

密级: _____



中国科学院大学
University of Chinese Academy of Sciences

博士学位论文

强流束流动力学研究及其模拟软件开发

作者姓名: _____ (作者姓名)

指导教师: _____ (导师姓名)

中国科学院高能物理研究所

学位类别: _____ 理学博士

学科专业: _____ 粒子物理与原子核物理

培养单位: _____ 中国科学院高能物理研究所

2018 年 04 月

Study on High Current Beam Dynamics
and the Simulation Code Development

by
(Author Name)

A thesis submitted to
The University of Chinese Academy of Sciences
in partial fulfillment of the requirements
for the degree of
Doctor of Nuclear and Particle Physics

Institute of High Energy Physics, Chinese Academy of Sciences

April, 2018

学位论文独创性声明

本人郑重声明：我所呈交的学位论文是本人在导师指导下进行的研究工作及所取得的研究成果。尽我所知，除了文中已经标注引用的内容外，本论文中不含其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中作了明确的说明或致谢。本人知道本声明的法律结果由自己承担。

作者签名：_____ 日期：_____

关于学位论文使用授权的说明

本人完全了解中国科学院高能物理研究所有关保留、使用学位论文的规定，即：中国科学院高能物理研究所有权保留送交论文的复印件，允许论文被查阅和借阅；可以公布论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存论文。

(保密的论文在解密后应遵守此规定)

作者签名：_____ 导师签名：_____ 日期：_____

摘要

在现代粒子加速器中，强流束流动力学的一个主要问题是空间电荷效应。空间电荷效应会引起束流和粒子的共振，导致束流的不稳定，其显著结果就是产生束晕和束损。一般来说，我们有两种方法来研究空间电荷效应，一种方法从非自治的模型出发，以给出粗略的物理图像；另一种方法从数值模拟出发，消耗更大的计算量给出更为精确的结果。空间电荷效应问题是一个多体耦合问题，这个问题没有解析解。为了更加深入地了解这个问题背后的机制，一个可以进行精确数值模拟的程序是必不可少的。

束流模拟软件在加速器研究和设计中起着非常重要的作用。随着加速器中束流流强的日益提高，模拟所需要的粒子数目提升了几个量级，模拟程序所需要的功能也越来越多。基于此，作者在 TOPO 的基础上进行了程序并行化处理和功能扩充，使用 C++ 语言开发了基于 PIC 算法的束流模拟程序 P-TOPO。我们对于程序的核心算法，PIC 算法，在 GPU 上和 CPU 集群上都进行了并行化实现，以提高模拟软件的效率。除了 PIC 算法之外，我们还引入了一种新型算法，无网格 Symplectic 算法，用来求解空间电荷效应。这种算法以更大的计算量为代价，显著降低了经典 PIC 算法中网格热噪声带来的发射度增长。无网格 Symplectic 算法很适合并行，有很好的可扩展性。我们也对 Symplectic 算法在 GPU 上进行了并行化实现，使其运行速度得到显著提升。

在程序开发和性能测试之后，我们使用 P-TOPO 进行了 C-ADS 注入器 I 的模拟和一些空间电荷相关的物理问题研究。首先，我们使用 P-TOPO 对 C-ADS 注入器 I 进行了完整的模拟，并对比了 P-TOPO 模拟结果和其他模拟程序的结果。这个模拟一方面验证了 C-ADS 注入器 I 设计的合理性，另一方面也验证了 P-TOPO 的正确性。我们还使用 Symplectic 算法对周期聚焦结构中由空间电荷效应导致的三阶共振现象进行了模拟和研究。此外，我们还使用 P-TOPO 研究了加速器中的束流如何自发地受到相干结构共振的影响，并对加速器中的共振区穿越问题进行了探索与研究。研究表明束流周期相移在穿越共振禁带时会出现振荡，而在向上和向下穿越过程中出现的“吸引”和“排斥”现象是束流对于共振的自发反应。研究还表明束流发射度增长与其在共振禁带内停留的时间正相关。

关键词： 空间电荷效应， 并行模拟软件， PIC， Symplectic， 共振穿越

Abstract

One of the main problems in high current beam dynamics is the space charge effect, which will lead to the resonance and instability of beam and particles. A direct result from resonance is beam halo and beam loss. In general, there are two approaches to the model the space charge effect. One approach starts from a non-self-consistent model which gives a rough physical picture. Another is from the numerical simulation which is more time-consuming but much more precise. Since the space charge effect is a multi-body coupling problem, we cannot get the analytical solution. In order to have a more sophisticated understanding of the mechanism behind the problem, a software which can carry on the accurate numerical simulation is necessary.

Beam simulation code plays an indispensable role in accelerator design and study. As the beam current increases, the number of particles to be simulated increases by several orders of magnitude and more features are required. Following this requirement and based on code TOPO and many other works, we developed a beam simulation code P-TOPO based on PIC method using C++ language. The code was implemented with paralleled PIC algorithm on both the GPU and the CPU cluster to improve the efficiency. Besides the PIC method, we also introduced a new space charge solver based on gridless Symplectic algorithm. It can effectively reduce the emittance growth associated with numerical grid heating compared with traditional PIC algorithm at the expense of lower computation efficiency. The Symplectic algorithm is very suitable for parallelism and can achieve very good speedup and scalability. We implemented it on GPU using CUDA library and achieved a significant speedup.

After the code development and performance test, we performed the simulation of C-ADS Injector I with P-TOPO and studied several space charge related beam dynamics problems. Firstly, a start-to-end simulation of C-ADS injector I was performed, and the results from P-TOPO and those from other simulation codes were compared. On one hand the correctness of P-TOPO is verified, and on the other hand it is considered as the verification of code by an existing tuning machine. Secondly, we simulated and studied of the single particle third-order resonance due to space charge effect in the periodic focusing structure using the Symplectic algorithm. Thirdly, we studied how the beam is spontaneously affected by the coherent structure resonances using P-TOPO and explored

the problem of the resonance stop band crossing. It shows that the beam phase advance will oscillate when crossing the resonance stop band. Related “attraction” and “repulsion” effects in the case of upward and downward crossing is a natural beam coherent reaction to the resonance. It also shows that the beam emittance growth is positively correlated with the time it spends in the resonance stop band.

Keywords: Space charge effect, Parallel simulation code, PIC, Symplectic, Resonance crossing

目 录

摘要	vii
Abstract	ix
目录	xi
图形列表	xv
表格列表	xix
符号列表	xxi
第一章 引言	1
1.1 课题研究的背景和意义	1
1.2 课题研究的现状	4
1.3 本论文主要工作	5
1.4 论文主要创新之处	6
第二章 强流束流动力学	7
2.1 基本理论	7
2.2 束流横向运动	7
2.3 束流纵向运动	10
2.4 空间电荷效应	12
2.5 国内外强流加速器简介	14
第三章 束流模拟程序的物理模型和算法	17
3.1 束核模型	17
3.2 束流模拟原理及PIC算法	21
3.2.1 权重插值	22
3.2.2 使用FFT解泊松方程	24
3.2.3 蛙跳法推动粒子	27
3.3 Symplectic算法	30
3.4 小结	33

第四章 束流模拟程序的设计及并行实现	35
4.1 P-TOPO程序	35
4.1.1 程序结构	36
4.1.2 正确性校验	37
4.2 PIC算法在GPU上的实现	39
4.2.1 粒子排序	40
4.2.2 权重插值	45
4.2.3 泊松方程	47
4.2.4 粒子推动	49
4.2.5 正确性校验	49
4.3 PIC算法在CPU集群上的实现	50
4.4 Symplectic算法在GPU上的实现	51
4.4.1 遍历三角函数	51
4.4.2 计算 Φ^{lmn}	52
4.4.3 计算 ep_i 并推动粒子	53
4.5 小结	54
第五章 束流模拟程序的优化和性能测试	55
5.1 PIC程序性能	55
5.1.1 泊松方程求解	55
5.1.2 单GPU性能-GTX1060	59
5.1.3 GPU集群性能-Titan	63
5.1.4 GPU集群性能-SummitDev	65
5.1.5 CPU集群性能-KNL	66
5.2 Symplectic算法性能	71
5.2.1 单GPU性能提升-GTX1060	71
5.2.2 多GPU性能提升-Titan	73
5.3 小结	75

第六章 束流动力学相关问题研究和模拟	77
6.1 C-ADS注入器I模拟	77
6.1.1 C-ADS简介	77
6.1.2 RFQ模拟	80
6.1.3 超导段模拟	81
6.1.4 总结	82
6.2 三阶共振模拟	86
6.3 共振穿越研究	87
6.3.1 物理模型	88
6.3.2 结构共振穿越 – 相干效应	90
6.3.3 单粒子动力学 – 非相干效应	95
6.3.4 总结	97
6.4 小结	97
第七章 论文总结与课题研究展望	99
附录 A 粒子模拟程序用户界面	101
附录 B 二阶集体模的雅克比矩阵	103
参考文献	105

图形列表

1.1 GPU与CPU的结构对比示意图	4
2.1 四极磁铁中磁场及粒子受力示意图	8
2.2 加速腔电场、势阱、相空间轨迹和相稳定区示意图	11
2.3 考虑阻尼项后的纵向相空间稳定区示意图	12
2.4 高流强高功率质子同步加速器汇总	15
3.1 束核包络与单粒子横向位移示意图	18
3.2 粒子的庞加莱截面示意图	19
3.3 不同相位压缩因子下的庞加莱截面	20
3.4 PIC算法块循环示意图	21
3.5 PIC算法分段流程示意图	21
3.6 权重插值中的分配方式示意图	23
3.7 权重插值中的形状因子曲线	25
3.8 使用FFT求解泊松方程流程图	26
4.1 P-TOPO程序结构示意图	37
4.2 同一均方根尺寸下的不同种类束团分布	38
4.3 P-TOPO与理论值的点电荷电势对比	39
4.4 零流强和15mA时，P-TOPO结果与包络方程的对比	39
4.5 权重插值中的线程冲突	41
4.6 网格分块示意图	42
4.7 GPU上的PIC算法分段流程图	42
4.8 粒子排序示意图	44
4.9 权重插值示意图	45
4.10 PIC算法在GPU上的正确性校验	50
5.1 单GPU域分解模式解泊松方程各部分耗时占比	56
5.2 $64 \times 64 \times 64$ 格点下，跨节点多GPU域分解模式解泊松方程时间随GPU个数的变化	58

5.3	$128 \times 128 \times 128$ 格点下, 跨节点多GPU域分解模式解泊松方程时间随GPU个数的变化	59
5.4	PIC程序在单GPU上的加速比	60
5.5	程序各部分耗时在不同粒子数时所占百分比	62
5.6	不同的格点数情况下单GPU求解泊松方程的加速比	63
5.7	$64 \times 64 \times 64$ 个格点, 160k个粒子时, Titan上PIC程序耗时随GPU个数的变化	64
5.8	$64 \times 64 \times 64$ 个格点, 160k个粒子时, Titan上域分解模式PIC程序耗时随GPU个数的变化	64
5.9	$64 \times 64 \times 64$ 个格点, 1.6M个粒子时, Titan上PIC程序耗时随GPU个数的变化	65
5.10	$64 \times 64 \times 64$ 个格点, 16M个粒子时, Titan上PIC程序耗时随GPU个数的变化	65
5.11	$64 \times 64 \times 64$ 个格点, 160k个粒子时, SummitDev上PIC程序耗时随GPU个数的变化	66
5.12	$64 \times 64 \times 64$ 个格点, 1.6M个粒子时, SummitDev上PIC程序耗时随GPU个数的变化	67
5.13	$64 \times 64 \times 64$ 个格点, 16M个粒子时, SummitDev上PIC程序耗时随GPU个数的变化	67
5.14	1.6M粒子数下, 单节点下不同混合并行配置的耗时与内存占用	68
5.15	160k粒子数下, 单节点下不同混合并行配置的耗时与内存占用	69
5.16	16M粒子数下, 单节点下不同混合并行配置的耗时与内存占用	69
5.17	PIC程序使用多个CPU节点的耗时	70
5.18	使用64个节点时程序各个部分消耗时间所占的百分比	70
5.19	不同并行配置下的多节点运行情况比较	71
5.20	Symplectic算法的单GPU加速比随着分解阶数和粒子数目的变化	72
5.21	Symplectic算法的多GPU加速比随着分解阶数和粒子数目的变化	74
6.1	C-ADS加速器整体布局示意图	78
6.2	C-ADS注入器I结构示意图	79
6.3	0mA (上) 和15mA (下) 的C-ADS注入器I的RFQ中的横向发射度变化	81

6.4 0mA (上) 和15mA (下) 的C-ADS注入器I的RFQ中的纵向发射度变化.....	83
6.5 C-ADS注入器I的RFQ中束团横向均方根尺寸变化	83
6.6 C-ADS注入器I的RFQ中束团纵向均方根尺寸和能散变化	84
6.7 C-ADS注入器I的超导段横向发射度和纵向发射度变化	84
6.8 C-ADS注入器I的超导段束团横纵向尺寸以及能散变化	85
6.9 不同流强下的发射度增长.....	86
6.10 三阶共振附近的庞加莱截面	87
6.11 二阶偶数模和奇数模的本征值 $Abs[\lambda_{j;k,l}]$ 和本征相位 $\Phi_{j;k,l}$ 随考虑空间电荷效应后的相移 σ 的变化.....	91
6.12 从下方穿越共振区（左）与从上方穿越共振区（右）的周期相移比较.....	93
6.13 从下方穿越共振区（左）与从上方穿越共振区（右）的发射度增长率(ϵ_f/ϵ_i)比较	93
6.14 从下方穿越共振区（上）与从上方穿越共振区（下）的穿越过程中的粒子相空间形状($x - p_x$)	94
6.15 a): 400个FODO周期后的最终发射度增长率与用于共振跨越控制的 FODO周期数的关系; b): 400个FODO周期后的最终发射度增长率与束流在结构共振禁带中的有效周期数的关系。	95
6.16 使用粒子束核模型得到的单粒子Poincaré截面	96
6.17 相空间分布上的两个共振岛结构示意图	96
A.1 Impact用户界面	102

表格列表

1.1 加速器束流模拟程序现状调研	5
2.1 国内外强流质子直线加速器参数表	15
2.2 国内外强流质子环形加速器参数表	16
5.1 单GPU域分解模式解泊松方程耗时（秒）	56
5.2 单节点双GPU域分解模式解泊松方程耗时（秒）	57
5.3 $64 \times 64 \times 64$ 格点下，跨节点多GPU 域分解模式解泊松方程各部分所用时间	58
5.4 $128 \times 128 \times 128$ 格点下，跨节点多GPU域分解模式解泊松方程各部分所用时间	59
5.5 PIC程序在单GPU上的加速比	61
6.1 C-ADS加速器主要参数	78
6.2 C-ADS注入器I基本参数	79

符号列表

Abbreviations

Acronym	Description
LHC	Large Hadron Collider
RHIC	Relativistic Heavy Ion Collider
BEPCII	Upgrade of Beijing Electron–Positron Collider
ALS	Advanced Light Source
ADS	Accelerator Driven Sub-critical system
CSNS	China Spallation Neutron Source
PIC	Particle-in-cell
P-TOPO	Parallel Trace-Of-Particle-Orbit
IBS	Intra-Beam Scattering
NGP	Nearest Grid Point
CIC	Cloud In Cell
TSC	Triangular Shaped Cloud
h.o.t.	Higher Order Term
FD	Focusing Defocusing
FODO	Focusing Drift Defocusing Drift
GPU	Graphics Processing Unit
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
FFT	Fast Fourier Transformation
BLAS	Basic Linear Algebra Subprograms
IHEP	Institute of High Energy Physics
LBNL	Lawrence Berkeley National Laboratory
GSI	GSI Helmholtz Centre for Heavy Ion Research
CERN	European Organization for Nuclear Research
ECRIS	Electron Cyclotron Resonance Ion-Source
RFQ	Radio-Frequency Quadrupole

第一章 引言

1.1 课题研究的背景和意义

粒子加速器是一种可以使带电粒子在高真空中受磁场力控制、电场力加速而达到高能量的装置。作为为基础科学的研究提供条件的工具，粒子加速器在理解各类原子核结构、原子核内部的相互作用，以及研究天体物理、宇宙学、生命科学、材料科学、核医学等方面都有着重要的意义。粒子加速器最开始的用途是高能物理研究，目标是研究粒子的微观结构和发现新的物理现象。高能物理和粒子物理研究中所用的粒子对撞机就是一种加速器，比如欧洲核子中心（CERN）的大型强子对撞机（Large Hadron Collider, LHC）、日本高能加速器研究机构（KEK）的SuperKEKB、美国布鲁克海文国家实验室（BNL）的（Relativistic Heavy Ion Collider, RHIC）、中国的北京正负电子对撞机（Upgrade of Beijing Electron–Positron Collider, BEPCII）等等。

除了在高能物理方面，加速器在其他方面也有着许多应用。粒子加速器可以用作同步辐射光源，比如美国先进光源（Advanced Light Source, ALS）、上海光源、以及北京在建的新光源等。此类的光源装置在物理、材料、化学、生物、医学等方面发挥着越来越重要的作用；加速器还在医疗方面有很多应用，比如质子加速器是目前最有效的治疗癌症的装置；除此之外，加速器还在新能源研究中起到重要作用，比如加速器驱动的次临界洁净核能系统（Accelerator Driven Sub-critical System, ADS）是目前产生核能、增殖核材料和嬗变核废物的解决方案之一。

ADS被认为是进行反应堆核废料嬗变的最佳技术方案之一，可以将铀-238转化为更易裂变的钚-239，或者开发和利用钍资源，使其能够充分利用可裂变的核资源。除此之外，ADS还可以嬗变长寿命核废料为短寿命核废料，降低放射性废物的储量和毒性；而ADS本身在产能过程中，基本不产生核废料，可以认为是一种洁净的核能。另外，ADS是一个次临界系统，需要不断从外界获取中子来维持反应，可以从根本上杜绝事故，其安全性得到保障。在ADS中，加速器的作用是将质子加速到高能，并通过高能质子束打靶产生中子，从而驱动次临界反应堆持续产生裂变反应。

粒子加速器的种类很多，它们的特点各有不同，可以按不同的原则加以分类。按照粒子运动的轨道形状分，加速器可以分为环形加速器和直线加速器。按照加速粒子的种类划分，可以分为电子加速器，离子加速器。本论文的主要研究范围是强流离子加速器。

因为加速器中的大部分实验事例与加速器束流流强正相关，所以强流加速器成为了未来的加速器科学重要的发展方向之一。强流加速器在核物理、生命科学、材料科学等基础科学研究，以及核医学、放射医学等应用研究方面有着越来越重要的作用。在最近二十年来，得益于高能物理、核物理、散裂中子源、加速器驱动的次临界核能系统以及其它应用领域的强烈需求，强流粒子加速器也得到了较大的发展[1, 2]。而随着加速器朝着高流强的方向发展，我们也面临着新的问题和挑战。

强流束流动力学主要研究的是加速器中的大量带电粒子在自生库伦场和外界电磁场中的演化问题。强流离子加速器的一个特点是束流粒子受空间电荷效应的影响很大，导致束流的集体不稳定性较为突出，从而导致束流的整体损失或束流品质变坏。空间电荷效应是指束团内部粒子之间的库仑作用，其作用的来源为带电粒子本身。束团中的粒子带相同的电荷，相互排斥，会受到其他带电粒子的斥力，导致带电束团的自然发散和粒子的相位改变。而集体不稳定的来源有两个，一个是空间电荷效应，一个是加速器的外部环境，比如束团与弯转磁铁，聚焦磁铁，加速腔等的相互作用。空间电荷效应和束流的集体不稳定性都可能导致束流的品质变差或者产生束损。

在加速器中研究空间电荷效应的目的主要是提升束流品质，控制束损。空间电荷效应和束流流强，束流能量，以及束团的大小有关。在现有聚焦强度下，流强越强，空间电荷效应带来的问题越严重，特别实在束流能量低流强大的时候，空间电荷效应非常明显。如果不加以控制，束流损失会很大。随着束流流强和能量的提高，即使很小的束流损失也会使机器维护无法有效进行、实验条件无法得到保障，所以束流损失率的控制也必须更加严格。国际上流行采用平均损失率为 1 W/m 的手动维护标准，这个标准最早是针对 1 GeV 量级的质子束流能量，但对较低的束流能量也基本上适合。譬如，对于能量为 1 GeV 、平均流强为 1 mA 的质子束，允许的束流损失率为 1 nA/m 或 10^{-4} %/m 。这个要求非常严格，需要从物理设计和技术措施两方面上保证实现。

强流束流动力学研究的主要问题，也是难点问题，是如何自洽的表述束流自身产生的非线性空间电荷效应。空间电荷效应导致的束流和粒子的共振和不稳定性、不同自由度之间的发射度交换、束晕粒子的产生、束流丢失等问题成为了强流束流物理中关注的焦点。如何定性，定量理解强流束流动力学中的由于非线性空间电荷效应导致的束流集体不稳定性、束晕粒子形成机制也成为了近年来加速器物理研究中的重要课题。这些问题背后的物理机制可以从物理模型和数值模拟两方面进行了研究。从物理模型方面，非自洽的空间电荷模型可以被用来对其进

行分析，以得到粗略的物理图像。然而，空间电荷效应是一个多体的耦合问题，无法得到解析解。为了更加精确地探寻其背后的物理机制，数值模拟是必不可少的。而为了获得更为精确地物理图像，粒子模拟程序的运行效率必须进行提高。

为了进一步提升模拟程序的运行效率，我们考虑对其进行并行化处理，并比较了不同的并行架构。CPU的英文全称为Central Processing Unit，中文为“中央处理器”，一般由逻辑运算单元、控制单元和存储单元组成。GPU的英文全称为Graphic Processing Unit，中文为“图形处理器”。GPU一开始是由于在现代的计算机中图形的处理变得越来越重要，需要一个专门的图形核心处理器，使显卡减少对CPU的依赖，并进行部分原本CPU的工作；近年来，GPU高速发展，极大的提高了计算机图形处理的速度和质量，不但促进了图像处理、虚拟现实、计算机仿真等相关应用领域的快速发展，同时也为人们利用GPU进行图形处理以外的通用计算提供了良好的运行平台。

如图(1.1)所示，一个CPU一般只拥有少数几个核，而一个GPU会有几百甚至几千个核。相对于CPU，GPU在并行处理和计算密集型问题方面具有很大优势。目前，GPU已成为普通计算机强大、高效的计算资源。从系统架构上看，GPU针对向量计算进行了高度并行的数据流优化处理，这种多核架构特别适合进行数据并行。这种以数据流作为处理单元的机制，在对数据流的处理上可以取得很高的速度。GPU 加速计算是指同时利用GPU 和 CPU，加快应用程序的运行速度[3]。GPU厂商为科学计算设计了自己的API，其中CUDA（Compute Unified Device Architecture）就是NVIDIA设计的并行计算平台和编程规范[4]。利用CUDA，我们能够有效的利用GPU，使程序的运行效率大大提高。

根据使用CUDA的测试结果显示，利用GPU求解FFT（Fast Fourier Transformation）、BLAS（Basic Linear Algebra Subprograms）、排序及线性方程组等科学计算问题，与单纯依靠CPU实现的算法相比，平均性能提高了近20倍。

随着GPU在可编程能力、并行处理能力和应用范围等方面得到不断提升和扩展，GPU已成为当前计算机系统中性能很高的部件。因此，我们对模拟程序中的核心算法，比如PIC算法和Symplectic算法，都进行了GPU化实现。这项工作充分利用了现有计算资源，发挥GPU的高性能计算能力，使程序在GPU与CPU之间进行协作，以提高模拟程序的运行效率。

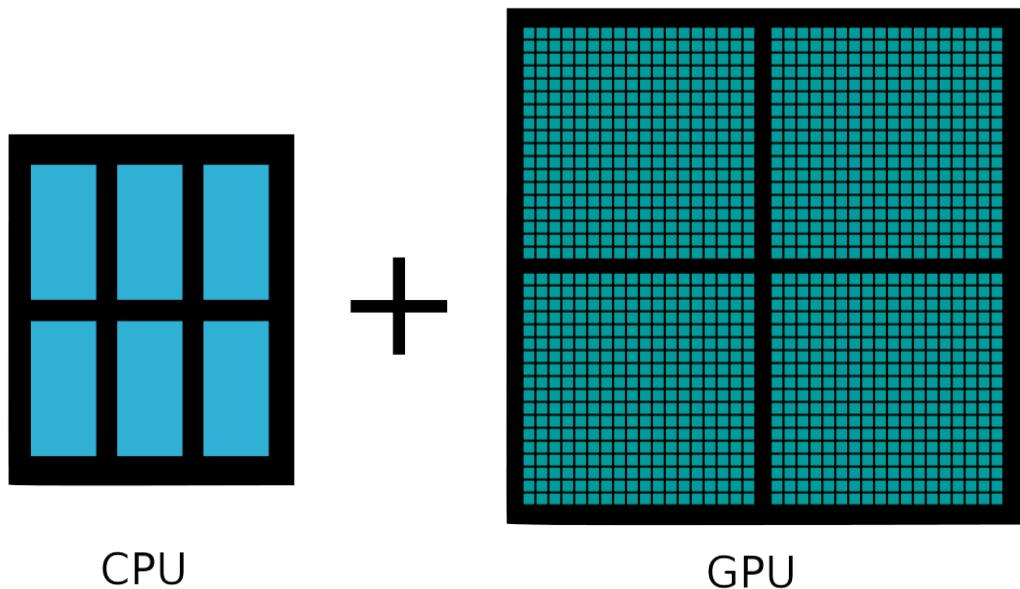


图 1.1: GPU与CPU的结构对比示意图

1.2 课题研究的现状

加速器中的束流是一个多体非线性耦合系统。由于多体非线性耦合系统的复杂性，强流束流物理问题的研究可以从解析近似和数值模拟两个角度出发。其中解析方法基本是在哈密顿力学的框架内加上一些必要的近似以建立起物理模型，比如束流包络模型，单粒子运动模型等，再使用这些模型进行分析。目前国际上流行的做法是使用非自治的模型，比如束核模型，对束晕产生的机制、束流集体效应等问题进行分析，给出基本的物理图像。而数值模拟方法主要是使用束流模拟程序在计算机上进行大规模计算，对带电粒子进行跟踪和模拟。模拟程序使用的基本模型有传输矩阵映射，PIC（Particle-in-cell）等。国际上普遍的做法是使用PIC方法求解空间电荷效应，进行粒子跟踪，得到粒子演化的图像，分析其背后的物理机制。

解析近似在某些特定的情况下可以得到较为准确的结果，可以定性的给出一些束流性质的预测和加速器设计的准则。其优点为物理模型较为简单，计算速度快；其缺点为对于更多的情况需要近似，与真实的加速器相差较远，对于一个具体的加速器无法给出具体定量的结论。而数值方法依赖于具体加速器的设计，在加速器建设中起到了必不可少的作用。其优点是可以针对具体的加速器，给出更为准确的结果；缺点是程序编写较为繁琐，运算速度较慢。在加速器研究中，我们一般是先用解析近似得到一个粗略的结果，再使用数值模拟进行验证和更为详细的研究。

表 1.1: 加速器束流模拟程序现状调研

名称	开源	界面	并行	后处理	匹配	速度	精确度	是否收费
Parmila	否	否	否	是	否	慢	低	免费
Impact	是	否	是	否	否	较快	较高	免费
TraceWin	否	是	是	是	是	较快	较高	收费
Dynac	是	否	否	是	否	快	较低	免费
Orbit	否	否	是	否	否	较快	较高	免费
Track	否	否	否	否	否	较快	较高	免费

如表(1.1)所示，目前在国际加速器界存在着一些束流模拟程序 [5–12]。国内加速器界同仁大部分依然使用国外的相关程序，并且对程序内部的计算过程和算法实现并不十分了解。但是，随着实际加速器的流强上升，我们的需求也越来越多，目前存在的程序无法完全满足我们的需求；随着需要模拟规模越来越大，受限于已有的程序的运行效率，模拟所需要的时间也越来越长。而且目前的程序大部分都不是开源，不方便我们根据自己的需求在底层进行改进。国内此前对束流模拟程序的探索和开发较少，目前还没有一套成熟完备的加速器束流模拟程序。目前，我们依托于国内近年来设计或在建ADS，中国散裂中子源（China Spallation Neutron Source, CSNS）等强流加速器项目，对强流束流模拟程序进行了很多研究，而且针对程序的效率提升方面也做出了很多探索，比如MPI, OpenMP等等CPU程序并行化，以及使用GPU并行计算。

总之，解析近似已经不能满足现代加速器的设计和研究需求，一个可以进行精确模拟的束流模拟程序是必须的。随着加速器束流流强的增大，模拟需要更大规模的粒子数目，而面对新的物理问题，程序所需的功能也越来越多。我们需要探索新的算法，编写新的粒子模拟程序，优化程序结构并提高程序运行效率，了解并掌握程序背后的计算过程。

