中国科学技术大学

硕士学位论文

基于 GPU 的等离子体快速平衡重建研究

作者姓名：岳小宁

学科专业：核能科学与工程

导师姓名：肖炳甲 研究员

完成时间：二〇一三年五月七日

University of Science and Technology of China

A dissertation for master's degree

Fast Equilibrium Reconstruction based on GPU

Author's Name： Xiaoning Yue

speciality： Nuclear Engineering

Supervisor： Prof. Bingjia Xiao

Finished time: May 7th, 2013

中国科学技术大学学位论文原创性声明

本人声明所呈交的学位论文,是本人在导师指导下进行研究工作所取得的成果。除已特别加以标注和致谢的地方外，论文中不包含任何他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的贡献均已在论文中作了明确的说明。

作者签名：_____ 签字日期：_____

中国科学技术大学学位论文授权使用声明

作为申请学位的条件之一，学位论文著作权拥有者授权中国科学技术大学拥有学位论文的部分使用权，即：学校有权按有关规定向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅，可以将学位论文编入《中国学位论文全文数据库》等有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。本人提交的电子文档的内容和纸质论文的内容相一致。

保密的学位论文在解密后也遵守此规定。

□公开 □保密（____年）

作者签名：_____ 导师签名：_____

签字日期：_____ 签字日期：_____

摘 要

实时平衡重建是托卡马克位形控制的基础，其精度对最终的位形控制质量有着重要影响。然而，平衡重建算法是一个十分耗时的算法，特别是在要求精度较高的时候，因此，目前 EAST 上所采用的 RT-EFIT 采用的是较为稀疏的 33×33 网格。本文提出了一个基于 GPU 并行技术的平衡重建算法 P-EFIT，可以在 220μs的时间内完成一次 65×65 网格的反演迭代，满足高精度实时平衡重建的要求。

本文着重描述了 P-EFIT 中对平衡重建算法的并行化处理，主要包括块三对角方程以及三对角方程求解的并行加速，一系列矩阵相乘的并行加速，以及边界磁通计算方法的并行加速。这些并行算法均针对 GPU 的硬件特征进行了专门优化，获得了明显的加速比。

文章中分析了磁测量信号中的随机误差对反演质量的影响，提出了减少这些影响的数值处理方法。

本文最后描述了验证 P-EFIT 正确性的一系列实验测试，实验结果，在磁测量信号的误差小于 3%的情况下，P-EFIT 重建结果具有足够的精度，利用 TSC 演化得来的精确放电数据验证了一次迭代策略的可行性，并且进行了与 PCS 系统整合的实验环境下的模拟测试，这些测试验证了 P-EFIT 应用于实时位形控制的可行性。

关键词：托卡马克， 平衡重建，P-EFIT，统一计算架构, CUDA

ABSTRACT

Real-time reconstruction is very important for plasma shape control in tokamaks. However, as the reconstruction algorithm is computational intensive, it is very difficult to improve its accuracy while keeping real-time property. This article described a parallel real-time code named P-EFIT.

P-EFIT could complete an equilibrium reconstruction iteration in 220μs. It is build with the CUDA™ architecture. Some optimization for middle-scale matrix multiplication on GPU and an algorithm which could solve block tri-diagonal linear system efficiently in parallel is described.

The implementation of P-EFIT into the PCS is described. Several benchmark test has been conducted. Static test proves the correctness of the P-EFIT and simulation-test shows the advantage of P-EFIT compared to RT-EFIT when used for discharge control.

Key Words: tokamak, equilibrium reconstruction, P-EFIT, CUDA

目录

第 1 章 绪论

1.1 托卡马克简介

1.1.1 受控核聚变的基本原理

原子核具有不同的结合能，直观上来讲，结合能就是原子核中的核子结合的紧密程度，结合能越大的核子越稳定。结合能的大小与原子核的质量数有关，中等质量的核具有最高的结合能，在质量数小于 30 的情况下，结合能呈现出周期震荡的特点，质量数为 4 的倍数的核子具有最大结合能。当两个结合能较小的核子结合为一个结合能较大的核子时，便会释放出巨大的能量[1]。目前人类最有希望实现的聚变为氘氚聚变，如式 1.1 所示[2]。

$$^{2}\mathrm{D} + {}^{3}\mathrm{T} = {}^{4}\mathrm{He}\ (3.5\mathrm{MeV}) + {}^{1}\mathrm{n}\ (14.1\mathrm{MeV}) \tag{1.1}$$

人们从最初认识核裂变到利用可控核裂变只用了不到十年的时间。然而尽管聚变的发现早于裂变，可控核聚变知道今日依然没能实现真正的实用化。其原因主要是因为原子核之间巨大的库仑势垒，这一势垒的存在使得聚变在常温甚至是人类通常认知下的高温情况下反应截面都微乎其微。于是，要实现聚变必须从两个方向努力，要么创造极端的高温，使得核子的动能大到足以越过库伦势垒，要么创造极端的高密度环境，使得单位体积单位时间内可以发生大量的聚变。同时，不管采取哪种方式，都必须将如此高的能量约束足够多的时间来产生足够的聚变反应。

类似于自持裂变，聚变也存在自持燃烧状态，式 2.1 中可以看出，聚变产生的能量中有大约 4/5 由高能中子携带，由于中子的强穿透力，这些能量几乎不会沉积在等离子体中，但是另外 1/5 由α 粒子携带的能量则可以大部分沉积在等离子体中，于是可以推断出，必然存在这样一种状态，即等离子体损失能量的功率小于自身聚变产生的α 里子的加热功率，此时等离子体可以在没有外界加热的情况下持续燃烧下去。氘氚等离子体要达到这一状态，其物理状态式 1.2 所示的称作劳逊判据的关系式。由于反应截面$\langle\sigma\upsilon\rangle$在 10~20KeV 范围内近似与$T^2$成正比，1.2 式可以转化为$n\tau_E T$大于某个值，即要达到点火条件，需要等离子体密度、约束时间、温度三者的乘积大于一定值。这个乘积被称做聚变三重积[3]。

$$n\tau_E > \frac{12}{\langle \sigma v \rangle} \frac{T}{\varepsilon_\alpha} \tag{1.2}$$

一种可能满足劳逊判据的约束形式是引力约束，只要天体的质量足够大，其核心处高压最终会在天体中心实现点火并向周围扩散，同时天体巨大的引力又使得聚变材料不会飞散，从而实现持续的燃烧。然而这种形式的聚变在地球上是不可能出现的。

另外一种可能的实现途径被称作惯性约束，这是通过利用高能激光、离子束或者间接产生的各向均匀 X 射线在同一时间加热靶丸，使其表面物质快速向外飞散，通过向内的反冲力是靶丸核心的聚变材料达到极端高温高压状态，在其靠其惯性维持的短暂时间内发生足够的聚变实现点火。这与氢弹中实现点火的原理一致[4]。

目前，研究开展最多的可控核聚变研究是磁约束聚变，这种聚变是利用高温等离子体带电这一特性，利用磁场产生的洛伦兹力将等离子体约束在一定范围内，并对其进行加热，知道其达到点火条件。磁约束聚变的装置种类很多，例如磁镜，托卡马克，仿星器等。其中研究最多的是托卡马克装置，这种装置通过外部线圈产生纵场，同时利用中心螺线管感应出很大的等离子体电流，等离子体电流产生的磁场与纵场耦合，形成螺旋线形的磁场，实现对高温等离子体的约束。

1.1.2 托卡马克的研究现状

如上一节所述，受控核聚变有很多种实现途径，但是目前被认为最有可能达到实用目标的是托卡马克方案。对托卡马克的研究始于上世纪五十年代，然而直到上世纪六十年代末苏联宣布自己在 T3 托卡马克上获得了 1keV（大约 10^7K）的电子温度之后，这一方案才开始得到各国的重视，冷战双方都开始投入大量的人力物力对托卡马克进行深入研究。由于鼓舞人心的实验结果不断出现，上世纪 70 年代末开始，当时世界上的四个主要经济体开始建造四个大型的托克马克，即欧洲联合建造的 JET，美国的 TFTR，日本的 JT-60 以及苏联的 T-20，在这些装置上获得了 Q 接近 1 或者大于 1 的实验结果[2]。但是大家也开始意识到托卡马克研究所面临的巨大挑战，进一步进行研究的投入和风险是当时各国都无法单独承受的，因此从上世纪八十年代开始，国际热核聚变实验堆（ITER）计划被提出，但由于种种原因，直到 2007 年，ITER 国际组织才正式成立，国际热核实验聚变堆进入正式设计建造阶段。

我国于 1974 年建成了第一台托卡马克装置 CT-6。1995 年，中科院等离子

体物理研究所建成了 HT-7 装置，2008 年在该装置上实现了长达 400 秒的等离子体放电。2006 年等离子体物理研究所建成了世界上第一个全超导托卡马克 HT-7U（又称 EAST 装置，东方超环，其主要参数如表 1.1 所示），该装置将为 ITER 的设计和运行提供重要参考[5]。在 EAST 的 2012 年春季实验中，获得了 400 秒长脉冲高参数偏滤器等离子体，以及 30 秒的高约束模放电。

图 1.1 中直观的展示了世界上主要托卡马克能够实现的三重积,可以看出，当前的托克马克研究已经进入了关键的阶段，下一步将由 ITER 来实现点火这一目标。

表 1.1 EAST 装置的主要设计参数

纵场	3.5T@1.75m
等离子体电流	1MA
大半径 R	1.75～ 1.98m
小半径 a	0.4 ～ 0.5m
拉长比 k	1.6 ～ 2.0
三角形变 d	0.6 ～ 0.8
放电长度	1 ～ 1000s
放电位形	双零/单零偏滤器位形

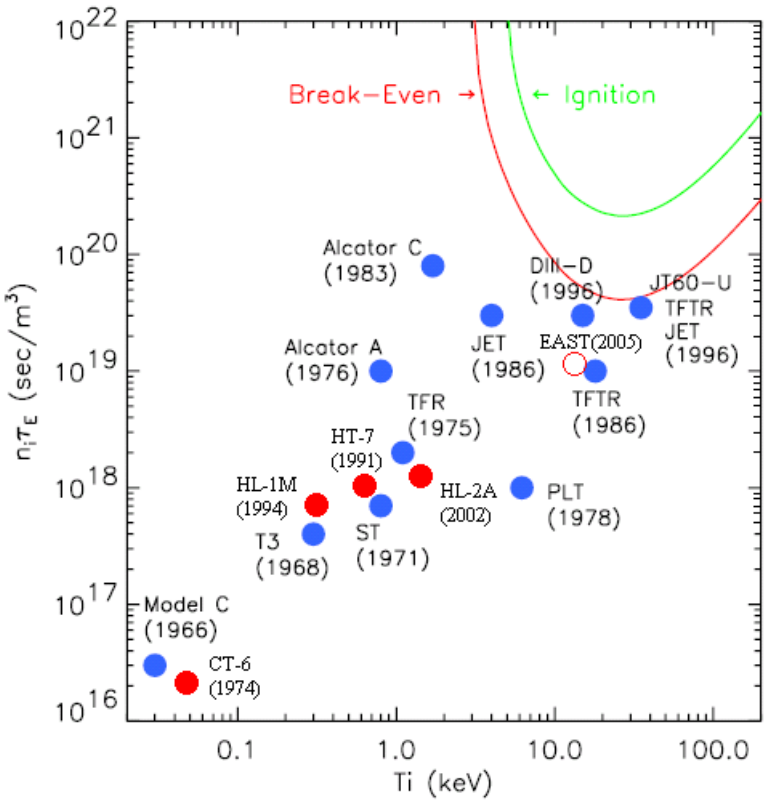

图 1.1 当前世界主要装置的三重积示意图[5]

1.2 等离子体平衡重建简介

托卡马克中等离子体的位形对于放电质量有着重要的影响，选取合适的等离子体位形有利于维持更高的等离子体比压，以及减少一些不稳定性的影响。但是在托卡马克实验中，等离子体的位形是无法直接测量获得的，必须通过被称作等离子平衡重建的过程来间接获得[6]。

平衡重建的算法有很多[7]，其中的一类算法避开电流的内部分布信息，只在真空区域根据 Grad-Shafranov 方程拟合外部磁测量来获得等离子体边界信息，采用这类算法的反演程序包括多级展开[8]以及 XLOC[9]，这种方法的优势是可以快速的获得等离子体边界信息，可以直接被用作实时位形控制，但是缺点也很明显。由于这类算法的实质是以诊断测量点处的磁通信息外推至边界处的，因此，必须要求诊断测量在真空室周围密集分布且非常靠近等离子体边界[10]。随着装置尺寸的逐渐增大，这类算法的应用愈加局限。

另外一种重建算法是考虑了等离子体内部的电流分布的，在算法中定义一种电流分布模型，然后将与外部磁测量进行拟合确定出电流模型中的未知量，从而得到完整的等离子体分布信息。最初使用的电流模型为电流丝模型[11]，然

而这种模型无法模拟等离子体拉长的情况。之后，J. L. Luxon，F.W. McClain 等人提出了 GAQ 电流模型，这种模型可以准确的模拟拉长位形下的等离子体电流分布[12]，但是，由于这一电流模型是非线性的，在计算中需要多次尝试来寻找最佳的模型参数，因此这一算法的时间复杂度非常高。为了满足大量实验数据分析的需求，L.L. LAO 等人发展出了基于 Piacard 迭代和多项式电流模型的反演算法[13, 14]。该算法用线性多项式模型来描述等离子体电流的分布，通过几次迭代便可以得到一个准确的电流分布，基于该方法发展出的 EFIT 程序目前被广泛用于实验数据的分析中。

虽然 EFIT 可以较快速的实现对等离子体电流分布的准确反演，但是其速度依然不足以被用于实时位形控制之中。为了解决这一问题，J.R. Ferron 对 EFIT 算法进行了一些修改，开发出了 RT-EFIT 程序[15] 。RT-EFIT 采用的电流模型与 EFIT 相同，但是 RT-EFIT 没有利用 Picard 迭代来寻找收敛的反演结果，而只进行一到两次迭代，然后就将结果输出，同时这次输出的结果会被作为下次反演的初始平衡输入。当格点较为稀疏（≤33×33）时，RT-EFIT 可以满足实时控制的需求。目前，RT-EFIT 在 DIII-D[15], KSTAR[16], NSTX[17], MAST[18] and EAST[19] 等众多装置上被用作为实时位形控制。

为了提升实时位形控制的精度，必须提高实时反演程序的精度，提升精度的一个最直接的办法是提高离散格点的密度。然而，格点数量的提升必然会导致时间复杂度的大幅度上升，从而失去实时性。为了在提升复杂度的同时保持足够的速度，便必须采用并行计算的方法。对平衡反演程序进行并行化的尝试很早便已开始，然而，由于平衡反演的过程的复杂性，计算过程中很难解耦出可以完全并发运行的独立部分，在计算过程中必然存在着大量的通信，因此，并行重建程序的开发难度很大[10, 20]。2012 年，德国的 M. RAMPP 等人利用 MPI 加速了反演程序中的 G-S 方程离散求解部分，获得了不错的效果[21]。但是采用这一架构的并行程序目前只完成了类泊松方程求解部分的并行，而这一部分只占总计算量的三分之一左右，而进一步的增加并行单元规模已经无法获得性能上的提升，因此采用这种方式的并行反演程序能够获得的总加速比是不够理想的。

通过对反演算法的分析，我们发现要获得理想的加速效果，采用的并行架构必须有足够的并行执行单元来完成反演中大量的浮点运算，同时又必须能够快速的实现线程之间的通信和同步。而新出现的通用 GPU 并行技术（GPGPU）是满足这一条件的。本文中将描述一种新的基于 GPGPU 技术的并行平衡重建程序 P-EFIT。

1.3 GPU 并行计算简介

GPU（Graphic Processing Unit）是图形处理器的简称，世界上的第一块 GPU 由英伟达公司（NVIDIA）在 1999 年推出，当时 GPU 的主要任务是加速图形渲染应用，例如高清视频，游戏等。这些任务的共同特点是可以高度并行，而且对处理器的吞吐（throughput）要求很高。于是。GPU 的硬件架构（如图 1.2）中分配了大多数的晶体管来执行浮点运算，而不是像 CPU 那样将大部分硬件资源用作逻辑控制。同时，GPU 具有很高的显存位宽，从而保证 GPU 能够完成高吞吐的任务。

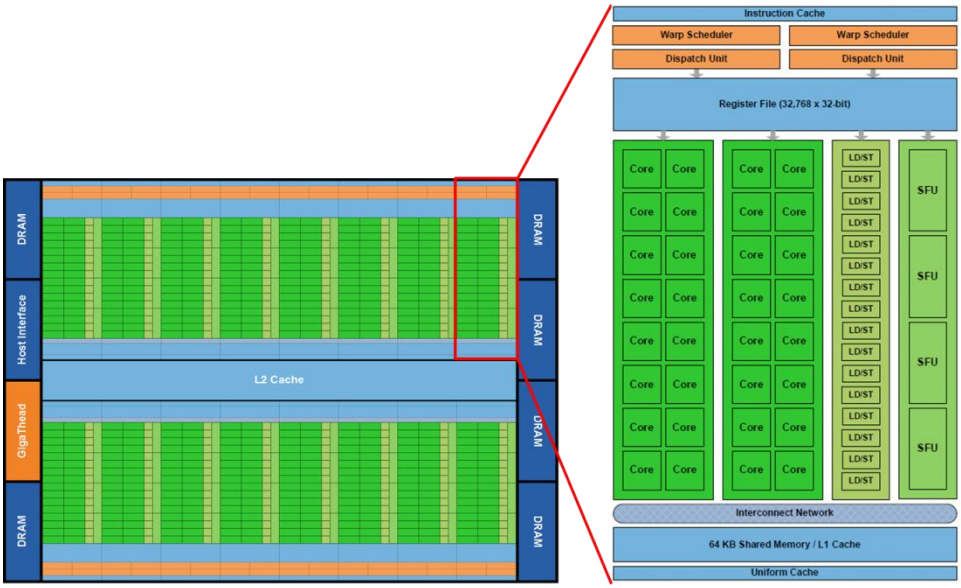

图 1.2 Fermi 硬件架构示意图

GPU 的这种硬件架构使得其理论浮点运算能力以及显存带宽远大于 CPU（如图 1.3 所示），为了充分利用显卡中的计算资源，在显卡出现后不久，GPGPU 的概念便被提出，然而，当时编写利用 GPU 进行通用计算的程序是很困难的，这一定程度上阻碍了 GPGPU 的推广。

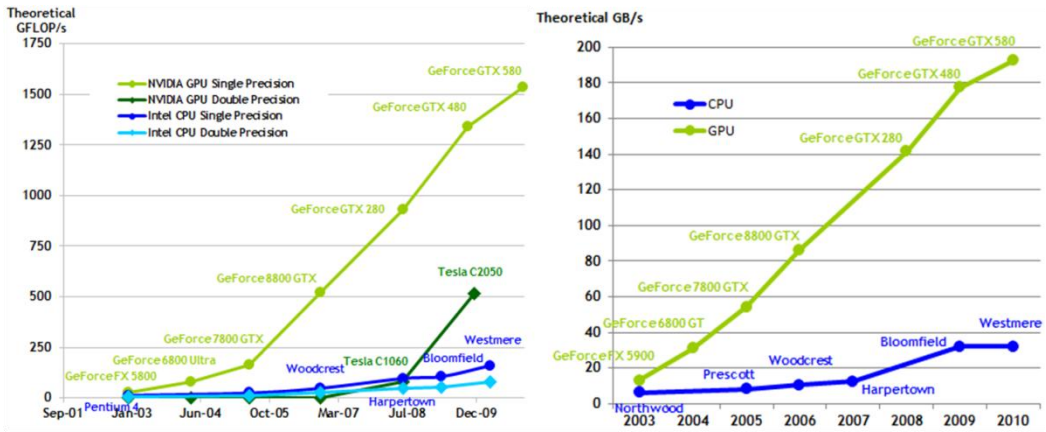

图 1.3 CPU 和 GPU 的浮点运算能力与带宽对比[22]

随着 PCIe 总线带宽的提高，CPU 现在可以相对较快的将计算任务传入 GPU 的内存中。于是如果 CPU 将那些计算量很大而且适合数据并行的任务传入 GPU 中，而 CPU 则专门处理串行的任务以及控制 GPU 的运行，则整个应用的执行效率可以大幅度的提升。当然，这里要求 GPU 需求的输入数据和输出数据相对于其运算量越小越好。

由于 GPGPU 具有广泛的应用前景，NVIDIA 公司终于在 2006 年推出了专门针对 GPGPU 的统一计算架构（CUDA），利用 CUDA，开发人员可以使用熟悉的高级语言，如 C、C++等编写在 NVIDIA GPU 上运行的程序。这使得在 GPU 上编写复杂数值计算程序的难度大大降低。同时，NVIDIA 也针对 GPGPU 应用对 GPU 的硬件做了很多专门的优化，例如对双精度浮点运算的支持等等。在最新的采用 Kepler 架构的 GK110 芯片中，更是添加了动态并行，Hyper-Q 等多种新特性，使得其计算能力进一步得到提升。

然而，编写 GPU 应用程序的难度依然远远大于普通的应用程序，这是因为在编写 GPU 应用程序是必须结合 GPU 的硬件架构来优化自己的代码，否则不仅得不到速度的提升，反而会降低其性能。下面将简要的介绍几点在利用 CUDA 编写 GPU 应用程序时最需要注意的几点事项，这些优化事项主要是针对计算能力 2.0 的 Fermi 架构来讲的。

1、仔细的划分原问题，要按照 GPU 硬件架构的特点，尽可能细粒度的划分原应用程序，对于只能粗粒度并行的应用程序而言，最好将其分配至不同的 block 上运行[23]。

2、对 Global memory 访问的优化。GPU 访问显存的延迟大，带宽相对较小，因此应该尽量减少访问显存的次数。GPU 中对 global memory 的读写是以 warp 为单位的（计算能力 2.0 以前的设备以 half-warp 为单位），如图 1.4 所示，计算能力 2.0 的设备可以一次读写 128Byte 的数据，因此在设计算法时要注意让一个 warp 内的显存读写连续的显存区域，这样可以明显的提高程序的效率。

3、注意其他影响较小的优化细节，例如 shared memory 的 bank conflict，global memory 的分区冲突。寄存器溢出等等。

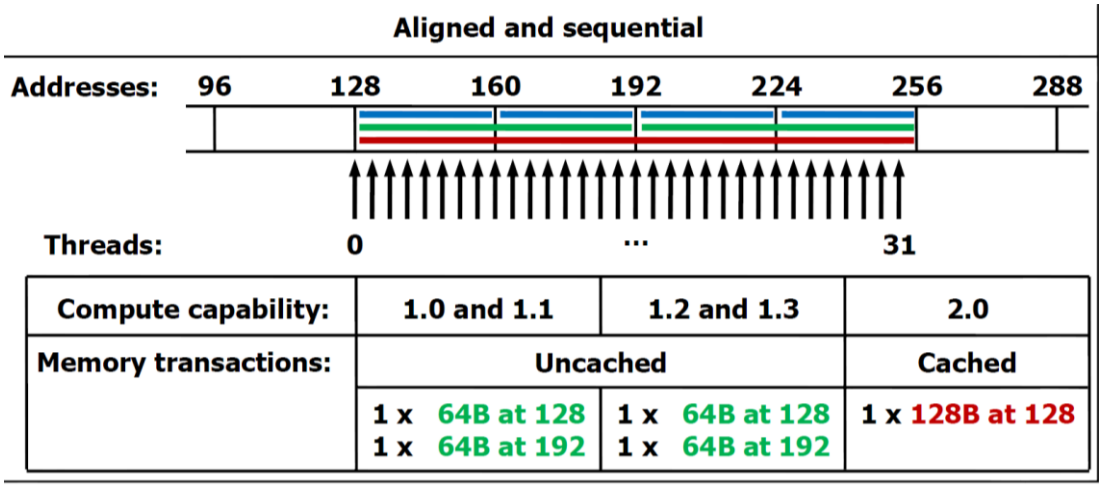

图 1.4 GPU 访存模式示意图[22]

