的误差；误差显示组件是把格式化后的误差结果在模拟区域中的分布情况可视化出来。

误差的产生和分布是与具体的数值算法和问题相关的，所以一些避免误差的通用原则往往在面对具体问题时不容易付诸实现，但在可视化的帮助下，具体问题中的误差出现模式是能够容易被发现的，从而使误差控制起来就更有针对性。由于大部分的数值误差都与运行时的自适应网格结构有联系，所以要结合网格结构以及相关模拟结果可视化，误差视图才能更好的发挥作用。如下图所示，在 W49B 模拟过程中，采用了热力图的形式来表现误差分布。左图是密度和温度的分布情况，右图表示离子丰度之和误差的分布，用于检查多流体一致性约束的满足情况（具体含义详见 2.4 节）。通过这三幅图的对比，可以很容易的发现，较大的数值误差都分布在物理变化剧烈的地方。变化剧烈的区域也意味着网格密度变化较大，这就要求自适应网格的划分标准要根据问题特点进行合理的设置。




(a)密度和温度分布；　　(b)离子丰度和的误差分布。

图 5-5　W49B 模拟中的离子丰度误差可视化。

## 5.3 天文模拟驾驭式计算环境的实现

驾驭式计算需要有可驾驭的目标，本文提出的可视化驾驭式计算环境还处于

原型阶段，目前可以连接到 FLASH 框架[6]，对基于 FLASH code 的天文模拟提供交互控制支持。虽然借助了 FLASH 框架，但该方法的核心功能并不依赖于具体的模拟及其底层的模拟框架。在实现上和 FLASH 的联系主要由两方面，一是向 FLASH 发送参数修改命令，这是通过介入 FLASH 的驱动模块将正确的设置同步到了各个 MPI 进程中来完成的，二是读取 FLASH 的配置文件和输出文件。这两部分都采用了模块化设计，可根据需要提供针对其他模拟框架的实现。图 5-6 描述了该集成环境的体系结构，主要分为三个部分：前端的用户界面，后端的控制逻辑和算法，以及基于 FLASH 的模拟程序。

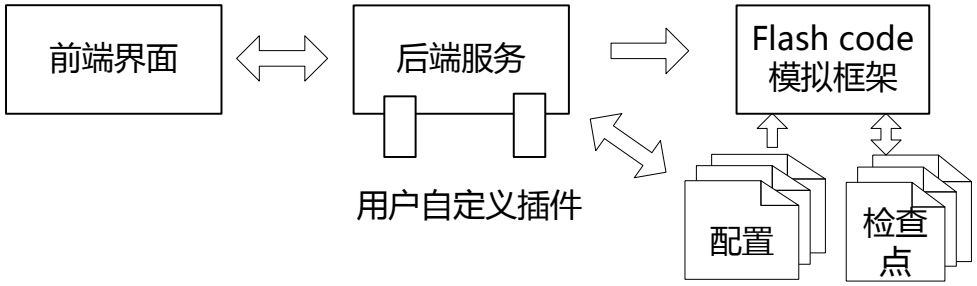



图 5-6　基于 FLASH code 实现的驾驭式计算框架结构。

前端表现层的功能和设计在第二章已经详细讨论过，这里不再赘述。表现层是基于 Web 方式的，其优点是可以方便的支持多用户的协同工作，只需要在运行模拟的集群上部署一个该环境的服务实例，用户就可以随时接入到系统中查看模拟的进展情况。

后端(Backend)是系统的核心，主要包括三个部分，模拟接口、插件结构、以及系统管理配置模块。这部分的主要功能是自动的收集配置信息和运行时信息，调用相关插件进行可视化，将可视化结果和数据发送到前端。模拟接口部分有一个守护进程，负责监视模拟结果的输出目录，收集配置信息、运行日志、中间结果文件，当有新的中间结果文件生成时，便调用对应的插件，直接生成图像或者是用于可视化的数据。而插件结构的设计主要是为了适用不同的模拟，无论是特征线图、组合分析视图、还是数值误差都是需要根据具体问题定义的，没有通用的可视化代码。可插拔的结构便于将特征函数的定义、误差计算方法、以及科学用户惯用的画图代码封装到可视化环境中。这部分的应用示例可以在后续的实验章节看到。系统管理模块主要是控制整个系统流程和提供一些通用的功能。在天文模拟领域，IDL 一直是主要的分析和画图工具，近年来，Python 已经成为天文模拟在后期处理中最常用的解释性语言之一，因此，系统的后端采用了 Python 语言实现，本文中出现的绝大部分的分析结果均是用基于 python 的 YT 可视化工具包[116]以及 Matplotlib[117]来实现的。

参数的自动提取和分类，以及运行时中间结果的收集都利用了 FLASH 框架

特有的结构。FLASH 主要有两类参数配置文件和三种输出文件[12]，系统中所有的参数信息都来自于对这两类参数文件的自动解析，而绝大部分的可视化结果都依赖于 FLASH 的输出文件。FLASH 两类参数文件包括初始化配置文件，和散落在各个模块内部的参数文件，这些模块专用的参数文件的存放位置是相对集中的，十分便于自动化的收集和分类。对于使用其他模拟框架（比如 ENZO）的程序来说，参数分区和调控这部分是需要重新开发的。

FLASH 的输出文件中，包括一个记录模拟初始信息和演化进程的日志文件，定期输出的检查点文件（checkpoint file，包含全部的中间结果，主要用于重启和详细分析），以低精度格式保存的部分物理量的文件（plot file，用于快速画图和分析）。目前系统并没有实现真正意义上的实时分析，即可视化的数据并不是直接来自模拟程序的内存，而是来自这些定期输出的文件。这样做的原因主要在遵守系统基本设计的前提下，减轻实现上的工作量，尽快完成可用的原型，以验证整个方法的可行性和效果。根据 5.1 节的分析可知，针对大规模模拟的实时可视化已经有了比较成熟的算法和工具，在生产环境中可以直接利用。虽然原型系统采用的是基于中间结果文件的可视化，但对可视分析和参数调控的效果并没有直接影响。为了尽量达到实时性能，本系统中可以通过触发式命令在任何时间点输出中间结果文件。

## 5.4 实验与评价

同前两章的内容一样，本文提出的基于层级结构的组合分析和可视化驾驭计算环境，也是作者所在科研团队与中科院紫金山天文台的领域专家紧密合作的成果。因此，本节从实际的模拟工作中选出了两个具有代表性的模拟实例来验证上面提到的方法。从数学角度，所有的模拟可以分为两大类，稳态和非稳态。前者的代表是星风（Stellar Wind），作为有稳态的物理现象，星风模拟会持续运行直到进入理论上的稳定状态，从统计角度看，模拟区域内的物理状态达到一种平衡，不再随时间而改变。超新星遗迹（SNR）可以作为后者的代表，此类模拟没有理论上的稳态，整个系统一直会随着时间而改变，模拟一般是在预定的时间尺度内进行，然后对其演化过程和最终结果进行分析。这里的稳态和非稳态都是相对的，演化后期各个物理量随时间变化都很小的模拟可以看作是进入了一个相对稳定的状态。

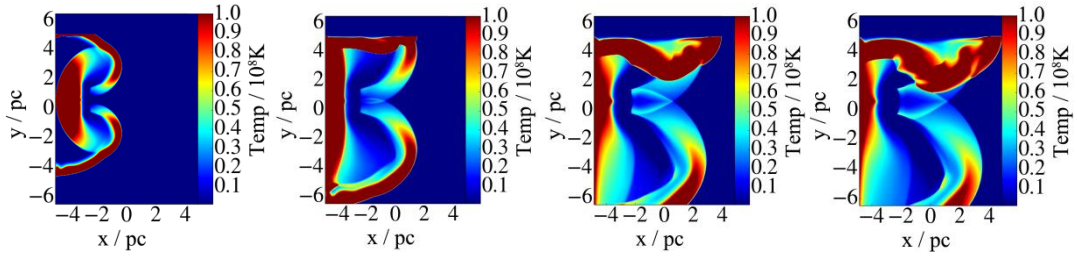
下面的实验分别针对这两类模拟展开。首先是可视化驾驭式计算系统在以 W49B 为目标天体的非电离平衡模拟中的应用，然后就该系统在星风模拟中的作用进行验证和评价。