1.3 本论文主要工作

本论文的工作主要分三个方面：首先是根据相关的基本理论，研究强流束流动力学的物理模型和求解空间电荷效应的算法；其次是多粒子追踪模拟程序P-TOPO（Parallel Trace-Of-Particle-Orbit）的开发和并行优化；最后是使用开发的束流模拟程序P-TOPO进行束流动力学研究。

本文第二章将会对强流束流动力学的基本理论进行简要介绍，其中包括束流

的横向运动与纵向运动、空间电荷效应等基本理论，并对国内外的强流加速器进行简要介绍。第三章介绍了束流模拟程序的物理模型，以及一些相关算法和逻辑流程。其中主要介绍了包括束核模型在内的一些物理模型，以及束流模拟程序中常用的PIC算法和一种新型的Symplectic算法。第四章介绍了模拟程序的顶层设计以及算法的实现，包括程序总体的设计、PIC算法在GPU上和CPU集群上的实现、Symplectic保辛算法在单GPU和GPU集群上的实现，并分别对每一种算法实现进行了正确性校验。第五章介绍了程序中的不同算法在各个计算平台上的优化和性能测试。第六章利用模拟程序对空间电荷效应和束流集体效应进行研究。首先对三阶共振现象进行了一个简单模拟；然后对加速器中束流如何自发的在共振禁带中进行演化、以及束流穿越共振禁带进行了讨论；最后对C-ADS注入器I进行模拟研究。第七章对论文工作进行了总结和展望。

1.4 论文主要创新之处

我们对不同的物理模型进行了比较，并开发了束流模拟程序P-TOPO。目前在国内还没有成熟的加速器束流模拟程序，本课题的有关工作将填补国内的空白。而且相对于其他束流模拟程序，P-TOPO在很多地方进行了创新。比如大多数模拟程序使用飞行时间法获得加速腔的相位，而P-TOPO使用类似于实际运行中的扫相方法获得加速腔相位，虽然花费时间要更长，但是结果更加精确。相比于其他模拟程序，我们的并行策略更为精细。我们在不同的子程序中使用了不同的并行策略，更加灵活的分配计算负载，提高了并行的效率。

除此之外，我们还在不同的空间电荷算法的并行化上进行了探索，并且在不同的并行计算架构中实现。据我们所知，Symplectic算法在GPU上的实现和PIC算法在GPU集群上的实现均为加速器界首次，这两个程序实现均取得了可观的加速比。在PIC算法在GPU集群的实现中，我们比较了不同的并行策略下求解泊松方程的效率，并且实现并测试了新的节点间通讯方法；而在Symplectic算法的实现中，我们针对GPU架构进行了大量优化，尤其是针对GPU的内存读写规则，重新组织了程序，使其数据吞吐速度更高。我们还测试了Symplectic算法中每一部分的加速比和瓶颈，为下一步升级做好准备。

最后，我们对束晕产生的机制进行了新的探索。在对C-ADS注入器I的模拟中，验证了其设计的合理性。我们对共振穿越的研究发现束流发射度增长几乎与束流在共振区内所停留的时间正相关，而我们通常理解的非相干共振并不能用来解释我们研究的问题，我们的模拟也证明了之前的理论预测，即低阶禁带被包含在高阶禁带中。

第二章 强流束流动力学

2.1 基本理论

束流动力学的基础是电动力学[13, 14]。一个电荷为 e 的带电粒子，在电场强度为 \vec{E} 、磁感应强度为 \vec{B} 的电磁场中运动，其运动方程为：

$$\frac{d\vec{p}}{dt} = e(\vec{E} + \vec{v} \times \vec{B}), \quad (2.1)$$

其中， $\vec{p} = \gamma m \vec{v}$ 为粒子的动量， \vec{v} 是带电粒子的速度， m 为粒子的固有质量， γ 为洛伦兹因子：

$$\gamma = \frac{1}{\sqrt{1 - \frac{v^2}{c^2}}}, \quad v = \sqrt{\vec{v} \cdot \vec{v}}, \quad (2.2)$$

这里 c 为光速。

电磁场满足麦克斯韦方程组。在真空中，麦克斯韦方程可以表达为：

$$\begin{aligned} \nabla \cdot \vec{E} &= \frac{\rho}{\epsilon_0}, \\ \nabla \cdot \vec{B} &= 0, \\ \nabla \times \vec{E} &= -\frac{\partial \vec{B}}{\partial t}, \\ \nabla \times \vec{B} &= \mu_0 \left(\vec{J} + \epsilon \frac{\partial \vec{E}}{\partial t} \right). \end{aligned} \quad (2.3)$$

其中， ρ 为电荷密度， ϵ_0 为真空介电常数， μ_0 为真空磁导率， \vec{J} 为电流密度。

2.2 束流横向运动

束流的横向运动已经得到了广泛研究[15–17]。我们定义束流横向运动与纵向运动方向的夹角为 x' ：

$$x' = \frac{dx}{dz} = \frac{p_x}{p_z}, \quad (2.4)$$

则

$$F_x = \frac{dp_x}{dt} = \frac{dp_x}{dz} \frac{dz}{dt} = v_z \frac{d(p_z x')}{dz} = v_z (p_z x'' + p'_z x'), \quad (2.5)$$

假设束流没有被加速，即 $p'_z = 0$ ，则粒子的横向运动方程为：

$$x'' = \frac{F_x}{p_z v_z}. \quad (2.6)$$

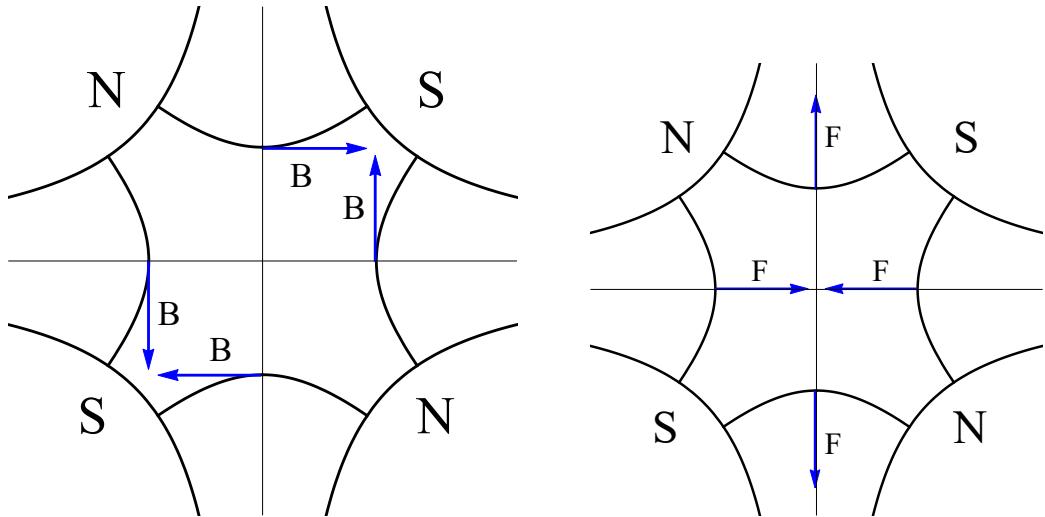


图 2.1: 四极磁铁中磁场及粒子受力示意图

现代加速器中一般使用二极磁铁来对束流进行偏转，使用四极磁铁来对束流进行横向聚焦。四极磁铁的磁场分布和粒子受力如图(2.1)所示[15]，其中带负电荷的粒子垂直于纸面向里运动。可以看出，四极铁产生的力在一个方向上聚焦，而在另一个方向上散焦。粒子所受到的力与粒子的位置成正比。以X方向为例，粒子所受到的力为：

$$F_x = -qv_z B_y = -qv_z g x, \quad g = \frac{\partial B_y}{\partial x}。 \quad (2.7)$$

则由式(2.6)和式(2.7)可得：

$$x'' = -\frac{qg x}{p_z} \approx -\frac{gx}{B\rho}, \quad (2.8)$$

其中 $B\rho$ 为磁刚度，即：

$$B\rho = \frac{mv}{q} \approx \frac{mv_z}{q}。 \quad (2.9)$$

我们定义四极磁铁的聚焦强度 K 为：

$$K \equiv \frac{g}{B\rho} = \frac{q}{mv} \frac{\partial B_y}{\partial x} = \frac{q}{\gamma m_0 \beta c} \frac{\partial B_y}{\partial x}, \quad (2.10)$$

则粒子的横向运动方程可以写为：

$$x'' + Kx = 0。 \quad (2.11)$$

在不同的加速器元件中， K 会发生变化，而且在水平方向和垂直方向可能不同，因此一个更普遍的描述单粒子横向运动的方程可以表示为：

$$\begin{cases} x'' + K_x(s)x = 0, \\ y'' + K_y(s)y = 0. \end{cases} \quad (2.12)$$

上式也被称为希尔方程，以X方向为例，其解可以表示为[18]:

$$x(s) = \begin{cases} a \cos(\sqrt{K}s + b), & K > 0, \\ as + b, & K = 0, \\ a \cosh(\sqrt{-K}s + b), & K < 0. \end{cases} \quad (2.13)$$

当 $K = 0$ 时，表示聚焦强度为0，即粒子在漂移节中运动；当 $K > 0$ 时，表示聚焦作用；当 $K < 0$ 时，表示散焦作用。

由式(2.13)可知， x 和 x' 都是连续的，而积分常数 a 和 b 则取决于初始条件 $x(0)$ 和 $x'(0)$ 。记 $\vec{x}(s) = [x(s), x'(s)]$ 为粒子运动的状态矢量，则方程(2.12)的解可以表示为矩阵形式：

$$\begin{bmatrix} x \\ x' \end{bmatrix} = \mathbf{M} \begin{bmatrix} x_0 \\ x'_0 \end{bmatrix}, \quad (2.14)$$

其中 \mathbf{M} 为传输矩阵：

$$\mathbf{M} = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} \quad (2.15)$$

对于聚焦强度 K 为常数的结构，我们可以得到传输矩阵的一般形式：

$$\mathbf{M} = \begin{cases} \begin{bmatrix} \cos \sqrt{K}L & \frac{1}{\sqrt{K}} \sin \sqrt{K}L \\ -L\sqrt{K} \sin \sqrt{K}L & \cos \sqrt{K}L \end{bmatrix}, & K > 0 : \text{聚焦}, \\ \begin{bmatrix} 1 & L \\ 0 & 1 \end{bmatrix}, & K = 0 : \text{漂移节}, \\ \begin{bmatrix} \cosh \sqrt{|K|}L & \frac{1}{\sqrt{|K|}} \sinh \sqrt{|K|}L \\ L\sqrt{|K|} \sinh \sqrt{|K|}L & \cosh \sqrt{|K|}L \end{bmatrix}, & K < 0 : \text{散焦}, \end{cases} \quad (2.16)$$

其中 L 为初位置和末位置之间的距离。

在薄透镜近似下，即 $L \rightarrow 0$ 情况下，传输矩阵可以简化为：

$$\mathbf{M} = \begin{cases} \begin{bmatrix} 1 & 0 \\ -\frac{1}{f} & 1 \end{bmatrix}, & K > 0 : \text{聚焦}, \\ \begin{bmatrix} 1 & L \\ 0 & 1 \end{bmatrix}, & K = 0 : \text{漂移节}, \\ \begin{bmatrix} 1 & 0 \\ \frac{1}{f} & 1 \end{bmatrix}, & K < 0 : \text{散焦}, \end{cases} \quad (2.17)$$

式中 $f = \lim_{L \rightarrow 0} \frac{1}{\sqrt{K}L}$ 定义为透镜焦距。

传输矩阵更一般的表达为:

$$\mathbf{M} = \begin{bmatrix} \cos \Phi + \alpha \sin \Phi & \beta \sin \Phi \\ -\gamma \sin \Phi & \cos \Phi - \alpha \sin \Phi \end{bmatrix} = \mathbf{I} \cos \Phi + \mathbf{J} \sin \Phi, \quad (2.18)$$

其中 α 、 β 、 γ 就是加速器中常用的Twiss参数，并且 $\beta\gamma = 1 + \alpha^2$ 。 Φ 为相移， \mathbf{I} 为单位矩阵，而 \mathbf{J} 形式如下:

$$\mathbf{J} = \begin{bmatrix} \alpha & \beta \\ -\gamma & -\alpha \end{bmatrix}。 \quad (2.19)$$

2.3 束流纵向运动

加速器一般使用加速腔对带电粒子进行加速[19–21]。带电粒子要被加速，必须满足同步加速条件。符合加速腔设计相位的理想粒子为同步粒子（参考粒子），而与设计相位存在偏差的粒子为非同步粒子。经过一个加速单元后，同步粒子的能量增益 δW_s 和非同步粒子的能量增益 δW 分别为:

$$\begin{aligned} \delta W_s &= eE_0TL_c \cos \varphi_s, \\ \delta W &= eE_0TL_c \cos \varphi. \end{aligned} \quad (2.20)$$

其中， e 为带电粒子电荷， L_c 和 E_0 分别为加速腔的长度和加速梯度， T 为渡越时间因子，与加速腔的结构有关，而 φ_s 和 φ 分别为同步粒子和非同步粒子的加速相位。于是有:

$$\frac{d\Delta W}{dz} = eE_0T(\cos \varphi - \cos \varphi_s), \quad (2.21)$$

其中

$$\Delta W = m_0c^2\gamma_s^3\beta_s\Delta\beta, \quad \Delta\beta = \beta - \beta_s。 \quad (2.22)$$

其中的 β 和 γ 分别为粒子的相对论速度和相对论因子。另外，相位变化也和速度变化有关:

$$\Delta\varphi = \varphi - \varphi_s = -\frac{z - z_s}{\beta_s\gamma}2\pi \rightarrow \Delta\beta = -\frac{\beta_s^2\gamma}{2\pi}\frac{d\Delta\varphi}{dz}。 \quad (2.23)$$

由式(2.21)和式(2.23)可得:

$$\Delta\gamma = \gamma - \gamma_s = \frac{\Delta W}{m_0c^2} = -\frac{\lambda}{2\pi}\beta_s^3\gamma_s^3\frac{d\Delta\varphi}{dz}。 \quad (2.24)$$

联立上式与式(2.21)，即可得到纵向运动方程:

$$\frac{1}{\beta_s^3\gamma_s^3}\frac{d}{dz}\left(\beta_s^3\gamma_s^3\frac{d\Delta\varphi}{dz}\right) + \frac{2\pi eE_0T}{m_0c^2\beta_s^3\gamma_s^3\lambda}(\cos \varphi - \cos \varphi_s) = 0。 \quad (2.25)$$

展开得到:

$$\frac{d^2\Delta\varphi}{dz^2} + \frac{3}{\beta_s\gamma_s} \frac{d\beta_s\gamma_s}{dz} \frac{d\Delta\varphi}{dz} + \frac{2\pi eE_0T}{m_0c^2\beta_s^3\gamma_s^3\lambda} (\cos\varphi - \cos\varphi_s) = 0. \quad (2.26)$$

纵向运动方程描述了粒子在 $(\Delta W, \Delta\varphi)$ 相空间中的运动。假设加速梯度足够小，我们可以忽略阻尼项 $\frac{d\beta_s\gamma_s}{dz}$ ，则纵向运动方程可以简化为:

$$\frac{d^2\Delta\varphi}{dz^2} + \frac{2\pi eE_0T}{m_0c^2\beta_s^3\gamma_s^3\lambda} (\cos\varphi - \cos\varphi_s) = 0. \quad (2.27)$$

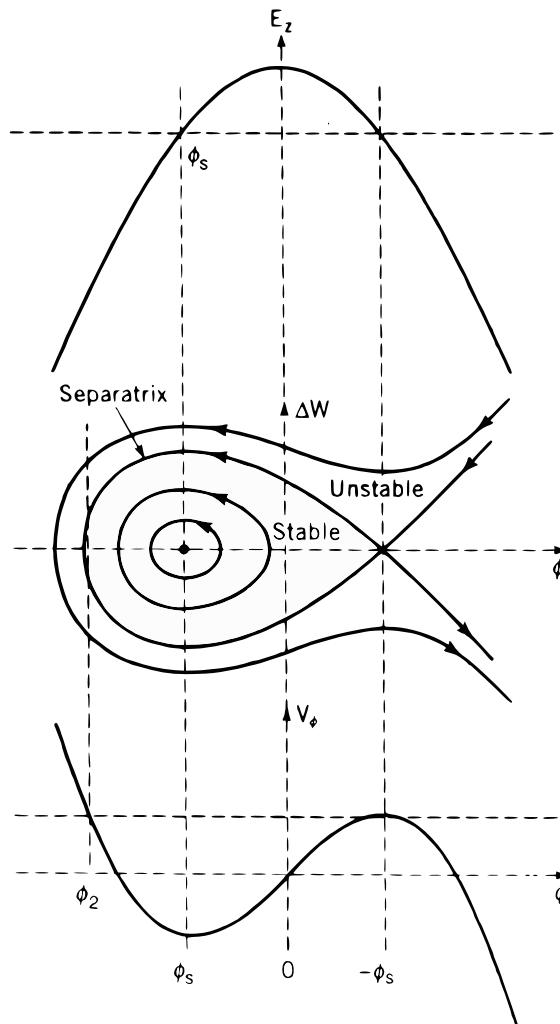


图 2.2: 加速腔电场、势阱、相空间轨迹和相稳定区示意图

加速腔梯度和相位的关系见图(2.2)最上方曲线[21]。当同步相位处于 $(-\frac{\pi}{2}, 0)$ 之间时，粒子纵向相空间存在势阱，相稳定区的边界为“鱼形”，如图(2.2)中图所示。势阱中的粒子围绕同步粒子 (φ_s, W_s) 作振荡，能够被稳定加速。其稳定区的范围为:

$$\varphi_2 < \varphi < -\varphi_s. \quad (2.28)$$

对于小角度振荡， $\varphi_2 \approx 2\varphi_s$ ，因此稳定区的宽度约为 $3|\varphi_s|$ 。当 $|\varphi_s|$ 增加时，稳定区会增大，但是由于同步相位为负，所以总能量增益会减小。

前面的分析中忽略了阻尼项，但在粒子能量较低时阻尼项不可忽略。当考虑了阻尼项之后，相空间的轨迹也有所变化。对于低能量的粒子，当经过加速后，同步粒子的速度和能量会有较大变化，相稳定区将有所增大，因而加速器的接收范围有所增加，相稳定区域的边界也由原来的“鱼形”变成“螺旋线形”形，如图(2.3)所示[21]。

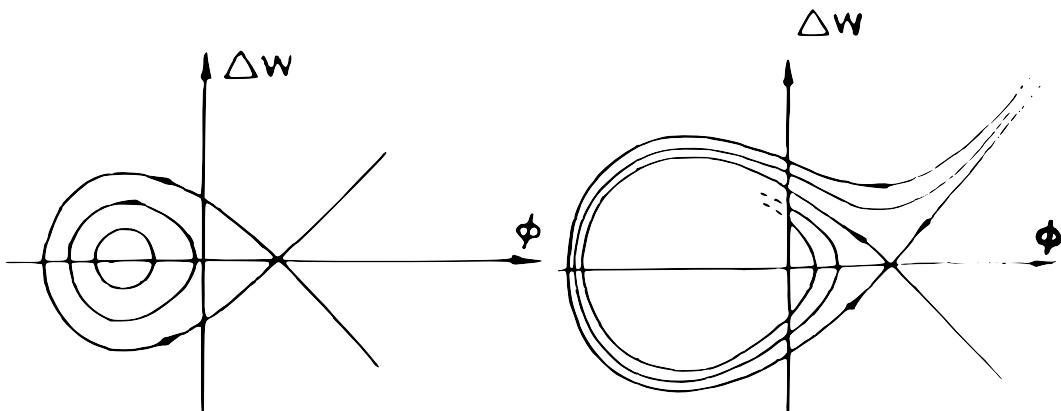


图 2.3: 考虑阻尼项后的纵向相空间稳定区示意图

2.4 空间电荷效应

如小节(1.1)中介绍，空间电荷效应的来源为束团内的粒子之间的库伦相互作用。随着加速器流强的不断提高，束团内粒子的相互作用与外场对粒子的作用相比已经不可忽略，甚至在极端流强下，空间电荷对粒子的运动起到了主导作用。

在一个多粒子的束团中，空间电荷效应产生的力可以分为两个方面：一方面来源于长程的电磁相互作用，即束团中所有粒子在空间中产生了一个近似光滑的电磁场，每一个处于这个电磁场中的粒子都受到作用；另一方面来源于短程的库伦碰撞作用。通常来讲，考虑到加速器中的一个束团的粒子数目，相比于短程的库伦碰撞作用，长程的电磁相互作用占据了主导。所以一般在空间电荷研究中，只考虑长程的相互作用，而忽略短程的碰撞作用。短程的库伦碰撞引起的散射叫做束内散射（Intra-Beam Scattering, IBS），其不在本文讨论范围内。在本文中，空间电荷效应指束团内部的长程电磁相互作用。

空间电荷作用与粒子在空间中的分布有关，而粒子的分布又是加速器元件产生的外场和束团自身产生的内场共同作用的结果。解析的计算需要联立求解牛顿

方程和麦克斯韦方程组（式(2.1)和(2.3)），其精确解只能使用数值的方法来得到。

在早期计算机性能不足时，人们使用很多方法对空间电荷效应进行了近似估计。比如空间电荷线性近似方法，在这种方法中，我们首先假定束团的分布形态，然后使用连续电荷分布所产生的场表达粒子间的相互作用。例如在三维情况下，如果粒子在实空间中的分布为一个均匀的椭球，则粒子在空间中任意位置的所受到的电场可以表示为[22]：

$$\begin{aligned} E_x &= \frac{3ITx}{4\pi\epsilon_0\gamma^2XYZ}\mu_x, \\ E_y &= \frac{3ITY}{4\pi\epsilon_0\gamma^2XYZ}\mu_y, \\ E_z &= \frac{3ITz}{4\pi\epsilon_0\gamma^2XYZ}\mu_z. \end{aligned} \quad (2.29)$$

其中 I 为束流流强， T 为相邻束团的时间间隔， X, Y, Z 分别是束流在三个方向上的尺寸， x, y, z 是粒子的位置， μ_x, μ_y, μ_z 是束团的形状因子：

$$\begin{aligned} \mu_x &= \frac{XYZ\gamma}{2} \int_0^\infty \frac{d\xi}{(X^2 + \xi)\sqrt{(X^2 + \xi)(Y^2 + \xi)(Z^2\gamma^2 + \xi)}}, \\ \mu_y &= \frac{XYZ\gamma}{2} \int_0^\infty \frac{d\xi}{(Y^2 + \xi)\sqrt{(X^2 + \xi)(Y^2 + \xi)(Z^2\gamma^2 + \xi)}}, \\ \mu_z &= \frac{XYZ\gamma}{2} \int_0^\infty \frac{d\xi}{(Z^2\gamma^2 + \xi)\sqrt{(X^2 + \xi)(Y^2 + \xi)(Z^2\gamma^2 + \xi)}}. \end{aligned} \quad (2.30)$$

但是，由于线性近似方法对束团分布的硬性假设，这种方法得到的解是非自洽的。而且这种方法只能得到线性空间电荷力的表达，几乎不能与实验相符。

计算机性能得到发展之后，人们更多的使用数值方法对空间电荷效应进行研究。一种最直接的数值求解空间电荷效应的方法是叠加计算，即对一个粒子逐个计算与其他粒子的相互作用力，然后将其作用力叠加起来，构成该粒子的总空间电荷力。这一种方法的计算复杂度为 $O(N_p^2)$ ， N_p 是粒子数目。由于其运算量与粒子数目平方成正比，这种算法在粒子数目较大时的计算开销很大，在实际研究中并不常用。

为了降低运算量，人们采取另一种求解空间电荷效应的方法，质点网格法（也叫粒子云算法，英文简称PIC算法）。其原理为通过对空间进行网格划分，先将粒子权重分配到网格上，再在网格上求解泊松方程，得到空间网格上的电势分布后，通过差分得到网格上的电场，再通过反向插值将电场作用到单个粒子上。通过使用网格，质点网格法的计算复杂度由原先直接粒子-粒子计算的 $O(N_p^2)$ 降低到了 $O(\alpha N_p + \beta N_{cells} \log N_{cells})$ ，其中 N_p 是粒子数，而 N_{cells} 是网格点数目，而 α 和 β 为

算法有关的常数。因为PIC算法能够有效地降低运算量，所以绝大多数束流模拟程序使用PIC算法来求解空间电荷力。关于PIC算法的详细介绍将会在第(3.2)节中展开。

PIC算法需要将粒子权重到网格上，这样不可避免的会带来网格热噪声。另外目前人们对于PIC算法是否可以保证辛条件仍然有较大的争议。如果不能保证辛条件，那么一些数值算法带来的非物理的效应就会被引入到模拟中。最近，无网格保辛多粒子追踪算法（Symplectic算法）被引入到加速器研究和模拟，被用作长距离模拟中空间电荷求解器 [23]。Symplectic算法并不利用网格，而是利用高阶分解来求解空间电荷效应，避免了网格噪声带来的影响。然而，Symplectic算法虽然能够保证辛条件，但其计算量要大得多，计算花费的时间比PIC算法要高两到三个数量级。幸运的是，无网格算法很适合并行加速运算，有很好的可扩展性。我们将在后文(3.3)节中对Symplectic算法的基本原理进行介绍。

2.5 国内外强流加速器简介

在早期，粒子加速器主要服务于高能物理研究，追求更高的能量，加速器发展也主要是朝着更高能量的方向发展。近年来，强流逐渐成为了另外一个重要的研究方向，其中一方面是因为某些物理现象为了提供足够高统计度，需要大量的事例累积；另一方面是因为某些极为稀少的事例只有在流强较大的加速器中才能观察到。图(2.4)展示了正在运行的、建造中的和计划中的高流强高功率质子同步加速器[24]，三条橙色虚线分别代表0.1MW、1MW、10MW的束流功率。可以看出，受到核物理、高能物理、散裂中子源、加速器驱动的次临界核能系统等应用的驱动，强流质子加速器得到了极大的发展。

表(2.1)和(2.2)是目前正在运行、建造和计划中的高功率质子直线加速器和同步加速器的主要参数[24]。目前运行的机器主要在1mA量级，功率在0.2-1MW之间。而在建和提出的加速器的功率基本在5MW附近。

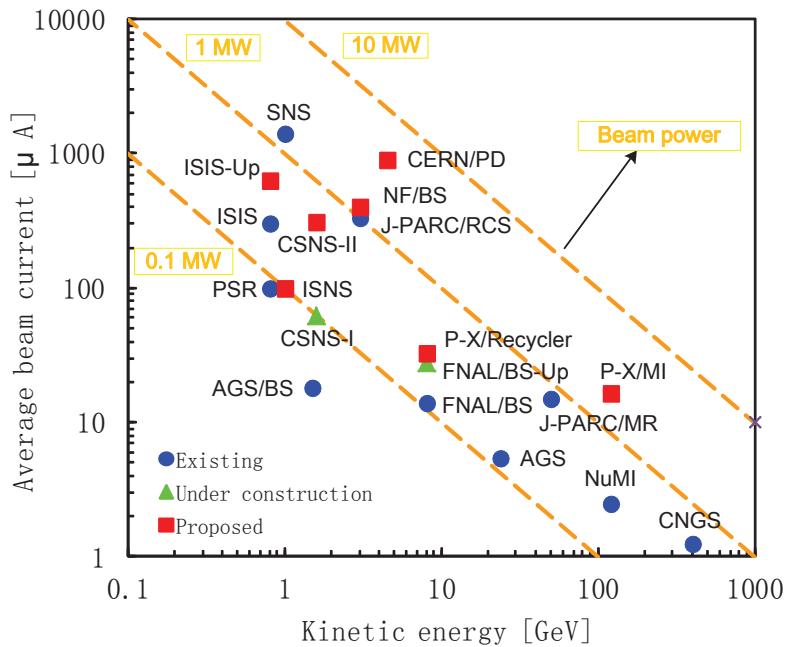


图 2.4: 高流强高功率质子同步加速器汇总

表 2.1: 国内外强流质子直线加速器参数表

	能量 (GeV)	脉冲束长 (ms)	重复频率 (Hz)	占空比 (%)	I_b (mA)	I_{ave} (mA)	P_{ave} (MW)
LANSCE	0.8	0.625	100/20	6.2/1.2	16/9.1	1.0/0.1	0.8/0.08
FNAL	0.4	0.05	15	0.04	35	0.014	0.007
SNS	1.0	1.0	60	6.0	38	1.4	1.4
J-PARC(1)	0.18	0.5	50/25	2.5	50	0.7	0.28/0.14
J-PARC(2)	0.6	0.5	25	1.25	50	0.35	0.21
CERN SPL	2.2	2.8	50	14	22	1.8	4.0
ESS SP	1.33	1.2	50	6.0	114	3.75	5.0
ESS LP	1.33	2.0/2.5	16.67	4.2	114/90	3.75	5.0
ESS-S	2.5	2.86	14	4.0	50	2	5.0
Project-X	3/8	Chopped	-	10/2.5	10	1/0.25	3/2
TRASCO	≥ 1.0	CW	-	100	30	30	≥ 30
IFMIF	0.04	CW	-	100	2*125	2*125	10
C-ADS	0.15	CW	-	100	10	10	1.5