1.4 论文结构

本文共分为 5 章。

第 1 章为绪论，介绍了受控核聚变以及托卡马克的一些基本知识，引入了等离子体平衡重建的概念并对其研究现状进行了介绍。之后介绍了 GPU 并行计算的基本情况。

第 2 章介绍了平衡算法重建的基本物理原理以及数值计算方法。

第 3 章介绍了 P-EFIT 中对平衡重建算法的并行化改造。详细介绍了 P-EFIT 中的一些算法实现细节。

第 4 章主要介绍了对 P-EFIT 进行的一系列测试，包括验证其准确性的静态测试，验证其在位形控制中采用的一次迭代策略的正确性以及在实验环境下的仿真测试。

第 5 章为总结与展望。

第 2 章 等离子体平衡重建算法

2.1 平衡重建的物理原理

具有轴对称结构的柱形等离子体在平衡之后磁力线会形成如图 2.1 所示的磁面结构[3]。

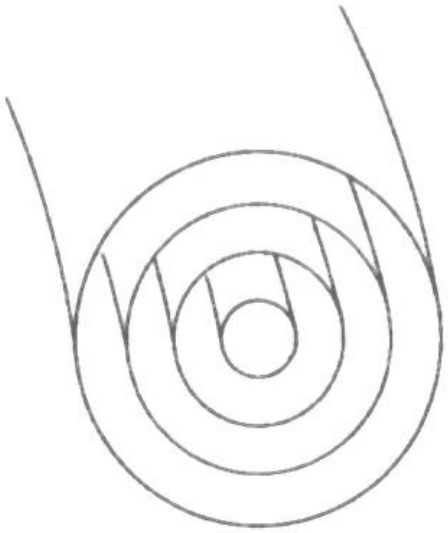

图 2.1 磁面示意图

在等离子体达到平衡时，等离子体每一点处所受的合力应该为 0，即等离子体所受的电磁力应当恰好抵消等离子体自身的热压力。由此可以得出：

$$j \times B = \nabla p \tag{2.1}$$

由于 $B \cdot (j \times B) = 0$，所以 $B \cdot \nabla p = 0$，由此可以得出磁面是等压力面。同理可以证明出电流 j 也是附着在磁面上的。为了研究方便，定义极向磁通函数 ψ，图 2.1 所示的磁面也是等 ψ 面，因此可以将压强 p 表达为 ψ 的函数 $p(\psi)$。同时可以得到极向磁通与 ψ 的如下表达式：

$$B_R = -\frac{1}{R}\frac{\partial \psi}{\partial z},\quad B_Z = \frac{1}{R}\frac{\partial \psi}{\partial R} \tag{2.2}$$

由 B 与 j 的对应关系可以推断必然存在电流通量函数 F，满足如下关系：

$$j_R = -\frac{1}{R}\frac{\partial F}{\partial z},\quad j_Z = \frac{1}{R}\frac{\partial F}{\partial R} \tag{2.3}$$

将 2.3 式与安培定律比较可以得出：

$$F = \frac{RB_\phi}{\mu_0} \tag{2.4}$$

可以证得 F 同样是 ψ 的函数，可以表达为 $F(\psi)$。

将式 2.1 展开，可以得到：

$$j_p \times i_\phi B_\phi + j_\phi i_\phi \times B_P = \nabla p \tag{2.5}$$

通过 2.2 和 2.3 式可以将极向磁通与极向电流函数表达为

$$B_P = \frac{1}{R}(\nabla \psi \times i_\phi) \tag{2.6}$$

和

$$j_P = \frac{1}{R}(\nabla F \times i_\phi) \tag{2.7}$$

将 2.6 和 2.7 代入 2.5 中，可以得到：

$$-\frac{B_\phi}{R}\nabla F + \frac{j_\phi}{R}\nabla \psi = \nabla p \tag{2.8}$$

由于 $\nabla F = \frac{dF}{d\psi}\nabla \psi$， $\nabla P = \frac{dP}{d\psi}\nabla \psi$，代入 2.8 式中可以得到：

$$j_\phi = Rp' + \frac{\mu_0}{R}FF' \tag{2.9}$$

由安培定律 $\mu_0 j_\phi = \nabla \times B$，并将磁场展开为 ψ 的函数，可以得到

$$R\frac{\partial}{\partial R}\frac{1}{R}\frac{\partial \psi}{\partial R} + \frac{\partial^2 \psi}{\partial z^2} = -\mu_0 R j_\phi \tag{2.10}$$

$$R\frac{\partial}{\partial R}\frac{1}{R}\frac{\partial \psi}{\partial R} + \frac{\partial^2 \psi}{\partial z^2} = -\mu_0 R(Rp'(\psi) - \frac{\mu_0}{R}F(\psi)F'(\psi)) \tag{2.11}$$

式 2.10 中的非线性偏微分方程被称作 Grad-Shafranov 方程（G-S 方程），这一方程描述了轴对称柱形等离子体中的平衡。托卡马克中等离子体重建的基本物理原理即为寻找满足式 2.10 的 $p'(\psi)$ 和 $F(\psi)F'(\psi)$ 来得到等离子体的内部信息。

2.2 等离子体平衡重建的数值算法

2.2.1 计算区域网格剖分

为了对式 2.10 进行数值求解，首先要对计算区域进行离散化，目前在 EAST 平衡重建中使用的网格剖分方式一般为 $(2^n+1)\times(2^n+1)$，其中 n 一般选取 5 到 7 的整数，这样选取的主要原因是增加求解块三对角方程的效率。划分的范围为 R 方向 1.2m~2.6m，Z 方向-1.2m~1.2m。划分后的网格点分布如图 2.2 所示。

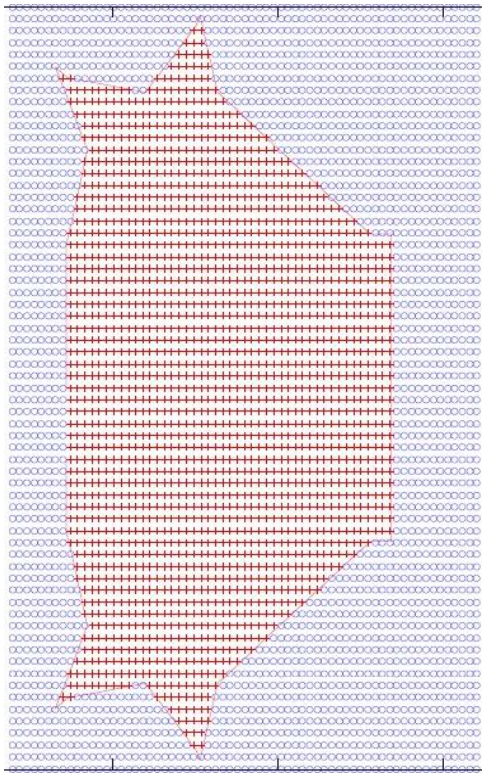

图 2.2 等离子体网格剖分示意图

2.2.2 等离子体电流的数值表示形式

由于式 2.10 没有直接的解析解，因此只能通过数值方法来近似求解。由于 $p(\psi)$ 和 $F(\psi)$ 的形式未定，进行完全重建必须要确定合适的电流函数模型，J.L. Luxon 和 B.B. Brown 提出了一种指数电流表达式，然而利用这种表达式进行平衡重建时必须反复多次的求解平衡来确定电流模型中的未知项，计算量过大。为了满足大量放电数据的处理要求，L.L. Lao 等人提出了利用 Picard 迭代来求解 G-S 方程获得平衡的方法，同时为了减少计算量，使用线性的电流模型取代了原来的非线性电流模型。EFIT 便是在这一算法的基础上发展而来的。

EFIT 所使用的多项式电流模型如下所示：

$$\begin{aligned} p'(\psi) &= \sum_{n=0}^{n_P} \alpha_n \psi_N^n \\ FF'(\psi) &= \sum_{n=0}^{n_F} \gamma_n \psi_N^n \end{aligned} \tag{2.12}$$

其中的 ψ_N 代表的是归一化磁通，其定义为：

$$\psi_{\mathrm{N}} = (\psi - \psi_{axis}) / (\psi_{bdy} - \psi_{axis}) \tag{2.13}$$

其中 ψ_{axis} 表示的是磁轴处的磁通，而 ψ_{bdy} 表示的则是边界处的磁通。

一般而言，对于 DIII-D 或者 EAST 之类的非圆截面托卡马克，如果只采用外部磁测量信号作为反演依据，多项式的参数选取应该遵循 $n_P + n_F$=3 或者 4。在 EAST 实验中，采用 $n_P = 2,\ n_F = 1$ 可以得到很好的反演效果。采用了这一表达形式之后的等离子体电流可以用下式表示：

$$j_\phi^0 = \alpha_0 R + \alpha_1 R\psi_N + \gamma_0 \frac{1}{R} \tag{2.14}$$

在实际求解中，为了能够给予电流模型在垂直方向调整的自由度，往往要增加一个额外的多项式参数 δ_Z。添加这一约束后的等离子体电流表达式为

$$j_\phi = \alpha_0 R + \alpha_1 R\psi_N + \gamma_0 \frac{1}{R} + \delta_Z R \frac{\delta\psi_N}{\delta Z} \tag{2.15}$$

δ_Z 项事实上不属于等离子体电流的多项式模型模型，在理想情况下，δ_Z 应该等于 0，然而在实际计算中，δ_Z 项的引入可以减少测量、截断等误差带来的影响，改善反演的结果，在正常情况下，反演出的 δ_Z 应该是一个很小的值。

在数值求解时，ψ_N、R、j_ϕ 等均是以向量形式存储的格点值存在的，因此可以将式 2.15 表达为如下所示的矩阵向量形式：

$$\mathbf{Ip}=\mathbf{H}\times\boldsymbol{\alpha} = (R, R\psi_N, \frac{1}{R}, R\frac{\delta\psi_N}{\delta Z})\times\boldsymbol{\alpha} \tag{2.16}$$

其中，$\boldsymbol{\alpha} = \begin{pmatrix}\alpha_0 & \alpha_1 & \gamma_0 & \delta_Z\end{pmatrix}^T$ 是未知多项式系数组成的向量。而 **H** 则是一个行数为电流格点数目，列数为未知多项式系数数量的矩阵。如果采用的是 65×65 的网格，则 **H** 的大小为（4225×4），由于这一矩阵的大小对整个反演程序的速度有着较大影响，而这 4225 个格点中，位于限制器以外的格点上的等离子体电流是始终为零的。因此，在实际求解中，这一步只需考虑位于限制器以内的格点（图 2.2 中的红色格点）。通过这一过滤，图 2.2 中的情况下 **H** 的规模可以被减小至（1956×4）。

2.2.3 响应矩阵的求解

等离子体的信息隐含在式 2.15 中的多项式参数中，为了通过外部诊断求得这几个未知的参数，首先必须建立未知多项式系数与外部诊断之间的关系，这一关系矩阵被称作响应矩阵。

计算区域的网格剖分完成以后，各个格点的位置都是预先可知的，而目前 EAST 用于反演的探针和单匝环位置也是已知的，因此通过毕奥-萨伐尔定律可以预先求出每一个网格上的电流在每一处磁测量点处所贡献的磁场或者磁通。这些数据组成的矩阵被称作格林函数矩阵。

等离子体电流与外部磁测量之间的关系可以用格林函数矩阵 $\mathbf{G_P}$ 来表示，设

等离子体电流格点数量为k_p，磁测量信号数量为k_d，则矩阵$\mathbf{G_P}$的大小为（$k_d \times k_p$），等离子体电流在磁测量点上的贡献值可以表示为$\mathbf{G_P} \times \mathbf{Ip} = \mathbf{G_P} \times \mathbf{H} \times \boldsymbol{\alpha}$，同时，磁测量点上的测量值还包括外部电流源的贡献。外部电流源主要包括极向场线圈（PF）电流和周围导体结构上的感应电流，对于 EAST 而言，感应电流的贡献是比较小的，因此目前只考虑 PF 电流的贡献。设 PF 线圈数量为k_F，由于 PF 线圈的位置是预先可知的，则 PF 电流大小与磁测量之间的关系可以用一个预先计算的大小为（$k_d \times k_F$）的格林函数矩阵$\mathbf{G_C}$表示。用$\mathbf{I_C}$来表示 PF 电流向量，则 PF 电流对磁测量值的贡献可以表示为$\mathbf{G_C} \times \mathbf{I_C}$。在 EAST 实验中，虽然可以获得 PF 电流的诊断值，但一般还是会将 PF 电流值作为反演的未知量，综上所述，磁测量值可以用下式表示：

$$\mathbf{D}=\left(\mathbf{G_C} \quad \mathbf{G_P} \times \mathbf{H}\right) \times \left(\mathbf{I_C} \quad \boldsymbol{\alpha}\right)^{\mathrm{T}} = \boldsymbol{\Gamma} \times \mathbf{U} \tag{2.17}$$

其中的$\boldsymbol{\Gamma} = \left(\mathbf{G_C} \quad \mathbf{G_P} \times \mathbf{H}\right)$被称作响应矩阵，$\mathbf{U} = \left(\mathbf{I_C} \quad \boldsymbol{\alpha}\right)^{\mathrm{T}}$是要反演的未知量所构成的向量。

2.2.4 未知量的求解

为了增加反演结果的准确度，磁测量的数量往往会远大于**U**中未知量的数量，因此式 2.17 属于超定方程组。可以应用最小二乘的方法来进行求解。

为了磁测量信号误差对最小二乘结果的影响，在进行最小二乘前会为式 2.17 的每一行乘以一个权重。最佳的权重值应该是磁测量方差的倒数，然而，磁测量的方差不是一个守恒量，因此目前只能用近似的方式来计算这一权重，EAST 上所采用的策略是每个测量量对应的权重为$\frac{1}{\sqrt{(\sigma_r \times d)^2 + {\sigma_d}^2}}$，其中的 d 为测量值，σ_r取磁测量的最大相对随机误差，一般取 5%，σ_d是整个磁测量系统的系统误差，这一误差属于系统的固有误差，目前主要考虑的是采集系统的量化误差。定义**F**为权重向量，其中的第 i 个元素为第 i 个诊断值所对应的权重，考虑权重后的式 2.17 可以转化为：

$$\mathbf{F} \cdot \mathbf{D} = (\mathbf{F} \cdot \boldsymbol{\Gamma}) \times \mathbf{U} \tag{2.18}$$

这里“·”运算符的含义是将矩阵的每一行都乘以向量中对应的元素。式 2.18 可以采用 SVD 分解、QR 分解或者正则式方法来进行求解。

2.2.5 计算格点上磁通值的更新

将式 2.10 写成式 2.19 所示的 Picard 迭代形式，根据前面的描写可以得知，在求得多项式位置参数之后，将其带入 2.15 式即可求得每个格点上的等离子体电流。相当于求得了式 2.19 中的 $j_\phi(\psi^{(n)})$，为了进行 Picard 迭代，必须求出 $\psi^{(n+1)}$。

$$R\frac{\partial}{\partial R}\frac{1}{R}\frac{\partial \psi^{(n+1)}}{\partial R}+\frac{\partial^2 \psi^{(n+1)}}{\partial z^2}=-\mu_0 R j_\phi(\psi^{(n)}) \tag{2.19}$$

一种直接的方法是采用类似 2.2.3 中响应矩阵求解时的方法，预先计算出每一个电流格点与其他格点之间的互感系数，存储为一个格林函数矩阵 $\mathbf{K_p}$，然后各个格点上的磁通可以用下式直接求出

$$\mathbf{\Psi_P}=\mathbf{K_p}\times\mathbf{I_p} \tag{2.20}$$

然而，式 2.20 中的计算是非常耗时的，假设网格划分为（$m\times m$），则矩阵 $\mathbf{K_p}$ 的大小为（$m\times m$）×（$m\times m$），排除限制器以外格点电流之后的大小为（$m\times m$）× k_p。计算式 2.20 的复杂度为 O(m^4)。为了提高反演程序的运行效率，一般不会采用这种直接求解的方式。

式 2.19 中左侧是一个椭圆微分算子，将其进行如 2.21 式所示有限差分离散形式：

$$\frac{\psi_{i-1,j}-2\psi_{i,j}+\psi_{i+1,j}}{(\Delta R)^2}+\frac{1}{R_i}\frac{\psi_{i-1,j}-\psi_{i+1,j}}{2\Delta R}+\frac{\psi_{i,j-1}-2\psi_{i,j}+\psi_{i,j+1}}{(\Delta Z)^2}=-\mu_0 R_i j_{i,j} \tag{2.21}$$

进一步将其整理为如下形式：

$$\left(\psi_{i-1,j}-2\psi_{i,j}+\psi_{i+1,j}\right)+\frac{\Delta R}{2R_i}\left(\psi_{i-1,j}-\psi_{i+1,j}\right)+\left(\frac{\Delta R}{\Delta Z}\right)^2\left(\psi_{i,j-1}-2\psi_{i,j}+\psi_{i,j+1}\right)=-\mu_0 R_i(\Delta R)^2 j_{i,j} \tag{2.22}$$

令 $\left(\frac{\Delta R}{\Delta Z}\right)^2=c$，$1+\frac{\Delta R}{2R_i}=b_i$，$1-\frac{\Delta R}{2R_i}=d_i$，则可将上式化简为：

$$-c\psi_{i,j-1}+2(1+c)\psi_{i,j}-c\psi_{i,j+1}-b_i\psi_{i-1,j}-d_i\psi_{i+1,j}=(\Delta R)^2\mu_0 R_i j_{i,j} \tag{2.23}$$

假设网格的数量为 $(M+2)\times(N+2)$，取 i=0,1,2,⋯,M+1，j=0,1,2,⋯,N+1，接下来，我们将同一 R 处的格点，即 i 相同的格点所列出的差分方程组合到一起，。可以得到如 2.24 式所示的关系式，其中的 i 和 j 的取值范围是不包括边界的，即 $i=1,2,3,\cdots,M$，$j=1,2,3,\cdots,N$，

$$-b_i\begin{bmatrix}\psi_{i-1,1}\\ \psi_{i-1,1}\\ \vdots\\ \psi_{i-1,N}\end{bmatrix}+\begin{bmatrix}2(1+c) & -c & & 0\\ -c & 2(1+c) & -c & \\ & \ddots & & \\ 0 & 0 & -c & 2(1+c)\end{bmatrix}\begin{bmatrix}\psi_{i,1}\\ \psi_{i,1}\\ \vdots\\ \psi_{i,N}\end{bmatrix}-d_i\begin{bmatrix}\psi_{i+1,1}\\ \psi_{i+1,1}\\ \vdots\\ \psi_{i+1,N}\end{bmatrix}=\begin{bmatrix}(\Delta R)^2\mu_0 R_i j_{i,1}+c\psi_{i,0}\\ (\Delta Z)^2\mu_0 R_i j_{i,2}\\ \vdots\\ (\Delta Z)^2\mu_0 R_i j_{i,N}+c\psi_{i,N+1}\end{bmatrix} \tag{2.24}$$

定义：

$$\Psi_i=\begin{bmatrix}\psi_{i,1}\\ \psi_{i,1}\\ \vdots\\ \psi_{i,N}\end{bmatrix},\quad \mathbf{A}=\begin{bmatrix}2(1+c) & -c & & 0\\ -c & 2(1+c) & -c & \\ & \ddots & & \\ 0 & 0 & -c & 2(1+c)\end{bmatrix},\quad \mathbf{S}_i=\begin{bmatrix}(\Delta R)^2\mu_0 R_i j_{i,1}+c\psi_{i,0}\\ (\Delta Z)^2\mu_0 R_i j_{i,2}\\ \vdots\\ (\Delta Z)^2\mu_0 R_i j_{i,N}+c\psi_{i,N+1}\end{bmatrix}$$

同时，由于采用的为狄拉克边界条件，即 Ψ_0 和 Ψ_{M+1} 均为已知项，将它们均移至方程右侧，于是，所有的格点的差分方程可以组织为如下形式

$$\begin{bmatrix}\mathbf{A} & -d_1\mathbf{I} & & & & \\ -b_2\mathbf{I} & \mathbf{A} & -d_2\mathbf{I} & & & \\ & -b_3\mathbf{I} & \mathbf{A} & -d_3\mathbf{I} & & \\ & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & -d_{M-1}\mathbf{I}\\ & & & & -b_M\mathbf{I} & \mathbf{A}\end{bmatrix}\begin{bmatrix}\boldsymbol{\psi}_1\\ \boldsymbol{\psi}_2\\ \boldsymbol{\psi}_3\\ \vdots\\ \boldsymbol{\psi}_M\end{bmatrix}=\begin{bmatrix}\mathbf{S}_2+b_1\boldsymbol{\psi}_0\\ \mathbf{S}_3\\ \mathbf{S}_4\\ \vdots\\ \mathbf{S}_M+d_M\boldsymbol{\psi}_{M+1}\end{bmatrix} \tag{2.25}$$

式 2.25 中的系数矩阵属于有着特殊结构的块三对角方程，这一类型的线性方程组的求解可以通过 Buneman 算法[24]或者离散正弦傅里叶变换（DST）方法[25-27]在 $O(m^2\log m)$ 的时间复杂度内完成。

式 2.25 的求解需要预先求得网格边界上的磁通，网格点处的磁通只能通过类似式 2.20 的方法来求解。只是这一计算的时间复杂度大大降低，因为只需要利用格林函数法求出网格边界点上的磁通，时间复杂度为 $O(m^2\times(4\times m))$。这一复杂度虽然远小于直接法求解格点磁通的方法，但是却大于前面描述的块三对角方程的求解，因此是格点磁通计算中的主要耗时部分之一。

完成上述计算后，得到的格点磁通只是等离子体电流产生的，因此还需要叠加上外部电流产生的磁通，这里只考虑 PF 线圈所产生的磁通，这一计算需要一个大小为（$m^2\times k_F$）的格林函数矩阵乘以一个含有 k_F 个元素的电流，在绝

大多数托卡马克上，k_F 都是一个较小的数字，因此这一计算的时间复杂度不大。

在求得格点上的磁通之后，为了计算ψ_N，还必须求出边界处的磁通ψ_{bdy}和磁轴处的磁通ψ_{axis}，同时，如果存在 X 点，还需要找出 X 点的位置。根据格点磁通信息之后也可以计算出具体的边界，确定边界和 X 点的方法将在下一章中详细介绍。

2.2.6 平衡反演的数值计算流程

根据之前的描述，可以总结出托卡马克中利用 Picard 迭代方法进行等离子体反演的基本流程。如图 2.3 所示，图中显示的只是计算中的基本流程，省略了中间的实现细节，这些在第 3 章中会有详细的逐一介绍。

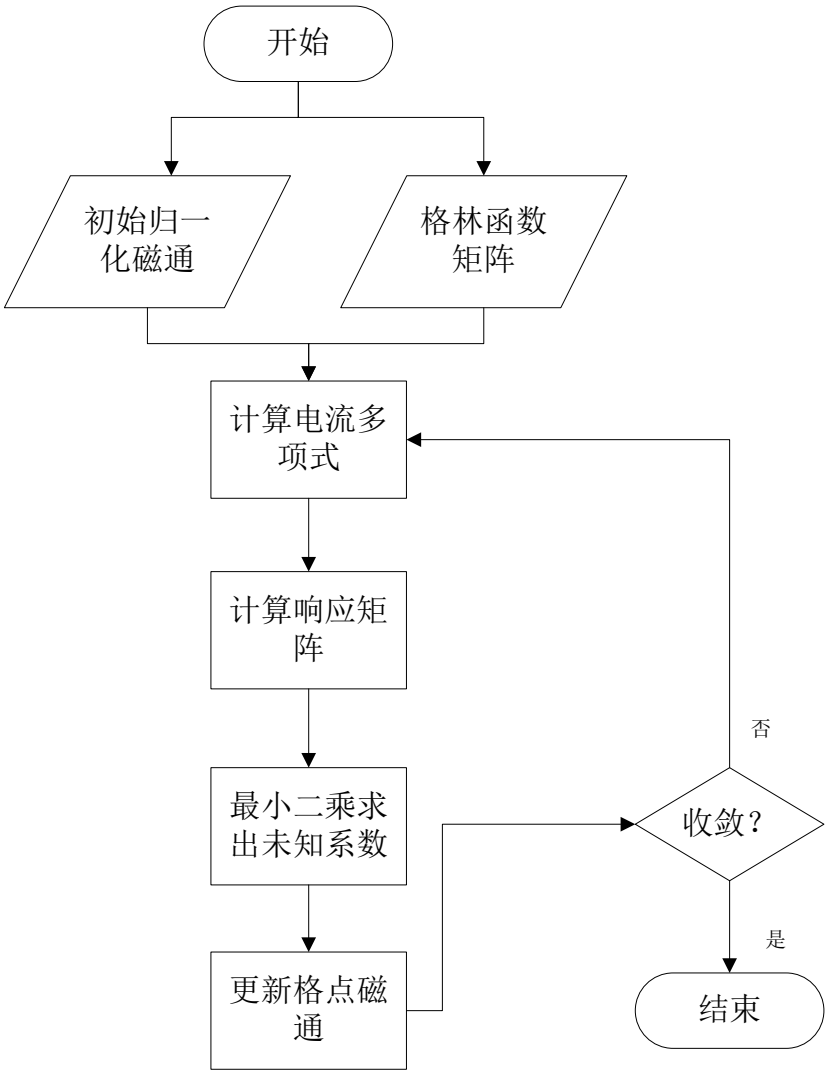

图 2.3 反演计算的基本流程

第 3 章 基于 GPU 的并行化实时平衡重建算法

第二章描述了一种在托卡马克中进行平衡重建的方法，这种方法的优点是考虑了反演出的平衡完全最大程度上拟合了 G-S 方程，具有较高的精确度，而且可以获得内部剖面的一些信息。然而，这个算法的一个缺点是计算量较大，以 65×65 的网格为例，一次完整的平衡重建过程往往需要花费上百毫秒的时间。因为这一原因，这一算法无法直接用于实时控制。这一章将介绍一种利用 GPU 并行计算技术的快速平衡重建算法。