### 5.4.1 W49B 的模拟

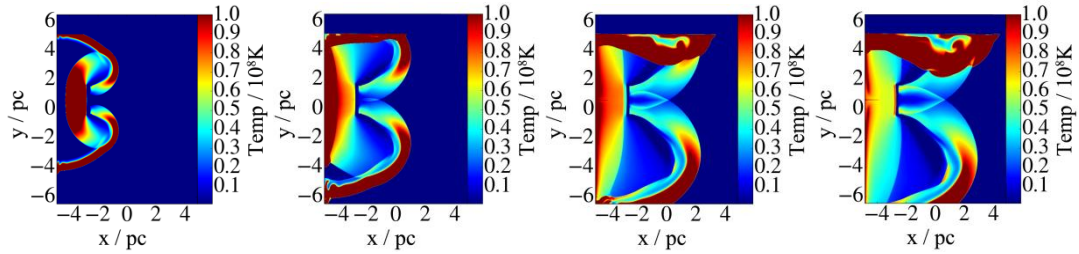
W49B 源于超新星爆炸后留下的遗迹。关于 W49B 的物理模型和对应的方程式可参见第二章和第三章中的相关内容。W49B 的模拟包括两大方面的内容，热动力学的演化历史和 Fe 离子电离状态的空间结构。

**低精度模式下的组合分析**

模拟开始前，需要确定物理模型，为后续高精度的模拟找到一个相对合理的初始配置。最初设计的物理模型中包含四个物理过程：气体动力模拟、电子热传导、辐射转移，非电离平衡，其中，电子热传导过程是否显著的影响了 W49B 的演化进程是无法通过理论推演确定的，只能根据模拟结果与观测结果的对比得知。当然，可以事先不做这个分析，而直接加入热传导过程，无论其对最终结果有没有显著影响，从理论上讲，结果总是对的，因为热传导总是会发生。但从第二章和第三章的实验数据得知，电子热传导涉及求解一个包含所有网格点的大规模线性方程组，十分消耗计算资源，如果能确定电子热传导在 W49B 的演化中不重要的话，那么可以节约 30%左右的计算资源和时间。




(a1) 950 years; (a2) 1585 years; (a3) 1900 years; (a4) 2220 years




(b1) 950 years; (b2) 1585 years; (b3) 1900 years; (b4) 2220 years

(a1-a4) 没有考虑热传导作用； (b1-b4)加入了热传导计算。

图 5-7 W49B 模拟中针对热传导现象的组合分析。

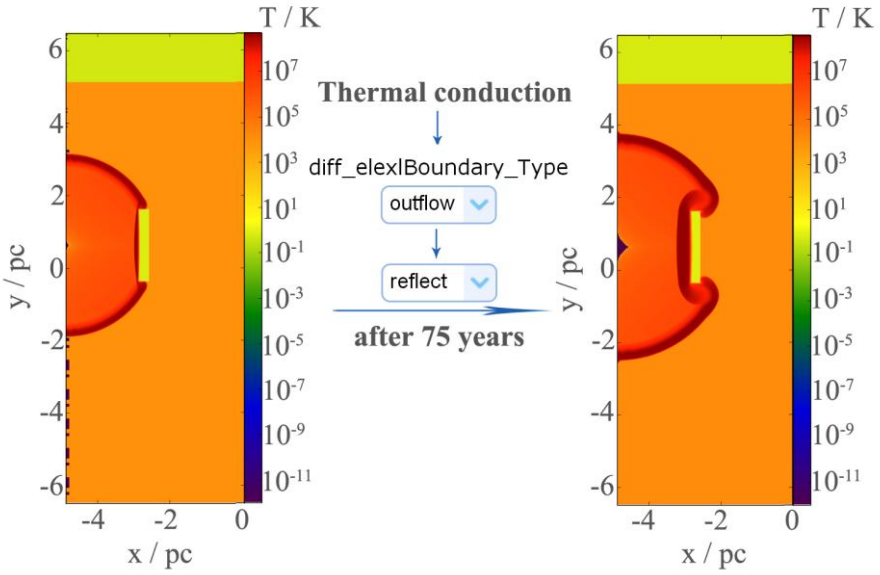
为了快速的了解电子热传导在 W49B 演化过程中的作用，在低精度下进行了两组实验，分别涉及和不涉及电子热传导的计算。此处利用了自适应网格的层级，低精度仅在网格最大划分级别为 5（lrefine_max = 5）的情况下进行，而正常精度所需的级别至少为 7。同时，为了进一步加快速度，演化步长的限制也被适当

的放宽了。

在组合分析视图（如图 5-7 所示）中，通过对比两个实验的结果，很容易确定了电子热传导在 W49B 的演化过程中不可忽略。低精度的组合分析总共花费的时间不到一天，而 W49B 模拟在正常精度的情况下，即使仅考虑 Fe 元素，72 颗 CPU 处理器连续运算了 20 天近 40 万步，产生了 180 个检查点文件，每个文件的大小从前期的 50MB 逐渐增加到后期的 500MB，还有 600 个较小的 plot 文件。由此可见，如果此处按照正常的精度来进行对比实验的话，将多耗费近 3.5 万个 CPU 小时。通过这组在低精度下的对比分析，快速的确定了 W49B 的模型中要包含的物理过程。接下来进入正常模拟阶段。

**数值算法的调控**

边界条件（border condition, BC）是求解偏微分方程时的一个重要参数，很多驾驭式计算框架都提供了运行时修改边界参数的功能[98][119]。然而在实际模拟中，尤其是涉及到多个物理场的模拟时，并不是所有的边界条件都能被准确的设置。当某些边界条件被使用者忽略时，默认值就会起作用，但这往往会造成非物理值。如图 5-8 所示，在 W49B 的早期模拟中，温度分布图上显示了多个很小的低温区块，而这些低温区块都沿着模拟的左边界排列。




(a) BC 修正前温度分布图（300 年）； (b) BC 修正后的温度分布图（375 年）。

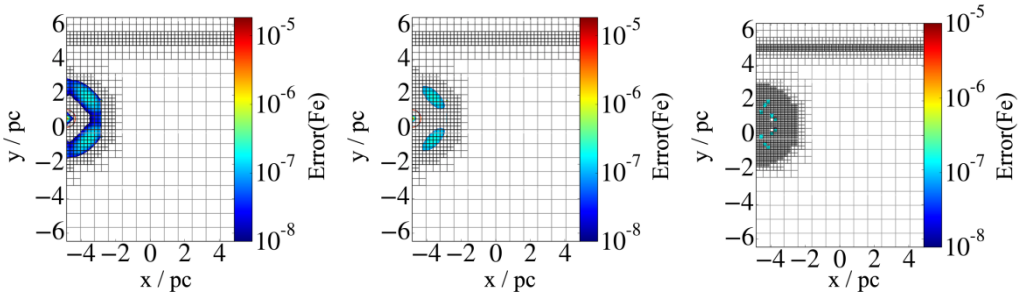
图 5-8 模拟运行中修改边界条件。

当天文学家注意到这些小的低温区域时，模型程序已经演化了 300 年。由于遗迹的快速绝热膨胀，原点附近出现的低温区属于正常的物理现象。而其他的低温块则明显是物理不能解释的。这时，最初的组合分析就派上了用场，通过对比原始尺寸的分析结果发现，在没有加载热传导模块时，这些低温区域并没有出现。

从而很容易联想到可能是电子热传导模块的配置出了问题。又考虑到这些非物理区块都沿着纵轴边界（左边界）分布，则很快确定了热传导求解器的左边界设置不正确（使用了默认值）。由于该模拟还出于演化的早期阶段，边界设置不当引入的错误能够在后续的演化中被逐渐抹平，所以修正边界参数之后就没有必要从头开始，而是可以如上图所示的那样，利用参数组件（图 5-3(b)）将边界值的修正信息直接发送到了正在运行的模拟中。

**数值误差的控制**

由于黎曼求解器的数值耗散作用，在非电离平衡的计算中，可能导致不能满足多流体流一致性约束（详见 2.4 节）。根据本文的测试结果，甚至在演化早期，在遗迹中心或爆炸边界等区域离子丰度和的误差就超过了可接受的误差上限，导致模拟不得不中止，这造成了模拟工作一度陷入停滞。为了了解此类数值误差出现的规律，以便在后面的模拟中将其限制在可接受的范围内，此处采用了组合分析的方法。如图 5-9 所示，通过对多组不同的网格细化层级和步长进行实验，将误差分布情况在系统提供的误差视图上表现出来，同时为了便于观察误差与网格以及其他物理量的关系，了解误差出现的模式，将误差显示在了自适应网格上。




(a) Filter=$10^{-8}$, refine=5; (b) Filter=$10^{-7}$, refine=5; (c) Filter=$10^{-7}$, refine=7;

(a)-(b) 相同网格层级不同过滤器下的误差分布图；(c) 在更细致的网格下的误差分布图。

图 5-9 数值误差可视化。

在上图中，误差定义为 Fe 元素所有离子的丰度之和与 Fe 元素的基准丰度的相对误差（经过了归一化处理），误差上限为 0.01%，调整误差过滤器去掉可以忽略的误差值，将主要误差突出的表现出来。通过对比在不同网格划分和步长组合的误差分布图，可以得出此类误差出现的大致模式为：多出现于物理量变化剧烈的区域，误差的大小跟网格划分层次和步长有关。步长越小，网格划分越密，误差也越小。但值得注意的是，在实际模拟中，不能比较随意地将步长限制在一个较小的范围，而又允许网格可以细分到比较高的层次，虽然精度会得到相应的提高，但这两种情况也都会导致计算量的大幅度上升。误差控制的主要目标是在可接受的误差范围内，找到精度和性能之间的最佳平衡点。经过多次实验和分析，