表 2.2: 国内外强流质子环形加速器参数表

	能量 (GeV)	注入能量 (GeV)	重复频率 (Hz)	累积粒子数 (10^{13})	I_{ave} (mA)	P_{ave} (MW)
IPNS	0.45	0.05	30	0.3	0.167	0.0075
ISIS	0.8	0.07	50	3.75	0.3	0.24
PSR-I	0.8	0.8	20	3.1	0.1	0.08
PSR-II	0.8	0.8	30	4.1	0.2	0.16
SNS	1.0	1.0	60	14.6	1.4	1.4
J-PARC/RCS	3.0	0.4	25	8.3	0.333	1.0
J-PARC/MR	50	3.0	0.3	33.2	0.015	0.75
FNAL/Booster	8	0.4	15	7.5	0.014	0.12
CSNS-I	1.6	0.8	25	1.56	0.065	0.1
CSNS-II	1.6	0.25	25	7.8	0.325	0.5
ISNS	1.0	0.1	25	2.4	0.1	0.1
Project-X/MI	120	8	0.4	16	0.0167	2

第三章 束流模拟程序的物理模型和算法

空间电荷效应是束流物理中非常重要的问题。特别是对于强流加速器，空间电荷效应更加明显，因此我们必须在加速器设计中正确处理。如我们在(2.4)节中所介绍，目前存在若干种不同的计算空间电荷效应的模型，比如线性近似、P2P方法、PIC模型、Symplectic模型，各个模型的计算速度、精度、近似程度都有所不同。

在本章中，主要介绍计算空间电荷效应的几种物理模型和模拟程序的一般性设计准则。首先，节(3.1)对一种近似求解单粒子空间电荷效应的物理模型 - 束核模型 - 进行介绍。其次，在(3.2)节中对束流模拟中的经典的PIC算法进行介绍。之后，在(3.3)节中对一种新型的求解空间电荷效应的算法 - Symplectic算法 - 进行介绍。最后，(3.4)节中对各个物理模型进行了小结。

3.1 束核模型

由于空间电荷效应这个多体耦合问题没有解析解，所以我们必须采用一些近似，建立一个更简单的模型，以得到这个问题的大致图像。R. L. Gluckstern和T. P. Wangler等人提出了一种束核模型（particle-core）[25, 26]，以描述空间电荷导致的束晕形成机制[27–31]。在束核模型中连续的束流通过一个轴对称的聚焦通道，假设粒子束核为均匀分布，则其包络大小可由包络方程描述。此时一个任意的粒子会受到束核产生的空间电荷力，当粒子处于束核内部时，空间电荷力为线性的；当粒子处于束核外部时，空间电荷力为非线性的。忽略单粒子对束核的影响，则束核的半径可以由下式表示：

$$\frac{d^2R}{dz^2} + k_0^2 R - \frac{\varepsilon^2}{R^3} - \frac{K}{R} = 0, \quad (3.1)$$

其中

$$K = \frac{qI}{2\pi\varepsilon m_0 c^3 \beta^3 \gamma^3}. \quad (3.2)$$

其中 R 为束核包络的大小， q ， m_0 ，和 βc 分别为粒子的电荷，质量，和运动速度， k_0 为外部聚焦强度， γ 为洛伦兹因子， I 为流强， ε 为发射度。

所以，单粒子的横向运动方程可以表示为：

$$\frac{d^2X}{dz^2} + k_0^2 X - F_{sc} = 0, \quad (3.3)$$

其中, X 为粒子的横向位置, F_{sc} 为由均匀分布束核产生的空间电荷力, 可以表示为:

$$F_{sc} = \begin{cases} KX/R^2, & |X| < R \\ K/X, & |X| \geq R \end{cases} \quad (3.4)$$

对式(3.1)和式(3.3)进行无量纲化, 我们引入变量 $r = R/R_0$, $x = X/R_0$, $\tau = k_0 z$, 并将空间电荷导致的tune depression表示为 $\eta = k/k_0 = \sqrt{1+u^2} - u$ 。则无量纲化的束核包络方程可以表示为:

$$\frac{d^2r}{d\tau^2} + r - \frac{\eta^2}{r^3} - \frac{1-\eta^2}{r} = 0. \quad (3.5)$$

无量纲化的单粒子运动方程可以表示为:

$$\frac{d^2x}{d\tau^2} + x = (1-\eta^2) \times \begin{cases} x/r^2, & |x| < r \\ 1/x, & |x| \geq r \end{cases} \quad (3.6)$$

式(3.5)和式(3.6)仅含有相位压缩因子 η 一个参数。当 $r_0 = 1$ 时, 束流包络为常数, 即束核匹配。为了描述束核的初始匹配程度, 我们将束核的初始的包络半径定位为失配度 $\mu = r_0$ 。失配度会影响束核包络的变化。当束核完全与聚焦强度相匹配时, 即 $\mu = 1$ 时, 束核的包络大小是不变的; 而当包络与聚焦强度不匹配的时候, 包络半径会发生周期性变化。图(3.1)为束核包络与单粒子横向位移示意图, 其横坐标为周期数, 纵坐标为粒子位置或束团半径, 单位为米。其中蓝色曲线为 $\eta = 0.5$, $\mu = 0.6$ 时的束核半径变化情况, 红色曲线为初始条件为 $x = 0.8$, $x' = 0$ 的粒子横向轨迹。可以看出, 束核的半径呈现规律的周期振荡, 而粒子和束核的相互作用会驱使粒子横向位置发生变化, 以表示空间电荷对粒子的影响。

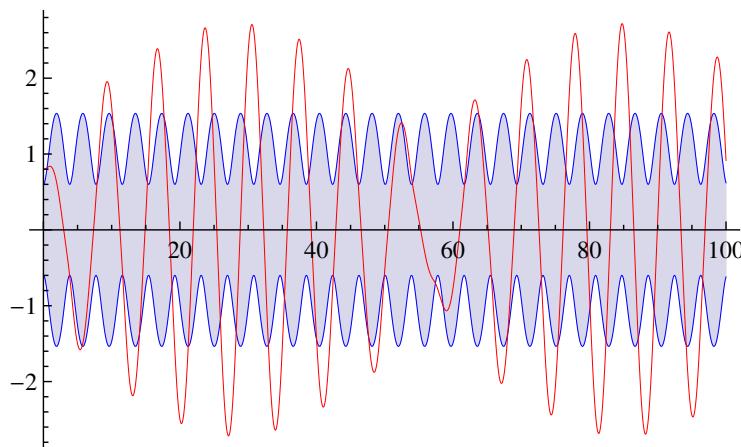


图 3.1: 束核包络与单粒子横向位移示意图

图(3.2)为每个周期的开始处粒子的相空间位置，其中横轴为粒子横向位置，纵轴为粒子横向动量，这张图被称为粒子的庞加莱截面（Poincaré surface of section）。在庞加莱截面中可以看到明显的规律，粒子沿着其中一条轨道进行运动。

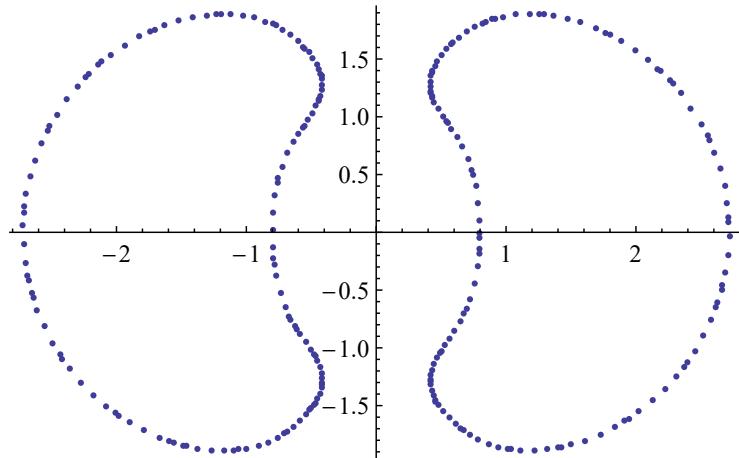


图 3.2: 粒子的庞加莱截面示意图

图(3.3)中的三幅图分别为不同的相位压缩因子 η 下的一组庞加莱截面，其中每一张图中的不同的颜色的点代表不同的初始相空间位置的粒子的轨迹。可以看出，庞加莱截面图可以分为三个区域：(1)内椭圆区，其大小约为束核半径及周边一小部分；(2)不动点区，或者叫共振区，包括x轴上的不动点以及周围的环线，其显示了参数共振的轨迹；(3)外部类椭圆轨迹。在空间电荷效应较弱的情况下，即 η 较大时，三个区的粒子都是作有规律的运动，如图(3.3a)所示。但是当空间电荷主导的时候，比如 $\eta = 0.1$ 时（图(3.3c)），在内椭圆区会出现混沌行为，并且随着相位压缩因子变小，混沌行为变得更加明显，参数区的粒子轨迹也开始对初始条件敏感。采用庞加莱截面我们可以对束晕形成的机制进行分析。束晕中的粒子主要来源于共振区中的粒子。这些粒子最开始分布于内椭圆区与共振区的分界线附近，随着粒子运动，其向外移动到共振区与外椭圆区的边界附近，在两个边界间振荡，形成了束晕。在空间电荷效应主导的情况下，内椭圆区的粒子也有可能形成束晕。比如图(3.3c)中的内椭圆区也出现了混沌行为，因此最内部的粒子也会移动到外部形成束晕。

束核模型在一定程度上可以解释束晕形成的机制，但是这个模型并不是自治的，与实际不相符。束核模型假设束核是稳定的，并且外部聚焦力是恒定的常数；而实际的束核有可能是不稳定的，外部的聚焦力也是可能变化的。因此，我们需要一个更加自治的系统来研究束流的行为。

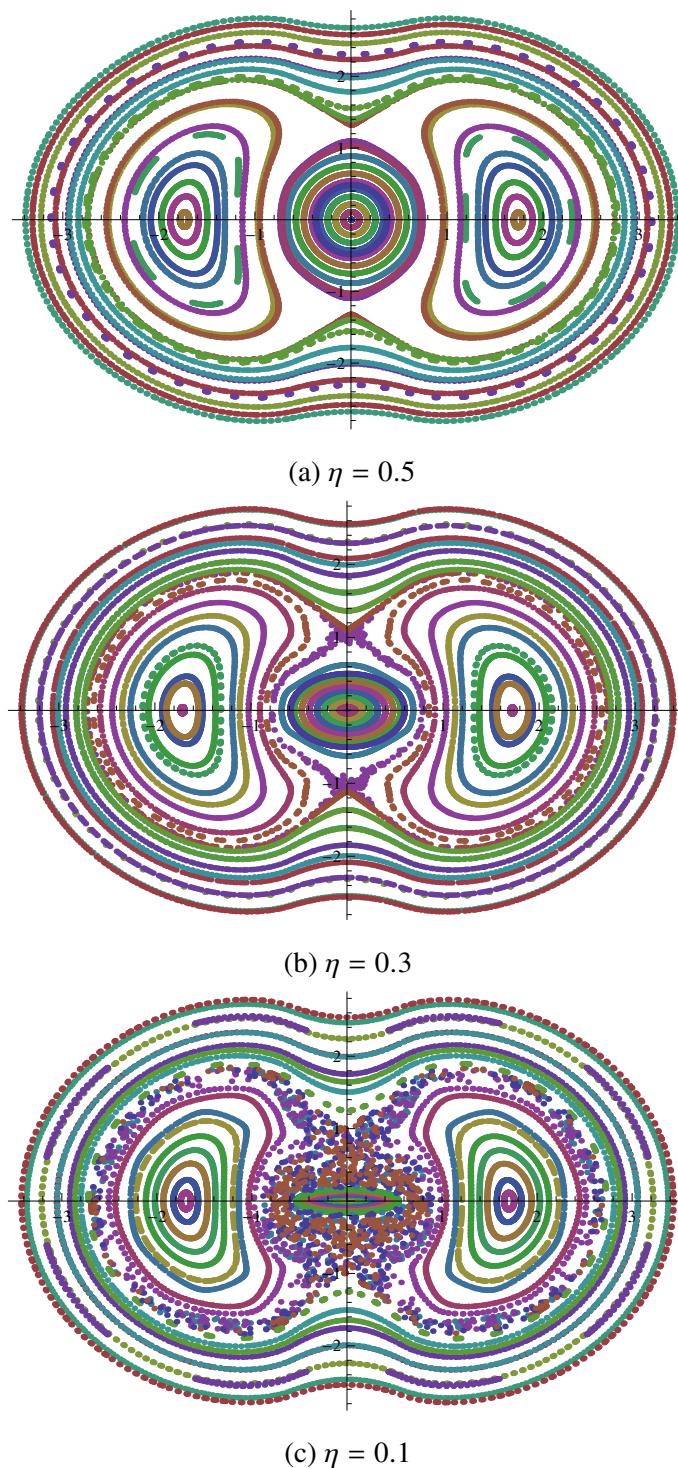


图 3.3: 不同相位压缩因子下的庞加莱截面

3.2 束流模拟原理及PIC算法

PIC算法是一种使用自治的系统研究空间电荷效应的一种数值模拟算法[32, 33]。PIC发展于上世纪七十年代，被广泛的应用于等离子体和加速器束流的模拟与研究。目前，主流的加速器模拟程序都是使用的PIC算法求解空间电荷效应[34–40]。使用PIC算法进行束流模拟的基本流程如图(3.4)和(3.5)所示。

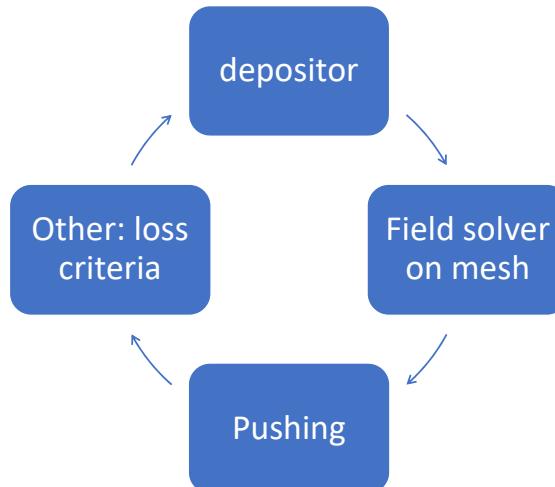


图 3.4: PIC算法块循环示意图

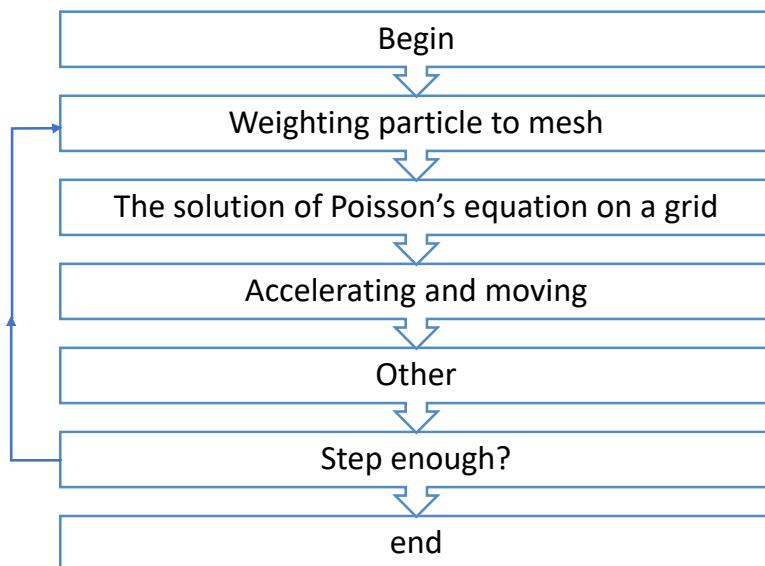


图 3.5: PIC算法分段流程示意图

根据粒子的位置和范围确定所用的空间网格大小之后，模拟程序中的一步可以表示为：

1. 将网格内的所有粒子的电荷根据粒子的位置权重到网格格点上，即从粒子分布得到获得空间点上的密度分布。
2. 根据网格点上的电荷密度分布，求解网格点上的泊松方程，得到网格点上的电势分布。
3. 根据电势分布，差分得到网格点上的电场分布。
4. 根据粒子位置和网格上的电场分布，反向权重得到静止坐标系下粒子所受到的电场，再通过洛伦兹变换得到实验室坐标系下粒子所处位置的电磁场。
5. 通过牛顿方程，推动粒子，更新粒子位置和动量。

下面，我们分别就权重插值、空间电荷效应求解、粒子推动三个主要方面进行详细讨论。

3.2.1 权重插值

PIC算法的第一步是将粒子电荷权重插值到网格上。粒子权重插值主要有以下几个方面进行考虑：

1. 如何将粒子电荷密度（3D）或者电流密度（2D）权重到空间网格上。
2. 求解泊松方程后，如何将网格上的电场权重插值回粒子上。
3. 如何确定网格点的数目和相对于粒子分布的位置及范围。

对于前两个问题，我们一般采用一个插值算法的正反两个过程来处理从粒子到网格和从网格到粒子的插值，从而避免算法上带来的非物理效应。如果采用不同的插值方法，可能会出现粒子自推动的错误。

以一维问题为例，将粒子电荷权重到空间网格上的过程，可以理解为粒子坐标 x_i 到网格点上的电荷密度 ρ_p 的一个映射，其中 p 为网格点的坐标。一些常见的权重方法如图(3.6)所示，有最近网格点法（Nearest Grid Point, NGP）、网格差分法（Cloud In Cell, CIC）、以及三角云分配法（Triangular Shaped Cloud, TSC）。图中，粒子处于 x 位置，而 $p - 1, p, p + 1$ 分别代表不同的网格格点，不同颜色的面积大小代表分配给相应格点的电荷权重，按照不同的形状分配可以得到不同的结果。NGP方法为零阶插值，只将电荷分配到离粒子位置最近的网格点上，有较大误差；CIC方法为一阶插值，根据粒子位置和格点位置的关系，如图(3.6b)所示，将粒子

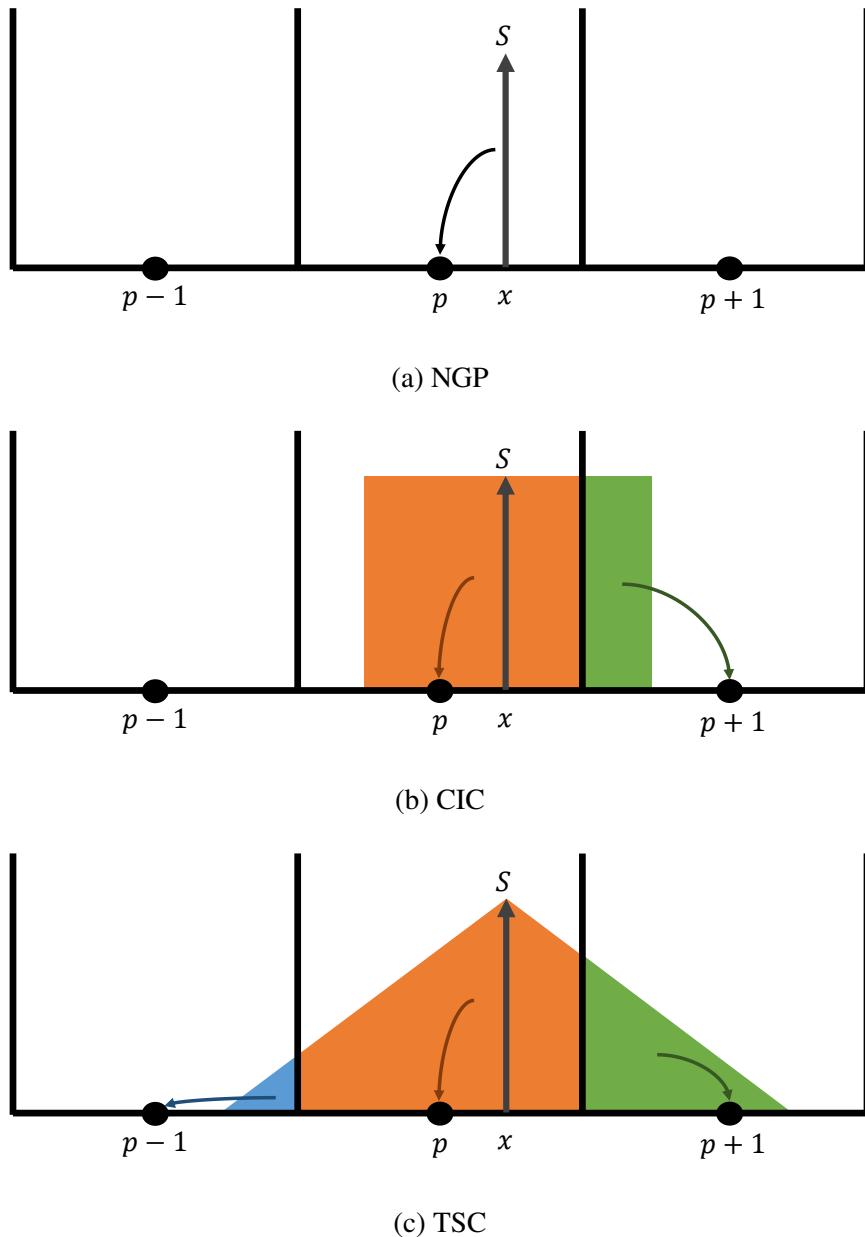


图 3.6: 权重插值中的分配方式示意图

按不同比例分配到临近的两个网格点上； TSC方法为二阶插值，如图(3.6c)所示，分配函数为三角形，将粒子按不同比例分配到临近的三个网格点上；

在数学上，这个权重过程可以表示为：

$$\rho_p = \sum_i^{N_p} q_i W(x_i - x_p), \quad (3.7)$$

其中， q_i 为粒子所带电荷量， $W(x)$ 为形状因子。如图(3.7)所示，根据插值的阶数不同， $W(x)$ 有很多形式。下式给出了零阶，一阶，和二阶的形状因子：

$$W_0(x) = \begin{cases} 1, & |x| \leq \frac{\Delta x}{2}, \\ 0, & |x| > \frac{\Delta x}{2}, \end{cases} \quad (3.8)$$

$$W_1(x) = \begin{cases} 1 - \frac{|x|}{\Delta x}, & |x| \leq \Delta x, \\ 0, & |x| > \Delta x, \end{cases} \quad (3.9)$$

$$W_2(x) = \begin{cases} \frac{1}{\Delta x} \left[\frac{3}{4} - \left(\frac{|x|}{\Delta x} \right)^2 \right], & |x| \leq \frac{\Delta x}{2}, \\ \frac{1}{2\Delta x} \left[\frac{3}{2} - \frac{|x|}{\Delta x} \right], & \frac{\Delta x}{2} < |x| \leq \frac{3\Delta x}{2}, \\ 0, & |x| > \frac{3\Delta x}{2}. \end{cases} \quad (3.10)$$

在数值模拟中，低阶的算法，比如NGP，会带来很大的粒子离散误差。为了避免数值误差，我们需要采取高阶精度的算法，但是高阶的算法会导致计算量的增大，不满足计算的要求。所以实际中往往采用CIC插值方法，兼顾精度和效率。在P-TOPO中，所有的插值形式都采用一阶的线性插值（CIC）。

对于实际的三维问题，处理方法也是类似，我们采用立方网格，使用体积权重法来计算网格上的电荷。

3.2.2 使用FFT解泊松方程

经过由粒子到空间网格的权重插值后，我们得到了分布在网格点上的电荷密度分布函数。接下来要做的就是求解网格点上的泊松方程，对于本程序而言，我们使用快速傅里叶变换（FFT）的方法来求解。以一维问题为例，泊松方程：

$$\nabla^2 \phi = -\frac{\rho}{\epsilon_0}, \quad (3.11)$$

在网格上可以离散表示为：

$$\frac{\phi_{j-1} - 2\phi_j + \phi_j}{\Delta_x^2} = -\frac{\rho_j}{\epsilon_0}. \quad (3.12)$$

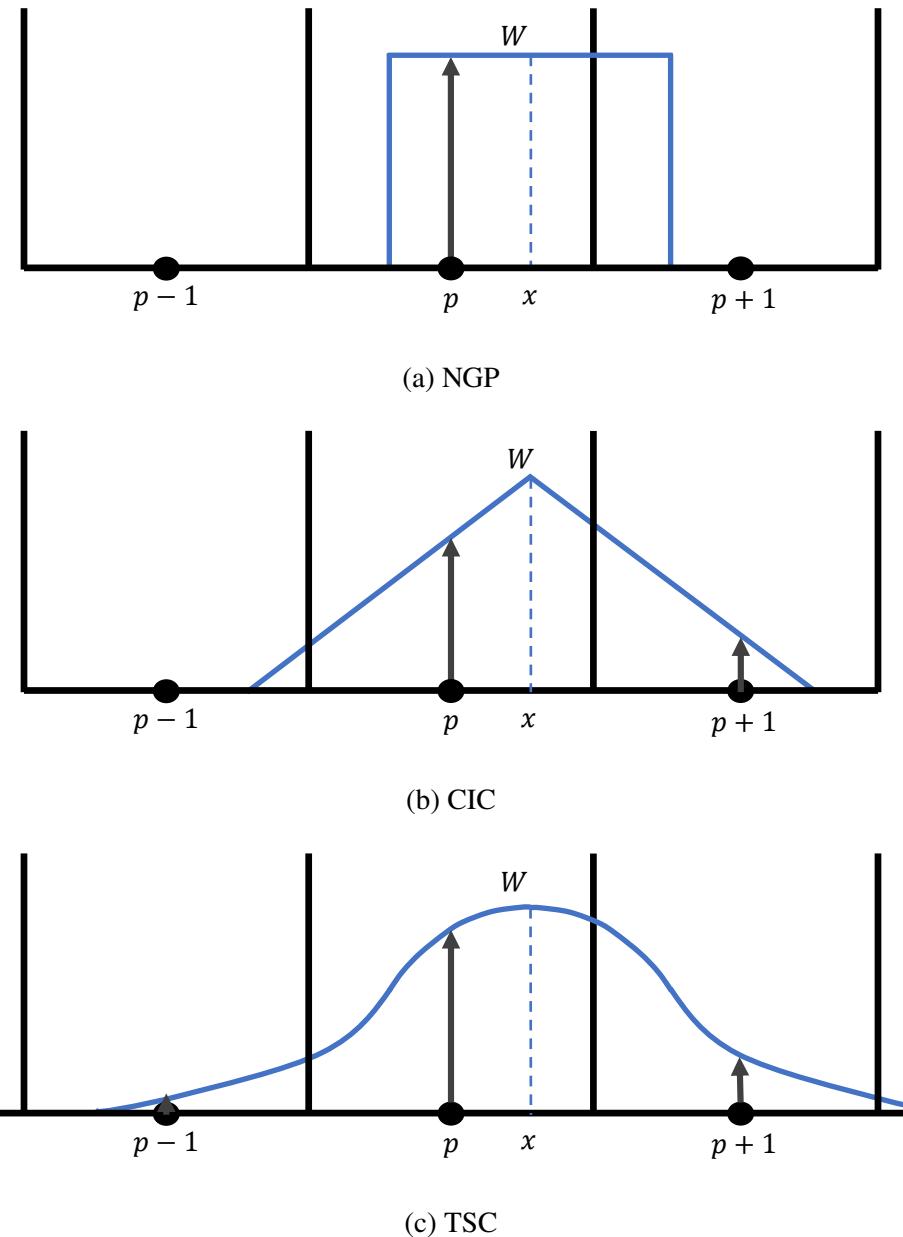


图 3.7: 权重插值中的形状因子曲线

其中, ϕ_j 为第j个格点上的电势, ρ_j 为第j个格点上的电荷量, Δ_x 为两个格点之间的距离。

将 ϕ_j 和 ρ_j 进行展开, 我们可以得到:

$$\phi_j = \frac{1}{N_x} \sum_{n=0}^{N_x} \phi_n \exp\left(-\frac{i2\pi nj}{N_x}\right), \quad (3.13)$$

$$\rho_j = \frac{1}{N_x} \sum_{n=0}^{N_x} \rho_n \exp\left(-\frac{i2\pi nj}{N_x}\right), \quad (3.14)$$

其中, N_x 为一个方向上网格点的数目。将式(3.13)和式(3.14)代回式(3.12)可得:

$$\begin{aligned} & \sum_{n=0}^{N_x} \phi_n \left(\exp\left(-\frac{i2\pi n(j-1)}{N_x}\right) - 2 \exp\left(-\frac{i2\pi nj}{N_x}\right) + \exp\left(-\frac{i2\pi n(j+1)}{N_x}\right) \right) \\ &= \frac{\Delta_x^2}{\epsilon_0} \sum_{n=0}^{N_x} \rho_n \exp\left(-\frac{i2\pi nj}{N_x}\right), \end{aligned} \quad (3.15)$$