式 3.1 是著名的阿姆达尔定律，式中的 P 是算法中可以并行执行部分的比例，S 是总的加速比，N 是用于并行计算的核数。由于我们的目标要将原算法提升数十倍以上，因此从式 3.1 可以看出，P 必须要非常接近与 1，也就是说整个反演算法的绝大多数部分都必须被并行加速，假如程序中有 10%的部分只能串行执行，则由 3.1 可以得出该程序的加速比不可能超过 10。然而，在原有的算法，有相当比例的运算是无法并行的，要达到既定的目标，必须对这些算法进行改造，同时，对于可以并行的部分，必须利用已有的计算资源达到尽量高的加速效率，以提升总加速比。在本章中将详细介绍 P-EFIT 各个模块的具体实现方法。

$$S(N)=\frac{1}{(1-P)+\frac{P}{N}} \tag{3.1}$$

3.1 构建电流多项式模型的并行化

3.1.1 筛选电流格点的并行化算法

P-EFIT 采用与 EFIT 相同的计算网格设置，以便利用已有的格林函数矩阵。因此，在进行电流多项式模型计算时必须考虑当前格点与限制器之间的关系，如果该格点位于限制器外部，则会将这一格点上的等离子体设为 0。否则将根据其归一化磁通值构造如式 2.16 所示的多项式电流。

首先考虑直接在 GPU 上进行电流格点位置的判断，则可以采用如下的策略，以 65×65 网格为例，可以分配 63 个 block（不包括边界格点），用每个 block

负责判断一行格点的判断，每个 block 中分配 63 个 threads，然后所有的 threads 并发执行来判断所有格点是否在限制器内。限制器在计算机中是以一系列的线段来表示的，这一判断过程可以认为是对这些线段一次进行是否在其右侧的判断（假定限制器描述是顺时针方向的），对于单个 thread 来说，这些判断只能利用一个 GPU 的标量处理器（SP）来顺序进行计算。而 GPU 上的 SP 只是一个简化的 CPU，因此这一运算的效率被大大降低，加速比很不理想。

在实际应用中，限制器的形状在反演计算开始之前就已经确定，而且这一设置在一轮实验中很少会变动，因此每个格点是否在限制器内的信息是可以在反演之前确定，不需要在每次反演计算中进行判断。具体的方法如下：

在每次实验之前，程序会根据当前的限制器形状计算出每个格点是否在限制器内部，并且会统计出每一行格点从第几个格点开始进入限制器内，这一行中有几个格点在限制器内，以及这一行的第一个限制器内点在最终的电流向量中的位置。假设计算格点为 65×65，其具体计算流程如下：

1、分配 63 个 block，每个 block 分配 63 个 threads。

2、每个 block 读入一行格点的归一化磁通，读入预先计算好的格点信息，包括该行对应的位于限制器内的格点数量，cu_num_block[blockIdx.x]。该 block 中被用于计算的 thread 要满足 threadIdx.x< cu_num_block[blockIdx.x]。该行的第一个有效格点（位于限制器内的，最终会被计入电流向量中的格点）在最终电流向量中的位置 offset = cu_blo_accu[blockIdx.x]。以及该行中位于限制器内的格点编号 cu_num_se[offset+threadIdx.x]。

3、读入相关数据后，筛选有效格点可以用以下简单操作来完成。

```
//具体实现在 data_eli 函数中，此为简化说明
if (threadIdx.x<cu_num_block[threadIdx.x])
{
        int offset = cu_blo_accu[blockIdx.x];
        int ori_index = cu_num_se[offset+threadIdx.x]
        int new_index = offset + threadIdx.x;
        effect_array [new_index] = ori_array[ori_index];
        //effect_array 为筛选得到的有效格点向量，
        //ori_array 为原始格点向量
  }
```

利用上述方法，每个 threaad 仅需要进行一次操作就可以完成筛选。整个筛选算法完全并发执行，效率很高。

3.1.2 边界磁通查找的并行算法

托卡马克中等离子体边界的定义是最后一个闭合的不与限制器相交的磁面。在 EFIT 中是利用称作 boundary tracing 的试探法来找出满足这一条件的磁面。然而这一算法是很耗时的，同时，对于实时位形控制来讲，详细的边界信息不是必须的，反演程序需要提供的只有控制点上的磁通与边界磁通之间的差值，因此，实时反演程序需要关注的是快速的找出边界磁通的方法。

首先考虑最外层磁面与限制器相交的情况。EAST 的限制器形状如图 3.1 所示，其中加粗的红色线条部分是等离子体接触点可能存在的范围。典型的限制器位形如图 3.2 所示，由于接触点 a 是限制器上唯一与最外层磁面相交的点，而磁面磁通是由中心向外递减的，因此图 3.2 中 a 点处的磁通值一定是限制器上所有点中最大的。因此，寻找接触点可以转化为寻找限制器上磁通值最大点的问题。

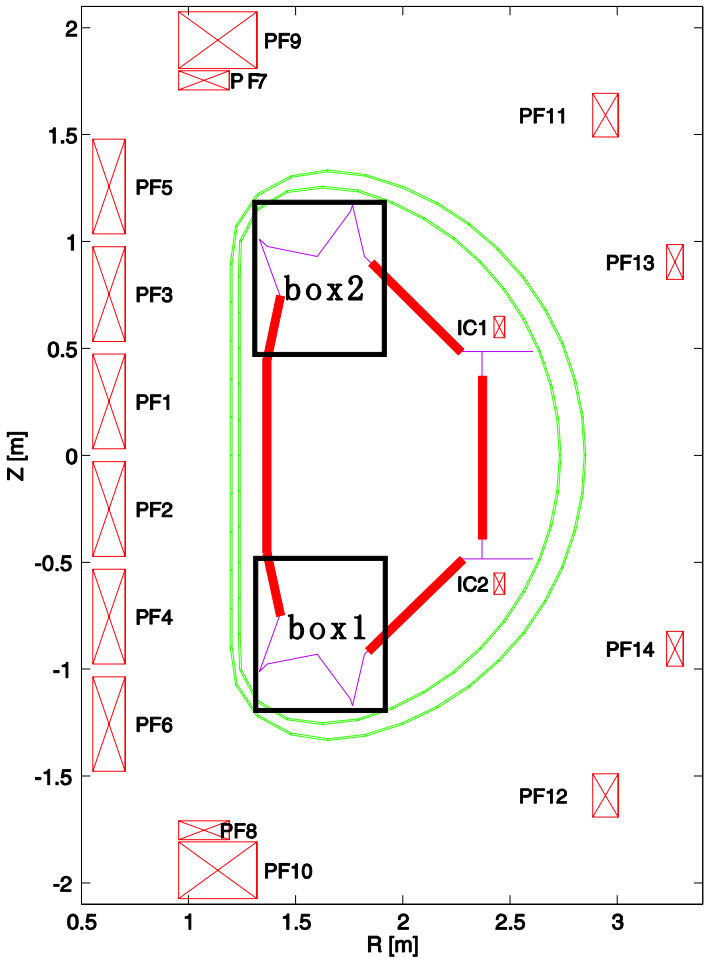

图 3. 1 EAST 真空室结构

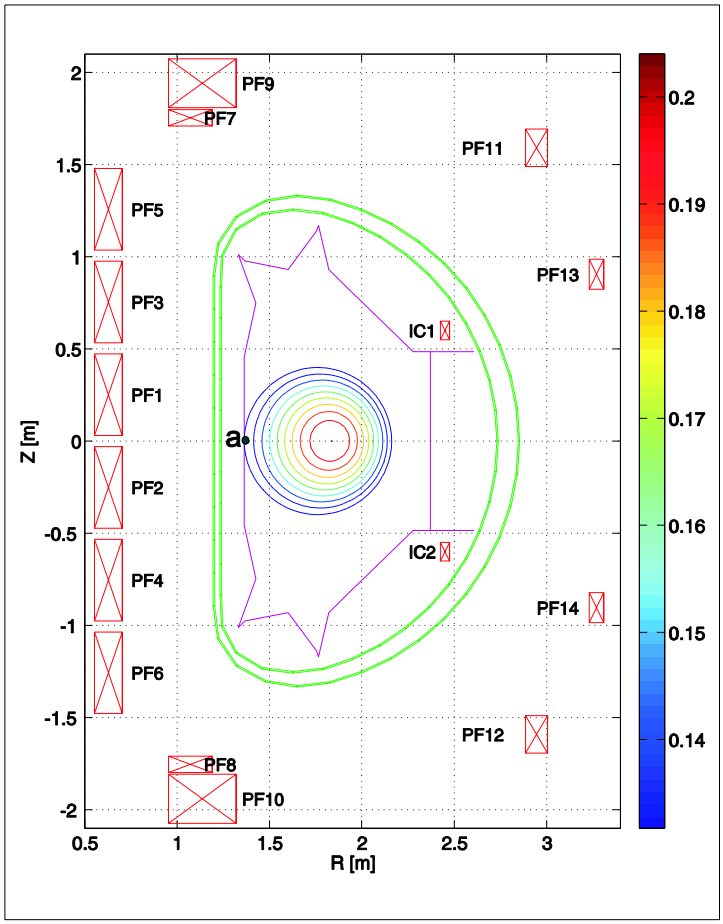

图 3.2 典型的限制器位型

然而，在拉长位形下，特别是在真空室内存在 X 点的情况下，在真空室靠近上下偏滤器的位置会形成 private flux 区，即图 4.3 中 X 点下部的等磁通线，这些等磁通线的存在会导致 X 点下部的区域磁通值大于真实的边界磁通，因此在计算时要排除位于下 X 点以下的限制器点以及上 X 点以上的限制器点。

为了保证反演程序的速度，磁通最大的限制器点是通过计算一个放电前确定好的限制器点序列的磁通最大值来寻找的，这一序列中的元素数量为 100~150，为了保证接触点的精度，这一序列点的选取要配合期望的放电位形，在期望位形确定之后，接触点可能出现的位置一般而言是处于一个很小的范围内的。因此只要在这些范围内较密集的选取，而在其他区域内较稀疏的选取即可。在限制器点的坐标确定以后，限制器上的磁通值会根据周围的四个格点的磁通值进行线性插值来得到。

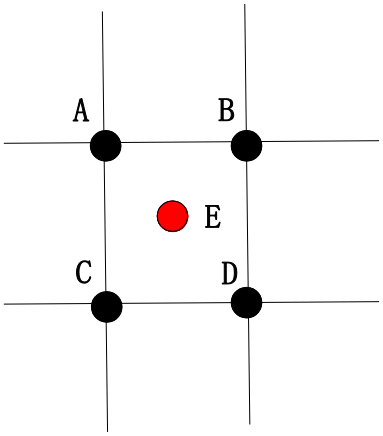

图 3.3 四点差分示意图

如图 3.3 所示，红色的 E 点如果是我们要求的限制器点，黑色的 A、B、C、D 则是该限制器点临近的四个网格点，这四个点上的磁通已知，分别为ψ_A，ψ_B，ψ_C和ψ_D，则 E 点上的磁通ψ_E可以由式 3.2 和 3.3 求出：

$$\psi_E = a\psi_A + b\psi_B + c\psi_C + d\psi_D \tag{3.2}$$

设 $rpos = (x_E - x_A)/(x_B - x_A)$， $zpos = (z_E - z_A)/(z_C - z_A)$

则 3.2 中的未知量可以用下式表示：

$$\begin{aligned} a &= 1 - rpos - zpos + rpos \times zpos \\ b &= rpos - rpos \times zpos \\ c &= zpos - rpos \times zpos \\ d &= rpos \times zpos \end{aligned} \tag{3.3}$$

由于在放电开始之前，选取的限制器点便已经确定，因此为了避免每个计算周期都计算相同的 *rpos*， *zpos* 以及 *a, b, c, d*，P-EFIT 采用的方法是在每次放电开始之前将这些不变量计算好，然后传入 GPU 的 global memory 中，这样在实时反演过程中就只需要从内存中读取相应的量来计算 3.2 式，减少了大量的除法以及乘法运算。这里利用的是线性插值的方法来求解限制器磁通，理论上讲，利用双三次样条插值的方法（bi-cubic interpolation）进行插值更为精确，但是对于格点数量为 65×65 或者以上的情况，这两种插值方法得到的结果差异其实很小（最终误差小于 0.1mm），对于实时控制而言利用利用线性插值精度已经足够。当然，从增强程序可靠性的角度讲，双三次样条插值是更好的选择，因此，有必要研究在 GPU 上的快速双三次样条插值算法。

对于 EAST 而言，很多情况下最外层闭合磁面并不与限制器相交，这就是所谓的偏滤器位形。以下单零位形（即只在下半部存在一个 X 点）为例，产生

的过程是 PF8 和 PF10 产生与等离子体同向的电流，随着这两个线圈电流的增加，等离子体逐渐被向下拉长，同时，如果将这两个 PF 线圈和等离子体电流视为两个独立电流源，则它们产生的二极场会存在一个零点，且该零点位于两者之间。随着 PF8 和 PF10 电流的增大，这一零点会被逐渐推向等离子体区域，如果这一零点被推入等离子体内部，则会在该零点区域形成一个分离点，沿着该点所在磁面运动的粒子会在这点被排出，打到偏滤器靶板上。而这之外的磁面都将是不闭合的，因此不会有等离子体存在。于是，按照等离子体边界的定义，X 点所在的磁面就近似地成为了新的等离子体边界。

根据上述描述，当前位形是限制器还是偏滤器的便可以通过判断两者所在磁面的位置关系来判断。如果限制器所接触的最内层磁面位于 X 点所在磁面之内，则可知此时的 X 点尚未被推至等离子体内部，当前放电位形依然是限制器位形，反之，则 X 点已经被推至等离子体内部，当前位形为偏滤器位形。而判断磁面相对关系则可以简单的通过比较两者的极向磁通大小来实现。

对于位形控制而言，X 点的位置也是需要的。因此，必须开发计算 X 点位置和 X 点磁通的算法。目前，离线 EFIT 所使用的寻找 X 点的算法是先找到最外层磁面，然后在这一磁面上找到极向磁场 B_p 最小的点，然后以此点为原点做一阶泰勒展开，找出 B_p 为零的点。而用于实时控制的 RT-EFIT 使用的方法是用上一次找到的 X 点作为展开原点。

对于 P-EFIT 而言，由于并没有像离线 EFIT 一样进行精确地 boundary tracing 计算，因此只能采用类似 RT-EFIT 的方法。不过 RT-EFIT 的寻找方法有一个很大的缺陷，如果新的 X 点与上一次找到的 X 点相比位置变化较大的话，其展开原点与真实的零点距离也会过大，则上述方法可能会失效。因此，结合 GPU 的高并发特性，P-EFIT 采用了如下所述的 X 点寻找算法：

首先，利用 kernel 函数 expansion_finder 中找出上下 X 点区域中 B_p 绝对值最小的点，作为展开的原点。方法如下：

为了防止找到 PF 线圈附近的零点，同时节省计算时间，我们定义了两个区域，如图 3.1 中的 box1 和 box2 所示，寻找 expansion 点将只在这两个 box 中进行。以 65×65 格点为例，假设两个 box 共圈住了 38 行 32 列的格点，则可以分配 38 个 block，每个 block 分配 32 个 thread，每个 block 负责一行的计算。

利用有限差分计算出 $B_R = -\frac{1}{R}\frac{\partial \psi}{\partial z}$ 和 $B_z = -\frac{1}{R}\frac{\partial \psi}{\partial R}$，然后，计算每一个格点的 $B_R^2 + B_z^2$，由于 $|B_P| = \sqrt{B_R^2 + B_z^2}$，因此 $|B_P|$ 最小的点也就是 $B_R^2 + B_z^2$ 最小的点。可以用并行的算法来寻找这一最小值。其基本原理可以用下面的代码描述，这

一代码描述了并行查找 32 个数字中最小值的方法。

```
//具体实现在 expansion_finder 函数中，此为简化说明
index[threadIdx.x] = threadIdx.x;
//index 是一个 shared memory 数组，存储最小值位置信息
for(unsigned int s=16;s>0;s>>=1)
{
   int k = threadIdx.x+s;
   if(threadIdx.x < s && k<32)
   {
        c = data[threadIdx.x];
        //data 是 shared memory 数组，存储数值信息
        d = data[k];
        if(c>d)
        {
             data[threadIdx.x] = d;
             index[threadIdx.x] = index[k];
             //__syncthreads();
        }
   }
}//计算后 data[0]存储了最小值，index[0]存储着最小值坐标
```

需要特别说明的是，由于上述代码中每个 block 只分配了 32 个 threads，在计算过程中每个 block 中始终只有一个 warp 在执行，因此可以省略掉同步的操作，否则每次对 data 和 index 数组操作都必须保证对其他所以线程可见之后才能继续执行，也就是要去掉__syncthreads()前的注释（同步操作是十分耗时的，在设计算法时要尽量避免）。

由于在一个 kernel 内不同 block 之间的线程互相之间无法通信，因此通过这一个 kernel 函数只能找到每一行中的最小值以及该最小值在这一行中的位置。这些被作为中间变量储存在 global memory 中，下一个 kernel 函数 xLocate 会利用相似的算法从这些每一行的最小值中找出全局的最小值以及它的位置，然后以此点为原点做一阶泰勒展开，找出 B_p 为零的点，具体原理如下：

设展开原点处的极向磁场为 $(B_r^{\,0}, B_z^{\,0})$，假设该点的坐标为 (r,z)，X 点的坐标为 $(r+\delta r, z+\delta z)$，则可以得出以下表达式：

$$\begin{cases} B_r^0 + \dfrac{dB_Z}{dR}\delta r + \dfrac{dB_Z}{dZ}\delta z = 0 \\ B_z^0 + \dfrac{dB_R}{dR}\delta r + \dfrac{dB_R}{dZ}\delta z = 0 \end{cases} \tag{3.4}$$

于是可以得到$(\delta r,\delta z)$的表达式 3.5，

$$\begin{cases} \delta r = \dfrac{(B_r^0 \dfrac{dB_Z}{dZ} - B_z^0 \dfrac{dB_R}{dZ})}{(\dfrac{dB_Z}{dR}\dfrac{dB_R}{dZ} - \dfrac{dB_R}{dR}\dfrac{dB_Z}{dZ})} \\ \delta z = \dfrac{(B_z^0 \dfrac{dB_R}{dR} - \dfrac{dB_Z}{dR} B_r^0)}{(\dfrac{dB_Z}{dR}\dfrac{dB_R}{dZ} - \dfrac{dB_R}{dR}\dfrac{dB_Z}{dZ})} \end{cases} \tag{3.5}$$

3.5 式中的磁场值以及磁场梯度可以利用极向磁通的有限差分来获得，但是更为精确的方法是预先制作好 box1 和 box2 中所有格点的(B_r,B_z)以及位于其右下角 0.5cm 处的(B_r,B_z)与电流向量之间的响应矩阵，在找到展开原点之后，只要利用该响应矩阵乘以电流向量即可求出该点以及其临近点处精确的极向磁场值，也就获得了精确的磁场梯度值。这一运算的时间复杂度并不高，以 65×65 格点为例，该响应矩阵大小仅为（8×1956）。

利用 3.5 式求得$(\delta r,\delta z)$之后，X 点的坐标便可以确定，此时还需要判断的是 X 点与展开原点之间的距离，如果这一距离超过了 2 个格点宽度，则我们认为这一次查找是不成功的。此外，如果最终找出的 X 点位于所规定的两个 box 之外，我们也认为这次查找是不成功的，这样做主要是为了避免找到的 X 点是 PF 线圈附近的零点，而非真正的 X 点。

在确定所找出的 X 点有效之后，可以利用前面所描述的线性插值方法求出这一点处的极向磁通ψ_X。

得到上下两个 X 点出的极向磁通之后，将它们的最大值与上一步找到的最大限制器点磁通比较，如果 X 点处磁通较大，则将 X 点处磁通作为边界磁通，此时放电为偏滤器位形，否则此时放电为限制器位形，边界磁通取限制器点上的最大磁通值。

磁轴处的磁通可以简单地利用最大格点磁通来代替，因为在磁轴附近的极向磁通梯度很小，这样的近似带来的误差很小。寻找最大磁通格点的方法与前面所描述的寻找最大限制器点磁通的算法类似。

在得到边界和磁轴磁通之后，可以利用式 2.13 求出所有格点上的归一化磁通，同时利用 3.1.1 中所描述的方法进行格点筛选。得到最终的多项式电流向量。

电流多项式模型的构建过程可以用图 3.4 来概括。可以看出这一过程需要的操作时十分复杂的。同时，这一过程也比较耗时，在目前的版本中，这一部分的运行时间占程序总运行时间的 15%以上，但这一部分的运行效率存在较大提升潜质。由于当前版本是针对 Fermi 架构来设计的，各个 kernel 之间完全顺

序执行的（即使采用 Stream 来 overlap 效率也很低），而最新的采用 kepler 架构的 K20 显卡利用 Hyper-Q 技术可以实现更高效的 kernel 间并发执行，这样图 2.13 之间不存在数据依赖的 kernel 便可以同步执行，同时，利用最新的 Dynamic Parallelism 技术可以将那些原本必须用多个 kernel 执行的任务合并至一个 kernel 中运行，减少 kernel launch 的时间消耗。

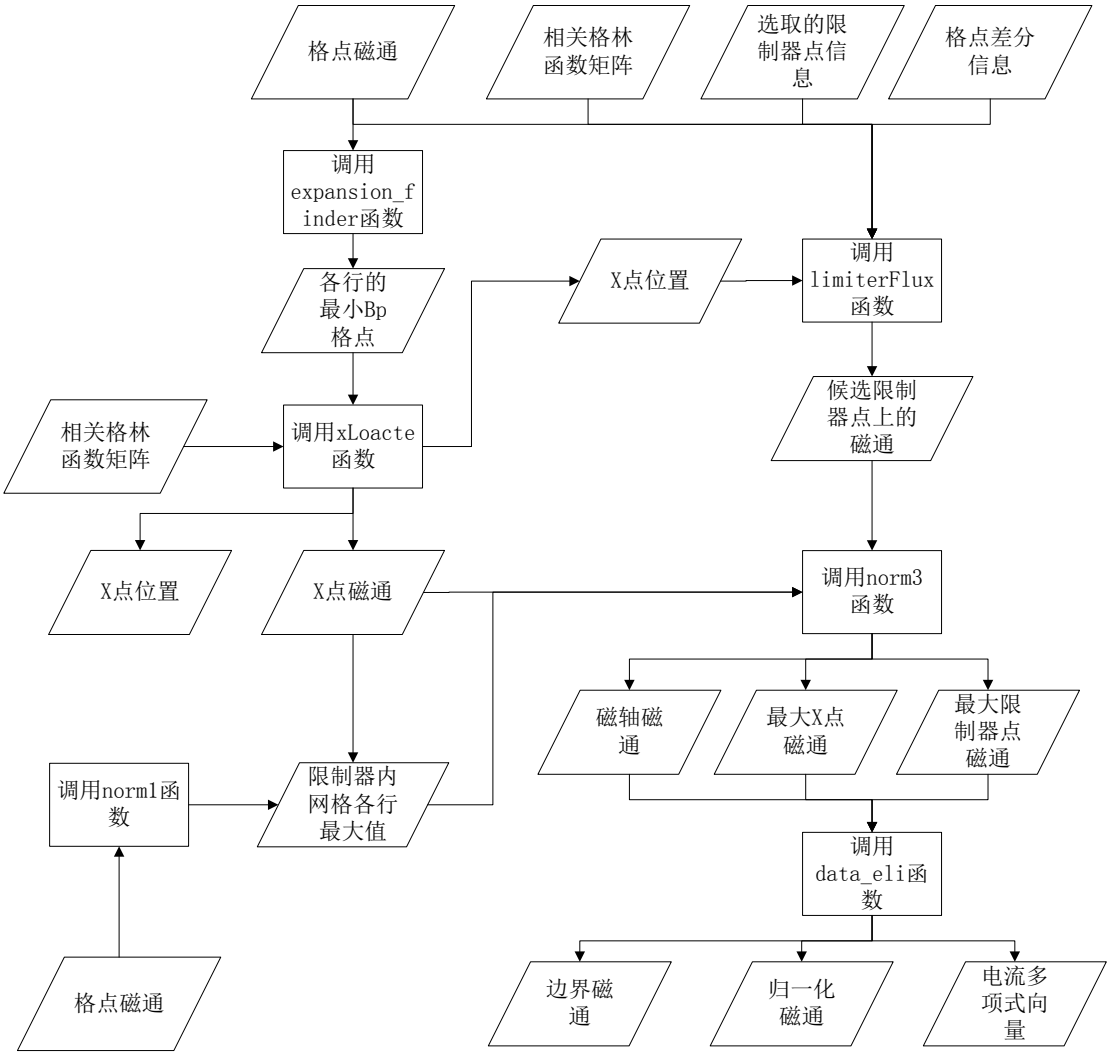

图 3.4 寻找边界磁通的基本流程

3.2 响应矩阵求解的并行算法

响应矩阵的求解方法在 2.2.3 节中有描述，其中主要的耗时部分是 $\mathbf{G_p} \times \mathbf{H}$ 的计算。在本节中，将以该矩阵相乘为例介绍 P-EFIT 中处理这类矩阵相乘的并行算法。