本文得出步长不超过$1.0 \times 10^4$ 秒且网格最大划分等于 *7* 的组合对于 W49B 的模拟来说是一个很好的折中。

在某些情况下，特征线图也能很好的辅助数值误差控制。用线图来监控误差，就是把误差也看作是模拟结果的一个特征。如图 5-10 所示，由于线图的限制，可以把全局或关键区域的最大数值误差作为特征。为了便于比较，还可以增加一条误差上限（最大可接收的误差），一旦特征线超过了上限，就要准备调整相关参数，然后折回到某个检查点重启模拟。

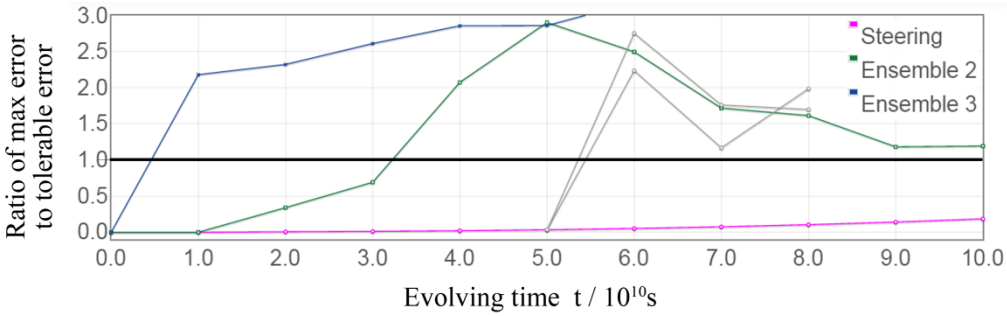



图 5-10　基于特征线图的数值误差可视化，黑色的粗实线代表误差上限，其他颜色的细线代表每次运行的误差变化，灰色线则表示当前运行中产生过的历史分支。

### 5.4.2 星风的模拟

星风（Stellar Wind）在宇宙中无处不在，是一类非常重要的天体物理现象。星风是星系内部向外快速喷射出的连续物质流，在宇宙的化学演化中扮演着重要的角色。星风的成因和物理模型的详细解释可参考[119]，此处的稳态星风模型由球面坐标下的欧拉方程来描述：

$$\frac{1}{r^2}\frac{\partial}{\partial r}(\rho v r^2) = \dot{m}(r) \tag{5-1}$$

$$\rho v \frac{\partial v}{\partial r} = -\frac{\partial P}{\partial r} - \dot{m}(r) v \tag{5-2}$$

$$\frac{1}{r^2}\frac{\partial}{\partial r}\left[\rho v r^2 \left(\frac{1}{2}v^2 + \frac{\gamma}{\gamma-1}\frac{P}{\rho}\right)\right] = \dot{E}(r) \tag{5-3}$$

其中，$v$，$\rho$，$P$ 分别是流体的速度、密度和压力，$\gamma$是绝热系数，$\dot{m}(r)$和$\dot{E}(r)$分别是单位体积内的质量和能量注入率，$r$ 是径向距离。与 W49B 中的流体模型（方程 2-1、2-2、2-3）不同，星风模型有质量和能量的注入，在 FLASH 的实现中，需要增加一个 Heat 模块来计算不断产生的质量和能量。

在已经达到稳态的星风中，流体的模式不随时间变化，沿着半径向外，星风中的物质密度、温度、喷射速度逐渐达到一个恒定的值，速度的这个恒定值被称

为终端速度（terminal velocity），这个值可以依据观测结果和相关理论推算出来。实际上本实验的最终目标是了解星风内部的物理分布特征，建立星风中的非电离平衡模型，此处是为了验证驾驭式计算环境的效果，所以只列出了相关的过程。

**降维模式下的组合分析**

除了数值算法本身的参数外，星风模拟要探索的参数主要有四个，质量注入率、终端速度、星团半径、以及星风内部的质量分布模型。其中，前两个参数已经通过观测数据和理论推导得到了。所以本模拟的重点是探索星团半径、质量分布模型与星风最终形态的联系。这种参数空间与结果空间的映射规律的探索，正是组合分析的优势所在。星风的三维模拟同样计算量巨大，为了快速确定几个合理的初始化配置，层级结构中的低分辨率阶段在此处具体表现为降维模拟，即借助球面坐标的对称性，将原始的三维模型退化到一维空间来进行计算。利用对称性来加速计算，在 W49B 的模拟中也使用过。理论上，降维处理对本阶段模拟的目标——为高精度的三维模拟确定星团半径、质量分布模型的合理组合，不仅没有影响，反而会大大加速探索和分析进程。

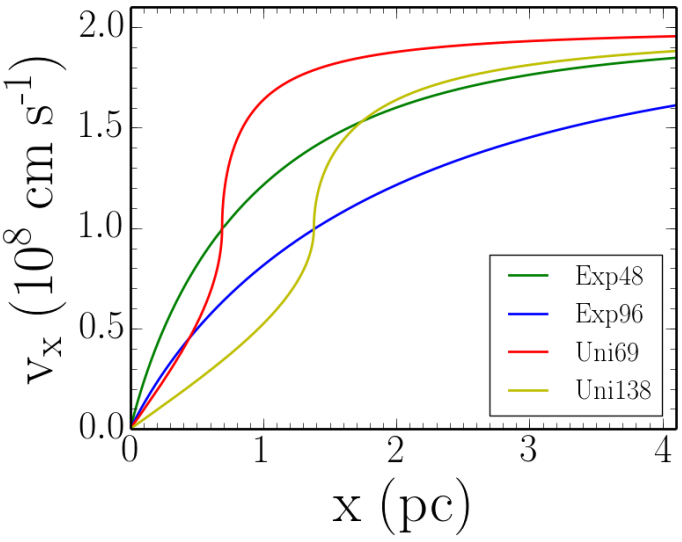
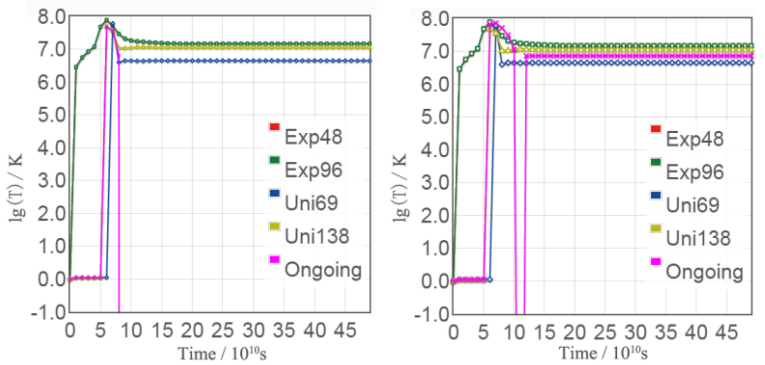



图 5-11 不同参数组合下的星风速度的对比分析。

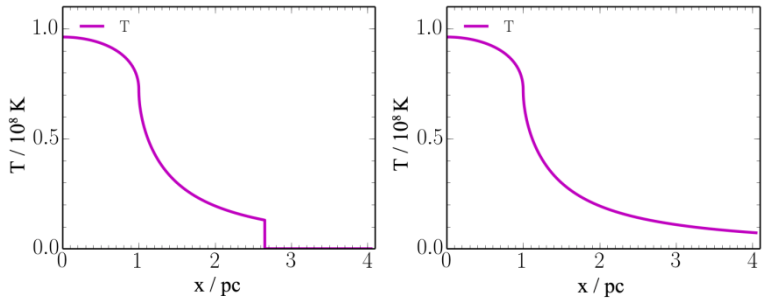
此阶段设计了星团半径和质量分布模型的四种组合，星团半径（声速半径）分别为 0.69*pc* 和 1.38*pc*，星云内部的质量分布模型分别为均匀（uniform）分布和指数分布。模拟过程中，系统监控了速度、密度、温度等三个物理量，图 5-3 显示了这一阶段的模拟的过程。组合分析视图（图 5-3(d)）能清楚的显示出模型的演化趋势和收敛（稳定）状态，再结合聚合视图（图 5-11），四种参数组合下的收敛规律也很容易看出来。这里的特征线图（图 5-3(c)）也可用于判断当前模拟是否达到了稳态，特征值可以定义为模拟边界处的速度和理论终端速度的差值。