对于傅里叶空间的电势和电荷密度, 我们可以得到 ϕ_n 和 ρ_n 的关系:

$$\phi_n \left(\exp\left(\frac{i2\pi n}{N_x}\right) + \exp\left(-\frac{i2\pi n}{N_x}\right) - 2 \right) = \frac{\Delta_x^2}{\epsilon_0} \rho_n. \quad (3.16)$$

化简得到

$$\phi_n = \frac{\rho_n}{\epsilon_0 K_n^2}, \quad (3.17)$$

其中

$$K_n = \left(\frac{2\pi n}{N_x \Delta_x}\right) \frac{\sin(\pi n/N_x)}{\pi n/N_x}. \quad (3.18)$$

可以看出, 通过傅里叶变换得到的傅里叶空间中的电荷密度 ρ_n , 进而可以得到傅里叶空间中的电势分布函数 ϕ_n 。然后再经过傅里叶逆变换, 我们就可以得到实空间中的电势分布 ϕ_j , 最后经过差分我们就可以得到空间电场分布函数。这个过程如图(3.8)所示。需要注意的是, 这种 e 指数的展开方式决定了我们使用的是周

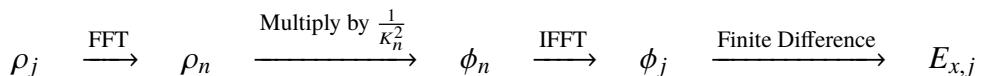


图 3.8: 使用FFT求解泊松方程流程图

期性边界条件。如果我们想要第一类边界条件, 我们需要采用 \sin 函数变换, 其变换过程与此类似。在P-TOPO中, 我们在横向采用第一类边界条件, 而在纵向采用周期性边界条件。

3.2.3 蛙跳法推动粒子

蛙跳法是一种束流模拟程序中常用的推动粒子的方法[41]，其计算的数学形式如下：

$$\mathbf{v}_{i+3/2} = \mathbf{v}_{i+1/2} + \frac{\mathbf{F}(\mathbf{x}_{i+1})}{m} \Delta T, \quad \mathbf{r}_{i+1} = \mathbf{r}_i + \mathbf{v}_{i+1/2} \Delta T. \quad (3.19)$$

蛙跳法利用1/2时间步长处的粒子速度来推动粒子位置。下面我们从一致性、同时性、高准确性、高稳定性四个方面来对蛙跳算法进行详细讨论。

3.2.3.1 一致性

一致性是指差分方程和微分方程的一致。任何数值算法都是使用有限大小的步长来模拟一个连续的过程。因此，我们应该在时间步长无限小的时候，使其效果与连续推动一致。

考虑经典的牛顿方程：

$$\frac{d\mathbf{x}}{dt} = \mathbf{v}, \quad (3.20)$$

$$m \frac{d\mathbf{v}}{dt} = \mathbf{F}. \quad (3.21)$$

如果我们采用有限步长 ΔT 来推动粒子位置，牛顿运动方程变为：

$$\frac{\mathbf{x}_{n+1} - \mathbf{x}_n}{\Delta T} = \mathbf{v}_n, \quad (3.22)$$

$$m \frac{\mathbf{v}_{n+1} - \mathbf{v}_n}{\Delta T} = \mathbf{F}_n. \quad (3.23)$$

当其步长 ΔT 足够小时，其结果应该与微分方程(3.20) 和(3.21)相一致。

3.2.3.2 同时性

同时性又叫做可逆性。微分方程(3.20)和(3.21)是可逆的。例如，一个粒子在给定的力场中沿时间正向积分运动，然后如果将其速度取反，并沿时间反向积分，粒子将会沿着原有的轨迹向回运动，并能回到起始点。

这一点在差分近似中并不能得到保证，比如，差分方程(3.22)和(3.23)中，粒子在第 $n + 1$ 步处取反，粒子位置变化为

$$\frac{\mathbf{x}_n - \mathbf{x}_{n+1}}{\Delta T} = -\mathbf{v}_{n+1}, \quad (3.24)$$

其速度为 \mathbf{v}_{n+1} 而不是 \mathbf{v}_n ，也就是说粒子并不能沿其轨迹回到初始位置。这是因为差分方程的左右的时间中心不相同。式(3.22)左侧，位置改变为 $\mathbf{x}_{n+1} - \mathbf{x}_n$ ，是以 $t_{n+1/2}$ 为

时间中心的；而右侧 \mathbf{v}_n 是时间 t_n 时的速度。因此，我们需要将其改变为：

$$\frac{\mathbf{x}_{n+1} - \mathbf{x}_n}{\Delta T} = \mathbf{v}_{n+1/2}, \quad (3.25)$$

$$m \frac{\mathbf{v}_{n+1/2} - \mathbf{v}_{n-1/2}}{\Delta T} = \mathbf{F}_n. \quad (3.26)$$

这也就是蛙跳法，即位置的时间与速度的时间并不在统一时刻，而是采用相差半个步长，然后交替推动粒子的速度和位置，以满足同时性条件。

3.2.3.3 准确性

准确性与一步计算中的数值解与解析解的误差有关。这个误差主要来源于两方面：第一，来自于计算机的精度，例如，计算程序中一般使用双精度浮点数，十进制下仅有十四或十五位有效数字；第二，来自于使用有限小的步长代表连续变量的截断误差。通常来讲，第一种来自计算机数值精度的误差非常小，如果算法是稳定的（关于稳定性的论述见下一条），这个误差是可以忽略的。第二种来自于有限小的步长的误差，可以由解析解与数值解的差值来表示，其差值通常与步长 ΔT 的 n 次方(ΔT) n 成正比，而不同算法之间的稳定性可由阶数 n 来表示。准确性越差，程序实际所需要的时间步长就越小，即程序的计算效率越差。

由式(3.25)和(3.26)可知，蛙跳法为解析解(3.20)和(3.21)的二阶近似。证明如下：由式(3.25)和(3.26)可得：

$$\frac{\mathbf{x}_{n+1} - 2\mathbf{x}_n + \mathbf{x}_{n-1}}{(\Delta T)^2} = \frac{\mathbf{F}(\mathbf{x}_n)}{m}. \quad (3.27)$$

如果 \mathbf{X} 为解析解：

$$\frac{d^2\mathbf{X}}{dt^2} = \frac{\mathbf{F}}{m}, \quad (3.28)$$

那么有限小步长的误差可以由 δ^n 表示：

$$\frac{\mathbf{X}_{n+1} - 2\mathbf{X}_n + \mathbf{X}_{n-1}}{(\Delta T)^2} = \frac{\mathbf{F}(\mathbf{X}_n)}{m} - \delta^n. \quad (3.29)$$

将 \mathbf{X}_{n+1} 和 \mathbf{X}_{n-1} 泰勒在 $\mathbf{X}_n = \mathbf{X}(t_n)$ 处展开，式(3.29)可以写作：

$$\frac{d^2\mathbf{X}}{dt^2} + \frac{(\Delta T)^2}{12} \frac{d^4\mathbf{X}}{dt^4} + h.o.t. = \frac{\mathbf{F}(\mathbf{X}_n)}{m} - \delta^n, \quad (3.30)$$

其中 $h.o.t.$ 为高阶项(higher order term)。式(3.29)与式(3.30)相减可得：

$$\delta^n = -\frac{(\Delta T)^2}{12} \frac{d^4\mathbf{X}}{dt^4} + h.o.t., \quad (3.31)$$

即，蛙跳法为二阶准确度($\delta^n \propto (\Delta T)^2$)，基本能够满足程序需求。

3.2.3.4 稳定性

稳定性与误差随时间的变化有关。即使算法每一步的误差非常小，最终误差也有可能由于累积效应导致变大。在一个数值算法中，如果每一步的误差不会导致更大的累积误差，那么这个算法是稳定的。

下面，我们讨论蛙跳法的稳定性。根据式(3.27)，如果我们给定 $\mathbf{x}_0 = \mathbf{X}_0, \mathbf{x}_1 = \mathbf{X}_1$ 作为初始条件，不考虑误差的情况下，我们可以得到之后的一系列解 $\mathbf{X}_2, \mathbf{X}_3, \mathbf{X}_4, \dots$ ，其中：

$$\mathbf{X}_2 - 2\mathbf{X}_1 + \mathbf{X}_0 = \frac{\mathbf{F}(\mathbf{X}_1)}{m}(\Delta T)^2, \quad (3.32)$$

$$\mathbf{X}_3 - 2\mathbf{X}_2 + \mathbf{X}_1 = \frac{\mathbf{F}(\mathbf{X}_2)}{m}(\Delta T)^2. \quad (3.33)$$

然而，由于误差的存在，我们无法得到微分方程中的精确解 $\mathbf{X}_2, \mathbf{X}_3, \mathbf{X}_4, \dots$ ，而是会得到一系列含有误差的近似解 $\mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \dots$ ：

$$\mathbf{x}_2 - 2\mathbf{X}_1 + \mathbf{X}_0 = \frac{\mathbf{F}(\mathbf{X}_1)}{m}(\Delta T)^2, \quad (3.34)$$

$$\mathbf{x}_3 - 2\mathbf{x}_2 + \mathbf{X}_1 = \frac{\mathbf{F}(\mathbf{X}_2)}{m}(\Delta T)^2. \quad (3.35)$$

这样，在第二步处的误差为

$$\epsilon_2 = \mathbf{x}_2 - \mathbf{X}_2. \quad (3.36)$$

在随后的每一步计算中，含有误差的近似结果都会被使用，我们需要知道的是 ϵ_2 是如何影响后续计算结果。由式(3.34)和式(3.35)可得：

$$(\mathbf{X}_3 + \epsilon_3) - 2(\mathbf{X}_2 + \epsilon_2) + \mathbf{X}_1 = \frac{\mathbf{F}(\mathbf{X}_2 + \epsilon_2)}{m}(\Delta T)^2. \quad (3.37)$$

与式(3.33)相比较，可得

$$\epsilon_3 - 2\epsilon_2 = (\mathbf{F}(\mathbf{X}_2 + \epsilon_2) - \mathbf{F}(\mathbf{X}_2)) \frac{(\Delta T)^2}{m}. \quad (3.38)$$

将上式右侧在 $\mathbf{x} = \mathbf{X}_2$ 处进行泰勒展开：

$$\epsilon_3 - 2\epsilon_2 \approx \epsilon_2 \left. \frac{\partial \mathbf{F}}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{X}_2} \frac{(\Delta T)^2}{m}. \quad (3.39)$$

类似可得：

$$\epsilon_4 - 2\epsilon_3 + \epsilon_2 = \epsilon_3 \left. \frac{\partial \mathbf{F}}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{X}_3} \frac{(\Delta T)^2}{m}, \quad (3.40)$$

则第 n 步的误差为

$$\epsilon_{n+1} - 2\epsilon_n + \epsilon_{n-1} = \epsilon_n \left. \frac{\partial \mathbf{F}}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{X}_n} \frac{(\Delta T)^2}{m}. \quad (3.41)$$

上式并不能得到一个准确的误差递进关系，除非 $\partial\mathbf{F}/\partial\mathbf{x}$ 为常量。由于我们的目标是判断算法是否稳定，而不是算法有多稳定，所以我们考虑最坏的情况，使用 $-|\partial\mathbf{F}/\partial\mathbf{x}|_{max}$ 代替 $\partial\mathbf{F}/\partial\mathbf{x}$ ，其中的负号是因为我们假设所计算的问题是有限边界，而不是会延伸到无穷远。则误差随时间的变化方程为

$$\epsilon_{n+1} - 2\epsilon_n + \epsilon_{n-1} = -\left|\frac{\partial\mathbf{F}}{\partial\mathbf{x}}\right|_{max} \frac{(\Delta T)^2}{m} \epsilon_n, \quad (3.42)$$

定义 $\Omega^2 \equiv \frac{1}{m} \left|\frac{\partial\mathbf{F}}{\partial\mathbf{x}}\right|_{max}$ ，则

$$\epsilon_{n+1} - 2\epsilon_n + \epsilon_{n-1} = -(\Omega\Delta T)^2 \epsilon_n, \quad (3.43)$$

其解的形式与振动方程形式类似，即 $\epsilon_n = \lambda_n = \exp(i\omega n\Delta T)$ 。两个特征解 λ_+ 和 λ_- 为

$$\lambda_{\pm} = 1 - \frac{(\Omega\Delta T)^2}{2} \pm \left[\frac{(\Omega\Delta T)^2}{2} \right] \left[1 - \frac{4}{(\Omega\Delta T)^2} \right]^{1/2}. \quad (3.44)$$

对于算法而言，想要保证数值上稳定，其误差不能随着时间而变大，误差推动方程的特征解必须位于单位圆内，即 $|\lambda| \leq 1$ 。因此，我们必须使时间步长足够小，才能保证算法稳定，对于蛙跳法而言，我们必须使

$$\Delta T \leq \frac{2}{\Omega}. \quad (3.45)$$

总之，权重插值、空间电荷效应求解、反向权重插值、粒子推动构成了基于PIC算法的模拟程序的主要步骤，一个有效的PIC程序必须满足以上各个部分的要求。使用这种方法，计算复杂度由直接的粒子-粒子方法的 N_p^2 降低到了 $\alpha N_p + \beta N_{cells} \log N_{cells}$ 。其中 N_p 是粒子数，而 N_{cells} 是网格点数目。

3.3 Symplectic算法

上一节中我们讨论了PIC算法，这种算法使用网格进行空间电势求解，不可避免的会带来网格热噪声，从而导致数值噪声引起的束流发射度增长。目前加速器界常用的PIC算法并不能保证辛条件（symplectic），其计算会被引入非物理效应，并对最终的强流束流物理的分析和讨论带来一些干扰。因此，在需要长程模拟（比如环形加速器）的物理分析中，我们需要一种保辛的算法。

一开始，Symplectic算法是为了保证哈密顿系统中的辛条件而被研究 [42, 43]。最近，无网格Symplectic模型被引入到加速器研究中作为空间电荷求解器 [23]。这种模型并不利用网格，而是利用高阶分解来求解空间电荷效应。相比PIC算法，这种方法能够显著的降低由于网格数值噪声带来的发射度增长。然而，虽然能够保

证辛条件，这种算法相比PIC算法也有缺陷，最显著的就是其计算量更大。无网格算法的计算复杂度为 $\alpha N_p * N_{modes}$ ，其中 N_{modes} 为分解的阶数，在通常模拟中我们一般使用 $16 \times 16 \times 16$ 阶，在这种配置下Symplectic算法花费的时间比典型的 $64 \times 64 \times 64$ 个格点的PIC算法要高两到三个数量级。所以我们必须提高这个算法的运行速度，以提高算法的实用性。幸运的是，由于无网格算法很适合并行的特性，这种算法能够很好的被加速，并且有很好的可扩展性。下面，我们简要介绍Symplectic算法的基本原理。

在束流动力学模拟中，当且仅当雅可比矩阵 \mathbb{M}_i 满足以下条件时，传输矩阵 \mathbf{M}_i 才是保辛的[16, 17]：

$$\mathbb{M}_i^T \mathbf{J} \mathbb{M}_i = \mathbf{J}, \quad (3.46)$$

其中， \mathbf{J} 是如下 $6N \times 6N$ 的矩阵：

$$\mathbf{J} = \begin{pmatrix} \mathbf{0} & \mathbf{I} \\ -\mathbf{I} & \mathbf{0} \end{pmatrix}. \quad (3.47)$$

对于考虑空间电荷力的多粒子系统，哈密顿量可以写为：

$$H = H_1 + H_2, \quad (3.48)$$

其中

$$H_1 = \sum_i p_i^2 / 2 + \sum_i q\psi(r_i), \quad (3.49)$$

$$H_2 = \frac{1}{2} \sum_i \sum_j q\varphi(r_i, r_j). \quad (3.50)$$

这里， H_1 只包含外场信息，而 H_2 只包括空间电荷效应。根据 H_1 和 H_2 得到的两个传输矩阵 \mathbf{M}_1 和 \mathbf{M}_2 ，一个二阶传输矩阵 $\mathbf{M}(\tau)$ 可以被定义为

$$\mathbf{M}(\tau) = \mathbf{M}_1(\tau/2) \mathbf{M}_2(\tau) \mathbf{M}_1(\tau/2). \quad (3.51)$$

如果 \mathbf{M}_1 和 \mathbf{M}_2 都是保辛的，那么 \mathbf{M} 就是保辛的。在大多数加速器元件中，我们可以通过单粒子动力学获得相应的外场传输矩阵 \mathbf{M}_1 ，而内场传输矩阵 \mathbf{M}_2 可以写为：

$$r_i(\tau) = r_i(0), \quad (3.52)$$

$$p_i(\tau) = p_i(0) - \frac{\partial H_2(r)}{\partial r_i} \tau. \quad (3.53)$$

其雅克比矩阵为：

$$\mathbb{M}_2 = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{L} & \mathbf{I} \end{pmatrix}, \quad (3.54)$$

其中 $L_{ij} = \frac{\partial p_i(\tau)}{\partial r_j} = -\frac{\partial^2 H_2(r)}{\partial r_i \partial r_j} \tau$ 是一个对称矩阵，所以 \mathbb{M}_2 满足保辛条件。

对于一个3D束团， H_2 可以表示为：

$$H_2 = \kappa \gamma_0 \sum_i \sum_j \varphi(r_i, r_j), \quad (3.55)$$

其中 $\kappa = q/(lmC^2\gamma_0^2\beta_0)$, $l = C/\omega$ 。而在束流坐标系下的静电势可以由求解泊松方程得到：

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2} = -\frac{\rho}{\epsilon_0}, \quad (3.56)$$

其边界条件为：

$$\begin{cases} \phi(x=0, y, z) = 0, & \phi(x=a, y, z) = 0, \\ \phi(x, y=0, z) = 0, & \phi(x, y=b, z) = 0, \\ \phi(x, y, z=0) = 0, & \phi(x, y, z=c) = 0. \end{cases} \quad (3.57)$$

这里， a, b, c 分别是X, Y, Z方向上的零电势边界长度。如果我们将 c 设的足够大，电势将在无穷远处为零。

泊松方程中的电势 ϕ 和电荷密度 ρ 可以展开为：

$$\rho(x, y, z) = \sum_{l=1}^{N_l} \sum_{m=1}^{N_m} \sum_{n=1}^{N_n} \rho^{lmn} \sin(\alpha_l x) \sin(\beta_m y) \sin(\gamma_n z), \quad (3.58)$$

$$\phi(x, y, z) = \sum_{l=1}^{N_l} \sum_{m=1}^{N_m} \sum_{n=1}^{N_n} \phi^{lmn} \sin(\alpha_l x) \sin(\beta_m y) \sin(\gamma_n z), \quad (3.59)$$

其中 N_l, N_m, N_n 分别为电势和电荷密度在X, Y, Z三个方向上展开的阶数，而 ρ^{lmn} 和 ϕ^{lmn} 可以表达为：

$$\rho^{lmn} = \frac{8}{abc} \int_0^a \int_0^b \int_0^b \rho(x, y, z) \sin(\alpha_l x) \sin(\beta_m y) \sin(\gamma_n z) dx dy dz, \quad (3.60)$$

$$\phi^{lmn} = \frac{8}{abc} \int_0^a \int_0^b \int_0^b \phi(x, y, z) \sin(\alpha_l x) \sin(\beta_m y) \sin(\gamma_n z) dx dy dz, \quad (3.61)$$

其中

$$\alpha_l = \frac{l\pi}{a}, \beta_m = \frac{m\pi}{b}, \gamma_n = \frac{n\pi}{c}. \quad (3.62)$$

将上面 ρ 和 ϕ 的展开代入泊松方程，我们可以得到：

$$\phi^{lmn} = \frac{\rho^{lmn}}{\epsilon_0(\alpha_l^2 + \beta_m^2 + \gamma_n^2)}. \quad (3.63)$$

据此，我们得到了粒子密度 ρ 和电势 ϕ 之间的关系：

$$\phi(x, y, z) = \frac{1}{\epsilon_0} \frac{8}{abc} \omega \times \sum_{j=1}^{N_j} \sum_{l=1}^{N_l} \sum_{m=1}^{N_m} \sum_{n=1}^{N_n} \frac{\sin(\alpha_l x_j) \sin(\beta_m y_j) \sin(\gamma_n z_j) \sin(\alpha_l x) \sin(\beta_m y) \sin(\gamma_n z)}{(\alpha_l^2 + \beta_m^2 + \gamma_n^2)}, \quad (3.64)$$

其中 ω 是粒子的电荷。于是哈密顿量 H_2 可以表示为：

$$H_2 = \frac{1}{\epsilon_0} \frac{8}{abc} \omega \kappa \gamma_0 \times \sum_{i=1}^{N_i} \sum_{j=1}^{N_j} \sum_{l=1}^{N_l} \sum_{m=1}^{N_m} \sum_{n=1}^{N_n} \frac{\sin(\alpha_l x_j) \sin(\beta_m y_j) \sin(\gamma_n z_j) \sin(\alpha_l x_i) \sin(\beta_m y_i) \sin(\gamma_n z_i)}{(\alpha_l^2 + \beta_m^2 + \gamma_n^2)}. \quad (3.65)$$

最终，我们得到了保辛的空间电荷传输矩阵 \mathbf{M}_2 。以X方向为例：

$$x_i(\tau) = x_i(0), \quad (3.66)$$

$$p_{xi}(\tau) = p_{xi}(0) - \tau \frac{1}{\epsilon_0} \frac{8}{abc} \omega \kappa \gamma_0 \times \sum_{j=1}^{N_j} \sum_{l=1}^{N_l} \sum_{m=1}^{N_m} \sum_{n=1}^{N_n} \frac{\alpha_l \sin(\alpha_l x_j) \sin(\beta_m y_j) \sin(\gamma_n z_j) \cos(\alpha_l x_i) \sin(\beta_m y_i) \sin(\gamma_n z_i)}{(\alpha_l^2 + \beta_m^2 + \gamma_n^2)}. \quad (3.67)$$

在Y和Z方向对应的粒子推动方程的形式与之类似。

3.4 小结

本章介绍了几种描述和求解空间电荷效应的模型。早期的束核模型使用稳定束核假设，并对空间电荷力进行了线性近似，是一个非自治的模型。只能在一定程度上定性解释束晕形成的机制，而无法定量的求解出实际加速器中的空间电荷效应的影响。

PIC算法是一个更加自治的用来描述空间电荷效应的方法。束流模拟程序广泛的采用PIC算法来求解空间电荷力，在PIC算法中，粒子首先根据位置被权重到网格上，然后在网格上的求解泊松方程，以得到网格上的电势，再根据粒子位置反推出粒子所处位置的电场。PIC有效地降低了计算量，成为了国际上主流的空间电荷效应算法。束流模拟程序一般使用蛙跳法来推动粒子，其满足一致性、同时性、高准确性、高稳定性四个条件。

然而，由于PIC算法含有网格热噪声，会被引入一些非物理效应，我们引入了Symplectic模型。这种模型利用对粒子的高阶分解来求解空间电荷效应。相比

PIC 算法， Symplectic 模型计算量要大得多，但是能够显著的降低 PIC 算法中由于网格热噪声带来的发射度增长。

由于 Symplectic 算法计算量巨大， PIC 算法也有提高效率的需求，我们需要对模拟程序进行并行化处理。在下一章中，我们将对程序整体架构和各种算法在不同的并行计算平台上的实现进行介绍。

第四章 束流模拟程序的设计及并行实现

随着加速器的流强越来越高，束流模拟程序在束流动力学研究中的作用也越來越重要，尤其是C-ADS，SNS，ESS等加速器[44–46]，其中的非线性空间电荷效应不能忽略。一个拥有更强大的计算能力以及更高的精确度的程序，是我们研究强流束流动力学所必须的。

根据前一章中介绍的物理模型，在本章中我们对模拟程序的架构设计、代码实现、以及并行化优化策略进行介绍。我们首先介绍P-TOPO程序的整体结构，之后分别介绍了PIC算法在GPU上和CPU集群上的实现，和Symplectic在GPU上的实现。

4.1 P-TOPO程序

如节(1.2)所述，现有的束流模拟程序存在着一些不足，随着加速器的发展，目前存在的程序在功能和运行效率两方面都已经无法完全满足我们的需求。

在程序功能方面，随着对束流动力学问题的进一步研究，以及物理模型的进一步发展，现有的程序不能满足我们的需求。比如对于加速器中的束流共振穿越问题（详见节6.3），我们无法使用现有的程序进行模拟研究。而且，现有程序大部分都是闭源的，不方便我们根据自己的需求在底层进行改进。另外，现有程序的输出结果都是固定的，我们无法在需要的位置获取数据，比如大部分束流模拟程序都支持在元件入口或出口输出束流信息，但是我们无法指定在元件内的任一位置进行数据输出。

在运行效率方面，随着加速器越来越大，研究的问题所需要的精度越来越高，需要模拟的规模越来越大，模拟所需要的时间也越来越长。比如对于一些环形加速器动力学孔径的计算，随着Lattice的增大，现有程序的运算速度已经不能满足需求。另外，加速器优化程序也是加速器设计中必不可少的，比如根据入口和出口的Twiss参数进行磁铁参数优化等。加速器优化程序的基础也是一个快速而精确的底层模拟程序。

针对以上功能和运行效率两方面的考虑，我们需要研究并开发一个功能强大、效率高、灵活、易用、开源的束流模拟软件。依托于国内近年来设计或在建的ADS、CSNS等加速器项目，我们对强流束流模拟程序进行了很多研究，并在TOPO[47]的基础上开发了一个新的粒子模拟程序，命名为 Parallel - Trace of Particle Orbits (P-TOPO)，致力于研究强流加速器中的空间电荷效应问题[48–51]。

4.1.1 程序结构

P-TOPO使用C++语言开发，并且使用OpenMP进行并行化，目标是在普通多核PC机上快速模拟粒子在加速器中的行为。P-TOPO使用时间t，而不是使用位置z，作为基本的独立变量，这是研究粒子的空间电荷效应非常自然的选择。即时使用位置z作为基本变量，在求解空间电荷力的过程中，也需要将粒子坐标转换为同一时间t下的空间坐标。在P-TOPO中，产生外场的一些加速器元件，例如各种磁铁、螺线管、和RFQ的外场使用元件的解析模型得到；而对于另外一些元件，例如超导腔，我们首先读取场文件[52]，然后采用线性插值的方法获取粒子所受到的场的大小和方向。对于束团内部的空间电荷效应，我们使用经典的PIC方法进行求解。

P-TOPO程序的结构如图(4.1)所示，图中每一个条目为一个类：

- 主类MAIN调用其他各个具体的类，比如从外部Lattice结构中获取外场，求解内部空间电荷力，以及根据粒子所受的场推动粒子的位置和动量。
- Lattice类根据外部输入文件，解析或者数值地构建加速器的结构，以及根据粒子坐标，给出粒子所受的外场力。
- Distribution类产生粒子的初始分布，目前可以产生 KV分布，水袋分布，抛物线分布，和高斯分布四种粒子分布，如图(4.2)所示。
- Beam类统计粒子的信息，并且计算出如均方根大小，发射度等束团参数，将其保存以供进一步分析和输出。
- Internal Field类通过PIC方法计算空间电荷效应，在PIC方法中，我们使用FFT来求解Poisson方程。
- Leapfrog类和Runge-Kutta 4类都是继承Pusher类，是不同的推动粒子位置和动量的方法。

程序主循环中的大部分都是并行化的，例如求解内场，推动粒子等。最基本的，程序采用粒子并行策略，即每个线程处理不同的粒子。以粒子推动部分为例，如果粒子总数为160000，线程数为8，那么每个线程推动20000个粒子。对可能发生线程冲突的部分，例如权重插值和求解泊松方程部分，程序采用了网格并行策略，即不同的线程处理不同的网格区域。除此之外，程序也使用了FFTW库的内部并行方法。

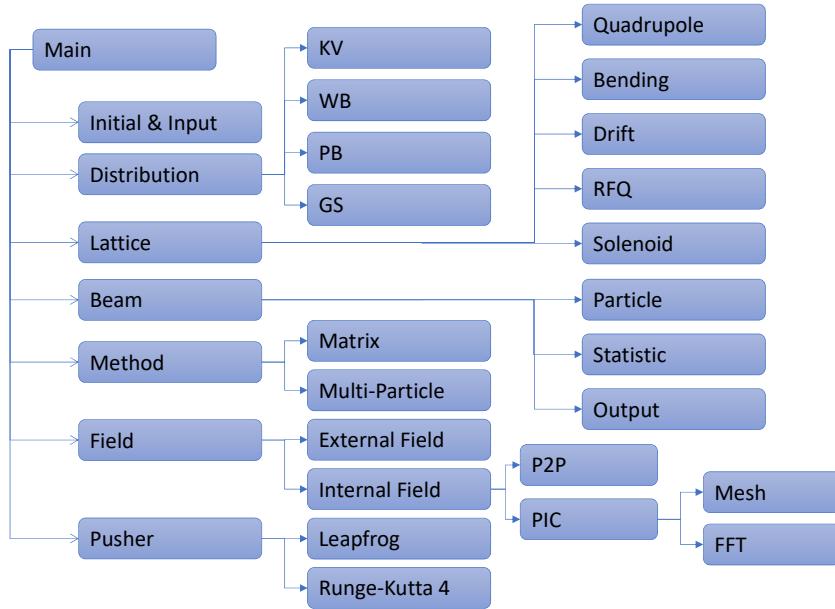


图 4.1: P-TOPO 程序结构示意图

我们使用了一个4核的普通PC机进行性能测试，4线程运行的程序的速度是串行运行程序的3.6倍。关于程序的更大规模的并行问题，我们会在后文做进一步讨论。

4.1.2 正确性校验

4.1.2.1 单粒子空间电荷场计算

我们使用一个点电荷来测试PIC部分的正确性。使用的格点数目为 $128 \times 128 \times 64$ 。横向边界为狄利克雷边界条件（第一类边界条件），纵向为周期性边界条件。图(4.3)是P-TOPO 的结果与理论值的对比，其中左图为横向电势（X方向），右图为纵向电势（Z方向）。图中红色实线为程序通过PIC方法计算得到的电势场的大小，绿色虚线为理论值，可以看出，程序的结果与理论值非常吻合。