以 65×65 的格点为例，假设此时限制器内格点数量为 1956，外部诊断数

量为 74，多项式电流参数为$n_p=2$，$n_f=1$，在这种情况下，$\mathbf{G_P}$为一个 74 行 1956 列的矩阵。$\mathbf{H}$是一个 1956 行 4 列的矩阵。

在使用 Intel Xeon E31230 3.2GHz CPU 的计算机上利用 BLAS 串行的计算这一矩阵相乘运算需要大约 900μs。而利用 NVIDIA 公司开发的 CUBLAS 库在 Tesla c2050 GPU 上并行的执行这一运算也要消耗 250μs以上。可以看出，这一运算的效率很低，c2050 的理论浮点运算能力在 1TFlops/s 以上。而这一运算的速度仅达到了 2GFlops/s。

效率如此之低的主要原因是在计算矩阵相乘的过程中，如果不能将这个矩阵一次加载入缓存中，则会存在反复读取数据的问题，在极端情形下，即缓存命中率为零的情况下，访存次数等于浮点运算次数。而 c2050 的访存带宽只有 144GB/s，访存的延迟更是多达 400~800 个时钟周期。由于$\mathbf{G_P}$和$\mathbf{H}$的规模对于 GPU 来讲并不算大，而且形状也比较特殊，因此利用当前的 CUBLAS 库尚无法有效的实现加速。为此，必须针对该矩阵相乘的特例设计专门的矩阵相乘算法，为了提高效率，必须结合 GPU 的硬件特点来提高访存效率。

图 3.5 展示了 GPU 的内存模型，由于$\mathbf{G_P}\times\mathbf{H}$的问题规模限制。Global memory 的访存延迟几乎不可能被掩盖，因此要尽量减少对 Global memory 的读写次数。

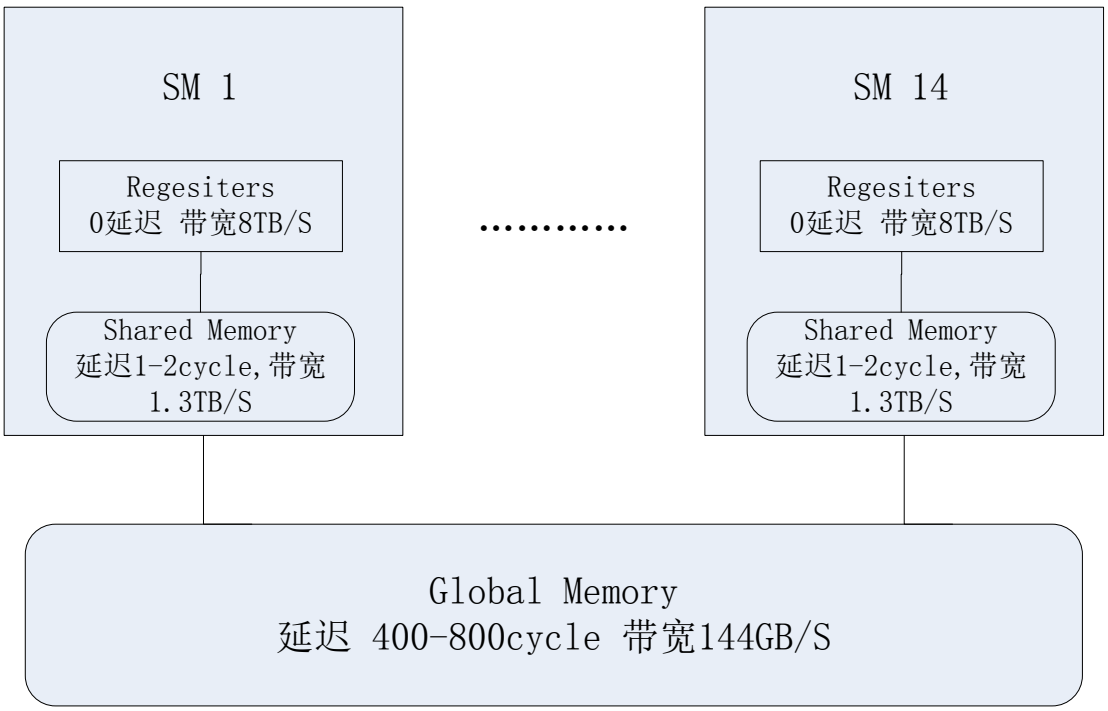

图 3.5 Tesla c2050 存储器结构示意图

对于$\mathbf{A}\times\mathbf{B}=\mathbf{C}$的矩阵相乘的问题，最直接的办法是用一个 thread 来负责求

解 C 中的一个元素，然而，在这种情况下，假设 A 有 m 行，B 有 n 列，则 A 中的每个元素会被从 global memory 中反复读取 n 次，而 B 中的每个元素会被从 global memory 中反复读取 m 次。显然，这样的策略是不可行的，如此大量的 global memory 访问会严重降低整个算法的效率。

上述策略中主要存在的问题在于各个 thread 是完全独立运行的，互相之间完全无法共享数据，因此每个都必须重新读入需要的数据才可以完成计算。事实上，可以通过线程间共享数据的方式来减少内存访问。例如，C 中第一行数据的计算都需要用到 A 中的第一行数据，如果负责第一行的 n 个线程之间可以共享数据的话，则 A 的第一行只需要从 global memory 中读入一次即可。假如 C 中每一行的 n 个线程之间数据都可以共享的话，则 A 中的每个元素都只需要从 global memory 中读入一次即可。然而，此时 B 中的每个元素依然需要读入 m 次。如果我们进一步假设所有的线程之间数据均可以互相共享的话，则 A 和 B 中的每个元素都只需要从 global memory 中读入一次。

然而，在实际中，线程之间的数据共享存在很多限制，这要从 GPU 的硬件架构来讲起。以 Tesla C2050 为例，它所采用的 GF100 核心共有 448 个标量处理器（SP），这 448 个处理器分布在 14 个流多处理器（SM）中，每个 SM 中有 32 个 SP。如图 3.5 所示，每个 SM 上存在有称作 shared memory 的高速缓存，利用这一缓存，正在这一 SM 上执行的线程可以实现数据的共享。而不同 SM 上运行的线程之间则无法通信或者共享数据。在 CUDA 的软件模型中，与之对应的是 block 和 thread 的设置，每个 block 中的 threads 互相之间是可以相互通信的，而一个 block 只能在一个 SM 上运行。尽管每个 block 中可以定义上千个 thread，但是同一时刻只会有 32 个 thread 在被执行。

从以上分析可以看出，如果要让计算 $\mathbf{A}\times\mathbf{B}=\mathbf{C}$ 所有的线程都实现贡献数据，这些线程都必须分配在一个 block 中，这样其实只利用了 GPU 的一个 SM，计算资源被严重浪费。

除了上述原因，shared memory 大小的限制也是一个重要的因素。GF100 核心每个 SM 上的 shared memory 最大容量只有 48KB。只能存放 12000 个 4Byte float 数，像 $\mathbf{G_P}\times\mathbf{H}$ 这类规模的问题显然不可能一次性放入 shared memory 中。

在 CUDA C Programming Guide 中，提出了一种对矩阵进行分块形式进行计算的方法。如图 3.6 所示，每一个 block 会负责计算 C 中的一个方块子矩阵。假设这个子矩阵的宽度和长度均为 BLOCK_SIZE，在这种情况下 A 中的每个元素会被读取 $\frac{n}{BLOCK\_SIZE}$ 次，而 B 中的每个元素会被读取 $\frac{m}{BLOCK\_SIZE}$ 次。

为了保证对 global memory 的合并访问，子矩阵的宽度应该大于 128 bit，即 32 个 4 Byte float 数或者 16 个 8 Byte float 数。然而，这种方法的应用范围是很有限的，对于特定形状的矩阵相乘，例如$\mathbf{G_P}\times\mathbf{H}$来讲并不适用，但是其部分思想是可以借鉴的，下面介绍针对$\mathbf{G_P}\times\mathbf{H}$的矩阵乘法。

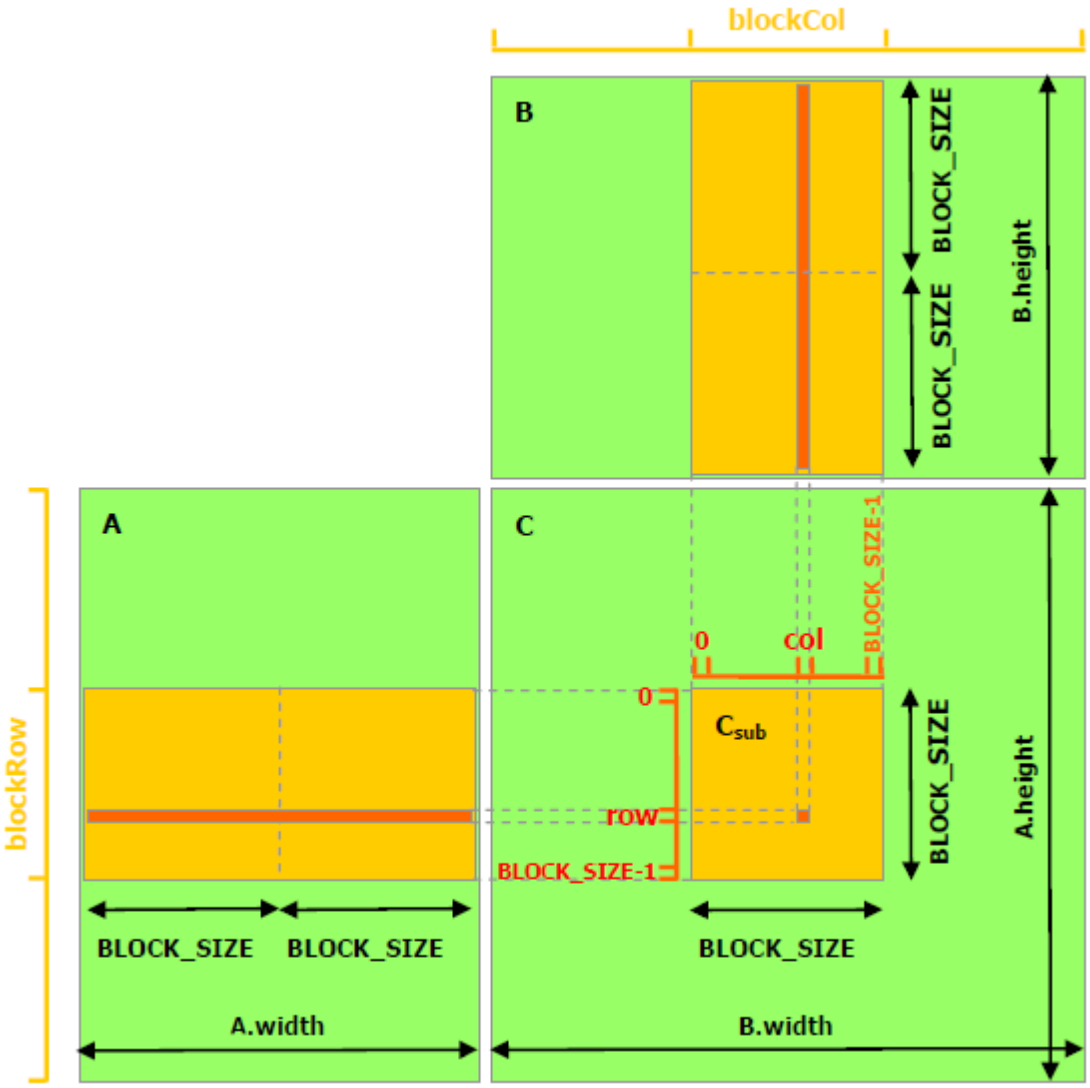

图 3.6 分块利用 shared memory 的矩阵相乘算法示意图

为了后续计算的方便，我们实际求解的是$(\mathbf{G_P}\times\mathbf{H})^{\mathbf{T}}$，即$\mathbf{H^T}\times\mathbf{G_P}^{\mathbf{T}}$，图 3.7 显示了 P-EFIT 对这一矩阵相乘的分割方法，考虑到计算中合并访问的问题，$\mathbf{G_P}$在内存中并没有采用转置的形式存储，不过在实际计算是按照正确的方式进行的。这一分割是在 tridaig_mul 函数中进行的，一共分配了 61 个 block，其中每个 block 分配了（32×24）个 thread，这些 thread 以二维形式组织的，即 threadDim.x = 32，threadDim.y = 24。在 tridaig_mul 函数中，每个 block 会负责求解$\mathbf{H^T}$中一

个（4×32）的子矩阵和$\mathbf{G_P}$中一个（74×32）的子矩阵的相乘运算。由于每个 SM 上最多分配了 5 个 block，在不影响 GPU 利用率的情况下每个 block 可以使用的 shared memory 大小为 9.6KB。不足以放下两个子矩阵，而受限于 thread 数量，最终采用的方法是将$\mathbf{G_P}$中的子矩阵分四次读入，每次读入后都与$\mathbf{H^T}$中的子矩阵进行相乘，最后将四次相乘得到的矩阵组合到一起，得到一个 4 行 74 列的矩阵。而子矩阵相乘的算法类似于 CUDA C Programming Guide 中的子矩阵相乘算法。每个 thread 负责结果中一个元素的求解。通过 tridaig_mul 函数的计算，每个 block 都会得到一个 4 行 74 列的子矩阵，将这 61 个子矩阵每个对应元素累加起来，最终得到的 4 行 74 列的矩阵便是最终的计算结果。由于 CUDA 没有提供全局同步的功能，因此没有办法在这个 kernel 函数中完成这 61 个子矩阵的累加。这一 kernel 函数会输出如图 3.8（a）所示的中间数据。

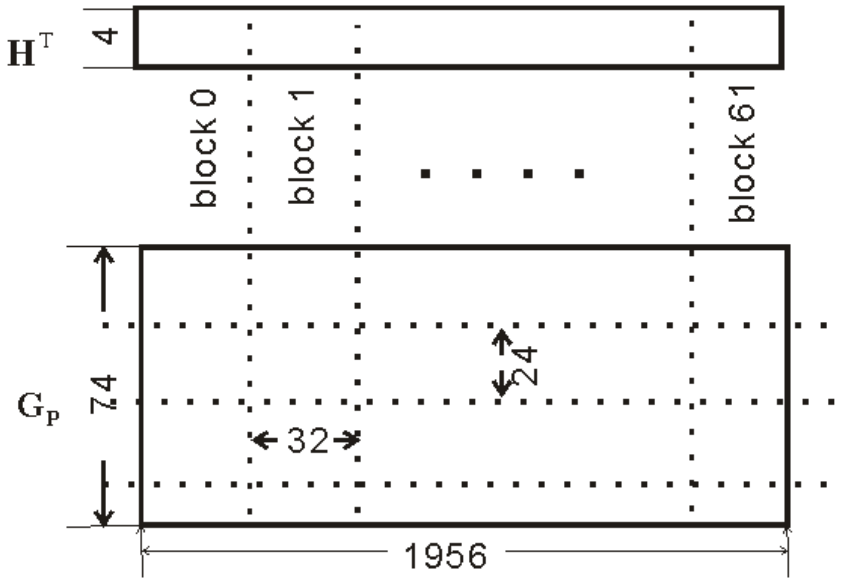

图 3. 7 tridaig_mul 函数中的矩阵划分

3.8(a)所示的中间数据接下来由 mul_postprocess 函数处理，这一个 kernel 函数使用了 74×4 个 thread，这些 thread 以二维形式组织，这些 thread 全部位于一个 block 中。在这个 kernel 函数中，每个 thread 负责最终结果中的一个元素，整个程序包含一个 61 次的循环，每次循环会将一个 3.8（a）中的一个子矩阵读入，然后进行累加，最后将得到如 3.8（c）所示的最终结果。

采用上述算法，$\mathbf{G_P}\times\mathbf{H}$最终在少于 40μs的时间内完成，速度的提高主要是由于对 global memory 访问次数的大幅减少，因为原问题的中的主要瓶颈便是访存的延迟。在这一算法中，原矩阵的每个元素只需要从 global memory 中读出一次即可。然而，这一算法依然存在很大的改进空间，例如，可以尝试减少线程

的数量，同时增加每个线程的独立操作，从而得到计算效率的提升（具体方法可参考 2010 年 GTC 大会的报告“Better Performance at Lower Occupancy”）。同时在新一代的 Kepler 架构中，threads 之间可以通过寄存器交换数据，而寄存器的带宽是远大于 shared memory 的，因此利用这一方法同样可以大幅的提升计算的效率。

由于$\mathbf{G_C}$已经事先求得，所以只要将$\mathbf{G_C}$以转置形式存储，然后将刚刚计算得到的矩阵存入$\mathbf{G_C}^{\mathbf{T}}$之后，于是得到$\mathbf{\Gamma^T}=\begin{pmatrix}\mathbf{G_C}^{\mathbf{T}}\\(\mathbf{G_P}\times\mathbf{H})^{\mathbf{T}}\end{pmatrix}$。至此便完成了响应矩阵的计算。

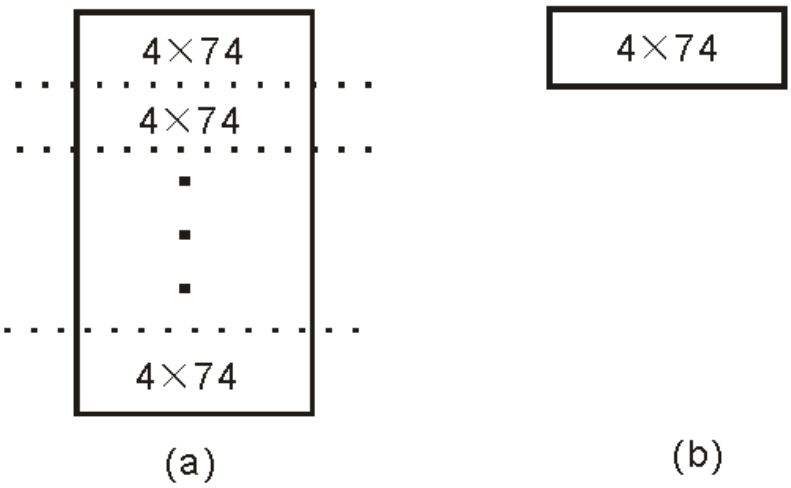

图 3.8 响应矩阵求解的中间数据

3.3 最小二乘拟合的并行算法

在得到$\mathbf{\Gamma^T}$之后，根据当前所对应的诊断信号，通过 data_process_right 函数计算得到权重矩阵$\mathbf{F}$并求出$\mathbf{F}\cdot\mathbf{D}$，记为$\mathbf{D}_{\mathrm{W}}$，接着在 data_process_left 函数中求解出加权后的响应矩阵$(\mathbf{F}\cdot\mathbf{\Gamma})$，记为$\mathbf{\Gamma}_{\mathrm{W}}$。

接下来要做的便是求解未知量，即求解超定方程$\mathbf{\Gamma}_{\mathrm{W}}\mathbf{U}=\mathbf{D}_{\mathrm{W}}$。这一问题的实质是极小化残差向量范数$\|\mathbf{\Gamma}_{\mathrm{W}}\mathbf{U}-\mathbf{D}_{\mathrm{W}}\|$的问题。

由于在大多数托卡马克装置上，$\mathbf{\Gamma}_{\mathrm{W}}$会呈现出严重的病态，即条件数很高的特点，而诊断量$\mathbf{D}_{\mathrm{W}}$又存在一定的误差，因此必须要考虑计算结果对误差的敏

感性。

考虑将$\mathbf{\Gamma}_{\mathrm{W}}$做 SVD 分解，得到

$$\mathbf{\Gamma}_{\mathrm{W}}=\mathbf{P}\sum\mathbf{V}^{\mathbf{T}} \tag{3.6}$$

其中 **P** 和 **V** 均为正交向量，$\sum$为对角向量，其对角线元素为$\mathbf{\Gamma}_{\mathrm{W}}$的奇异值，因而可以将$\mathbf{\Gamma}_{\mathrm{W}}$的伪逆表示为

$$\mathbf{\Gamma}_{\mathrm{W}}{}^{+}=\mathbf{V}\sum{}^{-1}\mathbf{P}^{\mathbf{T}} \tag{3.7}$$

于是，未知向量**U**中的第 i 个元素u_i可以表示为[28]

$$\mathbf{U}=\sum_{i=1}^{n}\frac{V_iP_i^{\mathbf{T}}\mathbf{D}_{\mathrm{W}}}{\sigma_i}=\sum_{i=1}^{n}\frac{P_i^{\mathbf{T}}\mathbf{D}_{\mathrm{W}}}{\sigma_i}V_i \tag{3.8}$$

上式中的σ_i为$\mathbf{\Gamma}_{\mathrm{W}}$的第 i 个奇异值，而 n 为$\mathbf{\Gamma}_{\mathrm{W}}$的奇异值数量。

现在考虑$\mathbf{D}_{\mathrm{W}}$中存在的误差，假设原有的磁测量诊断信号**D**中的精确测量信号为$\mathbf{D}^{*}$，误差向量为**e**，而**e**满足正态分布$N(0,\mathbf{S})$，其中**S**为诊断信号的方差矩阵，由于诊断信号的误差量是独立的，S为一个对角矩阵$diag(F_1,F_2\cdots F_m)$，其中的 m 是诊断信号的数量，由于$\mathbf{D}_{\mathrm{W}}=\mathbf{F}^{-1}\mathbf{D}$，因此$\mathbf{D}_{\mathrm{W}}$中含有的的误差向量$\boldsymbol{\eta}\sim N(0,\mathbf{I_m})$。于是可以将式 2.18 写为：

$$\mathbf{\Gamma}_{\mathrm{W}}\mathbf{U}^{*}+\boldsymbol{\eta}=\mathbf{D}_{\mathrm{W}}{}^{*} \tag{3.9}$$

其中的$\mathbf{U}^{*}$是真实系统的未知参数，$\mathbf{D}_{\mathrm{W}}{}^{*}$是测量系统无误差时的理论输出。将式 3.7 带入 3.9 中，并在左右两侧同时乘以$\mathbf{P}^{\mathbf{T}}$可以得到：

$$\sum\mathbf{V}^{\mathbf{T}}\mathbf{U}^{*}+\mathbf{P}^{\mathbf{T}}\boldsymbol{\eta}=\mathbf{P}^{\mathbf{T}}\mathbf{D}_{\mathrm{W}}{}^{*} \tag{3.10}$$

由于$\mathbf{P}^{\mathbf{T}}$是一个正交矩阵，因此$\mathbf{P}^{\mathbf{T}}\boldsymbol{\eta}$与$\boldsymbol{\eta}$相比只会转动一定角度，大小并不会发生变化，因此$\mathbf{P}^{\mathbf{T}}\boldsymbol{\eta}$依然满足$N(0,\mathbf{I_m})$分布。设$m=\dim(\mathbf{D_W})$，$n=\dim(\mathbf{U})$。令$\mathbf{P}=(\mathbf{P_1},\mathbf{P_2})$，其中$\mathbf{P_1}$为大小为 m×n 的子矩阵，$\mathbf{P_2}$为大小为 m×（m-n）的子矩阵，于是，

$$\min\left\|\mathbf{\Gamma}_{\mathrm{W}}\mathbf{U}-\mathbf{D}_{\mathrm{W}}\right\|^2=\min\left\|\begin{pmatrix}\Sigma\\0\end{pmatrix}\mathbf{V}^{\mathbf{T}}\mathbf{U}-\begin{pmatrix}\mathbf{P_1^T}\\\mathbf{P_2^T}\end{pmatrix}\mathbf{D}_{\mathrm{W}}\right\|^2 \tag{3.11}$$

当直接采用式 3.8 求解带有诊断误差的问题时，得到的结果$\hat{\mathbf{U}}$满足：

$$\mathbf{V}^{\mathbf{T}}\hat{\mathbf{U}}=\Sigma^{-1}\mathbf{P_1^T}\mathbf{D}_{\mathrm{W}} \tag{3.12}$$