### 数值算法的调控

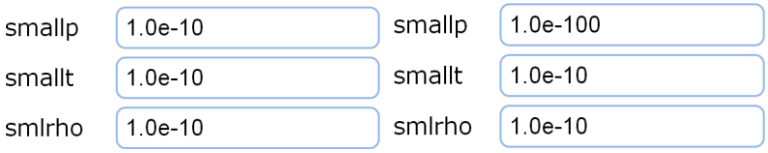
对天文学家而言，W49B 模拟中动态调整边界类型可能还相对容易，但很多数值错误隐蔽性更强，不容易找出原因。例如在星风模拟的探索阶段，在温度特征值的监控视图中接近外边界的区域，温度值突然下降到了一个非常低的水平，对于星风这样无激烈变化且连续的流体中，物理值的激烈变化是不符合规律的。理论上温度应逐渐下降并缓慢的接近背景温度。不过容易推断出，这个错误应该是源自流体求解器，通过使用垂直分类的参数视图（图 5-3(b)），发现流体模块的几个阈值的设置与物理模型本身有冲突，即星风中的合理值不在默认阈值规定的范围内，由于这类阈值仅是对数值算法中极值的限定，跟物理建模没有直接关系，所以在初始化中很容易被忽略，即使在 FLASH 的官方文档中也没有强调这类参数的副作用和注意事项。下图展示了在本文的可视化驾驭式计算环境中这个数值错误从发现到修正的过程。




(a1) 调控前的特征线图；　　(a2) 调控后的特征线图。




(b1) 调控前的温度曲线；　　(b2) 调控后的温度曲线。




(c1) 调控前的参数值；　　(c2) 调控后的参数值。

图 5-12　星风模拟中数值算法的参数调整。

## 5.5 应用中的实际问题

可视分析和驾驭式计算涉及范围和应用领域都比较广泛，目前已经有很多研究成果和应用。驾驭式计算需要可视化和可视分析的支持，二者的组合要在实际中发挥真正的作用，还需要与应用领域紧密结合，根据应用特点高度定制可视化策略和交互模式，目前并没有通用的工具。本文提出的基于层级结构的可视化驾驭计算环境，是与领域专家长期合作的结果，同时也是驾驭式计算在天文模拟领域的一次系统性尝试。该系统设计的最初目的是为长时间运行的天文模拟增加交互式控制的功能，文中所提到技术在其他可视化研究中也都有所体现，但并没有一个与本文类似的系统，能够对天文数值模拟的整个生命周期提供可视化驾驭式计算的支持。

### 5.5.1 可视化设计的欠缺和应对

在目前的实现中，系统在某些阶段并不能提供最佳的用户体验，比如组合分析阶段，仅支持特征线图和基于原始中间结果的简单可视化。针对这个缺点，将来要在合适的时候引入最新的可视分析技术来支持用户决策过程[107][120]，现阶段也可以通过增加系统插件的形式来应对某些特殊需求。图 5-6 所示的系统结构可以将用户自定义的可视化组件方便的连接到运行时环境，之所以这样设计，也是考虑到天文模拟的范围广泛，现阶段尚无法提供通用的可视化方案，用户和开发人员可以根据具体要求进行定制。

例如，在 W49B 的模拟中，为更直观的显示元素的电离状态，系统集成了一个基于矩阵图的可视化插件。如图 5-13 所示，矩阵图的左侧纵轴表示模拟区域中的采样点（在关键区域选取有代表性的点来进行分析是天文模拟中的常用手段），右侧纵轴记录这些采样点的时间序列，横轴则代表某种元素的各个离子。单元格的颜色用来标识对应离子的电离偏差。在该矩阵图中，来自多个不同演化阶段的离子的电离状态可以很方便的进行对比，按照时间分组，可以对比同一时刻不同区域的电离状态，按照空间分组，可以查看同一个区域内电离状态的演化趋势。此外，这种基于矩阵图的电离状态可视化也可以应用到其他多流体的模拟中，比如在核合成中表示元素丰度。

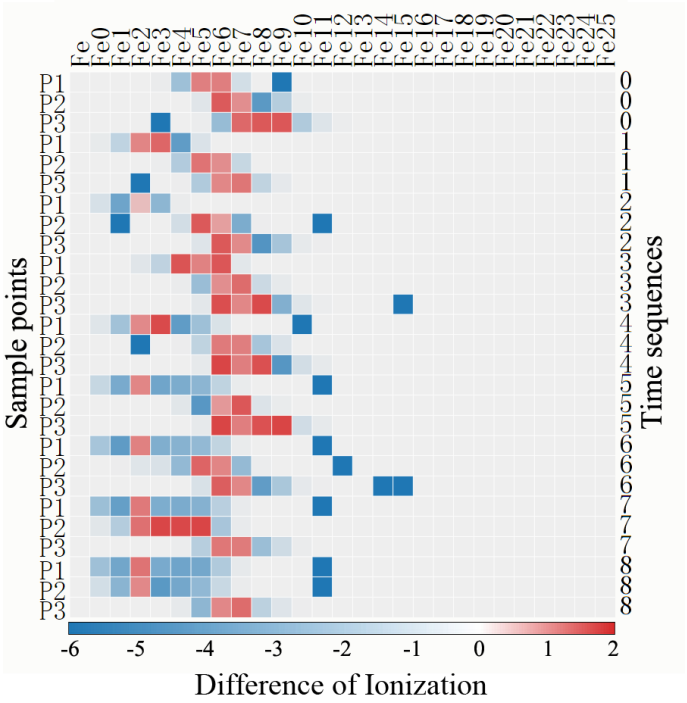



图 5-13 W49B 模拟中以矩阵图表示的 Fe 元素的电离状态。

### 5.5.2 低精度模拟的注意事项

本文提出的层级结构中，初始阶段推荐采用从低分辨率开始逐层细化的方式来快速得到对模拟全貌的一个大致了解并为后续模拟提供参考。在大多数情况下，低分辨率的组合分析是模拟正式运行之前的一个非常有用的准备阶段，能够帮助用户快速推导出一个相对合理的初始化配置，有效的缩小了后续模拟的探索空间，将对参数的精确调控留给了驾驭式计算阶段。但要提起注意的是，低分辨率并不能正确的反映出模拟的全部细节，因此，在应用渐进式的组合分析之前，确认低精度的模拟能够正确揭示所关心的部分或全部特性是非常重要的。

此外，降维处理是天文模拟中常用的手段之一，但仅限于模拟目标具有某种对称性。比如 W49B 是轴对称的圆柱状，可以退化为矩形的二维平面，星风则是球对称，可以沿着径向退化为一个线段。但降维处理一般由于受计算资源的限制，或是为了在初期简化计算，缩小探索范围，在实际中，三维模拟的结果最具有说服力。

因此，要在实际中应用本文提出的层级式驾驭计算方法，需要针对具体问题采用合适的策略，一般来说，基于低分辨率或降维的组合分析在天文模拟中有一定的普适意义。

### 5.5.3 其他不足和未来工作重点

对于大多数天文学家来讲，控制数值算法可能要比建立物理模型更复杂，本系统的参数视图在参数分区和调控方面能够为天文学家减轻一部分参数管理工作。但目前参数分区还是静态的，只能根据 FLASH 框架的特殊结构来进行。由于天文模拟运行的时间都比较长，现有的很多交互式参数聚类方法[108]并不适用，但文中对参数空间探索功能的支持还有很大的提升空间。

第三章采用了示踪粒子解决非电离平衡模拟问题，但本文尚未对粒子的可视化调控提供较好的支持。粒子的数量和初始分布对模拟结果的质量有很大的影响，如果能够在模拟过程中根据需要调整粒子的分辨率（增加粒子个数、拆分粒子等），将对提高模拟质量十分有益。

本文提出的驾驭式计算系统还处于原型阶段，仅将天文学家进行模拟工作的各个环节整合到了一个完整的集成环境中，但对某些环节的支持还相对薄弱，在后续的工作中，需要不断完善，重点加强交互式参数分区和通用的可视化设计两个方面。

## 5.6 本章小结

跟前面两章相比，本章的内容是从近几年的天文模拟工作中逐渐演化出来的成果，因而更具通用性。本章结合可视化、可视分析领域最新的研究成果，从如何加速整个天文模拟工作的流程出发，提出了一个基于层级结构的驾驭式计算系统；并将该系统成功的应用于两个基于 FLASH 框架的真实天文模拟实例。从应用的过程和效果看，该系统能够帮助天文学家快速的获得一个合理的初始化配置，并在后续的计算中，有效的增强了使用者对演化过程的交互式控制能力。