4.1.2.2 FD结构中与解析解的对比

我们以束流在FD结构中束流的演化为例，来对P-TOPO程序在周期结构中的束流传输的结果进行验证。连续束在周期聚焦结构中的束流包络方程可以表示为式(4.1)[26, 53]

$$\frac{d^2R}{dz^2} + k_0^2 R - \frac{\varepsilon^2}{R^3} - \frac{K}{R} = 0, \quad (4.1)$$

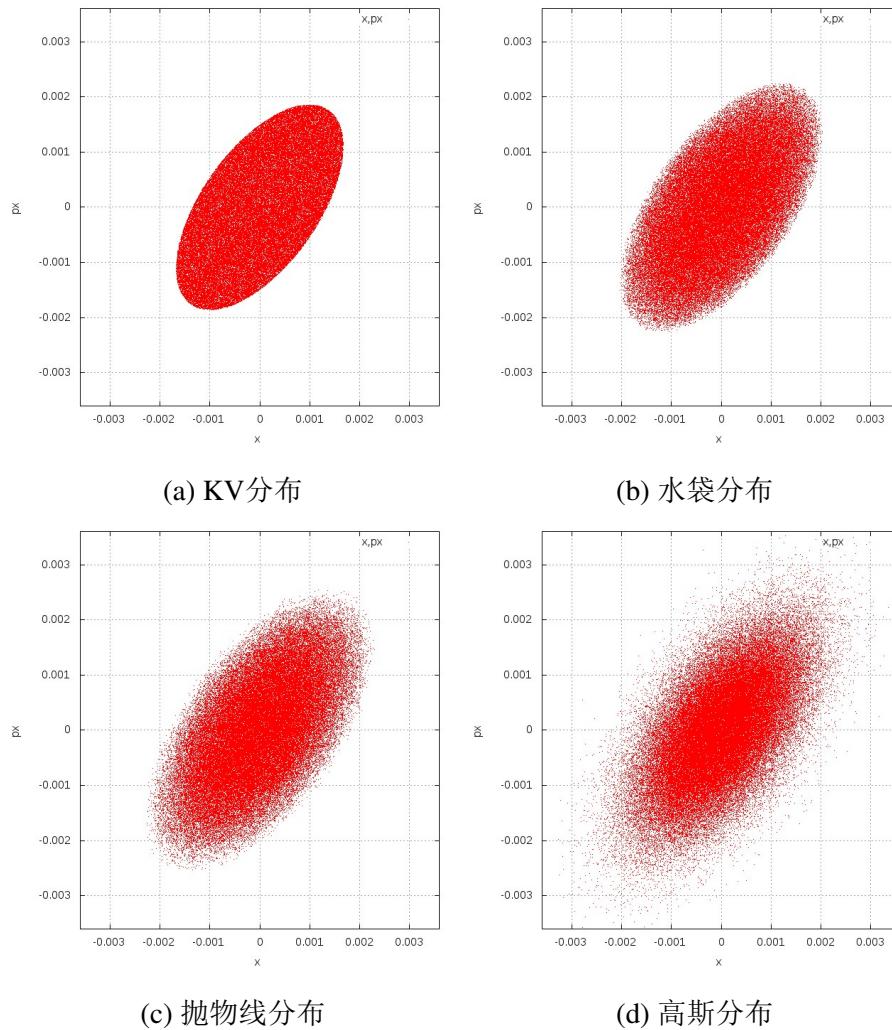


图 4.2: 同一均方根尺寸下的不同种类束团分布

其中 R 是束流横向均方根尺寸, k_0 是聚焦强度, 与外场力相关, ε 为发射度, K 是空间电荷力强度。

束流的横向均方根尺寸与初始匹配和相移有关。在这个验证中, 计算空间电荷力的网格数目为 64×64 , 我们使用10000个宏粒子在KV初始分布下, 分别使用流强为0mA和15mA在周期性FD结构中前进, 每个FD结构中推动400步。图(4.4)显示了由P-TOPO给出的束流均方根尺寸的演变和均方根包络方程的理论预期, 其中图片下部浅蓝色线表示外部四极铁的聚焦强度, 红色实线和绿色虚线分别为零流强下理论结果和P-TOPO程序计算结果, 而蓝色实线和紫色虚线分别代表15mA流强下理论预期和P-TOPO计算结果。可以看出, 无论是在零流强时还是15mA流强时, 计算结果与理论值都保持高度一致, 其中在15mA下束团演变的微小区别是因为包络方程计算中假定发射度为常数, 而实际的PIC计算中发射度会发生变化。

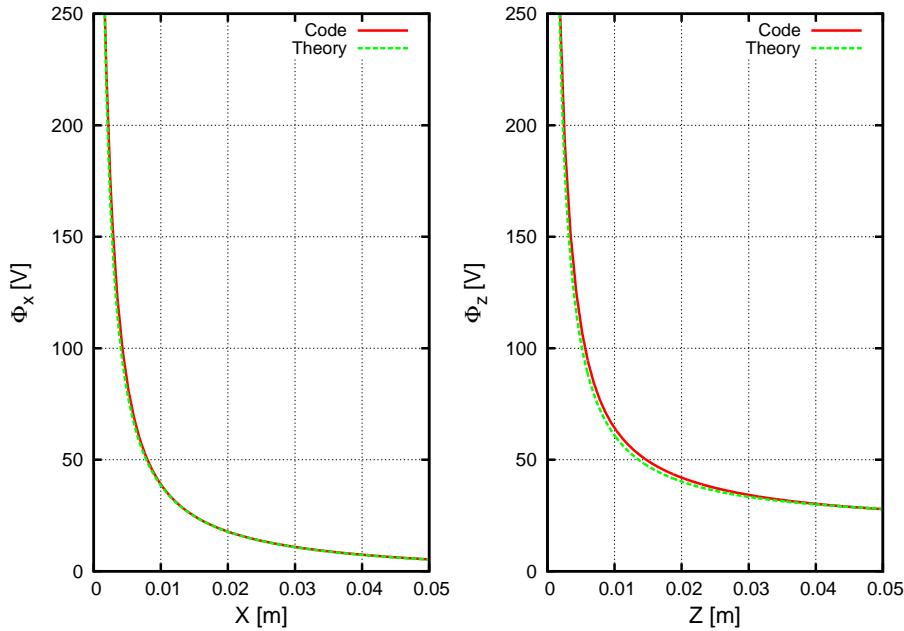


图 4.3: P-TOPO 与理论值的点电荷电势对比

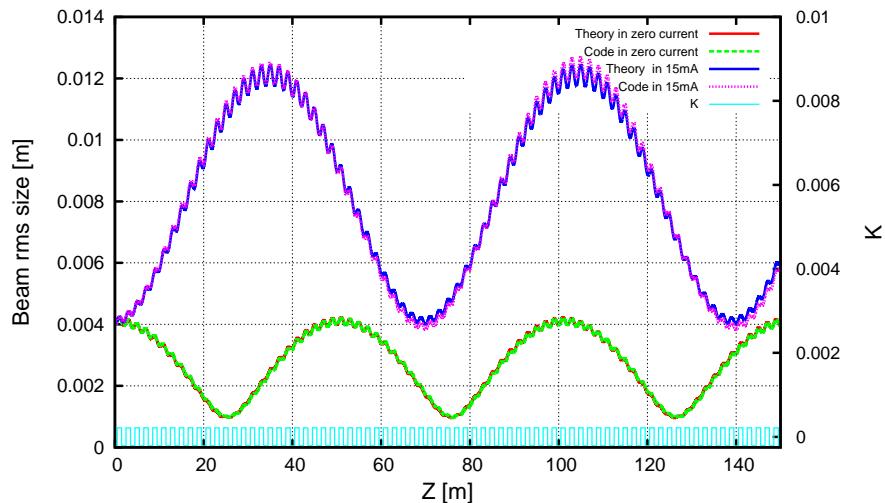


图 4.4: 零流强和15mA时, P-TOPO结果与包络方程的对比

4.2 PIC算法在GPU上的实现

在(3.2)节中我们已经讨论了基本的PIC算法。PIC算法主要包括四个部分：权重插值，解泊松方程得到电场，将电场反权重到粒子，推动粒子。本节将讨论如何在GPU上实现基于PIC算法的多粒子跟踪程序的并行化，提高程序的运行效率。

在GPU上实现PIC算法，主要难点在第一步权重插值上。第一步权重插值是将粒子逐个权重到网格上的过程，此步骤多线程并行运行时，会遇到不同线程相互竞争的问题（race condition），从而导致错误的结果。线程竞争是指在程序中，

协作的线程可能共享一些彼此都能读写的公用存储区，而竞争出现在当两个或多个线程同时访问同一内存地址，并且尝试写入数据的时候。由于线程调度器会以任意顺序运行多个线程，而我们无法得知线程尝试访问共享内存的顺序。因此，数据的最终结果会取决于线程调度器的算法，即多个线程“竞争”访问和改变数据。

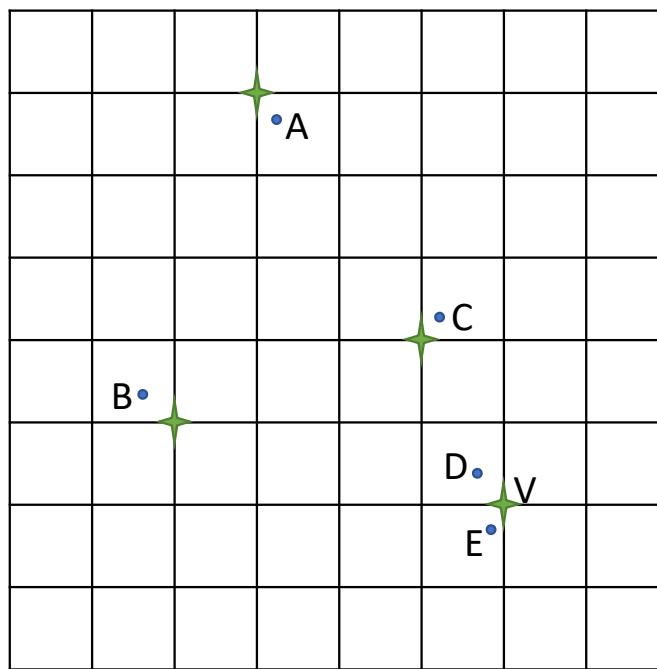
图(4.5)显示了PIC算法中权重插值部分的运算流程和多线程下可能会出现的竞争问题，其中蓝色点代表粒子，而绿色星状点代表与粒子相近的网格。使用多个线程运行权重插值时，通常使不同的线程处理不同的粒子。假设我们使用五个同一时钟的线程A-E，分别处理五个粒子，A, B, C三个线程并没有发生冲突，但是D线程和E线程会产生竞争，造成错误的运算结果。在第一个时钟周期内，线程D和E分别从同一内存地址M读取格点原本的电荷量V；在第二个时钟周期，线程D读取粒子D的电荷并计算相加到格点的电荷V+D，而线程E读取粒子E并计算得到V+E；在第三个时钟周期，线程D将V+D写入内存地址M，同时线程E将V+E写入同一内存地址M。但是两个写入都是错误，正确的结果应该是V+D+E，这就是PIC权重插值中的线程竞争。

为了避免线程之间的竞争和冲突，我们采取先将网格分块，再随后将网格上的电荷合并的做法。如图(4.6)所示，绿色部分为一个区块，且一个线程单独处理一个区块。想要分块处理，粒子必须是有序的，这样才能使相应的线程找到所对应区域的粒子。所以在权重插值之前，我们需要对粒子进行排序（reorder）。假设区块数目为N，我们需要声明N个数组，以分别对应N个区块中的粒子。在程序初始阶段，我们将粒子数据拷贝到不同的区块数组中，在随后每一步粒子被推动之后，粒子所处区块有可能会改变，所以我们要对粒子重新排序，以确保粒子处于所对应的区块数组。用这种方法，每一个线程能够单独处理自己所分配的区块，从而避免了线程之间的竞争和冲突。

加入了排序之后的PIC算法流程如图(4.7)所示，其中，标识为红色部分为粒子排序，是PIC算法在GPU上并行运行所必须的前置步骤；而黄色部分的GPU算法和普通CPU算法有较大区别，我们在下面，对流程图中的红色和黄色部分，即GPU上的PIC算法中的粒子排序，权重插值，解泊松方程，以及粒子推动做进一步讨论。

4.2.1 粒子排序

如上所述，在GPU上的PIC算法中，我们必须在每一步对粒子进行重新排序以避免线程竞争。因为排序算法的高度不规则，排序在GPU中并不能直接并行实现，特别是根据三维信息排序。在这里我们使用内存中的一块缓冲区作为数据中转空



线程 D	线程 E
1D: 读取变量 V	1E: 读取变量 V
2D: 计算 D + V	2E: 计算 E + V
3D: 写入 (V+D)	3E: 写入 (V+E)

但是正确结果应该为 $V + D + E$!

图 4.5: 权重插值中的线程冲突

间，以实现粒子排序算法在GPU上的并行运行。我们将整个网格划分为不同的区块，使每个线程只需处理对应的区块即可。例如，在一个网格数为 $64 \times 64 \times 64$ 的模拟中，我们将其分割成大小为 $4 \times 4 \times 4$ 的区块，总共 $16 \times 16 \times 16 = 4096$ 个区块，由每个线程单独处理相应的区块，这样可以避免线程的竞争。

在程序初始化的时候，我们首先需要为每一个区块声明数组并为其分配内存空间，如以下代码所示。在实际模拟中，每个区块的粒子数一般并不相同，使用内存预分配会导致浪费很多内存空间。但是不同于CPU内存，在GPU上分配内存耗时很长，所以无法在每一步计算中根据实际情况来进行内存的分配，只能采用预分配的方法，以提高程序运行效率。其中，我们根据实际使用的GPU的可用内存大小来确定每个区块的最大粒子数目，如果某一区块的粒子数目超过了最大粒子数，程序会报错并停止。在这种情况下，使用更少的粒子数目或者使用内存更大的显卡可以解决问题。

```
const int lth = BLOCKSIZE;
```

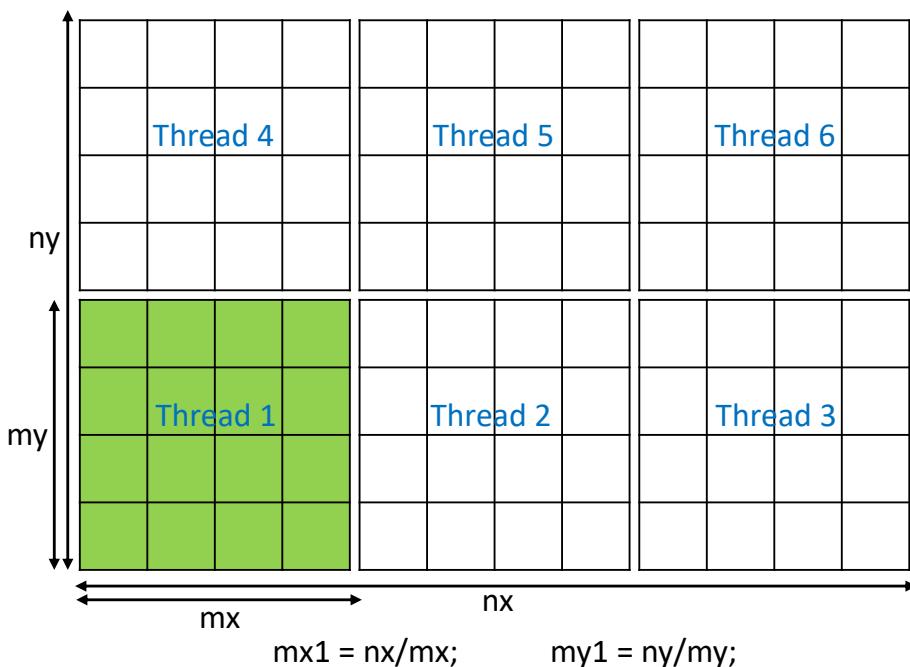


图 4.6: 网格分块示意图

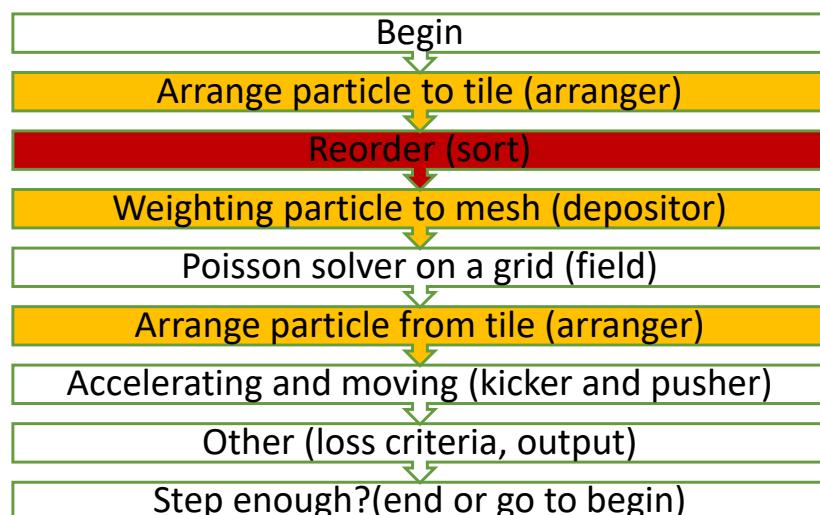


图 4.7: GPU上的PIC算法分段流程图

```

const int dim = _dimension;
int mth      = (mx1*my1*mz1-1)/lth+1; //number of tiles

double *ptc; //tiled particle
int   *kpic; //number of particles in each tile
int   *nhole; //number of hole left
int   *ncl;  //ndirec[k*26 + i]: number of particle going
             //to destination i[0-26) from tile k

```

```
double *pbuf;      //buffer for transfer particles

//get the maximum number of particle according to the GPU memory size
size_t freeMem, totalMem;
cuda_SafeCall(cudaMemGetInfo(&freeMem, &totalMem));
int npm = freeMem / 2 / sizeof(double) / dim / (mx1*my1*mz1);
if(npm>numberOfParticle) npm = numberOfParticle;
int MaxPtcTransf = npm / 2;

//allocate memory for particle array
size_t ptc_mem_size = sizeof(double)*lth*dim*npm*mth;
cuda_SafeCall(cudaMalloc((void**) &ptc,      ptc_mem_size));

//allocate memory for particle counting array
size_t kpic_mem_size = sizeof(int)*mx1*my1*mz1;
cuda_SafeCall(cudaMalloc((void**) &kpic,      kpic_mem_size));

//allocate memory for holes at tiles array
size_t nhole_mem_size = sizeof(double)*lth*2*(MaxPtcTransf+1)*mth;
cuda_SafeCall((cudaMalloc((void**) &nhole,  nhole_mem_size));

//allocate memory for particle buffer array
size_t pbuff_mem_size = sizeof(double)*lth*dim*MaxPtcTransf*mth;
cuda_SafeCall((cudaMalloc((void**) &pbuf,    pbuff_mem_size));

//allocate memory for transferred particle counting array
size_t ndirec_mem_size = sizeof(int)*mx1*my1*mz1*(3*3*3-1);
cuda_SafeCall((cudaMalloc((void**) &ndirec,  ndirec_mem_size));
```

在之后的每一步计算中，在空间电荷效应之前，我们都需要根据粒子空间位置对粒子进行排序，排序流程如图(4.8)所示。每个线程首先统计对应区块中将要离开当前区块的粒子信息，并使用数组nhole和ndirec记录粒子的序号和每个方向的数目，如图(4.8)中黄色箭头所示。在上面的代码中，我们使用最大粒子数目的一半作为最大穿越数目，而nhole已经根据最大穿越数目进行了内存预分配。

之后，我们将离开当前区块的粒子信息拷贝到一个缓冲区pbuf。作为全局内存，pbuf同样根据最大穿越数目进行了内存预分配。ndirec包括了每个方向上的粒子数目，对ndirec进行迭代求和（running sum），我们可以得到每个方向上的内存初始位置，以便于将前往某个方向的粒子在pbuf中连续排布。

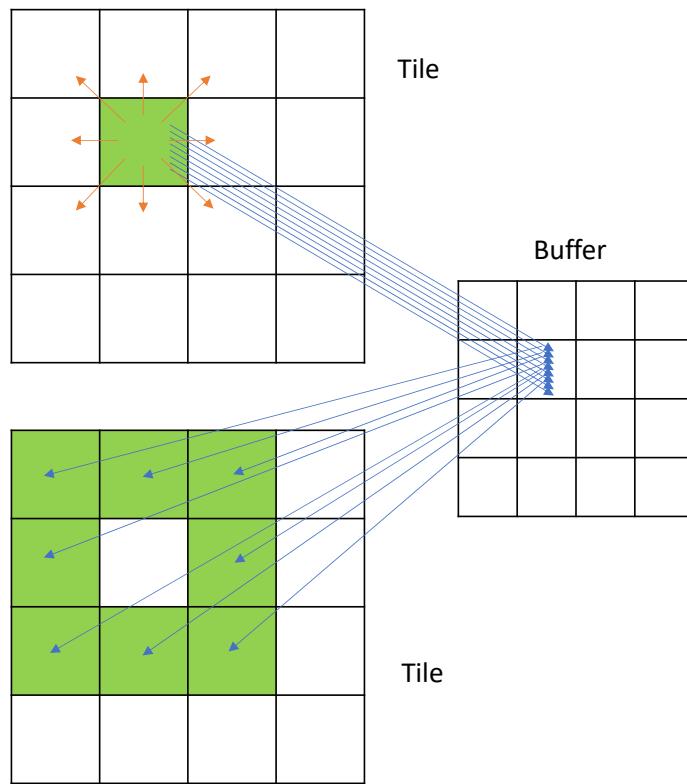


图 4.8: 粒子排序示意图

最后，对于每个区块，经过第一步的统计，我们可以知道会有多少粒子进入，以及它们在**pbuff**中的位置。在将粒子信息从**pbuff**拷贝到当前区块**ptc**的过程中，我们先填满因为粒子离开造成的空洞。如果进入的粒子数目大于空洞数目，即在所有空洞都被填满后，仍有粒子进入，我们将其写入到当前区块**ptc**的最后；但如果进入的粒子数目小于空洞数目，即写入所有进入粒子后仍有空洞存在，则从本区块粒子数组**ptc**尾部开始，依次移动粒子数据到空洞处。通过这种方式，粒子信息在内存中的连续性得到了保证。

我们对上述步骤做了一个简要总结：

1. 首先，对本区块内的中因为位置改变而不再属于本区块的粒子序号，以及其离开的方向记录下来，写入**nhole** 和 **ndirec**。
2. 其次，根据其序号和方向，将离开的粒子拷贝进入全局缓存区**pbuff**。
3. 最后，再根据相邻区块离开粒子的序号和方向，确定将要进入本区块的粒子，将缓存区内的数据拷贝到本区块。

通过这些流程，使每个线程仅处理对应的区块，来确保排序过程中不会出现线程竞争。

4.2.2 权重插值

通过以上粒子排序，我们使每一个区块的粒子相对独立，这样我们可以使每个线程处理一个区块，从而避免了线程冲突的情况。一个直接的权重插值流程如图所示(4.9) 首先，每个线程将自己所处理的粒子权重到一个小网格上，得到本地的网格密度分布。然后，我们将所有本地的密度分布结合到一起，得到全局密度分布。对于边界处的网格点我们使用各个子网格点的电荷密度叠加得到。

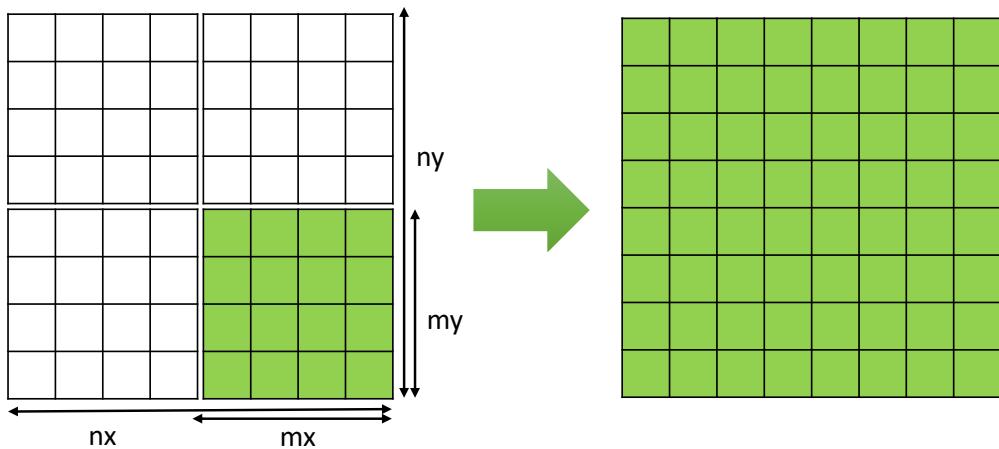


图 4.9: 权重插值示意图

上述为单GPU的权重插值方法，在使用多个GPU的时候，我们有两种方法：一种是使不同的GPU处理不同的空间域，在最后进行通讯，这种方法在CPU上的PIC程序中很常用，被称为“域分解”；另外一种是简单直接的使所有GPU做同样的工作，我们称这种模式为“复制模式”。下面，我们使用一个例子对这两种方法进行比较，该例使用 $64 \times 64 \times 64$ 个格点，区块大小为 $4 \times 4 \times 4$ ，区块的数目为 $16 \times 16 \times 16$ ，在TITAN上使用4个GPU运行。

4.2.2.1 域分解模式

在域分解模式中，每个GPU只需要处理相对应的域。假如区块数目为 $16 \times 16 \times 16$ ，在4个GPU上运行的话，每个GPU需要处理的域的大小为 $4 \times 16 \times 16$ 。但是，这种模式需要预先将粒子排序，使每个GPU只处理对应空间域内的粒子，因此需要额外的通讯和计算，具体步骤如下：

1. GPU之间的粒子排序：

- (a) 挑选粒子，每个线程处理一个区块，因此我们共有 $4 \times 16 \times 16 = 1024$ 个

线程。然而，在TITAN上每个GPU有2688个核，线程数小于核数，我们无法充分使用GPU；

- (b) GPU之间信息交换：
 - i. 从GPU内存拷贝到CPU内存，共需要拷贝 $4 \times 16 \times 16 \times (nGPU - 1) \times nPtcMax$ 个粒子，其中 $nPtcMax$ 为最大传输粒子数；
 - ii. CPU和CPU之间的信息交换，我们使用MPI_send/receive，共需要交换 $4 \times 16 \times 16 \times (nGPU - 1) \times nPtcMax$ 个粒子；
 - iii. 从CPU内存拷贝到GPU内存，共需要拷贝 $4 \times 16 \times 16 \times (nGPU - 1) \times nPtcMax$ 个粒子；
- 2. GPU内部的粒子排序，每个GPU的区块数目为 $4 \times 16 \times 16 = 1024$ ，小于在每个GPU上的核数2688；
- 3. GPU内部的权重插值，格点数目为 $16 \times 64 \times 64$ ，但是由于第一步进行了GPU之间的粒子排序，所以每个GPU上的粒子数目不同，会导致负载不平衡；
- 4. GPU之间的格点信息归约：
 - (a) 从GPU内存拷贝到CPU内存，共需要拷贝 $16 \times 64 \times 64$ 个格点；
 - (b) CPU和CPU之间的信息交换，我们使用MPI_Allgather，共需要交换 $64 \times 64 \times 64$ 个格点；
 - (c) 从CPU内存拷贝到GPU内存，共需要拷贝 $64 \times 64 \times 64$ 个格点。

4.2.2.2 复制模式

在复制模式中，每个GPU进行同样的工作，同域分解模式相比，复制模式无法使用多GPU来节省计算时间，但是也无需在GPU之间进行粒子信息的交换。

复制模式的流程如下所示，其中每一项和上面域分解模式相对应：

- 1. 无需GPU之间排序；
- 2. GPU内部的粒子排序，每个GPU的区块数目为 $16 \times 16 \times 16 = 4096$ ，大于在每个GPU上的核数2688；
- 3. GPU内部的权重插值，每个GPU上的粒子数都是相等的；
- 4. GPU之间的格点信息归约；