或者可以写作：

$$(\mathbf{V}^{\mathbf{T}}\hat{\mathbf{U}})_i=\frac{(\mathbf{P}^{\mathbf{T}}\mathbf{D}_{\mathrm{W}})_i}{\sigma_i},\quad i=1,2,3\cdots n \tag{3.13}$$

由式 3.10 可以看出，$\mathbf{P}^{\mathbf{T}}\mathbf{D}_{\mathrm{W}}$中的每一项都含有误差向量$(\mathbf{P}^{\mathbf{T}}\boldsymbol{\eta})_i$，对于$(\sum\mathbf{V}^{\mathbf{T}}\mathbf{U}^{*})_i$远大于$(\mathbf{P}^{\mathbf{T}}\boldsymbol{\eta})_i$的情况而言，式 3.13 中的该项不会对结果产生重大影响，

但是 $(\sum \mathbf{V}^{\mathbf{T}}\mathbf{U}^{*})_i$ 会随着 i 的增加而减小，如果到某一项以后，$(\sum \mathbf{V}^{\mathbf{T}}\mathbf{U}^{*})_i$ 的大小与 $(\mathbf{P}^{\mathbf{T}}\boldsymbol{\eta})_i$ 相当甚至远小于 $(\mathbf{P}^{\mathbf{T}}\boldsymbol{\eta})_i$，则 3.13 中的对应项会对最终结果带来重大影响。此时，$(\mathbf{P}^{\mathbf{T}}\mathbf{D}_{\mathrm{W}})_i$ 的大小与 $(\mathbf{P}^{\mathbf{T}}\boldsymbol{\eta})_i$ 相当，一种可以减小误差的方法是直接在 3.13 式中将这些项的 $(\mathbf{P}^{\mathbf{T}}\mathbf{D}_{\mathrm{W}})_i$ 置为 0。则 3.13 式变为了：

$$(\mathbf{V}^{\mathbf{T}}\hat{\mathbf{U}})_i = \begin{cases} \dfrac{(\mathbf{P}^{\mathbf{T}}\mathbf{D}_{\mathrm{W}})_i}{\sigma_i}, & i = 1,2,3,\cdots p \\ 0, & i = p, p+1\cdots n \end{cases} \tag{3.14}$$

该方法称作 truncated SVD，是处理此类病态问题的一种常用方法，当然，要使用该方法，必须确定式 3.14 中 p 的大小，这可以通过 L-CURVE 方法来确定，这种确定 p 的方法计算量极大，不可能实时进行，只能利用以前的实验数据来近似当前的数据，这一近似成立的前提是诊断信号的误差分布在前后几次放电中不会有明显变化。

由于奇异值 σ_i 总是随着 i 的增加而递减的，因此在确定 p 之后，式 3.14 所示的截断条件相当于为其中的 σ_i^{-1} 加入了一个过滤函数，该函数可以表示为

$$f(\sigma_i) = \begin{cases} 1, & \sigma_i > \sigma_p \\ 0, & \sigma_i < \sigma_p \end{cases} \tag{3.15}$$

我们也可以将这一过滤函数改写为 $g(\sigma_i) = \dfrac{\sigma_i^2}{\sigma_i^2 + \alpha^2}$，于是过滤后的 σ_i^{-1} 为

$\dfrac{\sigma_i}{\sigma_i^2 + \alpha^2}$，可以看出 $g(\sigma_i)$ 这一表达方式与 3.15 是近似等价的，当 σ_i 大于 α 时，

$g(\sigma_i)$ 接近于 1，反之，则 $g(\sigma_i)$ 会之间逼近 0。

于是，可以将式 3.13 写为：

$$\begin{aligned} \hat{\mathbf{U}} &= \mathbf{V}(\Sigma^2 + \alpha^2\mathbf{I})^{-1}\Sigma\mathbf{P}^{\mathbf{T}}\mathbf{D}_{\mathrm{W}} \\ &= (\mathbf{V}\Sigma^2\mathbf{V}^{\mathbf{T}} + \alpha^2\mathbf{I})^{-1}\mathbf{V}\Sigma\mathbf{P}^{\mathbf{T}}\mathbf{D}_{\mathrm{W}} \end{aligned} \tag{3.16}$$

上式可进一步化简为：

$$(\boldsymbol{\Gamma}_{\mathbf{w}}^{\mathbf{T}}\boldsymbol{\Gamma}_w + \alpha^2\mathbf{I})\hat{\mathbf{U}} = \boldsymbol{\Gamma}_{\mathbf{w}}^{\mathbf{T}}\mathbf{D}_{\mathrm{W}} \tag{3.17}$$

式 3.17 的求解不需要计算奇异值分解，而且主要的时间消耗是 $\boldsymbol{\Gamma}_{\mathbf{w}}^{\mathbf{T}}\boldsymbol{\Gamma}_w$ 这一矩阵相乘运算，因此利用 GPU 可以很容易的进行并行化加速，这种方法称作 norm equation 方法，而相应针对测量误差引入的操作称为 Tikhonov regularization[29]。

式 3.17 中这一普通线性方程组的求解可以采用 pivoting L-U 分解的方法进

行求解，由于这一方程组维数很少（<20），因此对于它的求解在 CPU 上反而可以获得更好的效果，因此，目前的 P-EFIT 中，是将相应的系数矩阵传回至 CPU 中进行 pivoting L-U 分解计算，之后再将计算结果传回 CPU 中。

在目前的 EAST 反演中，并没有采用前面所描述的截断 SVD 或者 Tikhonov regularization 方法，而是直接对反应矩阵进行 SVD 分解后求逆，对应的，在当前的 P-EFIT 中，当前设置的 Tikhonov regularization 参数 $\alpha = 0$，下一步，将在分析 EAST 以往实验数据的基础上，加入适当的 regularization 参数进行测试。

为了验证采用 norm equation + pivoting L-U 方法求解这一最小二乘的可行性，我们从 EAST 实验数据中挑选了 1000 组数据进行反演，在进行最小二乘求解时，分别采用了 SVD 分解求解响应矩阵伪逆以及 norm equation + pivoting L-U 方法。然后分别记录最小二乘过程中产生的残差向量，如图 3.9 所示。在实验中，均未对奇异值进行阶段，也未采用 tikhonov regularization 方法对 norm equation 进行处理。

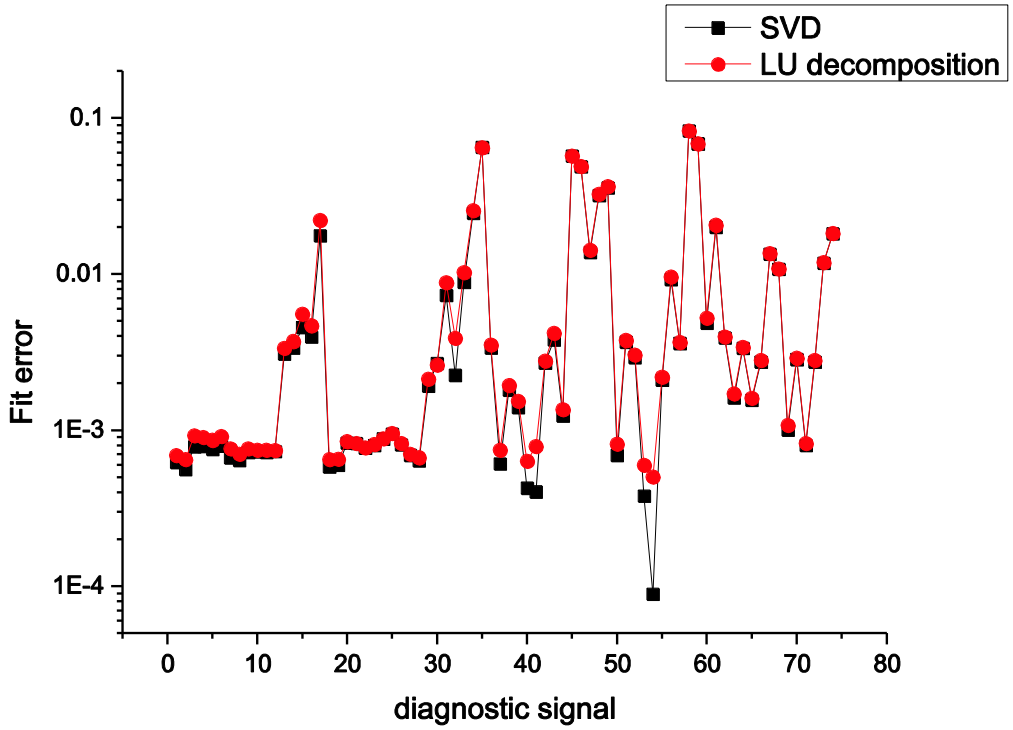

图 3.9 两种方法残差向量的比较

从图中可以看出，尽管采用 L-U 分解方法求得的结果中某些诊断数据的平均残差略大于 SVD 方法，两者的残差向量大小均处于极低的水平，因此，采用 norm equation + pivoting L-U 方法求解最小二乘是可行的，在 Krzysztof Bosak 的硕士论文中对此类问题有过仔细的研究，也得出了类似的结论[30]。

这一方法是非常容易实现并行的，首先是 2.18 式左右两侧的处理，分别在

rightHandSideInit 和 leftHandSideInit 函数中进行 $\mathbf{\Gamma}_W \times \mathbf{D_W}$ 和 $\mathbf{\Gamma}_W{}^T \times \mathbf{\Gamma}_W$ 的计算。接着要将处理后的方程左右两侧全部传回至 CPU 内存内，然后利用 LAPACK 库中的 pivoting L-U 分解进行最终方程的求解。得到需要求解的未知量，然后这些未知量会被传回 GPU 的 global memory 中进行下一步的计算。

为了减少在显卡 global memory 和内存之间进行数据传输的消耗，这里使用了 CUDA 提供的内存映射 API，这会将一段 page-locked host memory 映射到显卡的内存空间中。这样映射的内存在使用时不需要显式的调用数据传输 API，传输会在需要时隐式的进行，同时这些传输会隐式的与 kernel 函数的执行重叠。

在 Tesla c2050 上，上述最小二乘过程可以在 30μs的时间内完成，包括数据在 GPU global memory 和内存之间的两次传输时间。

3.4 格点磁通更新的并行算法

P-EFIT 采用如 2.2.5 节所描述的格点磁通计算方法。该方法通过在狄拉克边界条件下对 G-S 方程进行有限差分求解来获得各个格点上的磁通值。这一算法是十分耗时的，本节将描述 P-EFIT 中的求解算法。

3.4.1 网格边界点磁通的求解

如 2.2.5 所示，对网格边界上磁通值的求解时十分耗时的。为了减少这一时间消耗，可以采用一种近似的方法，该方法利用外部诊断点的磁通值插值来获得网格边界点上的磁通。然而这一方法的误差是远大于 2.25 节中所描述的利用格林函数法的。因此 P-EFIT 依然采用了格林函数法。以 65×65 计算网格为例，这一计算将是一个 256×1956 的矩阵 $\mathbf{G_b}$ 和一个 1956 的电流向量 $\mathbf{I}$ 相乘。计算量略大于 $\mathbf{G_P} \times \mathbf{H}$。这一运算在 CPU 上利用 BLAS 串行计算需要 800μs，而利用 CUBLAS 库并行求解也要消耗 200μs以上。

P-EFIT 采用的求解方法如图 3.10 所示，首先如 3.10（a）所示，为了提高计算效率，$\mathbf{G_b}$ 在内存中是以转置形式存储的，在第一个 kernel 函数 boundflux 中分配 245 个 block，每个 block 负责处理 $\mathbf{G_b}^{\mathbf{T}}$ 中的 8 行子矩阵，每个 block 分配 256 个 threads，每个 threads 负责子矩阵中的一列数据。之后，block 会读入电流向量中当前行所对应的元素，然后分别乘以每个当前行的元素。这之后的就转化成了将这一 1956 行 256 列的矩阵压缩成一个 1 行 256 列矩阵的问题，压缩的方法是将各行累加起来。在第一个 kernel 函数中，每个 block 会循环八次，将八行数据压缩为一行。这样在第一个 kernel 函数结束后，原问题就转化为了一个 245 行 256 列矩阵的压缩问题。由于 CUDA 中没有全局同步，所以只能在

一个新的 kernel 函数中继续压缩工作。在接下来的 post_boundflux 函数中，使用了 16 个 block，每个 block 会用 256 个 threads 来压缩 16 行的数据。在这个 kernel 函数结束后，原问题被转化为了一个 16 行 256 列矩阵的压缩。最后的处理工作在 final_boundflux 函数中由一个 block 完成。

上述算法在 Tesla c2050 GPU 上的执行时间只需要 30μs。

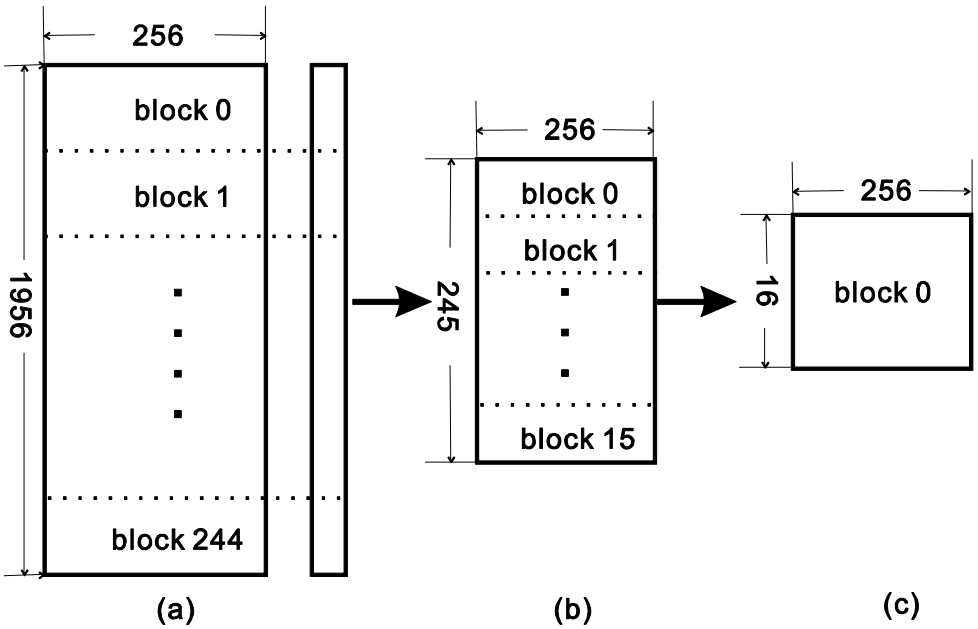

图 3.10 $\mathbf{G}_b \times \mathbf{I}$ 的并行实现示意图

对于这一矩阵相乘的问题，一种比较容易想到的策略是将 $\mathbf{G}_b$ 正常存储，然后分配 256 个 block，每个 block 负责求解结果向量中的一个元素，即负责将一行中的 1956 个元素乘以对应的 $\mathbf{I}$ 向量中的元素，然后将这 1956 个元素累加起来。这种大量数据的累加可以采用 reduction 算法以对数时间复杂度完成，如图 3.11 所示。

然而实际上，与图 3.10 所描述的方法相比，上述算法的运行效率是很低的，主要原因包括以下几个方面：

首先，采用 reduction 方法求解时，每个 block 都需要完整的读入 $\mathbf{I}$ 向量，即 $\mathbf{I}$ 向量中的每个元素都会被读入 256 次，而图 3.10 所描述的算法中所有元素都只需要读入一次。

其次，采用 reduction 算法时，虽然理论上每个线程的操作数少于图 3.10，但是 reduction 的每次操作之后都需要进行线程同步操作，而 256 个线程的多次同步代价是很大的。

最后，采用 reduction 算法必须首先将数据读入 shared memory 中，由于每个 block 必须完整读入一行数据，因此每个 SM 同时可运行的 block 数量会受 shared memory 资源限制，每个 SM 上最多只能有 6 个 block 被分配执行。而图 3.10 所描述的算法中，每个 block 只需要使用 256 个寄存器即可，完全不需要使用 shared memory，而寄存器的访问带宽远大于 shared memory。

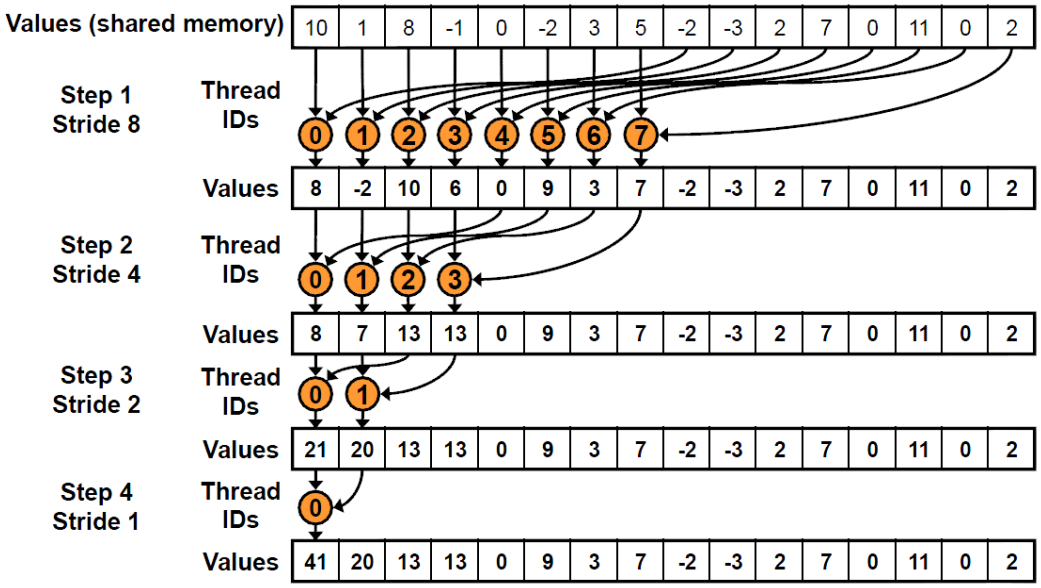

图 3.11 并行 reduction 算法示意图[31]

3.4.2 块三对角方程的并行求解

在求得网格边界点上的磁通以及所有格点上的电流之后，便可以求得 2.25 式右边的所有项。于是求解所有网格点磁通的问题便转化为了求解式 2.25 所示的块三对角方程的问题。这一方程组的系数矩阵（式 3.18）是一个有着特殊结构的稀疏矩阵，因此，当然可以采用针对一般稀疏矩阵的 SOR 等迭代方法来求解，或者采用针对一般正定矩阵的 Cholesky 分解方法来直接求解[32]。下面将介绍一种针对块三对角方程的快速直接求解方法。

$$\begin{bmatrix} \mathbf{A} & -d_1\mathbf{I} & & & & \\ -b_2\mathbf{I} & \mathbf{A} & -d_2\mathbf{I} & & & \\ & -b_3\mathbf{I} & \mathbf{A} & -d_3\mathbf{I} & & \\ & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & -d_{M-1}\mathbf{I} \\ & & & & -b_M\mathbf{I} & \mathbf{A} \end{bmatrix} \tag{3.18}$$

其中的 $\mathbf{A}=\begin{bmatrix} 2(1+c) & -c & & 0 \\ -c & 2(1+c) & -c & \\ & \ddots & & \\ 0 & 0 & -c & 2(1+c) \end{bmatrix}$ 是一个三对角方程，矩阵 A 的特征向量和对应特征值分别为 $q_{ij}=\sqrt{\frac{2}{m+1}}\sin\frac{ij\pi}{m+1}$ 和 $\lambda_i=c(2-2\cos\frac{i\pi}{m+1})+2$，因此对矩阵 A 作特征值分解，可以得到：

$$\mathbf{A}=\mathbf{Q}\Lambda\mathbf{Q}^{\mathbf{T}} \tag{3.19}$$

其中，

$$\mathbf{Q}=\sqrt{\frac{2}{m+1}}\begin{bmatrix} \sin\frac{\pi}{m+1} & \sin\frac{2\pi}{m+1} & \cdots & \sin\frac{m\pi}{m+1} \\ \sin\frac{2\pi}{m+1} & \sin\frac{4\pi}{m+1} & \cdots & \sin\frac{2m\pi}{m+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sin\frac{m\pi}{m+1} & \sin\frac{2m\pi}{m+1} & \cdots & \sin\frac{m^2\pi}{m+1} \end{bmatrix} \tag{3.20}$$

$\mathbf{\Lambda}$ 是由 λ_i 构成的对角矩阵，将 3.19 式带入线性方程组 2.25 中，并将右边项简化表示为 $\mathbf{g}_j$ 可以得到其一般项的如下形式：

$$-b_i\boldsymbol{\psi}_{i-1}+\mathbf{Q}\Lambda\mathbf{Q}^{\mathbf{T}}\boldsymbol{\psi}_i-d_i\boldsymbol{\psi}_{i+1}=\mathbf{g}_i,\ \ i=2,\cdots,M-1 \tag{3.21}$$

在 3.21 式的左右同乘以 $\mathbf{Q}^{\mathbf{T}}$，因为 $\mathbf{Q}$ 为正交矩阵，因此可以得到：

$$-b_i\mathbf{Q}^{\mathbf{T}}\boldsymbol{\psi}_{i-1}+\Lambda\mathbf{Q}^{\mathbf{T}}\boldsymbol{\psi}_i-d_i\mathbf{Q}^{\mathbf{T}}\boldsymbol{\psi}_{i+1}=\mathbf{Q}^{\mathbf{T}}\mathbf{g}_i \tag{3.22}$$

设 $\mathbf{Q}^{\mathbf{T}}\boldsymbol{\psi}_i=\hat{\boldsymbol{\psi}}_i$，$\mathbf{Q}^{\mathbf{T}}\mathbf{g}_i=\hat{\mathbf{g}}_i$，则上式可化为：

$$-b_i\hat{\boldsymbol{\psi}}_{i-1}+\Lambda\hat{\boldsymbol{\psi}}_i-d_i\hat{\boldsymbol{\psi}}_{i+1}=\hat{\mathbf{g}}_i,\qquad i=2,\cdots,M- \tag{3.23}$$

将 $\hat{\mathbf{u}}_i$ 中的元素进行重组，取每个 $\hat{\mathbf{u}}_i$ 中的第 j 个元素，重组得到：

$$\tilde{\boldsymbol{\psi}}_j=\begin{pmatrix}\tilde{\psi}_{1,j} & \tilde{\psi}_{2,j} & \cdots & \tilde{\psi}_{M,j}\end{pmatrix}^{\mathrm{T}},\ \ j=1,\cdots,N \tag{3.24}$$

相应的对 $\hat{\mathbf{g}}_j$ 重组得到 $\tilde{\mathbf{g}}_i$，于是可以得到新的线性系统：

$$\begin{pmatrix} \mathbf{T}_1 & & & \\ & \mathbf{T}_2 & \ddots & \\ & \ddots & \ddots & \\ & & & \mathbf{T}_N \end{pmatrix}\begin{pmatrix} \tilde{\boldsymbol{\psi}}_1 \\ \tilde{\boldsymbol{\psi}}_2 \\ \vdots \\ \tilde{\boldsymbol{\psi}}_N \end{pmatrix}=\begin{pmatrix} \tilde{\mathbf{g}}_1 \\ \tilde{\mathbf{g}}_2 \\ \vdots \\ \tilde{\mathbf{g}}_N \end{pmatrix} \tag{3.25}$$

其中 $\mathbf{T}_j=\begin{pmatrix}\lambda_j & d_1 & & \\ b_2 & \lambda_j & \ddots & \\ & \ddots & \ddots & d_{M-1}\\ & & b_M & \lambda_j\end{pmatrix}\begin{pmatrix}\lambda_j & d_1 & & \\ b_2 & \lambda_j & \ddots & \\ & \ddots & \ddots & d_{M-1}\\ & & b_M & \lambda_j\end{pmatrix}$，$j=1,\cdots,N$。

从 3.25 式可以看出，整个线性系统被划分为了 N 个完全独立的子线性方程组。

求出 $\tilde{\boldsymbol{\psi}}_j$ 之后，再进行一次重组，得到 $\hat{\boldsymbol{\psi}}_i$，因为 $\hat{\boldsymbol{\psi}}_i=\mathbf{Q}^{\mathbf{T}}\boldsymbol{\psi}_i$，因此通过 $\boldsymbol{\psi}_i=\mathbf{Q}\hat{\boldsymbol{\psi}}_i$ 便可以得到最终的计算结果。此外，观察矩阵 $\mathbf{Q}$ 可以发现，该矩阵是属于一类离散正弦变换（DST）矩阵，因此 $\mathbf{g}_i$ 和 $\hat{\mathbf{g}}_i$ 以及 $\hat{\boldsymbol{\psi}}_i$ 和 $\boldsymbol{\psi}_i$ 之间的相互转化是可以通过快速傅里叶变换（FFT）来进行的。综上所述，整个算法的基本流程可以概括为：