# 第六章 总结和展望

本文的研究内容属于计算机与天文数值计算的交叉学科，同时也是跟天文学家长期合作的成果。本文旨在借助计算机领域的相关技术提高计算密集、运行时间长的天文模拟的性能和效率。文中提到的诸多方案大都是基于计算机领域现有的技术，在此之前，MapReduce、GPU 加速、可视化、驾驭式计算等技术已经在其他很多领域，如生物信息学、化学动力学、流体力学、室内环境建模、心脏模拟等有了比较广泛的应用，但在天文模拟和分析中还没有引起足够的关注。另一方面，现代天文模拟普遍利用了高性能计算技术，但大部分的应用还限于比较浅层次的并行化，即基于 MPI 的模拟框架，对于一些有代表性的具体问题尚没有针对性的优化方法。

文本探索了天文模拟中的一些深层次优化方法，并尝试将通用的优化技术引入到一些有代表性的天文计算中。本文研究成果主要体现在性能加速和适应性上，本文并不是孤立的优化某个非电离平衡天文数值模拟个案，而是从现代大规模天文模拟软件的结构和模拟流程的共性出发，从不同的层次对天文数值模拟进行优化，进而提出了一个全方位的解决方案。除了成功解决了非电离平衡模拟的性能瓶颈外，本文的成果对优化其他天文模拟应用也极具参考价值。

## 全文工作小结

非电离平衡是天文领域中常见的计算，同时也是大部分模拟中比较耗时的部分。本文首先从理论和实验的角度出发，分析并验证了基于欧拉方法的传统模式在求解非电离平衡等反应流体时的性能瓶颈。然后分别从三个层次上对涉及反应流体的天文模拟提出了优化策略，利用可视分析和驾驭式计算技术优化整个模拟的流程，这是最高层次上的优化，该方法不仅限于非电离平衡，能够适用于大部分的长时间模拟。

在中间层次上，文本提出了一个两步走的优化策略，首先利用示踪粒子将反应计算从模拟的主程序中分离出来，对于非电离平衡来说，就是将非电离平衡求解器与流体求解器解耦；然后基于 MapReduce 的分布式架构来处理粒子轨迹，将后续的计算和分析任务纳入到一个并行的工作流中，同时支持后处理和就近分析两种模式。这个层次上的优化，主要体现在模拟程序架构上的改进，以及对粒子轨迹的处理上，优化的结果不仅使得程序的内部结构更加清晰，最重要的是避免了大量的内存消耗和额外的计算量（大量的平流方程），同时也大大减少了网

络通讯。优化后的方法，在相同的实验环境下，相比占用了全部资源的原有程序，在仅使用 1/4 的计算资源条件下，总体性能提升达到了 3 倍以上；同时这部分的工作也为进一步的细粒度优化提供了架构上的支持。

在细节层次上，本文结合非电离平衡方程的单体计算量小但数量巨大的特点，充分利用了 GPU 高度并行的结构，设计并实现了基于 GPU 的非电离平衡求解器，并针对多 CPU 和多 GPU 的混合架构，提出了基于任务队列和共享内存的任务分配策略，NEI 求解部分的性能提升达到了原算法的 15 倍。最后，还将该方法成功地用于光谱计算的加速。

文中的三个层面的优化方案并不是孤立的，基于 MapReduce 的并行工作流起到了承上启下的作用，一方面为可视化和驾驭式计算提供实时模式和就近模式的支持，另一方面为基于 GPU 的加速求解器提供任务输入和程序集成的接口。虽然这些成果主要都是针对非电离平衡模拟设计的，但其适用性都比较广泛，对于涉及多种元素反应变化的模拟（reactive flow）的性能优化问题都具有很好的参考价值。除了非电离平衡，文中还详细描述了本文方法在核合成、光谱计算和星风的模拟等方面的应用。

从整体上看，本文的核心工作集中在如何利用计算机领域的并行优化技术解决非电离平衡天文数值模拟一直以来面临的性能问题，虽然在计算机领域没有理论上的突破，但本文的应用效果很好。利用 MapReduce 和 GPU 来加速电离、光谱等计算，这在相关的天文模拟领域中尚无先例，更重要的是，本文的方法充分考虑了对实际物理问题的适应性，比较全面的总结了在不同物理条件下的应对策略，能够确保在获得性能提升的同时不损失计算精度，切实解决了天文学家面临的问题，对其他的天文模拟应用也有很好的借鉴意义。

### 不足和改进之处

鉴于本文中提出的大部分技术均是首次在非电离平衡、核合成、光谱计算中应用，所做的主要工作都集中在方案探索、流程疏通和可行性验证等方面，因此尚有很多不足和待改进的地方，除了各个具体方案中提到的一些特有的限制外，整体上的不足主要包括以下两个方面：

1. 由于实验环境以及时间关系，三个层次的优化方案还没有来得及在系统层面连接成一个统一的整体，各个优化策略都是在相对独立实验环境中完成的，没有能够提供一个同时涉及了文中所有方法的完整模拟案例。
2. 由于实验条件的限制，文中所有实验的规模都相对较小（240 个处理器，4 块 GPU 卡），没有能够在大规模的实验环境下进行验证是本文的一个明显不足。不过，文中所提到的基于 MapReduce 并行框架以及基于 GPU 的优化算法都具有很好的水平延展性，现有的实验能起到基本的验证效果。

## 未来工作

1. 本文目前的工作主要面向的是天文模拟软件的整体框架和结构，对于内部物理模型的具体计算，多为直接调用已有的成熟代码，或是将已有代码移植到 GPU 上。以后要进一步加强同领域专家的合作，在核心模型的建立和求解的细节上做好优化工作。
2. 优化模拟光谱和观测数据的拟合过程。对于天文领域而言，光谱含有丰富的物理状态的信息。理论模型中产生的大量光谱和观测数据的准确拟合是完善物理模型，正确解释物理现象的关键。拟合过程同样需要大量的计算，而目前天文领域在此方面的研究主要集中在评估模型本身，对计算过程的优化还没有深入展开。
3. 将非电离平衡模拟、可视分析、谱线拟合集成到一个统一的系统中，这同时也是本文长期的研究目标，即为天文数值领域提供一个开放、完整、并行化的非电离平衡数值模拟软件包。
4. 将文章中对非电离平衡的优化策略，全面扩展到核合成的计算。核合成的计算过程与非电离平衡类似，但其规模要大得多。由于其模型本身和计算时间上的复杂度，目前天文领域还没有一个通用的开源核合成计算代码库。著名的 FLASH 框架也仅提供了一个试验性的模块，很少能在实际模拟中发挥作用。核合成计算的研究可集中在通用的代码结构以及 GPU 加速方面。
5. 根据天文模拟程序运行时的内存结构以及自适应网格的特点，设计多 CPU 和多 GPU 混合体系下的并行可视化方案，为大规模天文模拟的驾驭式计算提供更好的支持。

# 参 考 文 献

[1] Keyes D E, McInnes L C, Woodward C, et al. Multiphysics simulations Challenges and opportunities [J]. International Journal of High Performance Computing Applications, 2013, 27(1): 4-83.

[2] Robert I. McLachlan and G. Reinout W. Quispel. Splitting methods. Acta Numerica [J], 2002, 11: 341-434.

[3] Ji L, Foster A R, Smith R K, et al. Non-equilibrium Ionization Diagnostics based on AtomDB [C]. NASA Laboratory Astrophysics Workshop, 2010. 1: 1.

[4] Smith R K, Hughes J P. Ionization equilibrium timescales in collisional plasmas [J]. The Astrophysical Journal, 2010, 718(1): 583.

[5] O. Kreylos, A. M. Tesdall, B. Hamann, et al. Interactive visualization and steering of CFD simulations [C]. In: *Proceedings of the symposium on Data Visualization, 2002*:25-34.

[6] B. Fryxell, K. Olson, P. Ricker, et al. FLASH: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes [J], The Astrophysical Journal Supplement Series, 2000, 131:273–334

[7] Bryan G L, Norman M L, O'Shea B W, et al. ENZO: An Adaptive Mesh Refinement Code for Astrophysics[J]. The Astrophysical Journal Supplement Series, 2014, 211: 19.

[8] Jian Xiao, Xingyu Xu, Jizhou Sun, et al. Acceleration of solving non-equilibrium ionization via tracer particles and mapreduce on eulerian mesh [C]. In: Algorithms and Architectures for Parallel Processing, 2014:29–42

[9] Gingold, R. A., Monaghan, J. J. Smoothed particle hydrodynamics - Theory and application to non-spherical stars [J]. Monthly Notices of the Royal Astronomical Society, 1977, 181: 375-389.

[10] Berger, M. J., Colella, P. Local adaptive mesh refinement for shock hydrodynamics [J]. Journal of Computational Physics, 1989,82:64-84

[11] Dubey A, Antypas K, Ganapathy M K, et al. Extensible component-based architecture for FLASH, a massively parallel, multiphysics simulation code [J]. Parallel Computing, 2009, 35(10): 512-522.