- (a) 从GPU内存拷贝到CPU内存，共需要拷贝 $64 \times 64 \times 64$ 个格点；
- (b) CPU和CPU之间的信息交换，我们使用MPI_Allreduce，共需要交换 $64 \times 64 \times 64$ 个格点；
- (c) 从CPU内存拷贝到GPU内存，共需要拷贝 $64 \times 64 \times 64$ 个格点。

可以看出，两个模式相比较的话，域分解模式在第一步中多了很多额外的通讯，在第二步与第三步中能够使用更少的格点数。然后，使用通讯时间来换取更少的计算量并不能带来效率的提升，而且，在域分解模式下，我们并不能充分利用GPU。因此，在我们的代码中以及接下来的性能测试中，我们均采用“复制模式”。

4.2.3 泊松方程

在将粒子权重到网格上之后，下一步即求解网格上的泊松方程。我们在此不再赘述单GPU上运行的泊松方程求解器，而在多GPU上的并行版本和权重插值相似，也存在两种方法：一种是参考经典的CPU上的PIC程序，使用不同的GPU处理不同的空间域，即“域分解模式”；另一种是直接使所有GPU做同样的工作，即“复制模式”。由于解泊松方程的运算量较大，是PIC方法中很重要的一步，我们对两种模式都进行了实现并进行了比较。

域分解的优点在于，通过使用不同的GPU处理不同的空间域，每个GPU可以降低计算量，从而提高程序运行速度。其缺点也很明显，域分解模式需要在不同的GPU之间交换信息，而在目前GPU与GPU之间，特别是在不同节点之间，并不能直接进行信息交换，而是必须通过CPU来进行，即信息交换需要三步：

1. 从GPU内存拷贝到CPU内存；
2. CPU和CPU之间的通过MPI或其他通讯接口进行信息交换；
3. 从CPU内存拷贝回GPU内存。

如节(3.2.2)所述，求解泊松方程最主要的部分是进行快速傅里叶变换（FFT）。在GPU代码中，我们使用NVIDIA公司的CUDA快速傅里叶变换库（cuFFT）[54]。在求解多GPU上的三维傅里叶变换时，最好是在三个方向上分别做一维的傅里叶变换，并在每次变换之间对数据进行转置。

假设在X, Y, Z三个方向上的格点数分别为 N_x, N_y, N_z ，那么当执行X方向的傅里叶变换的时候，变换的数组长度即为 N_x ，变换的次数为 $N_y \times N_z$ 。以使用四

个GPU为例，那么1号GPU需要处理的数据为 $\text{rho}[N_x][0 \rightarrow \frac{N_y}{4}][N_z]$ ，2号GPU需要处理的数据为 $\text{rho}[N_x][\frac{N_y}{4} \rightarrow 2\frac{N_y}{4}][N_z]$ ，3号GPU需要处理的数据为 $\text{rho}[N_x][2\frac{N_y}{4} \rightarrow 3\frac{N_y}{4}][N_z]$ ，4号GPU需要处理的数据为 $\text{rho}[N_x][3\frac{N_y}{4} \rightarrow N_y][N_z]$ 。每个GPU只需要进行 $\frac{N_y}{4} \times N_z$ 次变换即可。理想情况下，和单GPU运行相比，使用4个GPU运行只需要花费1/4的时间。

在X方向的傅里叶变换结束后，我们需要其他的数据来进行Y方向的运算，目前每个GPU上的数据为 $\text{rho}[N_x][(n-1)\frac{N_y}{4} \rightarrow (n)\frac{N_y}{4}][N_z]$ ，而进行Y方向的傅里叶变换需要的数据为 $\text{rho}[(n-1)\frac{N_x}{4} \rightarrow (n)\frac{N_x}{4}][N_y][N_z]$ 。我们必须对数据求转置并在GPU之间进行数据交换。而由于目前GPU与GPU之间并不能直接进行数据交换，我们必须将GPU中的数据拷贝回CPU内存中，并在CPU端进行通讯，这将需要额外的时间。所以域分解模式相比复制模式的效率高低将取决于额外的通信时间与降低的运算时间的比较。

以 $64 \times 64 \times 64$ 个格点数，使用4个GPU为例，域分解模式求解泊松方程的具体步骤可以表示如下：

1. X方向的FFT

- (a) 拷贝到GPU (64*16*64)
- (b) FFT (64*16*64)
- (c) 拷贝到CPU (64*16*64)

2. 转置及CPU与CPU之间的信息通讯

3. Y方向的FFT

- (a) 拷贝到GPU (64*16*64)
- (b) FFT (64*16*64)
- (c) 拷贝到CPU (64*16*64)

4. 转置及CPU与CPU之间的信息通讯

5. Z方向的FFT

- (a) 拷贝到GPU (64*16*64)
- (b) FFT (64*16*64)
- (c) 拷贝到CPU (64*16*64)

相比较而言，复制模式的步骤为：

1. X方向的FFT ($64 \times 64 \times 64$)
2. Y方向的FFT ($64 \times 64 \times 64$)
3. Z方向的FFT ($64 \times 64 \times 64$)

我们将在节(5.1.1)介绍更详细的性能比较。

4.2.4 粒子推动

在节(4.2.2)粒子权重过程中，粒子是按照区块排列的。因此在推动粒子的时候，最直接的方法是像权重过程中一样，由每个线程处理一个区块中的粒子。然而这种方法的缺陷是每个线程上的负载不均匀，可能出现某些区块中粒子数很多从而其线程运算量很大，然而另外某些区块中的粒子数较少，其线程运算量很小的情况。

为了保证负载的均衡，我们采取另外一种方法，先把区块中的粒子信息重组，拷贝到一个典型的 $6 \times N$ 的连续数组中，其中 N 是粒子数，6是维度；然后再并行推动。这种方法虽然需要额外的时间进行数据拷贝，但是相比第一种区块推动的负载更均匀，所以整体时间比区块推动更短，其流程如下所示：

1. 将粒子从区块格式`dev_ray_tile[lth*dim*npm*mth]`重新排列，并拷贝到经典的 $6 \times N$ 数组中`dev_ray[N][6]`；
2. 对`dev_ray[N][6]`进行并行推动；
3. 推动完成后，将粒子信息拷贝回原区块`dev_ray_tile[lth*dim*npm*mth]`中，以用来下一步的粒子排序和权重插值。

4.2.5 正确性校验

模拟程序最重要的是确保模拟结果的正确性，我们通过将GPU程序的结果与成熟的CPU程序的结果进行比较来验证GPU程序的正确性。图(4.10)是CPU程序和GPU程序的发射度比较，从图中可以看出，两个程序计算得到的结果完全相同。输出数据得到，两者的区别在 10^{-13} 量级，这可能是由于双精度浮点数的精度所限，双精度浮点数的精确度在 10^{-14} 左右。而一般我们输出只取九位有效数字，所以可以认为两个程序的输出结果完全相同，GPU程序的正确性得到了验证。

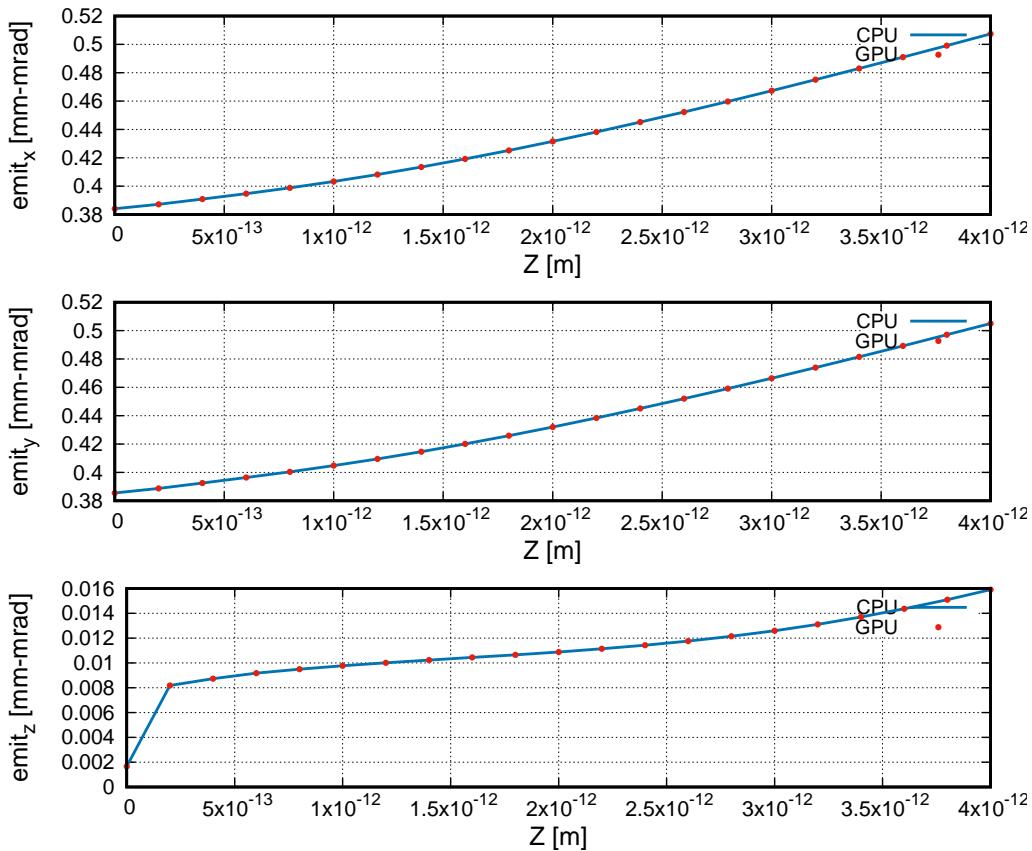


图 4.10: PIC 算法在 GPU 上的正确性校验

4.3 PIC算法在CPU集群上的实现

除了GPU集群之外，PIC代码还在基于因特尔众核（MIC）技术的超级计算机Cori Knight Landing（KNL）上进行了移植和测试。我们将OpenMP（OMP）指令添加到原先的纯MPI代码中，使用了MPI和OpenMP混合并行，以适应MIC架构，提高运行的效率。新的MPI/OpenMP混合PIC程序提供了节点间和节点内的两级并行机制，它结合了进程级的粗粒度并行和线程级的细粒度并行。

我们对 MPI 进程数量和 OpenMP 线程数量之间的平衡进行了精细调整，在一个节点上研究了不同OpenMP线程数和MPI进程数下程序的效率和内存占用情况，并找到最优的混合并行配置。对于程序中某些部分（如权重插值）中可能存在的线程竞争条件，我们使用了OpenMP内置的归约操作和原子操作符。而且，程序使用了新的编译器，以充分利用KNL上的512位矢量协处理器，可以显着提高程序中循环块的速度。KNL有16 GB的板载多通道DRAM(MCDRAM)内存，带宽超过460 GB/s。MCDRAM有两种模式：“缓存模式”（作为L3缓存）或“平坦模式”（作为更快的内存）。经过对两种模式的测试，最后程序选择以“缓存模式”

使用 MCDRAM 以获得较高的效率。之后，我们使用多个节点，研究程序在不同节点数下的效率变化。我们将在第(5.1.5)节介绍更详细的性能研究。

4.4 Symplectic算法在GPU上的实现

除了PIC算法，我们也对Symplectic算法进行了GPU实现，使用GPU可以显著加快Symplectic粒子跟踪代码的运行速度。我们已经在节(3.3)中介绍了 Symplectic 算法，在实际模拟程序中，对于大多数元件，粒子会被推动若干步。在每一步中，粒子将首先被外场传输矩阵推动半个时间步长，然后由空间电荷推动一个时间步长，再由外场传输矩阵传输推动半个时间步长。其中，求解空间电荷将消耗整个计算时间的90%以上。计算空间电荷效应的部分由遍历三角函数、计算 Φ^{lmn} 、计算 ep_i 并推动粒子三个子程序组成，而每个子程序由一个或两个内核组成。

我们对程序进行了许多优化，使其更适合GPU架构，显着改善了程序的性能。每个子程序的优化策略各不相同，下面介绍各个内核的详细优化方法。

4.4.1 遍历三角函数

根据粒子的位置，我们首先计算每个粒子的三角函数临时变量。设

$$\begin{aligned} S_j^l &= \sin(\alpha_l x_j), & C_j^l &= \cos(\alpha_l x_j), \\ S_j^m &= \sin(\alpha_m x_j), & C_j^m &= \cos(\alpha_m x_j), \\ S_j^n &= \sin(\alpha_n x_j), & C_j^n &= \cos(\alpha_n x_j), \end{aligned} \quad (4.2)$$

其中，下标 j 指不同的粒子，下标 l , m 和 n 指三个方向的频谱模数。X方向上的传输矩阵 \mathbf{M}_2 ，即式(3.67)，可以表示为：

$$p_{xi}(\tau) = p_{xi}(0) - \tau \frac{8}{\varepsilon_0 abc} \omega \kappa \gamma_0 \sum_{j=1}^{N_j} \sum_{l=1}^{N_l} \sum_{m=1}^{N_m} \sum_{n=1}^{N_n} \frac{\alpha_l S_j^l S_j^m S_j^n C_i^l S_i^m S_i^n}{(\alpha_l^2 + \beta_m^2 + \gamma_n^2)} \circ \quad (4.3)$$

与式(3.67)相比，新的传输矩阵节省了大量的计算量。计算 S_j 和 C_j 是非常有必要的，这样可以避免后期处理的重复计算。在这个子程序中，我们采用按照粒子分配线程的策略，即每个线程处理一个粒子。这段程序结构相对简单，只占整个空间电荷效应求解所需时间的2%左右。然而，它生成了 $2 \times (N_l + N_m + N_n) \times N_j$ 的数据，占了大部分的内存空间，而GPU的内存大小在给定型号的情况下是一个固定值，并不能像CPU端一样通过添加内存条来增加。在这个意义上，这段程序决定了模拟中粒子数目的最大值，是花费存储空间以节省计算量的典型示例。根

据我们的测试，考虑到不可避免的内存碎片，假设展开阶数为 $64 \times 64 \times 64$ ，程序可以在GeForce GTX 1060 6GB上处理大约100万个粒子。

由于需要遍历内存，该子程序的速度受到全局内存带宽的限制。为了改进内存读写，粒子数据是以阵列结构（SoA）排列的形式而不是结构数组（AoS），以达到对齐读取（coalesced memory read）。并且，在CPU侧分配页锁存存储器，以实现更快的CPU与GPU之间的数据复制。

4.4.2 计算 Φ^{lmn}

我们注意到在式(4.3)中，对下标 j 的求和是对每个粒子的求和，故我们可以改变求和的顺序以节省计算量。定义：

$$\Phi^{lmn} \equiv \sum_{j=1}^{N_j} S_j^l S_j^m S_j^n. \quad (4.4)$$

如果首先完成 $N_l \times N_m \times N_n$ 个 Φ^{lmn} 的计算。式(4.3)可以重写为：

$$p_{xi}(\tau) = p_{xi}(0) - \tau \frac{1}{\varepsilon_0} \frac{8}{abc} \omega \kappa \gamma_0 \sum_{l=1}^{N_l} \sum_{m=1}^{N_m} \sum_{n=1}^{N_n} \frac{\Phi^{lmn} \alpha_l C_i^l S_i^m S_i^n}{(\alpha_l^2 + \beta_m^2 + \gamma_n^2)}, \quad (4.5)$$

以这种方式，程序的计算复杂度从 $\alpha N_p^2 * N_{modes}$ 减少到了 $\alpha N_p * N_{modes}$ ，这使这个保辛粒子跟踪算法的计算量大大降低。

该子程序的目的是为每个展开模计算 Φ^{lmn} ，自然地使用每个线程处理一个模的方式。然而，在通常情况下，我们使用 $16 \times 16 \times 16$ 模式就可以得到收敛的结果，也就是说会有 $16 \times 16 \times 16 = 4096$ 个线程，而GPU通常具有几百甚至几千个核心，这个线程数量相对于的GPU的核心数目来说较少。为了实现负载平衡，我们将粒子分为几个部分，并采用CUDA的流技术来获得高并发性。我们定义临时变量 $\Phi_{temp,i}^{lmn}$ ：

$$\Phi_{temp,i}^{lmn} \equiv \sum_{j=Nstart_i}^{Nend_i} S_j^l S_j^m S_j^n, \quad (4.6)$$

那么， Φ^{lmn} 由以下求和得到：

$$\Phi^{lmn} = \sum_i \Phi_{temp,i}^{lmn}. \quad (4.7)$$

这个子程序的速度也是受到内存带宽的限制。另外，我们在计算 Φ^{lmn} 之前，会首先对 S_j 进行转置以达到对齐读取。

4.4.3 计算 ep_i 并推动粒子

以x方向为例，我们定义：

$$ep_{xi} \equiv \tau \frac{1}{\varepsilon_0} \frac{8}{abc} \omega \kappa \gamma_0 \sum_{l=1}^{N_l} \sum_{m=1}^{N_m} \sum_{n=1}^{N_n} \frac{\Phi^{lmn} \alpha_l C_i^l S_i^m S_i^n}{(\alpha_l^2 + \beta_m^2 + \gamma_n^2)} \circ \quad (4.8)$$

我们首先根据前两节中得到的 Φ^{lmn} ， S_j 和 C_j 计算得到 ep_{xi} ，随后，使用 ep_{xi} 对粒子进行推动：

$$p_{xi}(\tau) = p_{xi}(0) - ep_{xi} \circ \quad (4.9)$$

由于GPU寄存器数量的限制， ep_i 的计算和粒子的推动在X，Y，Z三个方向上分别进行，以提高GPU占用率和运算效率。在这个子程序中，三个方向的最内层循环会以不同的顺序访问 Φ^{lmn} ，所以在调用此子程序前我们需要对 Φ^{lmn} 进行转置，以实现对齐读取，提高效率。

在此段程序中我们根据粒子来分配线程。由于受到GPU的常量内存和共享内存的大小限制，这段子程序有两个分支：一个是阶数小于 $20 \times 20 \times 20$ 的情况，另一种是阶数大于 $20 \times 20 \times 20$ 的情况。

4.4.3.1 分支1（阶数小于 $20 \times 20 \times 20$ ）

当阶数小于 $20 \times 20 \times 20$ 时，我们使用常量内存保存 Φ^{lmn} 。常量内存专门针对内存广播进行了优化，多个线程同时访问同一常量内存地址的速度很快，正适合需要被每个线程访问的 Φ^{lmn} 。一个普通GPU中的常量内存总量为65536字节，只能容纳8192个双精度浮点数。正是这个常量内存的大小，决定了小于 $20 \times 20 \times 20$ 和大于 $20 \times 20 \times 20$ 两个分支的分界点。

这个分支中的内核程序使用共享内存来存储最内部的循环中 S_j 和 C_j 。共享内存是一种片上内存，内存大小较小，每个GPU的每个流处理器只有64kb，但是共享内存的速度远远快于全局内存。由共享内存大小限制，这段程序的GPU占用率仅为25%，但是共享内存的访问延迟要比全局内存低大约100倍。

我们还进行了一次全局内存和共享内存的速度测试。在这个测试中，我们通过数据结构的设计来使让warp中的所有线程访问连续的内存地址，尽量避免全局内存的访问延迟。由于不再受共享内存大小的限制，GPU占用率可以达到接近100%。然而，由于全局内存访问的延时，程序运行花费的时间接近使用共享内存程序的两倍。

4.4.3.2 分支2（阶数大于 $20 \times 20 \times 20$ ）

当阶数大于 $20 \times 20 \times 20$ 时，受限于共享内存和常量内存的大小，上述直接计算 ep_i 并推动粒子的方式将无法有效工作，我们必须进行改变。

首先， Φ^{lmn} 被存储在全局内存中。在这段程序中多个线程会访问相同内存地址，而由于GPU全局内存的合并读取机制，使用全局内存的速度只比使用常量内存的速度慢10%。另外，受限于共享内存的大小，我们将 ep_i 的计算和推送粒子分开。而在 ep_i 的计算中，不同的阶被分为若干部分来，分别使用共享内存。它类似于小节(4.4.2)中对 Φ^{lmn} 的计算。每个粒子会使用若干线程，而每个线程处理相应的阶，并获得临时变量 $ep_i^{temp,j}$ ：

$$ep_i^{temp,j} \equiv \tau \frac{1}{\varepsilon_0} \frac{8}{abc} \omega \kappa \gamma_0 \sum_{l=1}^{N_l} \sum_{m=1}^{N_m} \sum_{n=Nstart_j}^{Nend_j} \frac{\Phi^{lmn} \alpha_l C_i^l S_i^m S_i^n}{(\alpha_l^2 + \beta_m^2 + \gamma_n^2)} \quad (4.10)$$

然后我们对 $ep_i^{temp,j}$ 求和，并根据式(4.9)推动粒子：

$$ep_i = \sum_j ep_i^{temp,j} \quad (4.11)$$

将 ep_i 分成若干部分分别计算也有相应的缺点，分开计算需要更多的内存空间来保存 $ep_i^{temp,j}$ 。额外的内存使用量与粒子数和阶数成正比。在相同的内存大小分别计算的最大粒子数会比直接计算的减少约20%。

4.5 小结

本章首先介绍了根据PIC算法编写的粒子模拟程序P-TOPO，这个程序主要目标是处理强流加速器中的非线性效应，其正确性都得到了验证。

基于PIC算法提高效率的需求，我们在不同的并行计算机平台上对PIC程序进行了实现。在GPU上，我们使用CUDA库实现了基于PIC方法的多粒子模拟程序，并优化了GPU代码结构，使用特定的并行策略避免了线程竞争，使程序的性能得以提高。我们也新的CPU集群Cori Knight Landing上实现了PIC程序，并探索了其最佳性能的并行线程配置。

最后，我们在GPU上实现了Symplectic算法。由于Symplectic算法计算量巨大并且结构非常规则，而GPU在并行计算密集型问题方面具有很大优势，因此Symplectic算法非常适合使用GPU进行加速计算。我们针对每个子程序按照不同的方式进行并行GPU实现，以尽可能的提高程序的效率。在下一章中，我们会在多个计算平台上对束流模拟程序和不同的算法进行优化和测试。

第五章 束流模拟程序的优化和性能测试

随着加速器束流流强的提高，为了对束流进行精确的模拟和研究，我们需要模拟的粒子数目增加了几个量级。在这种趋势下，束流模拟程序的效率至关重要。本文在第(四)章介绍了束流模拟程序在GPU和CPU上的并行实现，接下来在本章中，节(5.1)将对PIC算法在多种计算平台的性能测试进行介绍；节 (5.2)将对Symplectic算法的GPU并行性能进行了测试。

5.1 PIC程序性能

我们使用了多种计算平台对并行PIC程序的性能进行优化和测试 [55–57]，如

- 单GPU - GTX1060
- GPU集群 - Titan
- GPU集群 - SummitDev
- CPU集群 - Cori Knight Landing

Titan是一台GPU/CPU混合架构的超级计算机，位于美国橡树岭国家实验室 (ORNL)，它使用的是由NVIDIA提供的Tesla计算卡和AMD提供的Opteron处理器 [58]。SummitDev同样位于ORNL，是下一代超级计算机Summit的早期测试平台，每个SummitDev节点都有2个IBM POWER8 CPU和4个NVIDIA Tesla P100 GPU [59]。Cori Knight Landing (KNL) 是Cori的新计算平台，位于美国国家能源研究科学计算中心 (NERSC)。KNL基于Intel众核架构 (MIC)，每个节点有68个内核和一个AVX-512矢量协处理器 [60]。

使用单GPU和不同集群的测试综合性地给出了不同计算机架构上PIC代码的性能评估。下面，如我们在节(4.2.3)中讨论的，我们首先比较泊松方程求解在域分解模式和复制模式下的性能，然后再介绍在各个计算平台上运行的PIC代码的性能结果。

5.1.1 泊松方程求解

5.1.1.1 单GPU测试

首先，我们在单GPU上对域分解模式进行了测试，并统计了每一部分所花费

的时间。测试的格点数为 $64 \times 64 \times 64$, 每部分所花费的时间如表(5.1), 表中的单位为秒, 每部分占比如图(5.1)所示。

表 5.1: 单GPU域分解模式解泊松方程耗时 (秒)

从CPU拷贝到GPU	1.306
从GPU拷贝到CPU	2.979
FFT计算	0.487
转置和通讯	3.772
总计	8.544

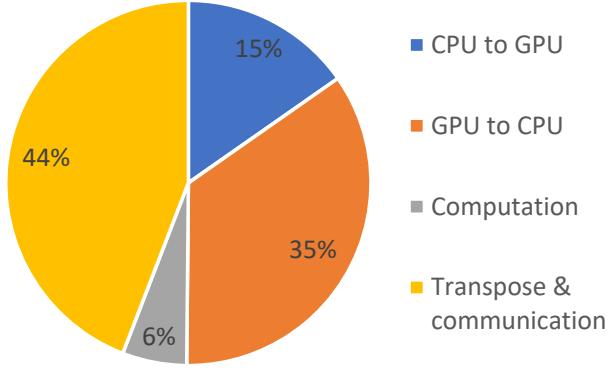


图 5.1: 单GPU域分解模式解泊松方程各部分耗时占比

可以看出, FFT计算所花费的时间为0.487秒, 相比于额外增加的数据拷贝时间 $1.306 + 2.979 + 3.772 = 4.933$ (秒), 其占比非常小, 仅占总时间的6%。

多GPU的一种方式是在同一节点内使用多个GPU, 因为节点内的总线宽度限制, 数据拷贝时间并不会随着使用GPU数目的增加而明显减少。而复制模式只需要花费FFT计算的时间, 因为FFT计算花费的时间0.487秒远小于数据拷贝的时间4.933秒, 所以即使在理想情况下, 使用N个GPU时FFT计算的时间减少到原来的 $1/N$, 域分解模式的总时间依然是比复制模式要大。

多GPU的另一种方式是在多个跨节点上使用多个GPU, 即每个节点只有一个GPU, 通过使用多个节点来使用多个GPU, 这也是目前超级计算机TITAN的运行模式。总线宽度会随着节点数增加而增加, 所以数据拷贝时间会下降, 假设其下降为线性的, 即在N个节点上运行的话拷贝数据时间只有原来的 $1/N$, 则总时间将会是 $4.933/N + 0.487/N$, 如果我们想要取得优于复制模式的速度, 即 $4.933/N + 0.487/N < 0.487$, GPU数目必须大于11。类似的, 如果我们想要取得两倍的速度, GPU数目必须大于22。然而, 这个计算忽略了CPU和CPU之间的通讯时间, 一般

来说，节点间的通讯时间随着节点数目增加而增加，因此我们很难取得理想情况的加速比。

下面，我们分别对节点内多GPU和跨节点GPU进行测试。

5.1.1.2 单节点多GPU测试

在同一节点中，我们进行了格点数为 $64 \times 64 \times 64$ 的多GPU测试，表(5.2)是各个部分所花费的时间，以及和单GPU时的时间对比。可以看出，双GPU的总时间更大了，由8.544秒增加到了9.537秒。其中，FFT计算的时间由0.487秒减少到了0.249秒，几乎减少了一半，证明了多GPU对于减少纯FFT计算时间是确实有效的。对于数据拷贝时间，双GPU略有减少，大概减少了三分之一。总时间中增加的部分来源于通讯的时间，几乎增加了一倍。对于单节点多GPU，由于通讯开销，

表 5.2: 单节点双GPU域分解模式解泊松方程耗时 (秒)

	1GPU	2GPU
从CPU拷贝到GPU	1.306	0.817
从GPU拷贝到CPU	2.979	1.991
FFT计算	0.487	0.249
转置和通讯	3.772	6.480
总计	8.544	9.537

多GPU花费时间反而更大，所以我们不推荐使用单节点上的多GPU。

5.1.1.3 跨节点多GPU测试

我们使用Titan测试了域分解模式下求解泊松方程在跨节点多GPU上的效率。使用常用的 $64 \times 64 \times 64$ 个格点，域分解模式下解泊松方程所花费的时间随GPU个数的变化如图(5.2)和表(5.3)所示，图中蓝线为总时间，各个柱代表各个部分所花费的时间。

可以看出，程序总耗时随着GPU数目增加而逐渐减少，并在32个GPU的时候到达最小值，随后开始逐渐增加。其中，拷贝数据所花费的时间基本随着GPU的数目线性减少；而通讯所花费的时间先随着GPU数目增加而逐渐减少，到达极值后又开始逐渐增加，此时通讯时间占程序所花费的时间的绝大部分；FFT计算所花费的时间随着GPU数目的增加而逐渐减少，但是FFT计算占总时间的百分比很小。