1、构造源项 $\mathbf{g}_i$，对 $\mathbf{g}_i$ 进行 DST 变化求得 $\hat{\mathbf{g}}_i$。

2、对 $\hat{\mathbf{g}}_i$ 进行重组，得到 $\tilde{\mathbf{g}}_j$

3、求解独立线性方程组 $\mathbf{T}_j\tilde{\boldsymbol{\psi}}_j=\tilde{\mathbf{g}}_j$，得到 $\tilde{\boldsymbol{\psi}}_j$。

4、对 $\tilde{\boldsymbol{\psi}}_j$ 进行重组，得到 $\hat{\boldsymbol{\psi}}_i$。

5、对 $\hat{\boldsymbol{\psi}}_i$ 进行 DST 逆变换，得到最终结果 $\boldsymbol{\psi}_i$

上述流程的可并行性是显而易见的，首先是两次 DST 变换，可以通过分配 M 个 block，在每个 block 内分配若干 thread 进行 DST 变换。或者直接调用 CUFFT 库中的函数进行变换。然而由于每个变换序列的长度太短，采用 CUFFT 库加速效果并不明显，目前在程序中是直接采用矩阵相乘的方式来进行这一变换的。下一步需要用离散傅里叶变换来替代这一部分。方法是先进行 zero padding，在变换序列前加一个 0，在变换序列后加 n+1 个 0，这样每个变换序列就变成了一个 2(n+1)的序列，对这一序列进行快速傅里叶变换之后，得到的第 2 到第 n+1 个结果中的虚部便是 DST 变换的结果[33]。

而对数据进行重组的过程，实际上是将这些数据所组成的矩阵进行转置的过程，利用 GPU 的高带宽，这一转置可以在很快的速度内完成。

而上述过程中的第 3 步，即求解独立方程的部分也是可以在不同的 block 中完全并发运行的。但是，在每个 block 内部对式 3.26 所示的三对角方程的求解则较难实现并行。

$$\begin{pmatrix}\lambda_j & d_1 & & \\ b_2 & \lambda_j & \ddots & \\ & \ddots & \ddots & d_{M-1}\\ & & b_M & \lambda_j\end{pmatrix}\begin{pmatrix}x_1\\ x_2\\ \vdots\\ x_M\end{pmatrix}=\begin{pmatrix}t_1\\ t_2\\ \vdots\\ t_n\end{pmatrix} \tag{3.26}$$

3.27 所示的三对角方程一般采用一种称作追赶发的特殊高斯消元方法来进

行求解，为了表述方便，将 3.17 式改写成如 3.18 式所示的一般形式的三对角方程。

$$\begin{pmatrix} b_1 & c_1 & & \\ a_1 & b_2 & \ddots & \\ & \ddots & \ddots & c_{n-1} \\ & & a_{n-1} & b_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{pmatrix} \tag{3.27}$$

这一追赶过程可以用式 3.28 和 3.29 所示。

向前消元阶段：

$$\begin{aligned} b_i' &= b_i - (a_{i-1} / b_{i-1}')c_{i-1} \\ d_i' &= d_i - (a_{i-1} / b_{i-1}')d_{i-1}' \end{aligned} \tag{3.28}$$

向后回代阶段：

$$\begin{aligned} x_n &= d_n' / b_n' \\ x_i &= (d_i' - c_i x_{i+1}) / b_i' \end{aligned} \tag{3.29}$$

从上面两个式子来看，整个追赶法过程是完全串行的，如果直接将这一算法用作 GPU 上求解，则意味着每个 block 内只有一个 thread 在真正执行，相当于每一个时刻只利用了每个 SM 上的一个标量处理器，效率显然是很低的。

设式 3.19 中 $k_{i-1} = a_{i-1} / b_{i-1}'$，3.20 式中 $m_i = c_i / b_i'$，对于 P-EFIT 而言，只要网格划分确定，式 3.18 中的线性方程系数就是固定的，因此可以预先将 k_{i-1}，m_i 和 b' 计算好存入 GPU 的 global memory 中，则 3.28 和 3.29 中的未知量只剩下 d' 和 x_i，将 3.28 和 3.29 式重写为 3.30 的形式，

$$\begin{aligned} d_i' &= d_i - k_{i-1} d_{i-1}' \\ x_i &= d_i' / b_i' - m_i x_{i+1} \end{aligned} \tag{3.30}$$

将向前消元截断展开，如式 3.31 所示，

$$\begin{aligned} & d_1' = d_1 \\ & d_2' = d_2 - k_1 d_1' \\ & d_3' = d_3 - k_2 d_2' = d_3 + (-k_2) d_2 + (-k_2)(-k_1) d_1 \\ & d_4' = d_4 + (-k_3) d_3 + (-k_3)(-k_2) d_2 + (-k_3)(-k_2)(-k_1) d_1 \\ & \vdots \\ & d_n' = d_n + (-k_{n-1}) d_{n-1} + (-k_{n-1})(-k_{n-2}) d_{n-2} + \cdots + \prod_{i=1}^{n-1} (-k_i) d_1 \end{aligned} \tag{3.31}$$

一种容易想到的并行策略是分配 n 个线程，每个线程计算一个 d_i'。但是，这样并行的话最终的速度实际甚至会慢于串行，因为串行的总计算量是 $n(1+n)$，而采用这种并行策略时，第 n 个 thread 执行的计算量为 $n(1+n)/2$，即使并行效

率为 100%，理论最大的加速比也只有 2。实际上由于线程调度，同步的时间消耗，以及 GPU 的单个线程执行效率问题，采用这种策略的 GPU 程序基本不会有加速效果。

仔细观察 3.22 式可以看出，采用上述并行策略时，每个线程都重复计算了前面那些线程所计算过的部分，例如，负责计算 d_3' 的线程实际上也计算了 d_2'，因此如果各个 thread 之间能够充分共享自己所计算过的资源，则可以大大提高并行的效率。在 W. DANIEL HILLIS 的文章[34]中描述了一个采用这一思想的算法，这一算法解决的问题称作 prefix sum 或者 plus scan 问题，如图 3.12 所示，有一个大小为 16 的数字序列 $x_0, x_1 \cdots x_{15}$，要求的是这个序列每个前缀子序列的和，即 x_0， $x_0 + x_1$， $x_0 + x_1 + x_2$， ……， $x_0 + x_1 + x_2 + \cdots x_{15}$。

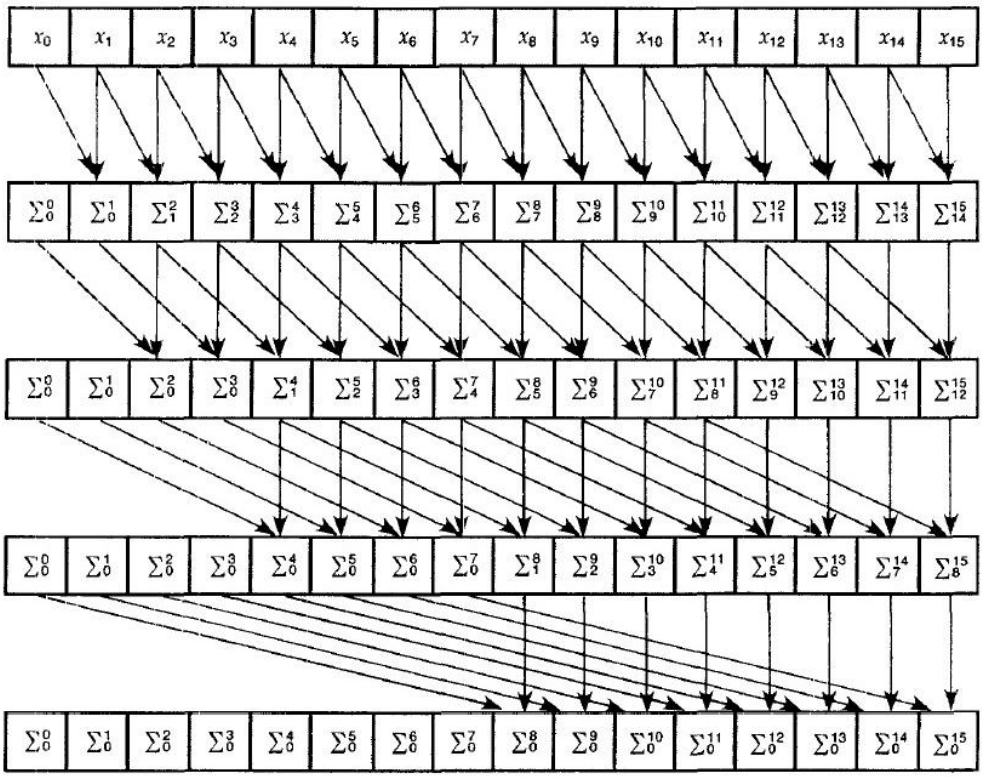

图 3.12 prefix sum 算法示意图

这一问题直观上看是完全串行的，因为我们必须先求出前面所有数的和才能去求解下一个数的和，对于只有一个处理器的情况来说，情况确实是这样，但是如果有 n 个处理器，则可以采取另外一种处理。设 $\Sigma_{i=j}^{k} x_i = \Sigma_j^k$，显然，有 $\Sigma_j^k + \Sigma_{k+1}^m = \Sigma_j^m$，于是容易想到这样一种算法，分配 n 个线程，从步长为 1 开始，每个线程同步的以这一步长向前累加。于是这一步长便会以 2 的指数形式增长，

使得这一累加过程可以在 $\log_2 n$ 步内完成。这一算法可以用图 3.12 直观的展示，其伪代码如下所示：

for j = 1 **to** $\log_2 n$ **do**
  for all m **in parallel do**
    if $m \geq 2^j$ **then**
      $x[m] = x[m-2^{j-1}] + x[m]$
    end if
  end do
end do

在有足够的并行执行单元的情况下，这一算法 $O\log_2 n$ 的时间复杂度内完成，与前面所描述的并行策略相比，效率提升十分明显。

现在回过头来看 3.22 式，可以如果忽略其中的 $(-k_i)$ 项，则这一问题就是一个典型的 prefix sum 问题。而进一步观察可以发现 d_i' 中的 $(-k_i)$ 项是以一种类似于 prefix sum 的问题，只要将上面伪代码中的‘+’号换成‘×’号即可计算出每个 d_i' 中的 $(-k_i)$ 项，于是，只要将上述伪代码做些修改即可，修改后的代码如下所示。其中我们定义 k_0 =1。

for j = 1 **to** $\log_2 n$ **do**
  for all m **in parallel do**
    if $m \geq 2^j$ **then**
      $d[m] = k[m-1] \times d[m-2^{j-1}] + d[m]$
      $k[m-1] = k[m-1] \times k[m-1-2^{j-1}]$
    end if
  end do
end do

上述代码很容易用 CUDA 实现，在 65×65 情况下，n 的值为 63，意味着如果有 63 个独立执行单元便可以在 $2\times \log_2 n$ 次的执行周期内完成上述计算，由于 Fermi 架构中每个 SM 上只有 32 个 SP，因此这一代码需要 $2\times(\log_2 n+1)$ 个执行周期。这样理论上每个 SP 最多只需要执行 14 次乘加运算即可，而如果串行的话则每个 SP 最多需要执行 126 次乘加运算。

上述算法的实现需要大量的线程间同步，因此在 fermi 架构上实现时必须将 $d[m]$ 和 $k[m-1]$ 存入 shared memory 中。于是，在每次更新这两个值的时候要特别注意线程间的同步问题，每次循环中，必须将新的值暂时放入寄存器中，然后所有线程进行同步，确保所有的 shared memory 读取已经完成之后才可以

将数据存入 shared memory 中，然后在进行一次同步，确保所有 shared memory 写入都已完成之后才可以开始下一步循环。为了减少同步的时间消耗，可以尝试只用 0 到 31 号 thread 来进行计算，虽然这样会导致计算量的略微增加，但是这 32 个 thread 始终是以一个 warp 的形式来运行的，完全不需要同步操作，因此可以节省同步的时间消耗。

之后的回代过程与向前消元过程十分类似，只需要稍作修改即可。

有了这个并行追赶算法，整个块三对角方程的并行效率得到了很大提高，首先是各个独立线性方程组之间的粗粒度并行，然后是各个独立线性方程组内部追赶发的细粒度并行，与 GPU 的硬件架构之间构成了直接的映射。图 3.13 显示了不同网格精度下的加速比以及运行时间。可以看出，随着网格精度的提高，这一算法的加速比也逐渐增高。

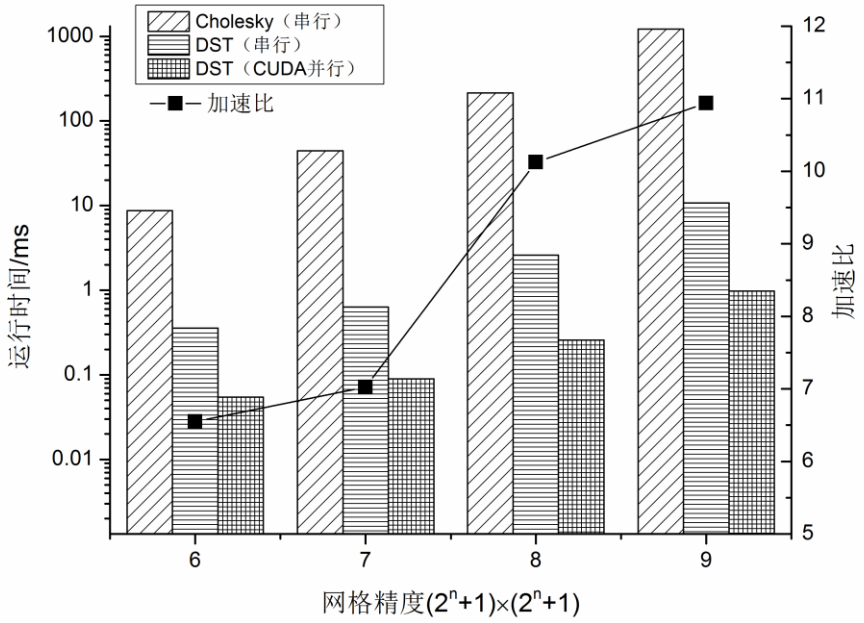

图 3.13 不同网格精度下的加速比

图 3.14 中显示的是不同的网格精度下 DST 块三对角方程求解算法在 GPU 上的运行时间构成。从中可以看出，在网格精度较低，如 65×65 的情况下，主要的时间消耗是 DST 的求解。而随着网格精度的提升，这一部分的运行时间所占比例快速下降，出现这一现象的主要原因是目前在 P-EFIT 中所应用的求解 DST 的算法是直接调用 CUFFT 库，而 CUFFT 在计算规模较小时的效率较低，因此下一步应该针对反演中所遇到的计算规模开发基于 GPU 的 DST 算法。

同时，可以看出在 65×65 的情况下，求解独立方程时间所占的比例很低，然而随着网格精度的进一步提升，这一部分所占的比例快速上升，导致这一现

象的主要原因是我们在开发求解独立方程算法时，针对的只是 65×65 的情况，在网格提升之后，如果不修改原有的代码，则这一算法的 GPU 利用率会严重下降。因此如果将来要开发适用于更高网格精度的 P-EFIT 程序，必须针对特定情况对代码进行重新的优化。

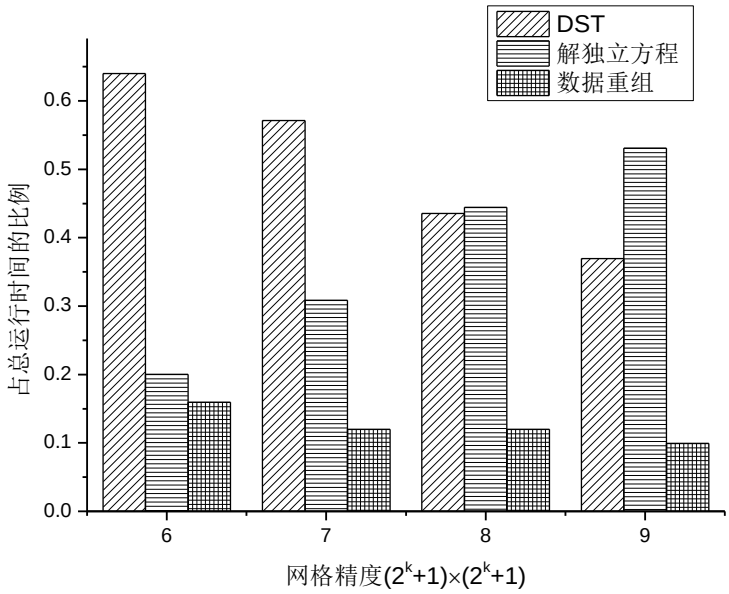

图 3.14 不同网格精度下算法的运行时间构成对比

由于这一算法采用的是大规模系数矩阵的直接解法，而且在并行算法中出现了一些累除累乘，因此必须考虑精度的问题，图 3.14 显示了在不同网格精度下的计算精度，计算精度的方法如式 3.32 所示。

$$error = \max\left(\frac{(\mathbf{Ku})_i - \mathbf{g}_i}{\mathbf{g}_i}, i = 1,2\cdots\dim(\mathbf{K})\right) \tag{3.32}$$

从图 3.15 中可以看出，该并行算法的精度要略低于一般的直接解法，但是即使在 512×512 的网格下，该算法的相对误差依然小于10^{-9}，在绝大多数的应用场景中，包括本文所描述的平衡反演应用中，都是完全可以忽略的。

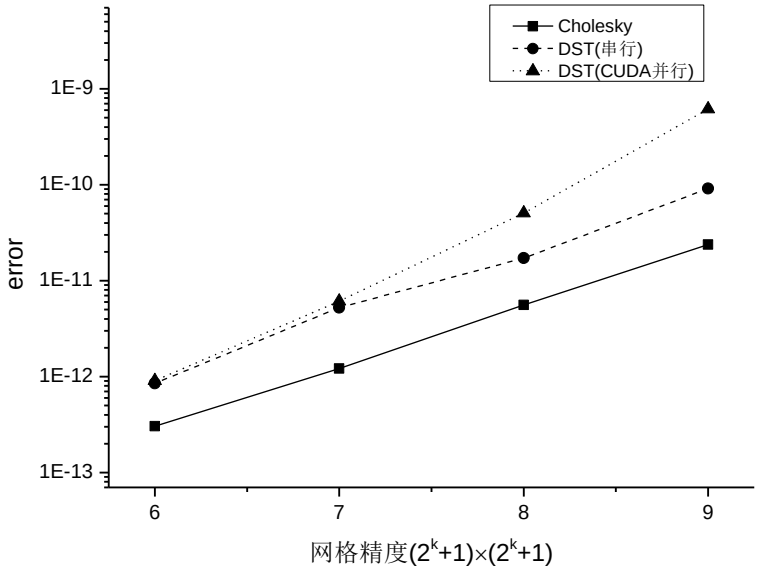

图 3. 15 不同网格精度下的计算精度对比

第 4 章 实验测试

为了验证 P-EFIT 的正确性以及用于实际放电控制的可行性，必须进行一系列的测试。在本章中，首先是介绍了用以验证 P-EFIT 算法正确性的静态测试。接下来介绍了验证一次循环进行放电控制模拟的可行性。最后模拟真实实验环境进行了测试。本节所有的测试中，P-EFIT 都是采用的 65×65 格点。

4.1 静态测试

用于验证 P-EFIT 正确性的标准平衡时一个由 EFIT 所制造出来的平衡，这一平衡是理想的，理论上是没有误差的。因此很适合用来进行这一测试。

实验平台配置为 Nvidia Tesla c2050 GPU，Intel Xeon® E3-1200 CPU，核数为 4 核 8 线程，主频 3.2GHz 操作系统为 Ubuntu 10.10，CUDA tookit 版本为 5.0。

首先，我们根据 EFIT 所制造的理想平衡的电流分布，求解出相应诊断点上的诊断值，这一诊断值理论上也是零误差的。然后用这一诊断值作为 P-EFIT 的输入，之后，P-EFIT 按照 Picard 迭代方法进行迭代重建，得到的平衡重建结果如图 4.1(c)中的红色线条所示，与 EFIT 所制造的原始平衡，即 4.1(c)中的黑色线条相比较，最大的位形误差也小于 1mm，4.1(a)中表示的是随着 Picard 迭代的进行，收敛误差的变化，其中收敛误差的定义如式 3.24 所示：

$$\frac{\max|(\psi^{N}-\psi^{N-1})|}{|\psi_a-\psi_b|} \tag{3.33}$$

其中的ψ^{N}表示的是第 N 次循环的格点极向磁通值，ψ_a 和ψ_b分别表示磁轴处的磁通以及边界处的磁通。可以看出，随着 Picard 迭代的进行，收敛误差迅速减小，在 10 次迭代之后，收敛误差已经开始低于10^{-4}，收敛速度与 EFIT 接近。图 4.1(b)表示的是这一平衡的最小二乘拟合残差，可以看出最大残差也只有10^{-3}，表明这一平衡重建的质量很高。

在实际放电中，作为平衡重建依据的诊断信号不可能是零误差的，因此必须考虑在加入误差以后 P-EFIT 的表现。我们在上面所描述的理想诊断信号上加入如图 4.2 所示的相对误差，然后对其进行反演，得到的反演结果如图 4.3 所示，图中的红色线条为 P-EFIT 结果，而黑色线条是原来的理想平衡。可以看出，这一误差的引入尚未对反演结果产生十分严重的影响，从图 4.4 可以看出反演依然能够在十次左右达到收敛。