[12] FLASH Center for Computational Science, University of Chicago. FLASH User’s Guide. 2012

[13] Almgren A S, Bell J B, Lijewski M J, et al. Nyx: A massively parallel amr code for computational cosmology [J]. The Astrophysical Journal, 2013, 765(1): 39

[14] Springel V., Yoshida N., White S. D. M. GADGET: a code for collisionless and gasdynamical cosmological simulations [J]. New Astronomy, 2001, 6(51): 79–117.

[15] Springel V., The cosmological simulation code GADGET-2 [J]. Monthly Notices of the Royal Astronomical Society, 2005, 364(4): 1105-2015.

[16] Springel V., E pur si muove: Galilean-invariant cosmological hydrodynamical simulations on a moving mesh [J]. Monthly Notices of the Royal Astronomical Society, 2010, 401(2): 791-851.

[17] Shi Y, Green W H, Wong H W, et al. Redesigning combustion modeling algorithms for the Graphics Processing Unit (GPU): Chemical kinetic rate evaluation and ordinary differential equation integration [J]. Combustion and Flame, 2011, 158(5): 836-847

[18] Shi Y, Green W H, Wong H W, et al. Accelerating multi-dimensional combustion simulations using GPU and hybrid explicit/implicit ODE integration [J]. Combustion and Flame, 2012, 159(7): 2388-2397

[19] Long M, Jordan IV G C, van Rossum D R, et al. Three-dimensional Simulations of Pure Deflagration Models for Thermonuclear Supernovae [J]. The Astrophysical Journal, 2014, 789(2): 103.

[20] Genel S, Vogelsberger M, Nelson D, et al. Following the flow: tracer particles in astrophysical fluid simulations [J]. Monthly Notices of the Royal Astronomical Society, 2013, 435(2): 1426-1442.

[21] Reale F, Orlando S. Nonequilibrium of ionization and the detection of hot plasma in nanoflare-heated coronal loops[J]. The Astrophysical Journal, 2008, 684(1): 715.

[22] Peter MacNeice, Kevin M. Olson, Clark Mobarry, et al. PARAMESH: A parallel adaptive mesh refinement community toolkit [J], Computer Physics Communications, 2000, 126(3): 330-354.

[23] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, et al. Introduction to Algorithms, third edition [M]. MIT Press, 2000

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

[25] Timmes F X. Integration of nuclear reaction networks for stellar hydrodynamics [J]. The Astrophysical Journal Supplement Series, 1999, 124(1): 241.

[26] Timmes F X, Hoffman R D, Woosley S E. An inexpensive nuclear energy generation network for stellar hydrodynamics [J]. The Astrophysical Journal Supplement Series, 2000, 129(1): 377.

[27] Ji L, Wang Q D, John K. A non-equilibrium ionization code and its applications[C]. Bulletin of the American Astronomical Society. 2005, 37: 1447.

[28] Plewa T, Müller E. The consistent multi-fluid advection method [J]. Astron. Astrophys, 1999, 342: 179-191.

[29] Toro E F. Riemann solvers and numerical methods for fluid dynamics: a practical introduction [M]. Springer Science & Business Media, 2009.

[30] Dubey A, Daley C, ZuHone J, et al. Imposing a Lagrangian particle framework on an Eulerian hydrodynamics infrastructure in FLASH [J]. The Astrophysical Journal Supplement Series, 2012, 201(2): 27.

[31] Dubey A, Antypas K, Daley C. Parallel algorithms for moving Lagrangian data on block structured Eulerian meshes [J]. Parallel Computing, 2011, 37(2): 101-113.

[32] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

[33] Stone, J.E., Gohara, D., Guochun Shi. OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems [J]. Computing in Science & Engineering, 2010, 12(3): 66-73.

[34] 张舒，褚艳丽，赵开勇等. GPU 之高性能运算之 CUDA [M]. 2009.

[35] Farber R. CUDA application design and development [M]. Elsevier, 2011.

[36] Michael McCool, James Reinders, Arch Robison. Structured Parallel Programming: Patterns for Efficient Computation. Elsevier, 2013.

[37] Lindholm, E., Nickolls, J., Oberman, S., et al. NVIDIA Tesla: A Unified Graphics and Computing Architecture [J]. Micro, IEEE, 2008, 28(2): 39-55.

[38] NVIDIA Corporation, CUDA Math Libraries Performance Report [EB/OL], http://developer.download.nvidia.com/compute/cuda/6_0/rel/docs/CUDA_6_Performance_Report.pdf，2014.

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

[40] Niemeyer K E, Sung C J. Accelerating moderately stiff chemical kinetics in reactive-flow simulations using GPUs [J]. Journal of Computational Physics, 2014, 256: 854-871.

[41] Brodlie K, Brooke J, Chen M, et al. Visual supercomputing: technologies, applications and challenges [C]. In: Computer Graphics Forum. Blackwell Publishing Ltd., 2005, 24(2): 217-245.

[42] Guo H, Yuan X, Huang J, et al. Coupled ensemble flow line advection and analysis [J]. IEEE Transactions on Visualization and Computer Graphics, 2013, 19(12): 2733-2742.

[43] Ekanayake, J., Pallickara, S., Fox, G. Mapreduce for data intensive scientific analyses [C]. In: IEEE Fourth International Conference on eScience, 2008: 277–284.

[44] Tu T, Rendleman C A, Borhani D W, et al. A scalable parallel framework for analyzing terascale molecular dynamics simulation trajectories[C]//High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for. IEEE, 2008: 1-12.

[45] Plimpton S J, Devine K D. MapReduce in MPI for large-scale graph algorithms [J]. Parallel Computing, 2011, 37(9): 610-632.

[46] Plimpton S J, Shead T. Streaming data analytics via message passing with application to graph algorithms [J]. Journal of Parallel and Distributed Computing, 2014, 74(8): 2687-2698.

[47] Li M, Vazhkudai S S, Butt A R, et al. Functional partitioning to optimize end-to-end performance on many-core architectures [C]. In: the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE Computer Society, 2010: 1-12.

[48] Dorier M, Antoniu G, Cappello F, et al. Damaris: how to efficiently leverage multicore parallelism to achieve scalable, jitter-free i/o [C]. In: the 2012 IEEE International Conference on Cluster Computing. IEEE, 2012: 155-163.

[49] Dorier, M, Sisneros, R, Peterka, T, et al. A Nonintrusive, Adaptable and User-Friendly In Situ Visualization Framework [C]. In: the 2013 IEEE Symposium on Large-Scale Data Analysis and Visualization. IEEE, 2013:67-75.

[50] Vishwanath V, Hereld M, Papka M E, et al. In Situ Data Analysis and I/O Acceleration of FLASH Astrophysics Simulation on Leadership-Class System Using GLEAN [J]. Proc. SciDAC, Journal of Physics: Conference Series. 2011.

[51] Vishwanath V, Hereld M, Papka M E. Toward simulation-time data analysis and I/O acceleration on leadership-class systems [C]. In: 2011 IEEE Symposium on Large Data Analysis and Visualization (LDAV), IEEE, 2011: 9-14

[52] Dubey A, Daley C, Weide K. Challenges of computing with flash on largest hpc platforms [C]. In: International Conference of Numerical Analysis and Applied Mathematics 2010. 2010, 1281(1): 1773-1776

[53] Latham R, Daley C, Liao W, et al. A case study for scientific I/O: improving the FLASH astrophysics code [J]. Computational Science & Discovery, 2012, 5(1): 015001.

[54] Fisher R T, Kadanoff L P, Lamb D Q, et al. Terascale turbulence computation using the FLASH3 application framework on the IBM Blue Gene/L system [J]. IBM Journal of Research and Development, 2008, 52(1.2): 127-136.

[55] Dubey A, Calder A C, Daley C, et al. Pragmatic optimizations for better scientific utilization of large supercomputers [J]. International Journal of High Performance Computing Applications, 2013, 27(3): 360-373.

[56] Bader G, Deuflhard P. A semi-implicit mid-point rule for stiff systems of ordinary differential equations[J]. Numerische Mathematik, 1983, 41(3): 373-398.

[57] Dubey A, Antypas K, Calder A C, et al. Evolution of FLASH, a multi-physics scientific simulation code for high-performance computing [J]. International Journal of High Performance Computing Applications, 2014, 28(2): 225-237

[58] W. Gropp, E. Lusk, and R. Thakur. Using MPI-2: Advanced Features of the Message-Passing Interface [M]. MIT Press, 1999

[59] Zhou X, Miceli M, Bocchino F, et al. Unveiling the spatial structure of the overionized plasma in the supernova remnant W49B [J]. Monthly Notices of the Royal Astronomical Society, 2011, 415(1): 244-250.