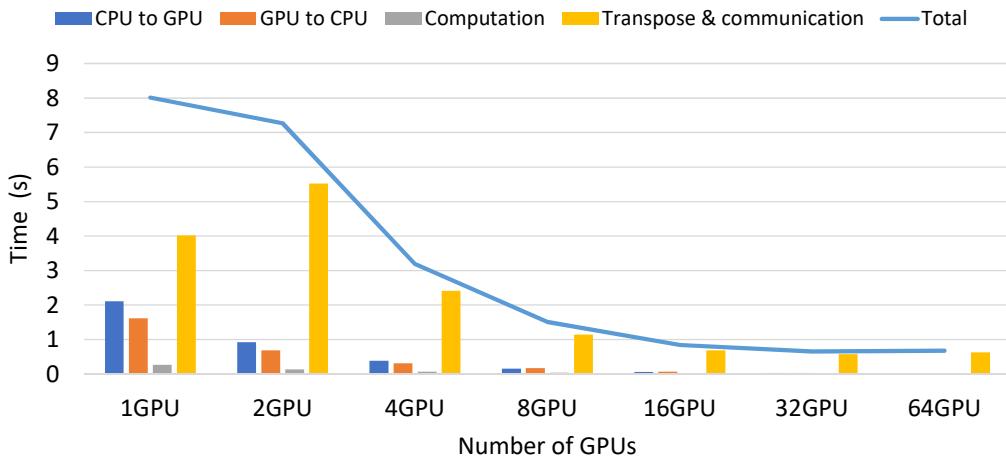


图 5.2: $64 \times 64 \times 64$ 格点下，跨节点多 GPU 域分解模式解泊松方程时间随 GPU 个数的变化

表 5.3: $64 \times 64 \times 64$ 格点下，跨节点多 GPU 域分解模式解泊松方程各部分所用时间

GPU数目	1	2	4	8	16	32	64
CPU to GPU	2.111	0.922	0.390	0.158	0.061	0.031	0.0155
GPU to CPU	1.618	0.694	0.314	0.167	0.066	0.033	0.0158
FFT计算	0.266	0.136	0.069	0.039	0.024	0.017	0.0150
转置和通讯	4.020	5.521	2.415	1.146	0.691	0.572	0.6320
总计	8.015	7.273	3.188	1.51	0.842	0.653	0.6783

我们也测试了格点数目更大的情况，格点数目越多，所需要的计算量越大。当格点数为 $128 \times 128 \times 128$ 时，程序在不同的 GPU 数目下消耗的时间如图(5.3)和表(5.4)所示，其整体的趋势和 $64 \times 64 \times 64$ 时类似，因为更大的计算量，其总时间在使用 128 个 GPU 时到达最小值。

在 $64 \times 64 \times 64$ 和 $128 \times 128 \times 128$ 两种情况下，总时间开始都随着 GPU 数目增加而减小，然后到达最小值，随后由于通讯所需要的时间变大，总时间也随之变大。然而，无论是使用 $64 \times 64 \times 64$ 个格点还是 $128 \times 128 \times 128$ 个格点，总时间的最小值都大于只用一个 GPU 的纯计算时间。

通过上面对于“域分解模式”的实现和测试，可以看出程序运行速度主要受限于 CPU 和 GPU 之间数据拷贝的速度与 CPU 和 CPU 之间的通讯速度。如果 GPU 带宽足够大，或者以后能够实现 GPU 与 GPU 之间的直接通讯的话，“域分解模式”可以作为一个可行的方案。但是在目前，与“复制模式”相比，“域分解模式”由于其额外的数据拷贝和通讯开销，并不能通过减少运算量来达到提高速度的目

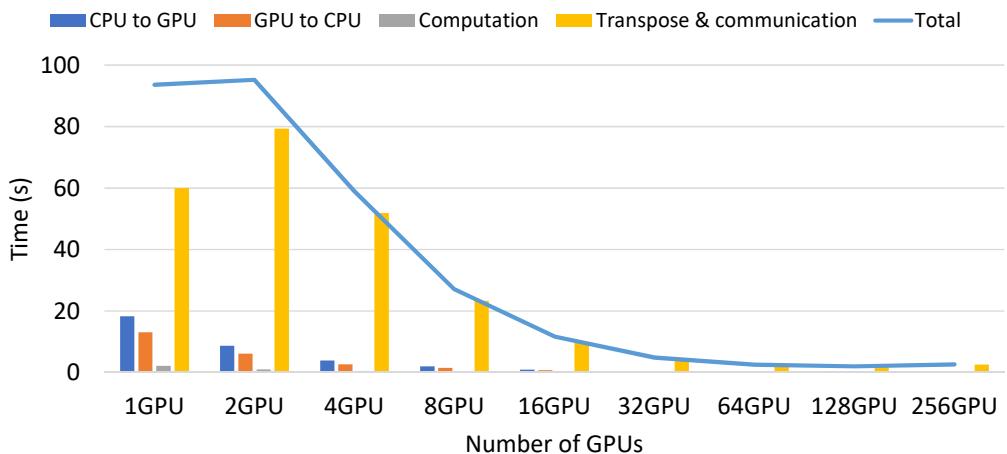


图 5.3: $128 \times 128 \times 128$ 格点下，跨节点多 GPU 域分解模式解泊松方程时间随 GPU 个数的变化

表 5.4: $128 \times 128 \times 128$ 格点下，跨节点多 GPU 域分解模式解泊松方程各部分所用时间

GPU数目	1	2	4	8	16	32	64	128	256
CPU to GPU	18.29	8.687	3.946	2.023	0.953	0.404	0.148	0.064	0.033
GPU to CPU	13.11	6.15	2.618	1.463	0.795	0.368	0.139	0.07	0.036
FFT计算	2.185	1.034	0.522	0.266	0.138	0.071	0.036	0.021	0.017
转置和通讯	59.99	79.34	51.85	23.30	9.797	4.041	2.237	1.859	2.506
总计	93.58	95.21	58.94	27.05	11.68	4.884	2.56	2.014	2.592

的。故我们在程序中以及在下文中都使用“复制模式”（除非另加说明），即使所有的 GPU 都同时运行同样的程序。

5.1.2 单 GPU 性能-GTX1060

在测试了泊松方程求解器之后，我们通过对比 CPU 程序和单个 GPU 程序的运行速度，得到 GPU 程序的加速比。测试使用的格点数为 $64 \times 64 \times 64$ ，加速比等于 CPU 程序耗时除以 GPU 程序耗时。图(5.4)和表(5.5)是粒子数从 16k 到 1.6M 时 CPU 程序和 GPU 程序各部分耗时以及加速比。可以看出，程序总耗时的加速比大约有 40，程序各个部分的加速比有很大不同。

其中，橙色柱行代表求解泊松方程，其加速比大约是 64，而且并不随粒子数目的变化而变化。事实上，求解泊松方程的计算量主要和格点数相关，而此次

测试中我们都是用同样的格点数，因此其耗时和加速比基本不变，之后我们会比较在不同格点数情况下求解泊松方程的加速比变化。灰色柱行代表推动粒子的加速比，其随着粒子数目增加而变大，在粒子数较大时取得超过了70的加速比。浅蓝色和深蓝色柱行分别是权重插值和粒子信息输出的加速比，其中GPU的权重插值部分包括了粒子排序运算，而且由于其运算的不规则性，加速比较低；而粒子信息输出部分也包含了束团参数的计算，这部分加速比较低是因为输出带宽的限制。

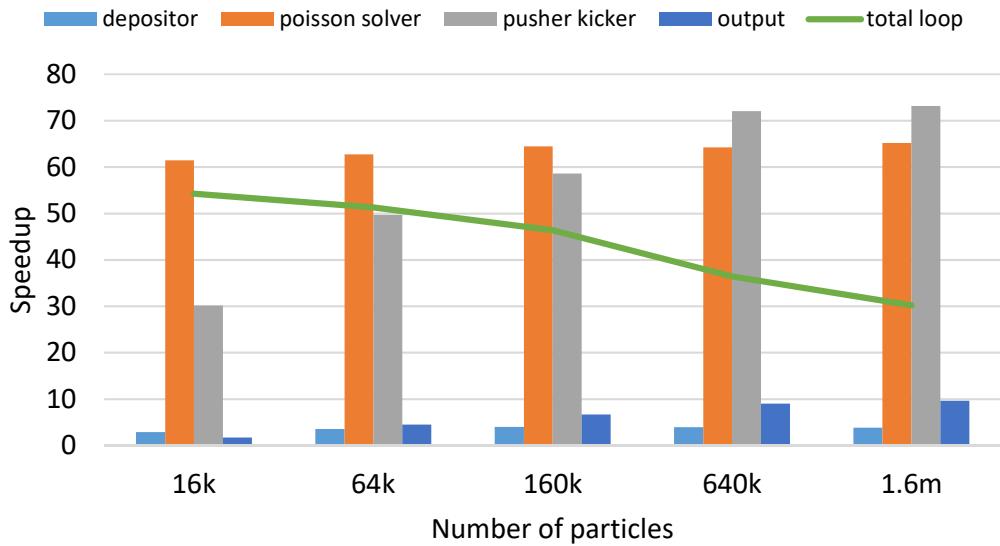


图 5.4: PIC 程序在单 GPU 上的加速比

权重插值和粒子信息输出的加速比较低，从而拉低了整体的加速比。整体的加速比随着粒子数的增加而逐渐变小，其原因是当粒子数目较小时的时候，求解泊松方程所占得比重较大，从而整体加速比较大；而当粒子数目变大的时候，权重插值所占的时间比重变大，因而整体加速比随之变小。程序各部分耗时在不同粒子数所占比重如图(5.5)所示。

在上述测试中使用的都是 $64 \times 64 \times 64$ 个格点数，因此求解泊松方程的时间和加速比都基本不变。在不同的格点数情况下，求解泊松方程的加速比如图(5.6)所示。可以看出，求解泊松方程的加速比随着格点数的增加逐渐变大，最终达到了接近70，这是因为格点数越大，其计算量越多，GPU上的负载能够分布得更均匀。对于实际模拟中常用的 $64 \times 64 \times 64$ 个格点和 $128 \times 128 \times 128$ 个格点中，我们都取得了令人满意的加速比。

总之，在单GPU上，程序总体的加速比达到了40，而求解泊松方程的加速比超过了60。在一个普通的家用机GPU上运行的速度相当于在64核机器上的两倍。

表 5.5: PIC程序在单GPU上的加速比

16k	CPU(s)	GPU(s)	Speedup
权重插值(包括排序)	0.16888	0.05874	2.875043
求解泊松方程	26.13173	0.42511	61.47051
粒子推动	0.53748	0.01781	30.17855
输出	0.01977	0.01149	1.720627
总耗时	27.78216	0.51199	54.26309
64k	CPU(s)	GPU(s)	Speedup
权重插值(包括排序)	0.3897	0.10847	3.592698
求解泊松方程	26.08269	0.41554	62.76818
粒子推动	2.00422	0.04032	49.70784
输出	0.0641	0.01433	4.473133
总耗时	29.54123	0.57598	51.28864
160k	CPU(s)	GPU(s)	Speedup
权重插值(包括排序)	0.82705	0.20731	3.989436
求解泊松方程	25.9208	0.40208	64.46677
粒子推动	4.88644	0.08343	58.56934
输出	0.15477	0.02316	6.682642
总耗时	32.94187	0.71	46.397
640k	CPU(s)	GPU(s)	Speedup
权重插值(包括排序)	2.79413	0.71529	3.90629
求解泊松方程	25.72129	0.40045	64.23097
粒子推动	22.48269	0.31207	72.04374
输出	0.62289	0.06931	8.987015
总耗时	53.51193	1.46745	36.46593
1.6M	CPU(s)	GPU(s)	Speedup
权重插值(包括排序)	6.7528	1.73779	3.885855
求解泊松方程	26.03071	0.39894	65.24969
粒子推动	56.05512	0.76562	73.21533
输出	1.56528	0.16288	9.61002
总耗时	90.40391	2.99053	30.23006

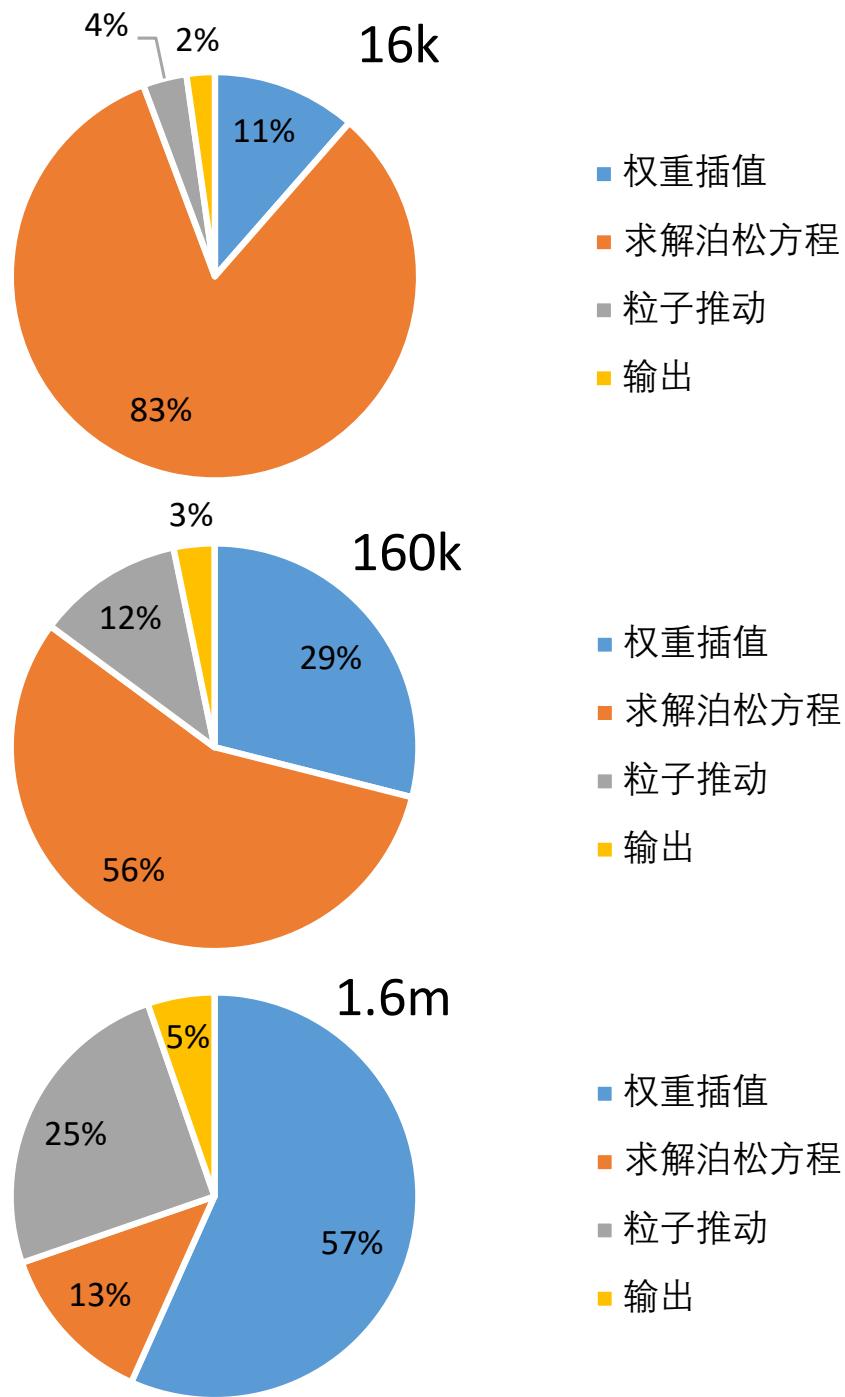


图 5.5: 程序各部分耗时在不同粒子数时所占百分比

另外，如小节(4.2.1)所述，因为GPU内存大小一般是固定的，而不像CPU内存那样较容易的扩展，所以单GPU的粒子数目存在某个上限，当超过最大粒子数时，应该使用多GPU来进行计算。在接下来的一节中，我们将讨论PIC程序在多GPU上的效率。

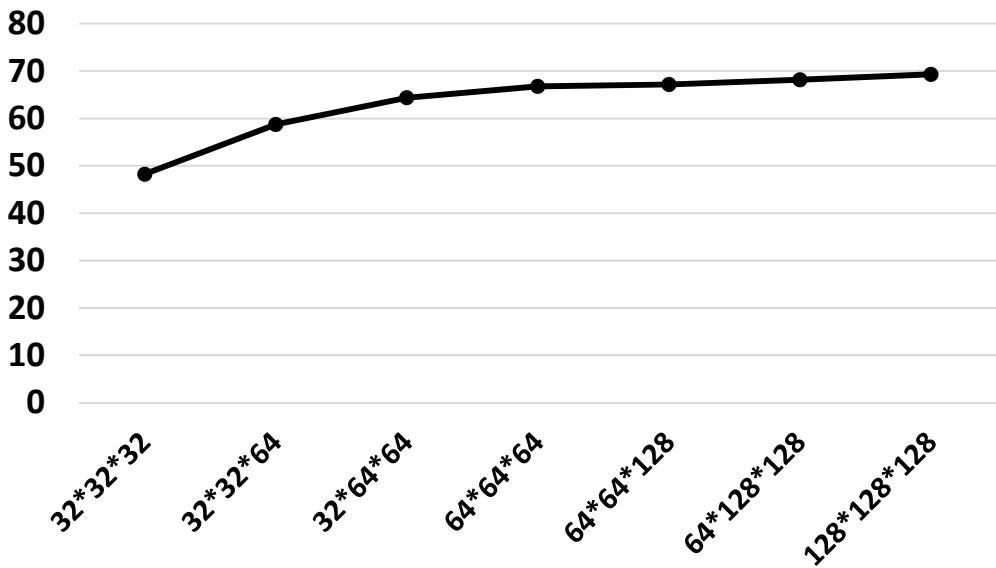


图 5.6: 不同的格点数情况下单GPU求解泊松方程的加速比

5.1.3 GPU集群性能-Titan

在单GPU测试之后，我们使用GPU集群Titan对PIC Cuda程序进行了多GPU性能测试。Titan使用CPU和GPU混合架构，是目前世界上速度最快、规模最大的计算机集群之一。Titan中每个节点只有一个GPU，因此只能通过使用多个节点来使用多个GPU。在这种条件下，GPU之间的通讯只能够通过先将数据拷贝到CPU端，再通过CPU端的节点间网络进行通讯。

图(5.7)为求解泊松方程时各个部分所花费的时间随GPU个数的变化。可以看出，因为处于复制模式，求解泊松方程的耗时基本保持不变。更多的GPU个数意味着每个GPU上的粒子数随之减小，所以粒子推动的耗时随着GPU数目的增加而减少；而由于通讯成本随着GPU数目增加而增加，信息统计和输出耗时也随之增加。各个部分对于GPU的数目变化的反应并不相同，单从总体来看，程序总耗时基本保持不变。

图(5.8)和图(5.7)类似，也是 $64 \times 64 \times 64$ 各个点，160k个粒子下程序总耗时随着GPU数目的变化。但是在图(5.8)中，GPU数目大于等于2的时候，求解泊松方程使用了域分解模式。其在各种情况下，都相较复制模式耗时更多，主要是拷贝数据花费了大量时间，这个结果和我们在节(4.2.3)中讨论的相符。

当粒子数更大时，耗时的趋势发生了变化。图(5.9)是粒子数为一百六十万（1.6M）的结果，粒子数较图(5.7)变大了十倍。在使用1.6M个粒子时，总耗时随着使用的GPU数目增加而明显下降，并在32个GPU处到达最小值。这是因为在大粒子数目情况下，推动粒子和权重插值所消耗的时间占总时间的比重更大。而由于

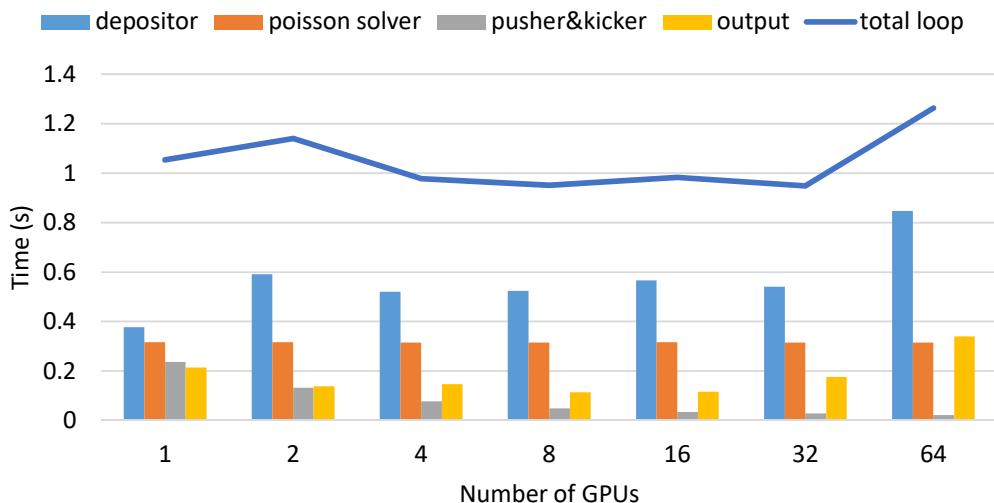


图 5.7: $64 \times 64 \times 64$ 个格点， $160k$ 个粒子时，Titan 上 PIC 程序耗时随 GPU 个数的变化

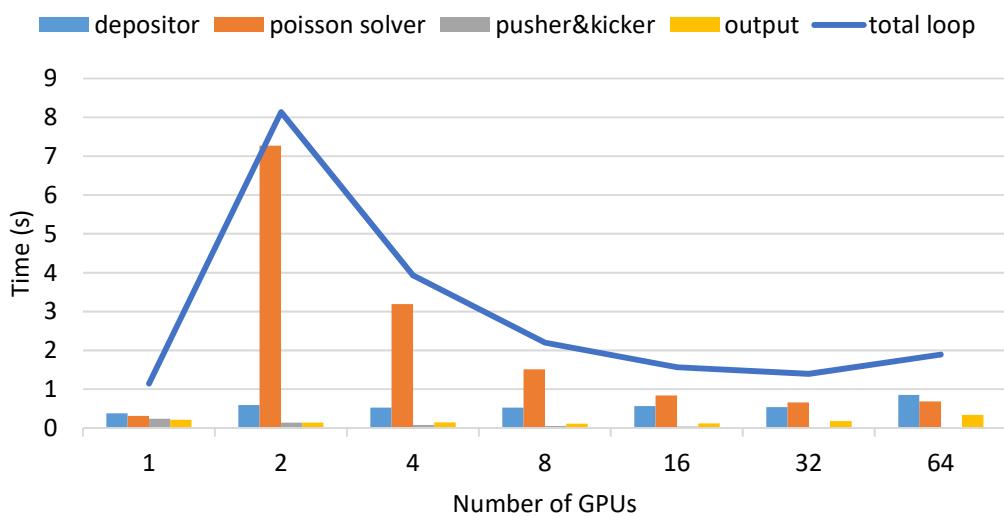


图 5.8: $64 \times 64 \times 64$ 个格点， $160k$ 个粒子时，Titan 上域分解模式 PIC 程序耗时随 GPU 个数的变化

使用多GPU带来的每个GPU上的粒子数目变少，推动粒子和权重插值能够很好的被多GPU加速。

我们尝试了使用更大的粒子数，一千六百万个粒子（16M），更十倍于前，如图(5.10)所示。由于GPU内存大小的限制，程序在这个粒子数下不能仅仅使用一个或两个 GPU 运行，因此在图(5.10)中，GPU数目等于1和等于2的时候没有数据。对于更多的GPU数目，程序耗时随着GPU数目增加而减小。

超级计算机Titan上使用的GPU型号为NVIDIA K20x，每个GPU有5GB的内存。理想情况下，一个5GB内存能够使用的最大粒子数为六千万（60M）。但这个粒子数是在使用空间均匀分布下才能达到，而实际上，我们很难达到这个粒子数。在

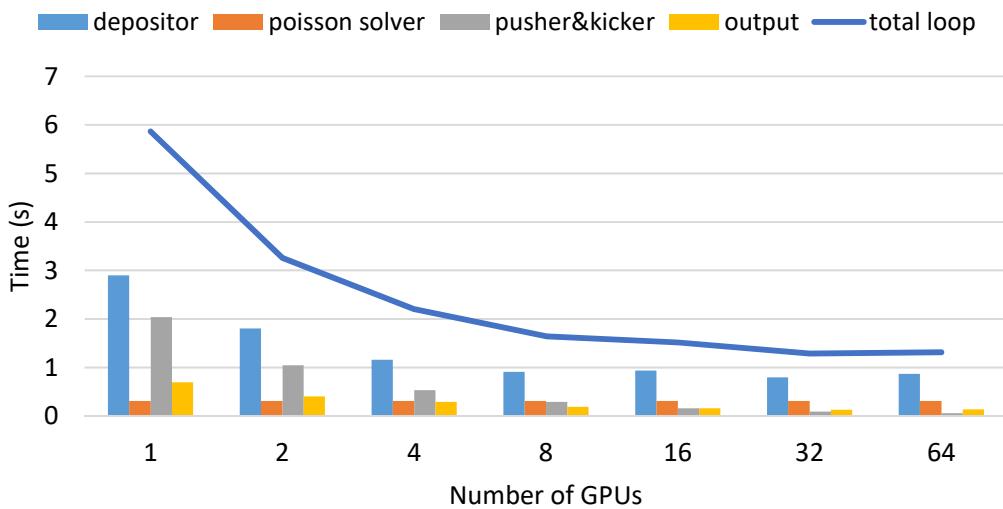


图 5.9: $64 \times 64 \times 64$ 个格点, 1.6M 个粒子时, Titan 上 PIC 程序耗时随 GPU 个数的变化

一个实际的加速器模拟中, 我们通常使用水袋分布或者高斯分布, 因此其可用的粒子数目远小于理想数目。

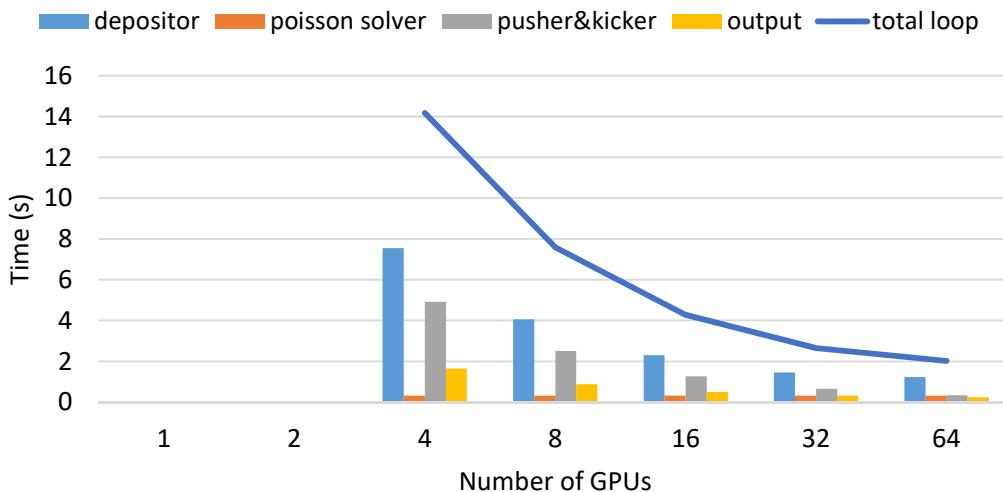


图 5.10: $64 \times 64 \times 64$ 个格点, 16M 个粒子时, Titan 上 PIC 程序耗时随 GPU 个数的变化

5.1.4 GPU 集群性能-SummitDev

和上一节中在 Titan 上进行的性能研究类似, 我们还在 ORNL 新一代的 GPU 集群 SummitDev 上对 PIC-Cuda 程序进行了测试。图 (5.11) 为使用 $64 \times 64 \times 64$ 个格点和 160k 个粒子的程序耗时。和 Titan 的耗时趋势一样, 随着 GPU 数量的增加,

粒子推动所消耗的时间减少，权重粒子所消耗的时间有所增加。而总时间并不随着GPU数量的增加而减少。多GPU不能提高效率的原因是SummitDev上GPU的计算能力非常强大，对于160k个粒子来说，纯计算在整体耗时中所占比例不大，使用一个GPU已经足够。