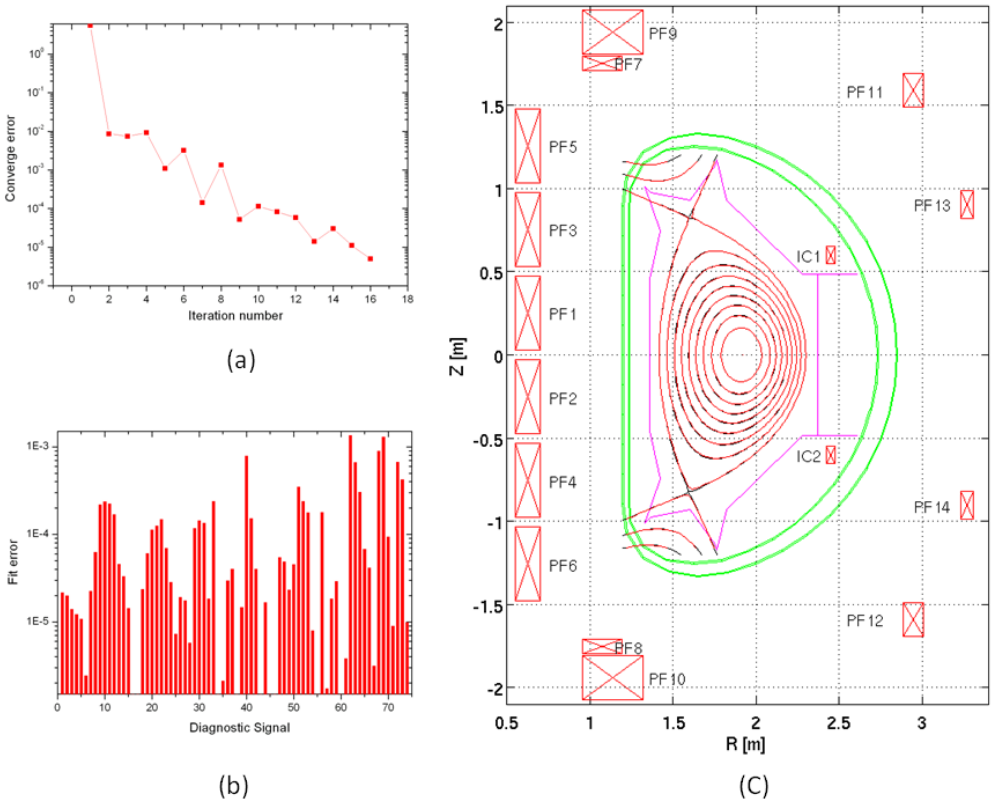

图 4.1 零误差平衡重建测试

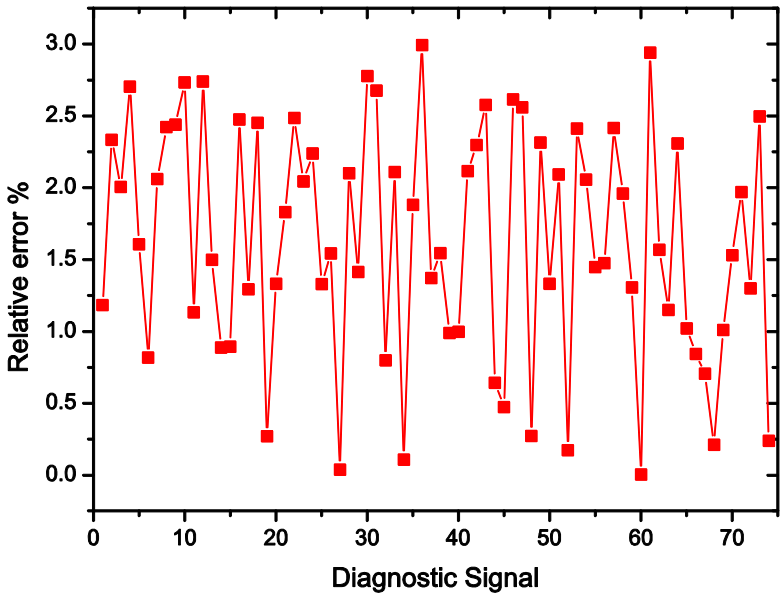

图 4.2 加入诊断信号中的相对误差示意图

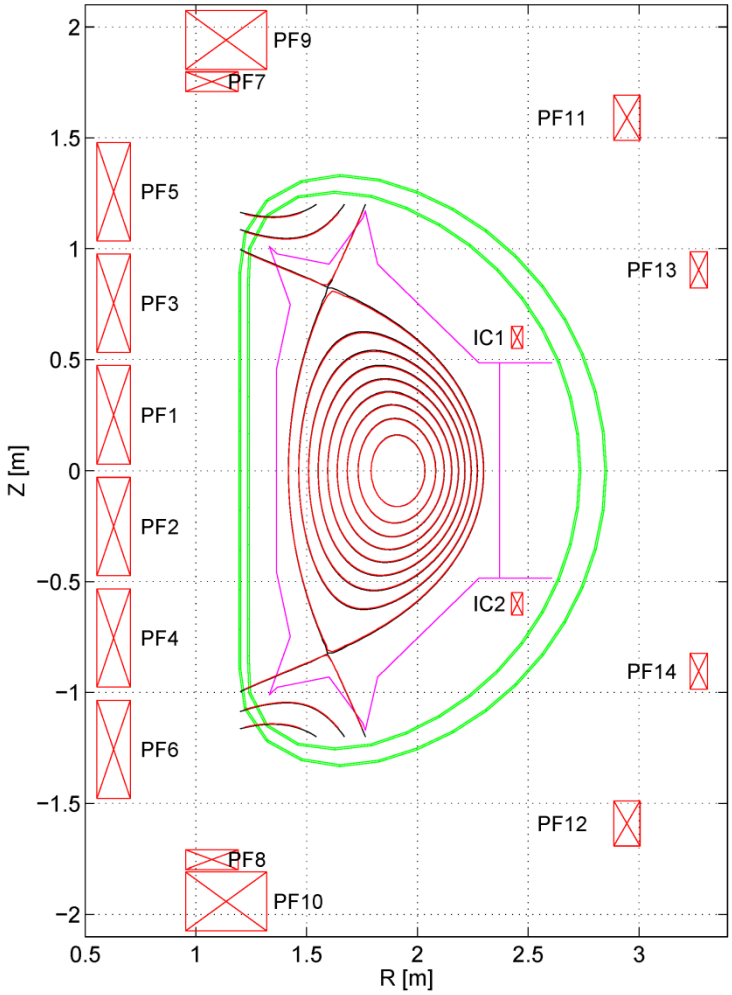

图 4.3 引入 3%误差之后的反演结果与标准平衡对比

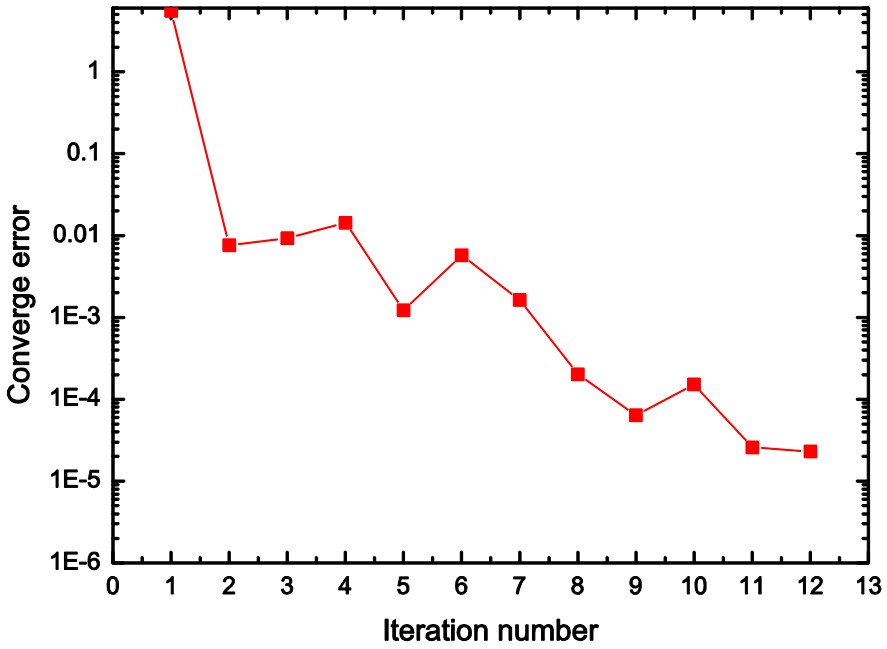

图 4.4 引入 3%误差之后的收敛误差示意图

但是，如果进一步将相对误差增加至 5%，则如图 4.5 所示，位形出现了较大变形，在外中平面上有大约 1cm 的误差，而在下偏滤器区域却出现了巨大的误差。因此，可以看出诊断数据中的误差在小于一点范围时，对 P-EFIT 的计算结果没有太大影响，但是如果误差大于一定范围时，反演的结果会出现很大的偏差，因此，在实验中必须严格控制诊断信号的质量，使其相对误差保持在 3%以下。

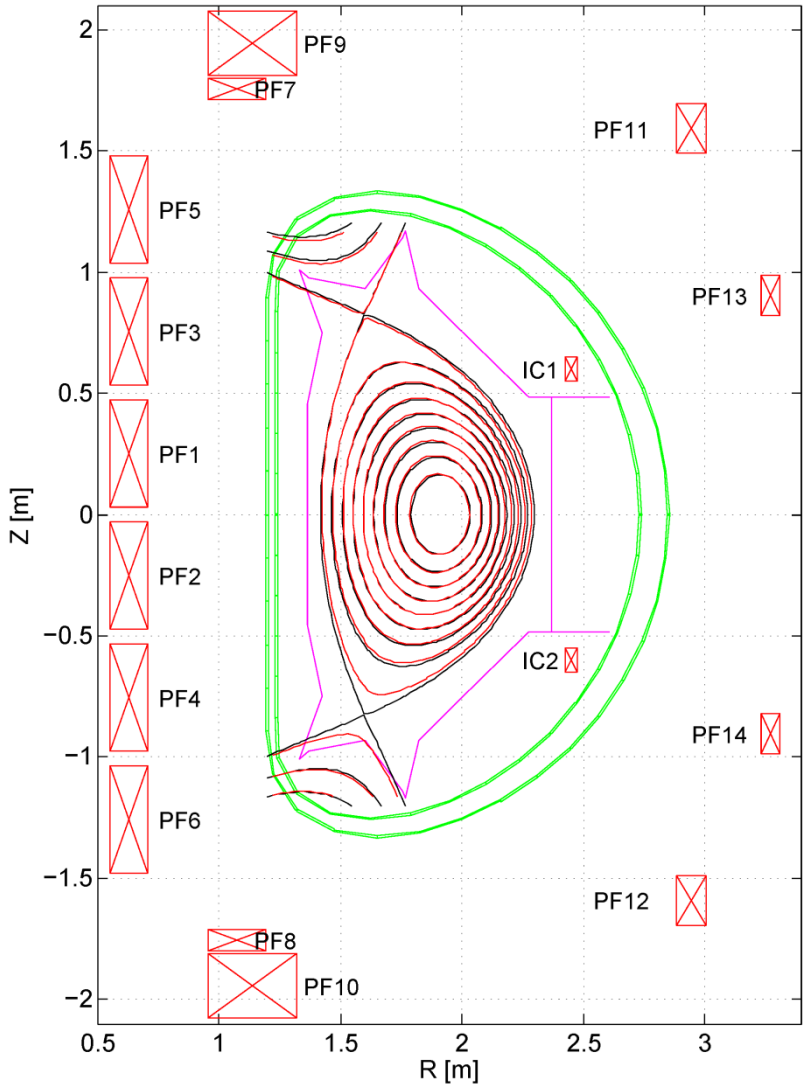

图 4.5 引入 5%误差之后的反演结果与标准平衡对比

速度方面，P-EFIT 完成一次迭代需要大约 220μs，完成一次收敛的平衡重建需要大约 2ms 左右，速度远快于 EFIT。

4.2 一次迭代模式可行性验证

从上一节的实验测试结果可以看出，在相对误差小于 3%时，P-EFIT 的反演结果是足够精确的，但是，从收敛误差的示意图可以看出，要完全收敛，需要的迭代次数在十次左右，总时间需要大约 2ms，对于实时控制而言，这一速度依然是不够的。而 RT-EFIT 中，对于类似情况的处理方法是只迭代一次或者两次就将结果输出，同时下一个时间片的计算以上一次计算的结果做为初始平衡。这样做在一定程度上是可行的，因为在平衡重建中，如果初始平衡和目标平衡位形相差不大时，一次或者两次迭代就可以得到一个近似收敛的结果。但

是如果这两者相差很大，则一次迭代的结果可能与真实结果相差也很大。

为了验证 P-EFIT 采取这种方式时的正确性，必须进行相应的模拟测试，而且在模拟测试中要让位形存在快速的明显变化。

测试的方法是先准备一个序列的诊断信号，这些诊断信号从 0s 开始，以 250μs的时间间隔加入时间信息。在反演开始时计时，一次迭代结束后，查看当前时间，选取时间与之最接近的诊断信号作为依据继续循环。

实验中所使用的诊断信号序列来自于 TSC[35]模拟的结果。获得的方法是用 TSC 演化一次 EAST 典型的放电，然后记录各个时间片内诊断点位置处的理论诊断值。这样获得的诊断信号序列理论上是零误差的。而且有零误差的平衡，即 TSC 的输出作为重建结果的参考。

图 4.6 显示的是这次试验所采用的 TSC 模拟炮 2.5 秒到 3.5 秒的等离子体电流信息，可以看出这一段中电流变化有一定变化，但变化不是特别剧烈。因此在真空室壁上的涡流不会很大，由于 P-EFIT 暂时没有对涡流进行反演，因此并不适用于涡流特别大的情形。

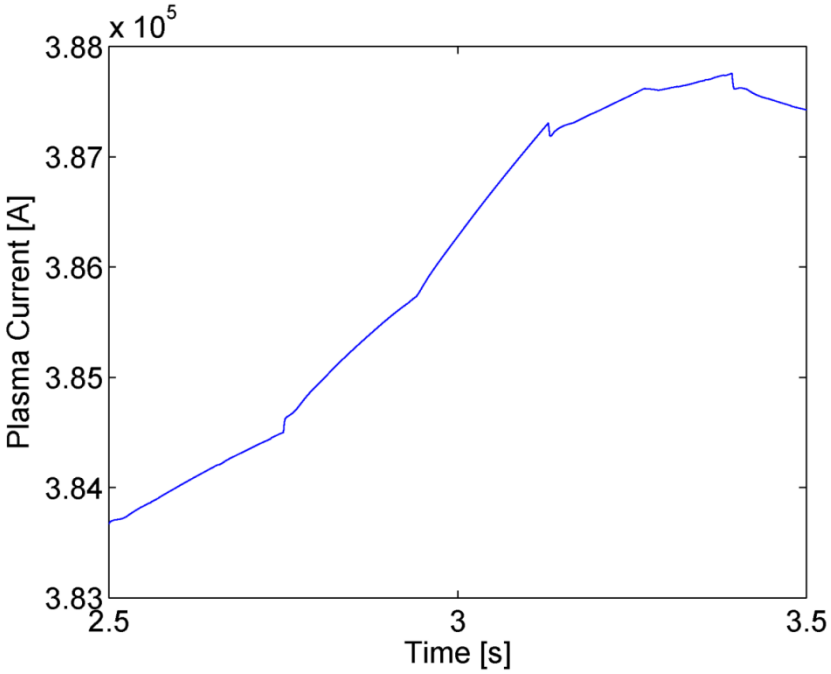

图 4.6 实验中所采用的 TSC 模拟炮的电流信息

我们所选取的用于对比的时间段是 2.5 秒到 3.5 秒，在这段时间内，等离子体正在由双零位形转化为下单零位形，如图 4.7 显示了这一时间段内三个时间片的边界位形，可以看出这一段时间内的位形存在着较大的变化。

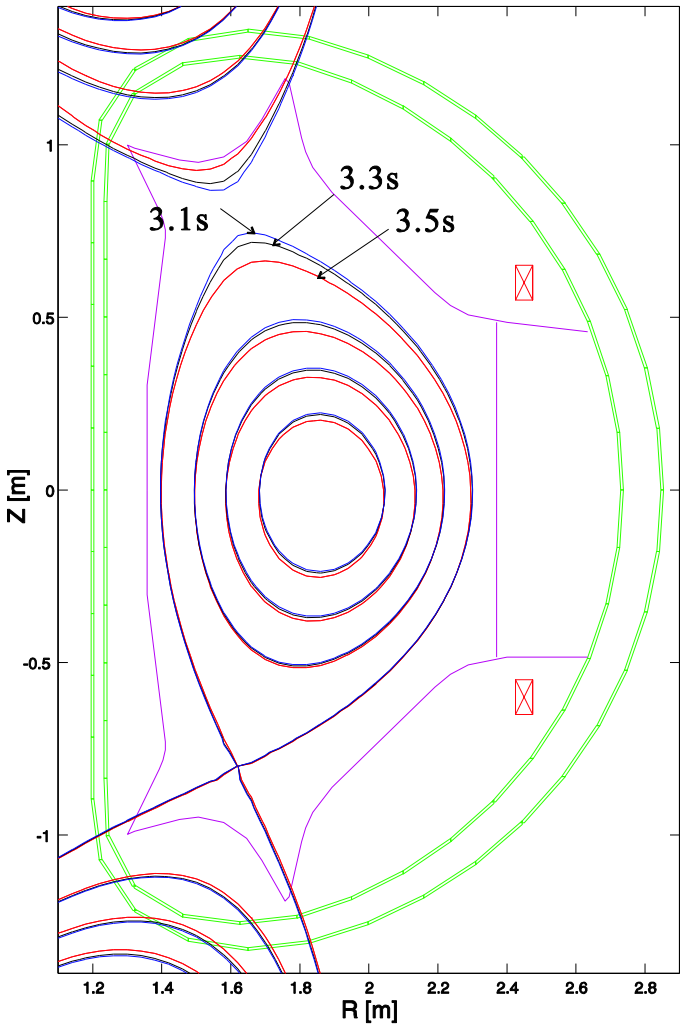

图 4. 7 TSC 模拟炮中的三个时间片信息

由于 TSC 给出的数据理论上是零误差的，为了模拟实际放电的情况，我们在其诊断数据中加入了 3%的随机误差。

测试的平台与静态测试略有不同，这次的实验是在曙光的天阔 I620-G15 GPU 上进行的，GPU 依然是采用 tesla c2050，但 CPU 换为了 Intel XeonE5-2609，4 核 4 线程，主频 2.4GHz，操作系统为 Fedora 16，CUDA tookit 版本依然是 5.0。

图 4.8 展示了主要位形参数的测试结果，图中从上到下的三幅图分别是大半径、小半径、下 X 点横坐标以及下 X 点纵坐标的对比。其中红色线条是 TSC 的理想输出，而蓝色的结果是 P-EFIT 的输出。P-EFIT 在这一秒钟内产生了 3985 个输出，平均大约每 250 微秒产生一个输出，与静态测试时相比，速度慢了 30 个微秒，这主要是由于测试平台 CPU 的不同所致。

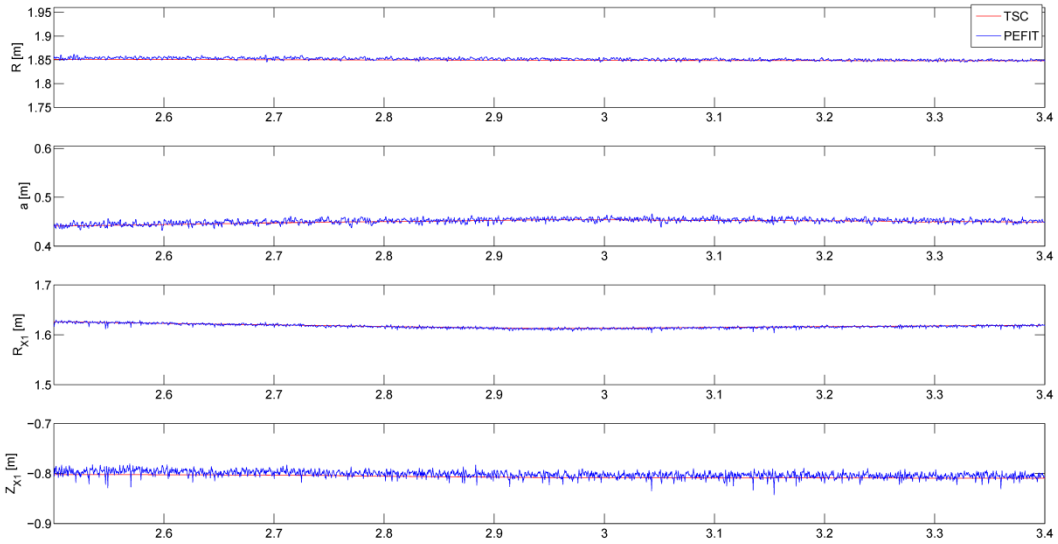

图 4.8 加入 3%随机误差后主要位形参数的测试结果

从图 4.8 中可以看出，P-EFIT 计算的位形参数中大小半径，以及下 X 点的横坐标与 TSC 精确结果相比差别并不大，最大误差也只有 5mm 左右。存在较大误差的是下 X 点的纵坐标，在某些时间片，瞬间的位置误差达到了 1cm 以上，这个主要是由于目前 P-EFIT 所采用的 X 点找寻算法还不是特别完善。如 3.1.2 节所描述的那样，目前算法中展开点处的磁场以及磁场梯度均是用有限差分方法计算的，而纵向上格点之间的距离为 3.75cm，而横向上格点之间的距离仅为 2.1875cm，因此，纵向上有限差分带来的误差便远大于横向。解决的方法是用 3.1.2 节中提到的动态调用格林函数矩阵直接求解格点磁场值的方法。这一方法实现的难度并不大，可以作为下一步的首要工作来完成。

尽管 X 点纵坐标存在一定误差，但这一误差也是对于位形控制而言也是很小的，总的来讲，P-EFIT 采用一次迭代模式反演出的位形参数精度是足够高的，图 4.9 中展示了两个位形变化剧烈时刻 P-EFIT 所反演出的位形（红色线条）与 TSC 所输出的精确位形（黑色线条）之间的对比，可以看出，P-EFIT 的结果与精确结果吻合的很好，说明 P-EFIT 采用这种一次迭代模式来进行实时位形控制是完全可行的。数据结构如

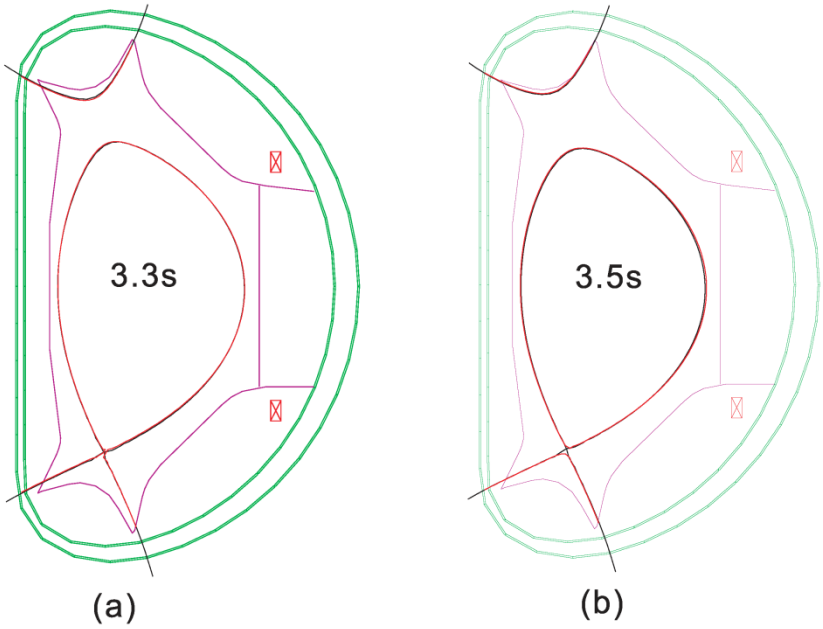

图 4.9 两个时刻的位形比较

4.3 实验环境下的仿真测试

为了能够利用 P-EFIT 在实验中进行位形控制，必须将 P-EFIT 与目前在 EAST 装置上使用的 PCS 系统结合起来[36]，在 EAST 实验中中目前采用的通信方式主要有两种，包括 Socket 通信以及通过反射内存卡实现的共享内存通信。

P-EFIT 采用的通信策略为：通过控制网与 EAST 总控之间实现 Socket 通信，通过反射内存卡与 PCS 系统相连实现共享内存通信。在放电准备阶段共享内存区域的数据结构如图 4.11 所示。其中的 flag 是标志位，sig_setup 数组存储的是诊断信号的设置信息，target shape number 位置存储的是目标位形的时间片数量，接下来存储的是控制线段和 X 点区域的数量。target shape times 数组存储的是目标位形变化的时间点，control point location 数组存储各个控制点在各个控制时刻的坐标。

在 P-EFIT 服务器收到总控发来的放电准备信号之后，便进入放电准备阶段，开始读取共享内存区域的第一个值，即标志位，当标志位变为 1 时，表明 PCS 已经将所有的初始化信息写入了共享内存，P-EFIT 便会将这些值读入，然后完成初始化。

初始化的主要内容包括根据读入的控制点坐标信息，利用 3.1.2 节中描述的方法，生成在 GPU 上进行四点插值所需要的参数信息。同时，将控制算法规定

的关键时间点读入。

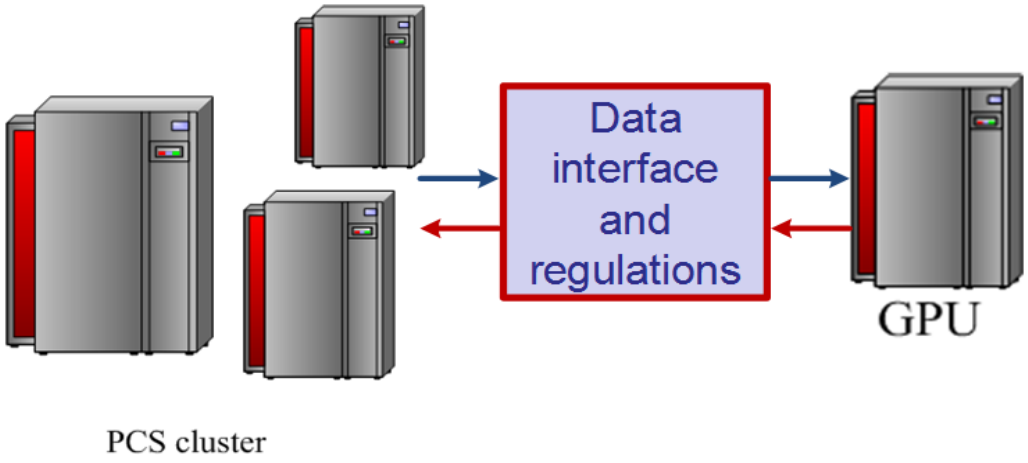

图 4.10 P-EFIT 与 PCS 之间的通信架构

在初始化完成之后，P-EFIT 便进入放电阶段，开始读另外一段共享内存，这段共享内存主要包括一个当前时间信息以及反演需要的磁测量诊断信号。同时 P-EFIT 会实时的将 Isoflux 算法[37]需要的信息实时的写入另一段共享内存中，这段内存的数据结构为：各个控制线段上的控制点处极向磁通误差、各个控制点处的极向磁通梯度、X 点处的极向磁通误差、X 点位置误差、X 点处的 B_r 和 B_z 值。这些数据会实时的被 PCS 读取。然后将这些数据传递给 Isoflux 算法进行位形控制。