[60] Robert D. Falgout, Ulrike Meier Yang. HYPRE: a Library of High Performance Preconditioners [C]. Lecture Notes in Computer Science, 2002: 2331:632-641.

[61] Dokainish M A, Subbaraj K. A survey of direct time-integration methods in computational structural dynamics - I. Explicit methods [J]. Computers & Structures, 1989, 32(6): 1371-1386.

[62] Subbaraj K, Dokainish M A. A survey of direct time-integration methods in computational structural dynamics - II. Implicit methods [J]. Computers & Structures, 1989, 32(6): 1387-1401.

[63] Brian Gough. GNU Scientific Library Reference Manual - Third Edition [M]. Network Theory Ltd, 2009.

[64] Scott D. Cohen, Alan C. Hindmarsh. CVODE, a stiff/nonstiff ODE solver in C. Computers in Physics. 1996, 10(2): 138-143.

[65] A. C. Hindmarsh. ODEPACK, A Systematized Collection of ODE Solvers. IMACS Transactions on Scientific Computation, 1983(1): 55-64.

[66] Intel Corporation. Intel Math Kernel Library (Intel MKL) [EB/OL]. http://software.intel.com/en-us/intel-mkl, 2015.

[67] Kuniaki Masai. X-ray emission spectra from ionizing plasmas [J]. Astrophysics and Space Science, 1984, 98(2): 367-395.

[68] Agullo, E., Demmel, J., Dongarra, J., et al. Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects [J]. Journal of Physics: Conference Series, 2009, 180:1.

[69] Demidov D, Ahnert K, Rupp K, et al. Programming CUDA and OpenCL: a case study using modern C++ libraries [J]. SIAM Journal on Scientific Computing, 2013, 35(5): C453-C472.

[70] K. Rupp, F. Rudolf, J. Weinbub. ViennaCL - A High Level Linear Algebra Library for GPUs and Multi-Core CPUs [C]. In: International Workshop on GPUs and Scientific Applications, 2010: 51-56.

[71] Shuo Chen; Xiaoming Li, A hybrid GPU/CPU FFT library for large FFT problems [C]. In: the IEEE 32nd International Performance Computing and Communications Conference (IPCCC), 2013: 1-10.

[72] Alejandro Acosta, Vicente Blanco, Francisco Almeida. Dynamic load balancing on heterogeneous multi-GPU systems [J]. Computers & Electrical Engineering, 2013, 39(8): 2591-2602.

[73] Yang B, Lu K, Liu J, et al. Hybrid Embarrassingly Parallel algorithm for heterogeneous CPU/GPU clusters[C]. In: the IEEE 7th International Conference on Computing and Convergence Technology (ICCCT), 2012: 373-378.

[74] Mulansky M, Ahnert K. Metaprogramming Applied to Numerical Problems [C]. In: American Institute of Physics Conference Series. 2011, 1389: 1582-1585.

[75] Karsten Ahnert, Mario Mulansky. Odeint – Solving Ordinary Differential Equations in C++ [C]. In: Proceedings of the International Conference on Numerical Analysis and Applied Mathematics 2011, 1389:1586-1589

[76] Lionetti F V, McCulloch A D, Baden S B. Source-to-source optimization of CUDA C for GPU accelerated cardiac cell modeling [M]. Euro-Par 2010-Parallel Processing. Springer Berlin Heidelberg, 2010: 38-49.

[77] Garcia V M, Liberos A, Climent A M, et al. An adaptive step size GPU ODE solver for simulating the electric cardiac activity[C]. Computing in Cardiology, 2011. IEEE, 2011: 233-236

[78] 罗力, 杨超, 赵宇波, 等. CPU/GPU 集群上求解偏微分方程的可扩展混合算法[J]. 集成技术, 2012 (1): 84-88.

[79] Burgess A, Summers H P. The Effects of Electron and Radiation Density on Dielectronic Recombination [J]. The Astrophysical Journal, 1969, 157: 1007.

[80] S. I. Feldman, D. M. Gay, M. W. Maimone, et al. Availability of f2c—a Fortran to C converter [J]. SIGPLAN Fortran Forum, 1991, 10(2): 14-15.

[81] Noble M S. Getting more from your multicore: exploiting OpenMP from an open-source numerical scripting language [J]. Concurrency and Computation: Practice and Experience, 2008, 20(16): 1877-1891.

[82] Noble M S, Nowak M A. Beyond XSPEC: Toward highly configurable astrophysical analysis [J]. Publications of the Astronomical Society of the Pacific, 2008, 120(869): 821-837.

[83] Noble M S, Ji L, Young A, et al. Parallelizing the XSTAR Photoionization Code [C]. In: the Astronomical Data Analysis Software and Systems XVIII. 2009, 411: 301.

[84] Randall Smith. Calculating Radiative Recombination Continuum from Hot Plasma [BE/OL]. http:// www.atomdb.org/Physics/rrc.pdf, 2008.

[85] Smith R K, Brickhouse N S, Liedahl D A, et al. Collisional plasma models with APEC/APED: emission-line diagnostics of hydrogen-like and helium-like ions [J]. The Astrophysical Journal Letters, 2001, 556(2): L91.

[86] Arumugam K, Godunov A, Ranjan D, et al. An efficient deterministic parallel algorithm for adaptive multidimensional numerical integration on GPUs [C]. In: 42nd International Conference on Parallel Processing (ICPP), 2013: 486-491.

[87] Arumugam K, Godunov A, Ranjan D, et al. A memory efficient algorithm for adaptive multidimensional integration with multiple GPUs [C]. In: 20th International Conference on High Performance Computing (HiPC), 2013: 169-175.

[88] Daniel Thuerck, Sven Widmer, Arjan Kuijper et al. Efficient heuristic adaptive quadrature on GPUs: Design and Evaluation [C]. In: 10th International Conference on Parallel Processing and Applied Mathematics, 2013: 652-662.

[89] Giuliano Laccetti, Marco Lapegna, Valeria Mele, et al. A study on adaptive algorithms for numerical quadrature on heterogeneous GPU and multicore based systems [C]. In: 10th International Conference on Parallel Processing and Applied Mathematics, 2013: 704-713.

[90] Petra Wenisch, Computational Steering of CFD Simulations on Teraflop Supercomputers [D]. Technical University of Munich, Germany, 2007.

[91] Mulder J D, van Wijk J J, van Liere R. A survey of computational steering environments [J]. Future generation computer systems, 1999, 15(1): 119-129

[92] Wright H, Crompton R H, Kharche S, et al. Steering and visualization: Enabling technologies for computational science [J]. Future Generation Computer Systems, 2010, 26(3): 506-513.

[93] G.A. Geist II, J.A. Kohl, P.M. Papadopoulos et al. CUMULVS: Providing fault tolerance, visualization, and steering of parallel applications [C]. In: The International Journal of Supercomputer Applications and High Performance Computing, 1997, 11(3):224–235.

[94] Parker, S.G., Johnson, C.R., SCIRun: A Scientific Programming Environment for Computational Steering [C]. In: Proceedings of Supercomputing, 1995: 52

[95] J. Knezevic, R.-P. Mundani, E. Rank, et al. Extending the SCIRun Problem Solving Environment to Large-Scale Applications [C]. In: Proceedings of Applied Computing, 2012: 171-178

[96] Pickles S M, Haines R, Pinning R L, et al. A practical toolkit for computational steering [J]. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 2005, 363(1833): 1843-1853.

[97] Riedel M, Eickermann T, Habbinga S, et al. Computational steering and online visualization of scientific applications on large-scale hpc systems within e-science infrastructures [C]. In: IEEE International Conference on e-Science and Grid Computing, 2007: 483-490.

[98] Atanasov A, Bungartz H J, Frisch J et al. Computational steering of complex flow simulations [C]. In: Transactions of the Fourth Joint HLRB and KONWIHR Review and Results Workshop. Munich, Germany, 2009: 63-74.

[99] Esnard A, Richart N, Coulaud O. A steering environment for online parallel visualization of legacy parallel simulations [C]. In: 10th IEEE International Symposium on Distributed Simulation and Real-Time Applications, Terremolinos, Spain, 2006: 7-14.

[100] N. Richart, A. Esnard, O. Coulaud. Toward a computational steering environment for legacy coupled simulations [C]. In: 6th International Symposium on Parallel and Distributed Computing, Hagenberg, Austria, 2007: 319-326.

[101] C. Wagner, M. Flatken, M. Meinel, et al. Fssteering: A distributed framework for computational steering in a script-based CFD simulation environment [C]. In: 6th High-End Visualization Workshop, Obergurgl, Austria, 2010.

[102] D. Jenz and M. Bernreuther. The computational steering framework stereo [C]. In: 10th International Conference on Applied Parallel and Scientific Computing, Reykjav ́k, Iceland, 2010.