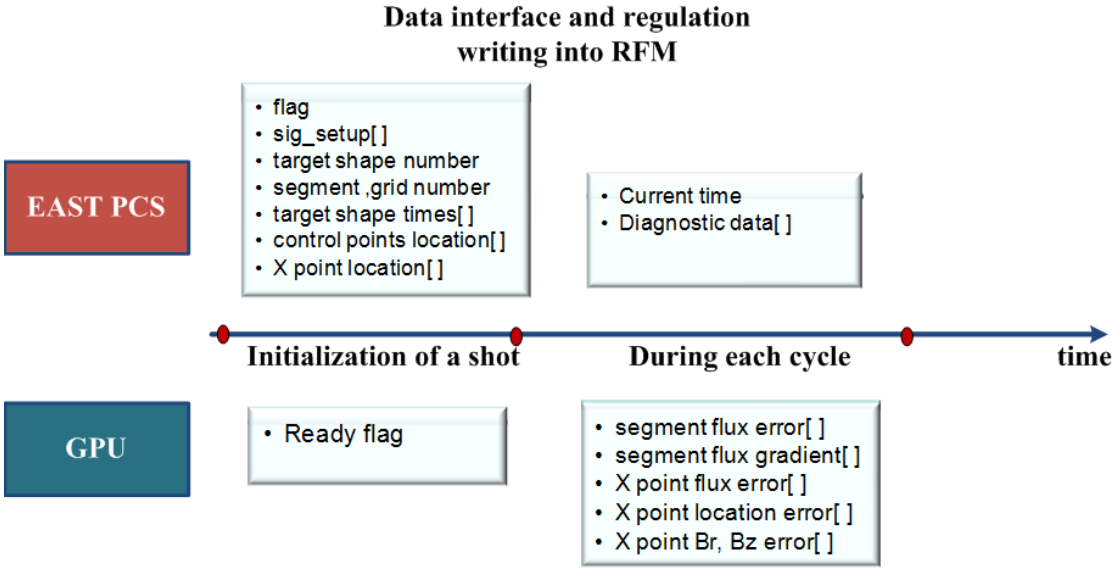

图 4.11 P-EFIT 与 PCS 的通信模式

读写反射内存卡是比较耗时的操作，P-EFIT 放电过程中每次迭代所需的反射内存要耗时 50μs左右，对平衡反演的速度有着严重的影响。

为了减小这一影响，P-EFIT 采取了如下措施，P-EFIT 中定义了两个线程，

并将两个线程利用 linux 系统的硬亲和性设置分别绑定到 CPU0 和 CPU1 上运行，其中一个线程专门负责读写反射内存卡，在完成对反射内存卡的一次读写之后，会将数据写入到 exchange buffer 中去，同时将 exchange buffer 中需要输出的数据取出。而另外一个线程则专门负责控制 GPU 的并行计算，在 P-EFIT 完成一次迭代之后，这一线程会负责将计算结果写入 exchange buffer 的对应区域。两个线程读写 exchange buffer 的操作通过一个互斥锁来保证原子性。采用这一操作后，反射内存卡的读写对 P-EFIT 运行速度的影响变得很小，P-EFIT 一次迭代的运行时间仅仅增加了 10μs左右。

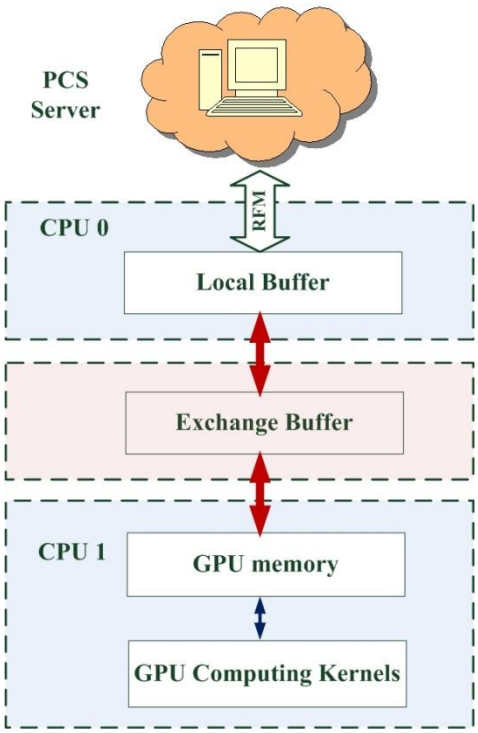

图 4. 12 P-EFIT 读写反射内存卡策略示意图

对于一些非实时需求的重要数据，例如格点磁通值等，P-EFIT 会将它们暂时存储在 GPU 的 global memory 中，这些数据的数据量是很大的。如果 P-EFIT 每 1ms 记录一次这些计算结果，则只需要大概 50 秒就会将 Tesla c2050（3GB global memory）的内存填满。因此为了满足 EAST 长脉冲放电的需求，必须设计在放电过程中实时将数据传输至 Host memory 的方法。

尽管 Fermi GPU 有两个数据传输引擎，但是，如果真的利用其中一个进行如此大数据量的传输，其对 PCIe 总线的占用会严重影响程序的性能。所以目前所设想的办法是利用两台 GPU 服务器交替运行，每台运行大约 40 秒左右，运行结束后将计算数据传回 Host memory。PCS 系统会根据当前时间决定取哪台服务期上的数据。

由于 PCIe×16 总线的带宽有 8GB/s，而我们的数据生产速度只有 60MB/s，所以这样的策略是可行的。

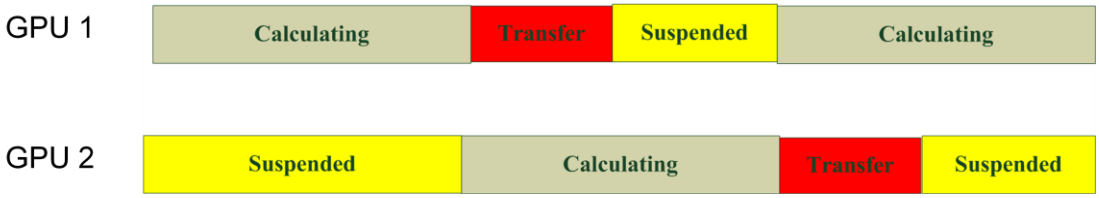

图 4.13 长脉冲下的数据传输示意图

由于目前的 EAST 还处于整修阶段，因此只能利用上轮实验中保存的实验数据进行仿真测试。我们选取的是 43362 炮。

仿真测试的方法为 4.2 节中所描述的一次迭代模式，唯一的不同是数据的来源，该测试的放点设置以及实时诊断信息是由 PCS 系统按照前面所描述的方法实时写入的。测试所用的 GPU 服务器配置同 4.2 节也一致。测试得到的 P-EFIT 位形反演结果如图 4.14 中的蓝色线条所示，其中从上到下一次是大半径，小半径，下 X 点 R 坐标，下 X 点 Z 坐标以及上 X 点 R 坐标和上 X 点 Z 坐标。

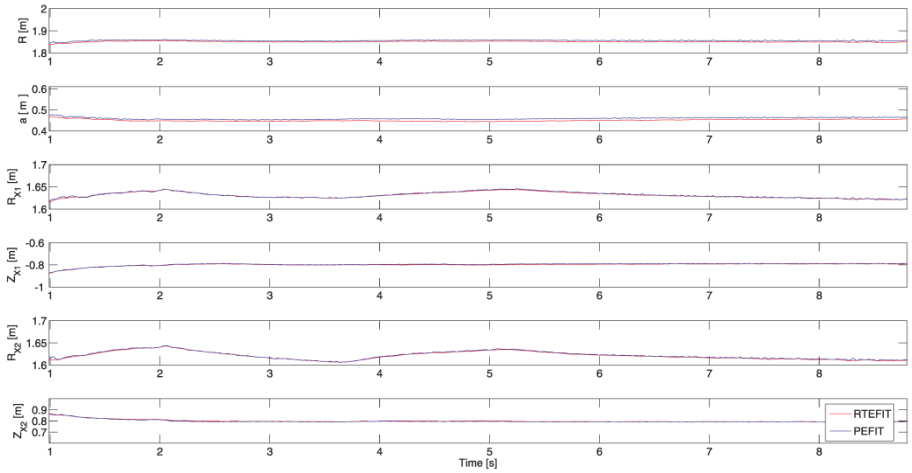

图 4.14 P-EFIT 仿真测试结果与 EFIT 结果的对比

从图中可以看出，P-EFIT 的反演结果与 EFIT 相比十分接近。图 4.15 中展示了 P-EFIT 和 RT-EFIT 的下 X 点 R 坐标输出对比，其中的数据是以散点形式表示的，可以直观的看出 P-EFIT 的速度优势，在这 0.5s 的时间内，采用 65×65 网格的 P-EFIT 共输出了 1985 组数据，而采用 33×33 的 RT-EFIT 仅输出了 94 组数据。

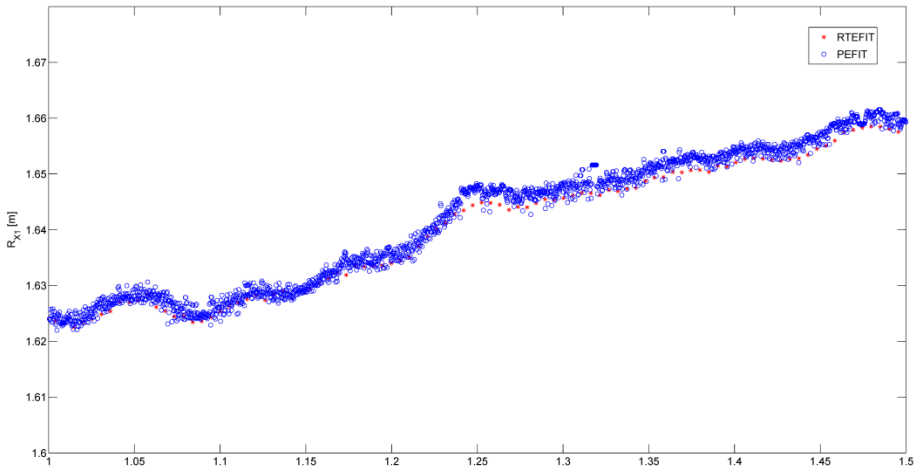

图 4. 15 P-EFIT 与 RT-EFIT 输出结果对比

图 4.16 显示的是 P-EFIT 和 RT-EFIT 所计算出的 segment error 值的对比，可以看出，两者的计算结果接近，震荡趋势几乎完全一致，这说明了 P-EFIT 系统的延迟与 RT-EFIT 系统没有明显区别

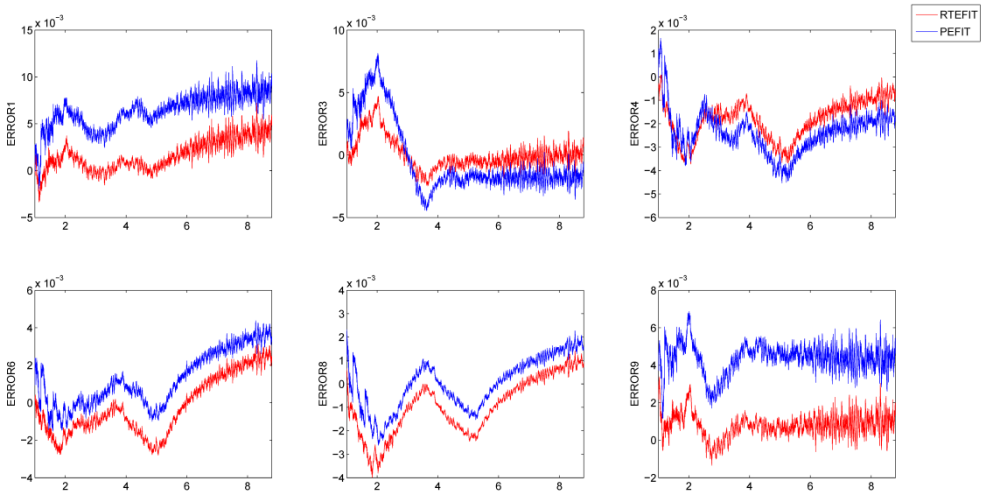

图 4. 16 P-EFIT 和 RT-EFIT 输出的 segment error 对比

从图 4.16 可以看出，两个算法输出的 segment 1 和 segment 9 上存在着较大的误差，为了解释这一原因，我们抽出了 4s 时刻两者所输出的平衡结果，如图 4.17 所示，其中黑色的等磁通线是 RT-EFIT 的计算结果，而红色等磁通线是 P-EFIT 的计算结果。加粗的黑色直线显示的是 segment 的设置，可以看出，segment 1 和 segment 9 所在的位置两种算法所计算出的磁面位置存在较大差别，以 segment 1 为例，两者在这一位置的差别大约为 1.3cm，而沿着这个 segment

的极向磁通梯度大约为 4×10^{-3}/cm，因而可以估算出两者在这一时刻所计算出的 segment error 会存在大约 5.2×10^{-3} 的区别，与图 4.16 中所展示出的结果吻合。这说明了 P-EFIT 可以正确的计算出相应的 segment error，也就意味着 P-EFIT 可以提供给 Iso-flux 正确的参数，将位形按照 P-EFIT 的反演结果进行控制。从而说明了 P-EFIT 具备了应用于实时位形控制的条件。

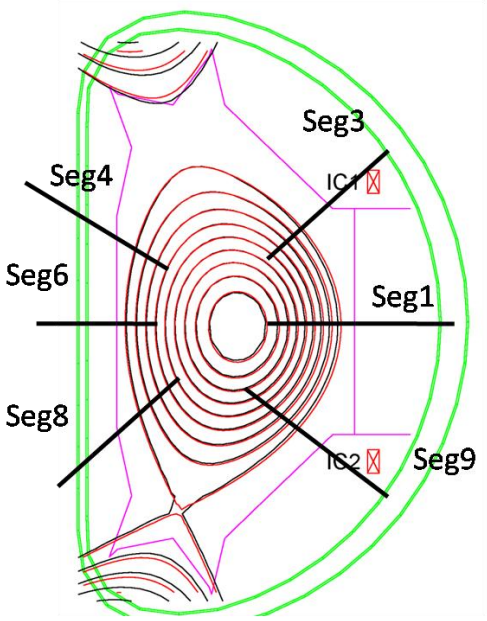

图 4. 17 43362 炮 4s 平衡计算结果对比及 segment 设置

第 5 章 总结与展望

本文描述了一种 GPU 的并行等离子体反演程序 P-EFIT，现总结如下：

P-EFIT 的算法基于 EFIT 反演算法，通过求解等离子体区域的 Grad-Shafranov 方程来获得等离子体的电流剖面和边界信息。

1、开发了适用于 GPU 的高效电流格点筛选程序。开发了寻找边界磁通的快速算法。

2、针对在 P-EFIT 中遇到的特定维度矩阵相乘运算，结合 GPU 的硬件架构开发了高效的并行矩阵相乘算法。

3、验证了采用 norm equation + pivoting L-U 方法进行最小二乘求解的可行性，并利用 CPU 和 GPU 协同运算获得了很好的效果。

4、通过 DST 算法和修改的前缀和算法实现了基于 GPU 的块三对角方程快速求解算法。

5、开展了静态测试，实验结果表明在磁测量诊断值误差小于一定程度的情况下 P-EFIT 可以获得很好的收敛结果。

6、通过一次迭代测试验证了 P-EFIT 采用一次迭代模式进行位形控制的可行性。

7、通过实验环境下的测试，验证与 PCS 系统整合的可行性。

P-EFIT 目前还有许多需要完成的工作，主要包括以下方面：

1、程序算法中依然存在一些不足需要修正，例如 X 点 Z 坐标的误差问题，需要用格林函数法取代目前算法中采用的线性插值法。

2、程序中尚需要添加许多必要的模块，许多重要的物理量，例如 β_P，l_i 和 q_0 的计算，根据现在已经得到的信息，这些量很容易求得。

3、要完全取代 RT-EFIT 必须由 P-EFIT 计算输出实时显示系统需要的等离子体的边界几何坐标。当前的 P-EFIT 中，有输出边界与横向水平线交点的功能，但是在某些位形，特别是圆位形下，会在上下顶端出现最大 3.75cm 的偏差，因此，下一步必须完善这一算法，一种思路是把边界与纵向水平线的交点也输出。

4、目前已经验证了 P-EFIT 在 129×129 格点下的部分性能，可以推断在这一格点设置下会获得更高的加速比，下一步要完善 129×129 甚至更高格点密度的反演功能[38]。

5、目前的 P-EFIT 代码可读性以及可移植性较差，下一步将改善这些方面。

参考文献

[1] 卢希庭. 原子核物理[M]. 北京市: 原子能出版社, 2000.

[2] 朱士尧. 核聚变原理[M]. 合肥: 中国科学技术大学出版社, 1992.

[3] Wesson J. Tokamaks[M]. USA: Oxford University Press, 2011.

[4] 王淦昌, 袁之尚. 惯性约束核聚变[M]. 北京市: 原子能出版社, 2005.

[5] 中国国际核聚变能源计划执行中心. 中国聚变研究[EB/OL]. http://www.iterchina.cn/Research/Introduction.html.

[6] Pironti A, Walker M. Control of tokamak plasmas: introduction to a special section[J]. Control Systems, IEEE, 2005,25(5):24-29.

[7] Beghi A, Cenedese A. Advances in real-time plasma boundary reconstruction - From gaps to snakes[J]. Ieee Control Systems Magazine, 2005,25(5):44-64.

[8] ALLADIO F, CRISANTI F. ANALYSIS OF MHD EQUILIBRIA BY TOROIDAL MULTIPOLAR EXPANSIONS[J]. NUCLEAR FUSION, 1986,26(9):1143-1164.

[9] Sartori F, Cenedese A, Milani F. JET real-time object-oriented code for plasma boundary reconstruction[J]. FUSION ENGINEERING AND DESIGN, 2003,66-68:735-739.

[10] 罗正平. EAST等离子体平衡重建可靠性与快速性研究[D]. 合肥: 中国科学院合肥物质科学研究院, 2011.

[11] SWAIN D W, NEILSON G H. AN EFFICIENT TECHNIQUE FOR MAGNETIC ANALYSIS OF NON-CIRCULAR, HIGH-BETA TOKAMAK EQUILIBRIA[J]. NUCLEAR FUSION, 1982,22(8):1015-1030.

[12] Luxon J L, Brown B B. Magnetic analysis of non-circular cross-section tokamaks[J]. Nuclear Fusion, 1982,22(6):813.

[13] Lao L L, Ferron J R, Groebner R J, et al. Equilibrium analysis of current profiles in tokamaks[J]. Nuclear Fusion, 1990,30(6):1035.

[14] Lao L L, John H S, Stambaugh R D, et al. Reconstruction of current profile parameters and plasma shapes in tokamaks[J]. Nuclear fusion, 1985,25(11):1611.

[15] Ferron J R, Walker M L, Lag L L, et al. Real time equilibrium reconstruction for tokamak discharge control[J]. Nuclear Fusion, 1998,38(7):1055-1066.

[16] Kwak J G, Oh Y K, Kim K P, et al. Key Features in the Operation of KSTAR[J]. Ieee Transactions on Plasma Science, 2012,40(3):697-704.

[17] Gates D A, Ferron J R, Bell M, et al. Plasma shape control on the National Spherical Torus

Experiment (NSTX) using real-time equilibrium reconstruction[J]. Nuclear Fusion, 2006,46(1):17-23.

[18] Pangione L, McArdle G, Storrs J. New magnetic real time shape control for MAST[J]. Fusion Engineering and Design, (0).

[19] Xiao B J, Yuan Q P, Humphreys D A, et al. Recent plasma control progress on EAST[J]. Fusion Engineering and Design, 2012,87(12):1887-1890.

[20] Luo Z P, Xiao B J, Zhu Y F, et al. Online Plasma Shape Reconstruction for EAST Tokamak[J]. Plasma Science & Technology, 2010,12(4):412-415.

[21] Rampp M, Preuss R, Fischer R, et al. A Parallel Grad-Shafranov Solver for Real-Time Control of Tokamak Plasmas[J]. Fusion Science and Technology, 2012,62(3):409-418.

[22] nVIDIA. nVIDIA CUDA C Programming guide v. 4.0.[M/OL].

[23] nVIDIA. CUDA C Programming - Best Practices Guide v. 4.0[M/OL].

[24] Buneman O. A Compact Non-iterative Poisson Solver[M]. Calif.: Institute for Plasma Research, Stanford University, 1969.

[25] Buzbee B L, Golub G H, Nielson C W. Direct Methods for Solving Poissons Equations[J]. Siam Journal on Numerical Analysis, 1970,7(4):627.

[26] Hockney R W. A Fast Direct Solution of Poissons Equation Using Fourier Analysis[J]. Journal of the Acm, 1965,12(1):95.

[27] Swarztrauber P N. The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle[J]. Siam Review, 1977,19(3):490-501.

[28] Rust B W. Truncating the singular value decomposition for ill-posed problems[M]. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 1998.

[29] Tikhonov A N, Leonov A S. Ill-posed Problems in Natural Sciences: Proceedings of the International Conference Held in Moscow, August 19-25, 1991[M]. Vsp, 1992.

[30] Bosak K. Real-time numerical identification of plasma in tokamak fusion reactor[D]. Master’s thesis University of Wroclaw, Poland URL http://panoramix. ift. uni. wroc. pl/∼ bosy/mgr/mgr. pdf, 2001.

[31] Harris M. Optimizing parallel reduction in CUDA[J]. NVIDIA Developer Technology, 2007,6.

[32] David Kincaid Ward Cheney. 王国荣 俞耀明. 数值分析[M]. 北京市: 机械工业出版社, 2005.

[33] Chan S C, Ho K L. Direct methods for computing discrete sinusoidal transforms, 1990[C].

[34] Hillis W D, Steele G L. Data Parallel Algorithms[J]. Communications of the Acm, 1986,29(12):1170-1183.

[35] Jardin S C, Pomphrey N, Delucia J. Dynamic modeling of transport and positional control of tokamaks[J]. Journal of computational Physics, 1986,66(2):481-507.

[36] Yuan Q P, Xiao B J, Luo Z P, et al. Plasma current, position and shape feedback control on EAST[J]. Nuclear Fusion, 2013,53(4):43009.

[37] Hofmann F, Jardin S C. Plasma Shape and Position Control in Highly Elongated Tokamaks[J]. Nuclear Fusion, 1990,30(10):2013-2022.

[38] Ren Q, Chu M S, Lao L L, et al. High spatial resolution equilibrium reconstruction[J]. Plasma Physics and Controlled Fusion, 2011,53(9).

致谢

时光荏苒，光阴飞逝，三年的研究生生活马上就要结束了，同时我的学生时代也将告一段落。回首过去，需要感谢的人实在是太多太多。

首先要感谢我的导师肖炳甲研究员，感谢您一直以来对我的指导以及生活上的关心和照顾，您严谨的治学精神，渊博的学术知识，雷厉风行的行事风格是我这三年里获益良多，这些将是我最宝贵的精神财富，谢谢您肖老师。

感谢 7 室罗正平博士三年来对我的指导和帮助，感谢您耐心的传授我 EFIT 算法的知识，帮助我查找程序中的问题。您认真、努力的工作态度是我学习的榜样。

感谢青年科学基金(11205191)的支持。

感谢袁旗平老师、杨飞老师、孙有文老师、郭勇博士、黄耀同学、裴晓芳同学以及已经毕业的蒋坤师兄对我论文工作的帮助。感谢邢哲、刘磊、陈树亮、刘广君、章勇、张祖超等师兄对我的帮助。感谢同一级的颜涛、陈颖、杨万彩、夏金瑶、许光俊、陈大龙和黄文君同学，感谢你们的陪伴和帮助。感谢 7 室其他所有的老师和同学。

感谢中国科学技术大学核科学技术学院的王炜老师和邱友凤老师，感谢你们一直以来对我的照顾。

感谢我的本科同学朱康，读研期间我们是最好的朋友，也一起经历了跨专业找工作，那段一起学习，互相鼓励的日子我永远不会忘记。感谢王伟宁同学，你那积极乐观的精神给我带来了很多的正能量。感谢我的舍友鲁忠涛同学，是你给了我跨专业找工作的勇气。

感谢我的女友，感谢你一直以来对我的支持和理解。

最后要感谢的是我的父母和姐姐，感谢你们一直以来对我无私的帮助和无条件的支持，谢谢你们。

在读期间发表的学术论文与取得的其他研究成果

待发表论文：

[1] Xiaoning Yue, Bingjia Xiao, Zhengping Luo Yong Guo, “Fast Plasma Equilibrium Reconstruction based on GPU parallel computing”, *Plasma Physics and Controlled Fusion* (录用)

[2] **岳小宁**，肖炳甲，罗正平；“基于CUDA的二维泊松方程快速直接求解”，*计算机科学*（录用）

会议论文：

[1] Q.P. Yuan, X.N. Yue, X.F. Pei, Z.P. Luo, R.R. Zhang, B.J. Xiao, “The implementation of real time parallel equilibrium reconstruction in EAST PCS”, 9th IAEA Technical Meeting on Control, Data Acquisition, and Remote Participation for Fusion Research, May, 2013.