[103] Daniel Butnaru, Dirk Pflüger, Hans-Joachim Bungartz. Towards High-Dimensional Computational Steering of Precomputed Simulation Data using Sparse Grids [J]. Procedia Computer Science, 2011, 4:56-65

[104] Butnaru D, Buse G, Pfluger D. A parallel and distributed surrogate model implementation for computational steering[C]. In: IEEE 11th International Symposium on Parallel and Distributed Computing, 2012: 203-210.

[105] H.-J. Bungartz, M. Griebel. Sparse grids [J]. Acta Numerica, 2004, 13: 147-269.

[106] Matkovic K, Gracanin D, Jelovic M, et al. Interactive visual steering—rapid visual prototyping of a common rail injection system [J]. IEEE Transactions on Visualization and Computer Graphics, 2008, 14(6): 1699-1706.

[107] Matkovic K, Gracanin D, Splechtna R, et al. Visual Analytics for Complex Engineering Systems: Hybrid Visual Steering of Simulation Ensembles [J]. IEEE Transactions on Visualization and Computer Graphics, 2014, 20(12).1803-1812

[108] Bergner S, Sedlmair M, Moller T, et al. ParaGlide: Interactive parameter space partitioning for computer simulations [J]. IEEE Transactions on Visualization and Computer Graphics, 2013, 19(9): 1499-1512.

[109] Ribicic H, Waser J, Fuchs R, et al. Visual analysis and steering of flooding simulations [J]. IEEE Transactions on Visualization and Computer Graphics, 2013, 19(6): 1062-1075.

[110] Pretorius A J, Bray M A P, Carpenter A E, et al. Visualization of parameter space for image analysis [J]. IEEE Transactions on Visualization and Computer Graphics, 2011, 17(12): 2402-2411.

[111] W. Berger, H. Piringer, P. Filzmoser, et al. Uncertainty-aware exploration of continuous parameter spaces using multivariate prediction [C]. In: Proceedings of the 13th Eurographics / IEEE - VGTC conference on Visualization, Bergen, Norway, 2011: 911-920.

[112] Walker R, Kenny P, Miao J. Balancing Resolution and Response in Computational Steering with Simulation Trails [C]. In: Proceedings of the 2009 International Conference on Modeling, Simulation and Visualization Methods. 2009: 167-172.

[113] Hudson R, Norris J, Reid L B, et al. Experiences using smaash to manage data-intensive simulations [C]. In: Proceedings of the 20th international symposium on High performance distributed computing, ACM, 2011: 205-216.

[114] Brodlie, K., Poon, A., Wright, H., GRASPARC-A problem solving environment integrating computation and visualization [C]. In: IEEE Conference on Visualization, 1993: 102-109.

[115] Li S. Comparison of refinement criteria for structured adaptive mesh refinement [J]. Journal of computational and applied mathematics, 2010, 233(12): 3139-3147.

[116] Turk M J, Smith B D, Oishi J S, et al. yt: A multi-code analysis toolkit for astrophysical simulation data [J]. The Astrophysical Journal Supplement Series, 2011, 192(1): 9.

[117] Hunter, J. D., Matplotlib: A 2D graphics environment [J]. Computing In Science & Engineering, 2007, 9(3): 90-95.

[118] M.Ament, S. Frey, F.Sadlo et al. GPU-based two-dimensional flow simulation steering using Coherent Structures [C]. In: proceedings of the 2nd International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering, Stirlingshire, UK, 2011.

[119] Ji L, Wang Q D, Kwan J. Non-equilibrium ionization model for stellar cluster winds and its application [J]. Monthly Notices of the Royal Astronomical Society, 2006, 372(2): 497-509.

[120] Dan R. Lipşa, Robert S. Laramee, Simon J. Cox, et al. Visualization for the Physical Sciences [J]. Computer Graphics Forum, 2012, 31(8): 2317-2347.

# 发表论文和科研情况说明

## 发表的主要论文：

[1] **Jian Xiao**, Xingyu Xu, Jizhou Sun, et al. A Pipeline for Accelerating Non-equilibrium Ionization on Eulerian Mesh via Tracer Particles and MapReduce [J]. Journal of Multiple-Valued Logic and Soft Computing, accepted, (SCI 四区，Impact factor: 0.667)

[2] **Jian Xiao**, Jiawan Zhang, Ye Yuan, et al. A Hierarchical Visual Analysis and Steering Framework for Astrophysical Simulations [J]. Transactions of Tianjin University, 2015, 21(6): 507-514. (EI Journal)

[3] **Jian Xiao**, Xingyu Xu, Ce Yu, et al. Accelerating Spectral Calculation through hybrid GPU-based computing [C]. In: International Conference on Parallel Processing (ICPP), 2015:41-50. (CCF B 类)

[4] **Jian Xiao**, Xingyu Xu, Jizhou Sun, et al. Acceleration of solving non-equilibrium ionization via tracer particles and mapreduce on eulerian mesh [C]. In: Algorithms and Architectures for Parallel Processing, 2014:29–4. (CCF C 类, **EI**: 20143518106631)

[5] **Jian Xiao**, Jizhou Sun, Gang Li, et al. MicroSSB: A lightweight framework for on-line distributed application based on soft system bus [C]. In: Proceedings of the 6th International Conference on Evaluation of Novel Approaches to Software Engineering, 2011: 211-217. (**EI**: 20123415363081)

[6] RunTao Wang, **Jian Xiao**, Jizhou Sun, et al. Application-Aware Storage Strategy for Scientific Data [J]. Communications in Computer and Information Science, 2013, 332:671-681 (**EI**: 20132616453513，通讯作者)

[7] Ce Yu, Runtao Wang, **Jian Xiao**, et al. High Performance Indexing for Massive Audio Fingerprint Data [J]. IEEE Transactions on Consumer Electronics, 2014, 60(4): 690-695. (SCI 四区，Impact factor: 1.157)

## 参与的科研项目：

[1] 无缝式天文数据访问关键技术研究（U1231108），国家自然科学基金—天文联合基金项目，主要参加人

# 致 谢

本论文得以最终完成，首先要感谢我的导师孙济洲教授，从硕士起就承蒙老师的教诲，导师严谨认真的科研态度、一丝不苟的工作作风、和蔼可亲的处事风格、以及对弟子无微不至的爱护，均使我受益终生。其次感谢我的大师兄张加万教授，师兄严格的鞭策、以及在我迷茫时的激励和指导，最终促使我克服了读博期间的懒惰和恐惧，找到了做科研的感觉。还要特别感谢紫金山天文台的纪丽研究员，是纪老师将我领入了天文模拟的世界，又一直为我答疑解惑，帮我渡过科研过程中的一个个难关。同时感谢埼玉大学程京德教授在科研方法和论文写作上的热心指导。

尤其要感谢我的本科同窗兼同事于策博士，如果没有他的鼓励和帮助，本文的研究成果将很难发表，同时也感谢于策老师在工作上对我的支持和信任，读博五年，也是我们课题组成长最快的五年，这是大家共同努力的结果。感谢国家天文台信息中心的老师们，感谢紫金山天文台的周鑫博士、张水乃博士、雷士俊博士，以及芝加哥大学 FLASH 中心的龙旻博士，他们不厌其烦地回答我那些浅显无聊的问题，教会了我天文模拟的基础知识和技巧。感谢学院领导对我读博的支持和理解；感谢廖士忠教授在计算理论方面提供的无私指导；感谢李罡老师替我分担了服务器维护的压力；感谢胡静老师对我的开导以及在小论文写作上的帮助；感谢软件学院实验室的同事们对我工作的支持。感谢于瑞国老师、张亚平老师、吕良福老师、李春老师、张怡老师、许光全老师、饶国政老师、张坤龙老师、赵来平老师对我的热心帮助。

非常感谢国家天文台—天津大学天文信息技术联合研究中心的同学们，特别感谢徐星宇同学，本文的大部分成果都是在她的鼎力协助下完成的，所有实验的背后都有她辛勤的汗水；特别感谢原野同学在可视化部分提供的帮助以及给我的鼓励。感谢洪智、尹树成、罗壹文、李晨、裘实等同学对天文领域云项目做出的贡献，使得我能抽出更多的精力来完成博士学业。还要感谢汤善江博士对本论文的组织结构提出的宝贵建议。

读博五年，冷暖自知，唯一不变的是家人对我的理解包容以及自始至终的默默奉献。感谢岳父、岳母、父亲、母亲在我读博期间为我所做的一切，感谢爱人和儿子对我忙于学业忽视家庭的理解，她们的鼓励和支持，是我前进的最大动力。感谢小妹对我的关心和嘘寒问暖，感谢舅舅在我上学期间对我的照顾和教诲。感谢陈武星老师教给我为人处事的道理，学生定会牢记在心，身体力行。最后，感谢所有对本文有过帮助的人。