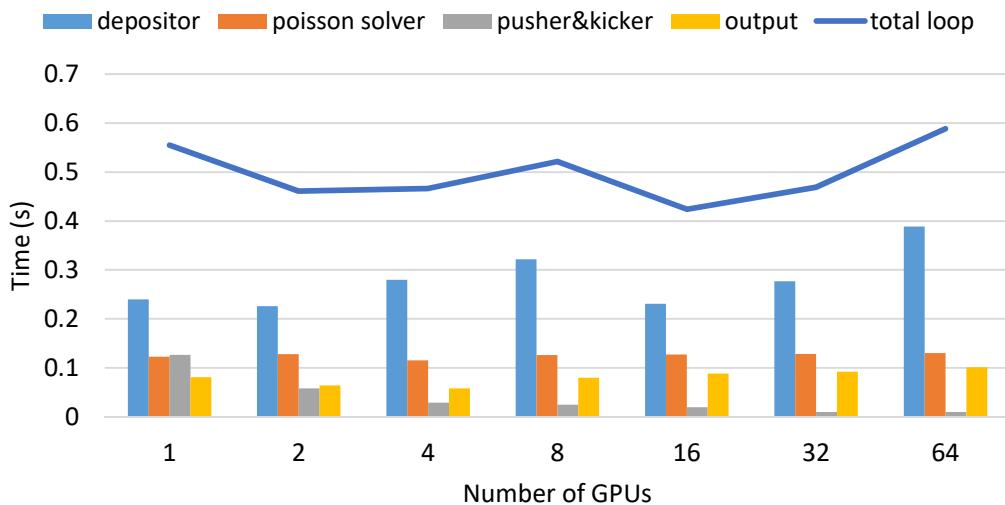


图 5.11: $64 \times 64 \times 64$ 个格点，160k 个粒子时，SummitDev 上 PIC 程序耗时随 GPU 个数的变化

当粒子数量变大时，耗时趋势发生了变化。图 (5.12)显示了1.6M个粒子的结果。总耗时随着GPU数量的增加而减少，并在16个GPU时到达最小值。与Titan上的运行时间相比，由于计算能力的提高，对于同一规模的问题在SummitDev上所需要的时间减少了50%。图 (5.13)为了16M个粒子的结果，受GPU内存大小的限制，程序必须使用两个或两个以上GPU运行。在粒子数很大的情况下，由于计算量较大，程序能够充分使用多个GPU的性能，所以总耗时随着GPU数目的增加而单调递减。

5.1.5 CPU 集群性能-KNL

为了和GPU做比较，我们也在CPU集群KNL上使用MPI 和OpenMP混合并行实现了高效率的PIC程序。首先，我们使用集群的一个节点来研究不同 OpenMP 线程数和MPI进程数下程序的效率和内存占用情况，并找到最优混合并行配置。之后，我们使用多个节点，研究程序在不同节点数下的效率变化。

5.1.5.1 单节点

首先，我们使用 $64 \times 64 \times 64$ 个网格，在1.6M粒子下测试不同的混合并行配置，

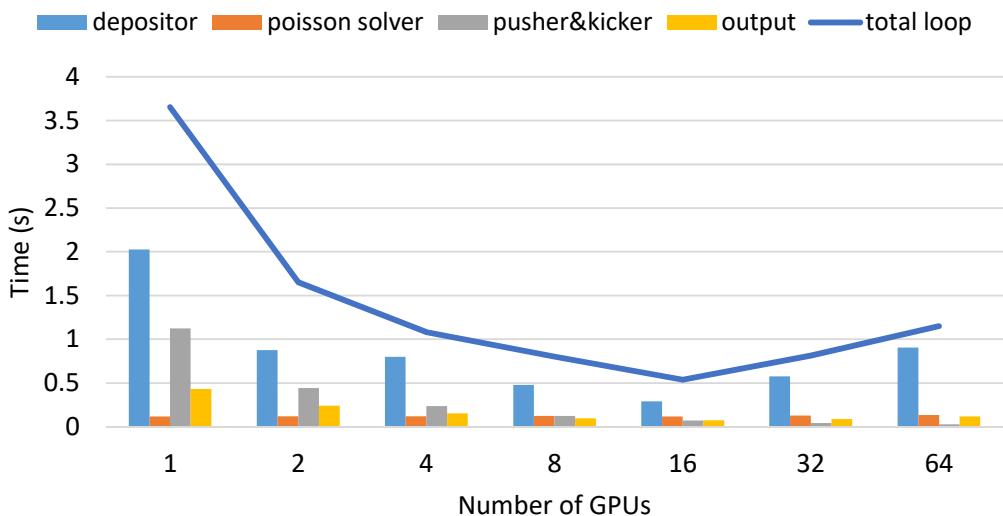


图 5.12: $64 \times 64 \times 64$ 个格点, 1.6M 个粒子时, SummitDev 上 PIC 程序耗时随 GPU 个数的变化

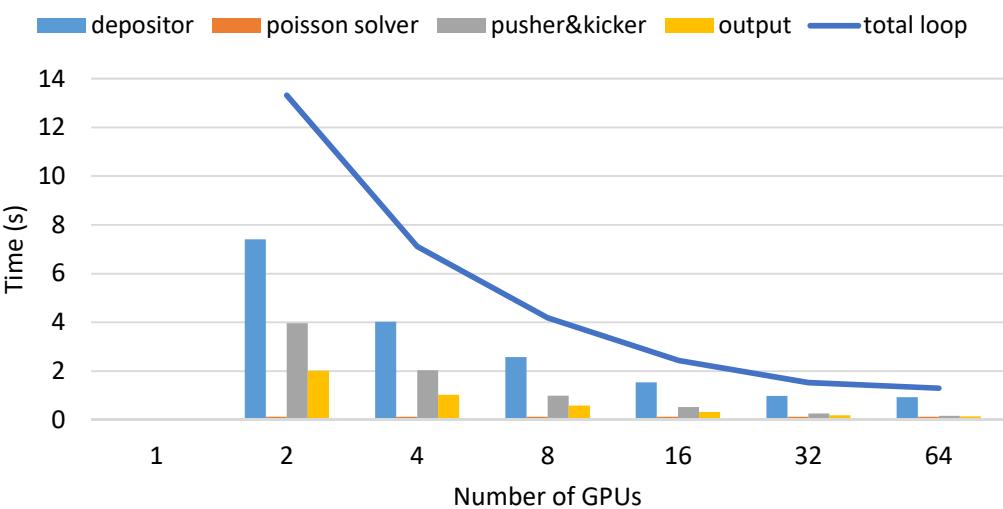


图 5.13: $64 \times 64 \times 64$ 个格点, 16M 个粒子时, SummitDev 上 PIC 程序耗时随 GPU 个数的变化

如图(5.14)所示。其中横轴为不同的并行配置，其中MPI进程数由1指数增加到64，而 OpenMP 线程数有64指数得减小到1，线程数乘以进程数则保持不变。在每种并行配置下，总时间（浅蓝色实线），和权重插值，求解泊松方程，推动粒子，信息输出（各色柱行）所耗时间由左纵轴以秒为单位表示，而内存占用（绿色实线）由右纵轴以GB为单位表示。

从图(5.14)可以看出，除了纯OpenMP并行（ $\text{MPI}=1, \text{OMP}=64$ ）明显较慢外，其他各个并行配置下所消耗的总时间的差别并不大，其中耗时最小的并行配置为使用32个MPI进程，每个进程使用2个OpenMP线程。而内存占用基本随着MPI进程

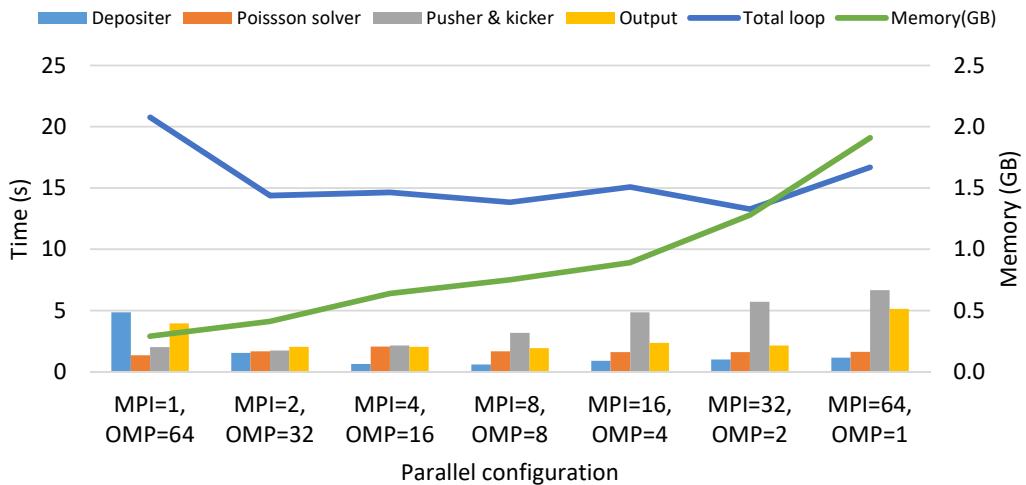


图 5.14: 1.6M 粒子数下，单节点下不同混合并行配置的耗时与内存占用

数目线性增加。

程序的不同部分对于并行配置的反应并不相同。随着 MPI 进程数的增加和每个进程所用线程数的减小，权重插值的耗时先减小后增加，一开始先从 MPI=64, OMP=1 时的 1.14 秒减少到了 MPI=8, OMP=8 时的 0.58 秒，然后其开始剧烈增加，最终到达 MPI=1, OMP=64 时的 4.85 秒。其原因是权重插值需要在不同的线程之间进行归约以避免线程冲突，而归约操作在线程数很大时有一个较大的启动时间。粒子推动耗时随着 MPI 数目变大单调增加，这是因为 OpenMP 更适合 KNL 众核架构，并能更有效的利用矢量处理器。

我们也测试了不同的粒子数下的并行配置，测试的粒子数为 160k 和 16M，分别是之前粒子数的十分之一和十倍，其结果如图(5.15)和图(5.16)所示。在 160k 个粒子的情况下，总时间随着 MPI 进程数的增加单调减少，这表明小粒子数情况更适合纯 MPI 并行。而在 1.6M 个粒子的情况下，纯 OpenMP 并行由于无 MPI 进程间通讯，显示出一些优势，但是其总耗时依然大于 MPI 并行，其耗时最小的配置为 MPI=64, OMP=1。在两种情况下，内存占用总是随着 MPI 进程数变大而单调增加。

5.1.5.2 多节点

从上面单节点的测试中可以得到，使用较大的 MPI 进程数和较小的 OpenMP 线程数是更有效率的并行配置。因此在多节点的测试中，我们首先选用纯 MPI 并行，测试在不同的节点数下程序总体以及各个部分的耗时情况。之后，我们也测试了 OMP=2 和 OMP=4 的情况，并与纯 MPI 程序进行了比较。

图(5.17)是纯 MPI 配置下 PIC 程序在 CPU 集群多节点下的耗时情况。图中横轴为节点数目，每个节点使用 64 个核；左纵轴为时间，以秒为单位；右纵轴为内存

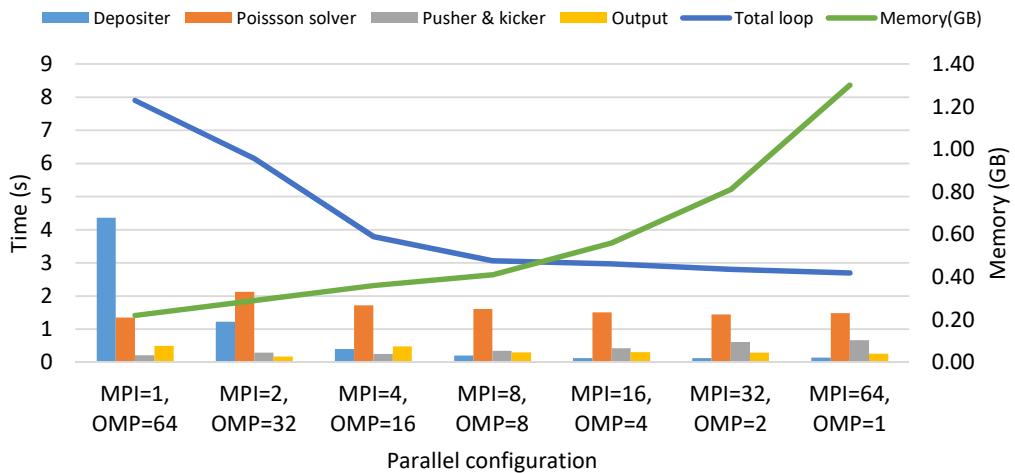


图 5.15: 160k 粒子数下，单节点下不同混合并行配置的耗时与内存占用

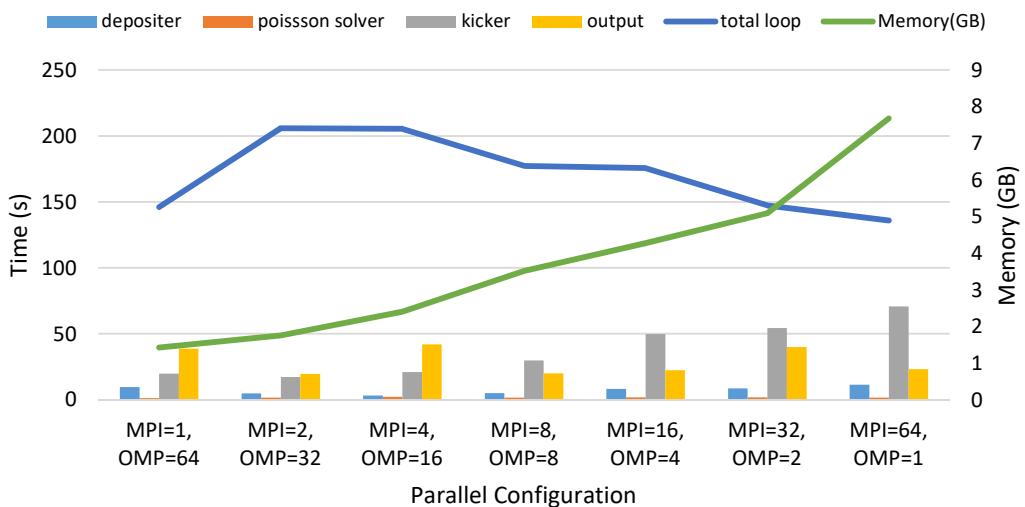


图 5.16: 16M 粒子数下，单节点下不同混合并行配置的耗时与内存占用

使用，以GB为单位。随着使用更多的节点，程序总耗时先降低后增加，在32个节点处到达最小值。总耗时先减小是因为使用的节点数越多，每个节点上需要进行的运算越少；后增加是因为随着节点数上升，节点间的通讯时间也会随之增加。图(5.18)为使用64个节点时程序各个部分消耗时间所占的百分比，可以看出此时通讯耗时已经占了总耗时的60%。

接下来，我们对OMP=2和OMP=4进行了测试，并与纯MPI程序(OMP=1)进行了比较，如图(5.19)所示。在大部分情况下，不同并行配置的耗时差别很小；但是在某些情况下差别很大，比如节点数为16时，纯MPI的速度是OMP=4的1.8倍。对于内存使用情况，使用更多的OpenMP线程和更少的MPI进程总是占优势。综合考虑，在内存足够大的情况下，使用纯MPI并行配置在KNL上运行PIC程序依然是一个很好的选择。

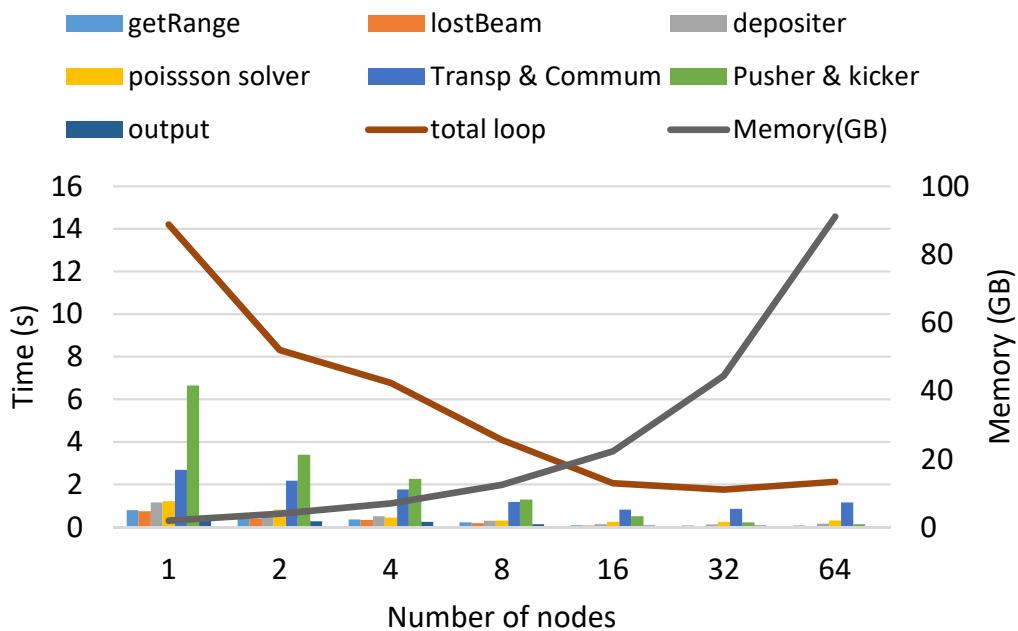


图 5.17: PIC 程序使用多个 CPU 节点的耗时

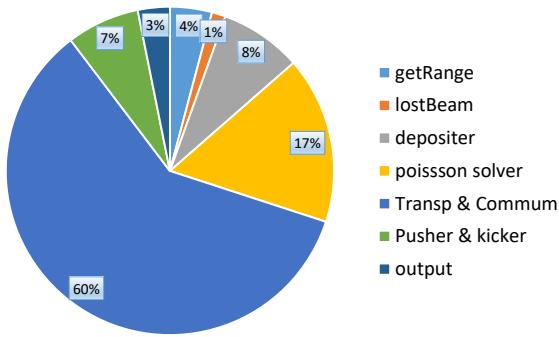


图 5.18: 使用 64 个节点时程序各个部分消耗时间所占的百分比

5.1.5.3 与 GPU 对比

在之前节(5.1.2)的单GPU测试中，我们使用的GPU型号为GTX 1060，这个GPU有1280个核，时钟频率为1.71GHz。而在本节我们使用的CPU型号为Intel® Xeon Phi™ Processor 7250 (KNL)，共包含68个核，时钟频率为1.40GHz，但是实际上为了方便比较，我们只是使用了64个核。

而由于GPU程序做了一些简化，减少了一部分步骤，例如粒子丢失判据等，我们在进行比较的时候也在CPU程序的耗时中减去了相应部分。在 $64 \times 64 \times 64$ 个格点数，1.6M个粒子的情况下，对于同样长度的加速器，使用一个GPU代码的总耗时为3.56秒，这类似于CPU程序在300个核心上的运行时间。换而言之，对于我们实现的PIC程序，一个GPU卡的运算效率与4到8个CPU节点的运行效率相当。

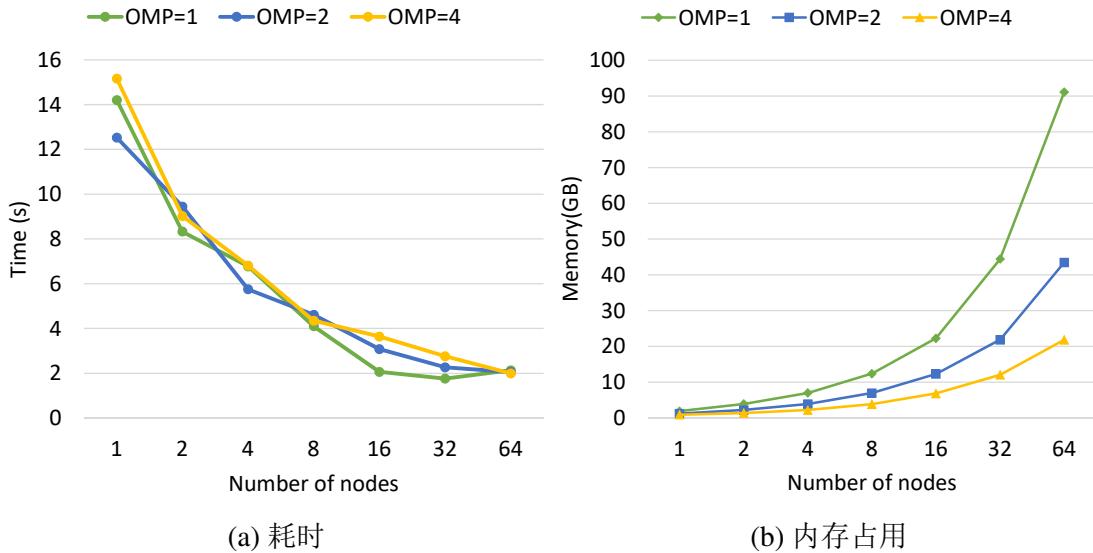


图 5.19: 不同并行配置下的多节点运行情况比较

5.2 Symplectic算法性能

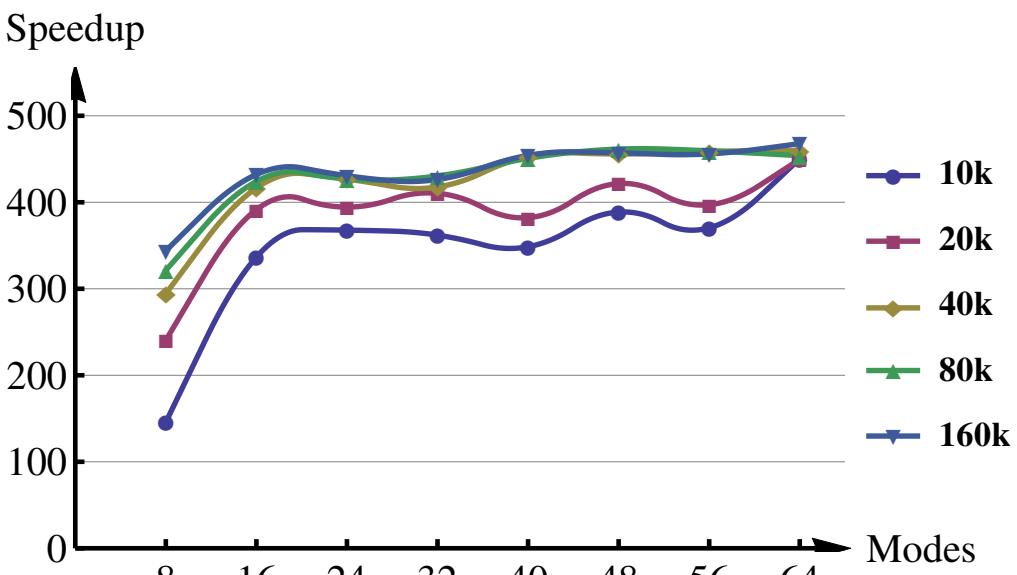
使用一个GPU，Symplectic算法与CPU串行相比实现了超过400倍的加速。而且，加速比会随着GPU的数目几乎线性增长。我们对GPU程序的性能和可拓展性进行了两个测试，第一个测试使用一个普通的家用GPU：GTX 1060 6GB，测试的结果与在CPU串行运行的程序进行比较。第二个测试使用ORNL的GPU集群Titan，用于测试程序的可扩展性，即程序在多GPU下的表现。

5.2.1 单GPU性能提升-GTX1060

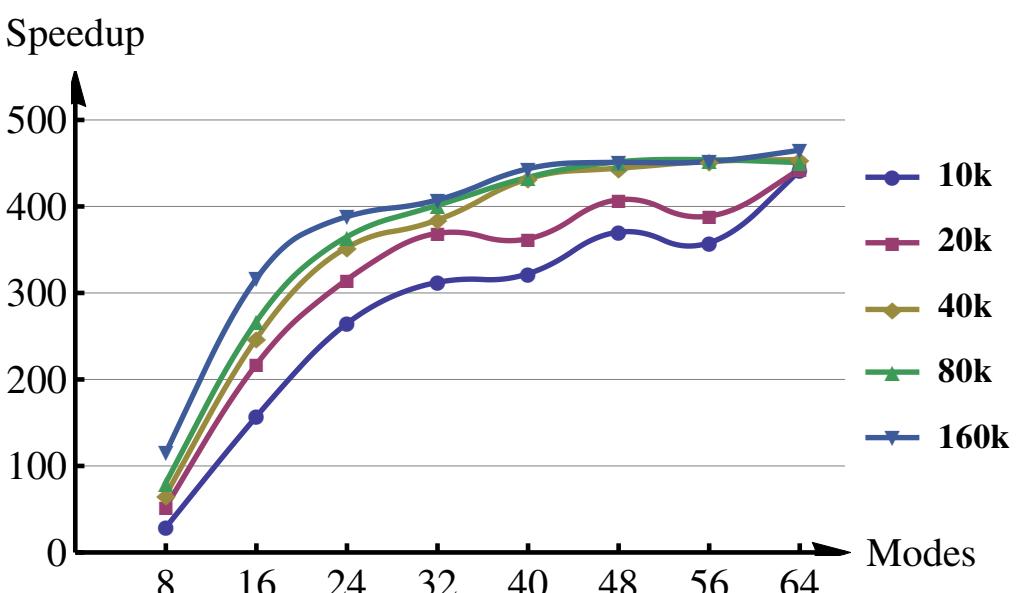
首先，我们将GPU代码的性能与CPU代码进行比较。GPU代码使用GeForce GTX 1060 6GB（Pascal架构）进行测试，而作为对比的CPU代码使用AMD Opteron 6134的一个核心运行，测试的软件环境为Ubuntu 16.04，使用CUDA 8.0版。

加速比由CPU版本运行的运行时间除以GPU版本的运行时间得到。在本次测试中，我们对空间电荷求解器和整个程序分别进行了比较。空间电荷求解器包括将数据从CPU侧复制到GPU侧，计算空间电荷效应，推动粒子，并将数据复制回CPU侧，而整个程序则包括了除了空间电荷求解器之外的所有其他部分，比如外场推动，输入，统计，输出等等。

从CPU代码简单的移植到GPU上就可以实现大约200倍的加速，并且通过优化可以获得更大的加速比。在采用了小节(4.4)中的优化策略后，程序能够充分利用GPU，取得了超过400的加速比。接下来，如图(5.20)所示，我们会讨论分别讨论空间电荷效应和整个程序在不同的问题规模大小下的加速比。



(a) 空间电荷效应求解器加速比



(b) 程序整体加速比

图 5.20: Symplectic 算法的单GPU加速比随着分解阶数和粒子数目的变化

图(5.20a)为优化后的空间电荷效应求解器的加速比在不同的问题规模下的变化。其中，横坐标为展开的阶数，而纵坐标是加速比，不同的曲线是不同粒子数目的结果。可以看到，阶数越大加速比越高，这与我们的预期符合，因为阶数越大意味着空间电荷求解器占用的时间在总时间中的比例越大。其中，在 $8 \times 8 \times 8$ 阶时加速比很低是因为在这种模式下的计算量很低。随着阶数的增加程序计算量也随之增加，GPU的负载更为平衡，因此取得了更大的加速比。另一方面，粒子数

目对于加速比的影响很小，在大阶数的情况下尤为如此。这是因为粒子数目本身远远超出了一个GPU的核心数目（在GTX 1060 上为1280个核心），即使在最低粒子数（10000个粒子）的情况下，程序也能有效的使用所有的核心。而随着粒子数目的轻微提升是因为GPU能够在更大运算量的情况下更好的协调和平衡计算资源。

图(5.20b)为程序整体运行时间的加速比，除了空间电荷效应求解之外，程序总体运行时间还包括外部传输矩阵，从Z坐标到T坐标的变换，粒子信息统计，以及输入输出。同空间电荷效应求解器相比，整体时间的加速比在各个问题规模下都略有下降，但是加速比变化的趋势却保持一致。其中，在低阶情况下，比如 $8 \times 8 \times 8$ 阶或 $16 \times 16 \times 16$ 阶，因为空间电荷效应求解器所占总时间的比重不大，所以加速比的下降更为严重。然而，在高阶情况下，空间电荷效应求解所占的时间会占总时间的绝大部分，所以总时间的加速比和空间电荷效应求解的加速比能够基本保持一致。

总之，Symplectic算法在单GPU上取得了比较高的加速比。对于程序整体的总运行时间，GPU代码比CPU代码的加速比最高超过了450；而如果仅仅比较空间电荷效应求解器，其最大加速比超过了460。

5.2.2 多GPU性能提升-Titan

我们使用超级计算机Titan对Symplectic算法进行了GPU集群上的性能测试。为了测试多GPU的程序可扩展性，我们最多使用了1024个节点进行测试。其中，为了在不同节点的GPU上交换信息，我们需要先把GPU上的数据拷贝到CPU侧，再使用MPI协议在不用节点之间交换数据，最后再将交换后的信息拷贝回GPU侧。在本次测试中，我们使用 $16 \times 16 \times 16$ 阶进行测试，使用这个阶数是为了精确度和计算速度之间的平衡，也是我们在实际模拟中最常用的阶数。如图(5.21)所示，我们分别讨论了空间电荷效应和整个程序在不同的粒子数目下，在不同的节点数上的加速比。其中，横轴为节点数目，纵轴为加速比，加速比有多节点的运行时间除以单节点的运行时间得到，不同曲线是程序在不同粒子数目时的耗时情况。

由图(5.21a)可得，在一开始，空间电荷效应求解器的加速比随着GPU的数目几乎线性增加，直到逐渐到达一个极限。一方面，加速比的线性增长主要是因为GPU之间的数据交换量很少，导致通讯时间很少。这是无网格Symplectic粒子跟踪算法的一个很大的优势，各个进程之间的数据交换量仅与阶数有关，而与粒子数量无关。另一方面，它可以实现的最大加速度主要受粒子数量的限制，线性增加的范围和极限也随着粒子数量的增加而增加。以160k粒子为例，加速比最大可以达到40。在Titan集群上，每个GPU包含2688个内核，当使用64个GPU时，我们

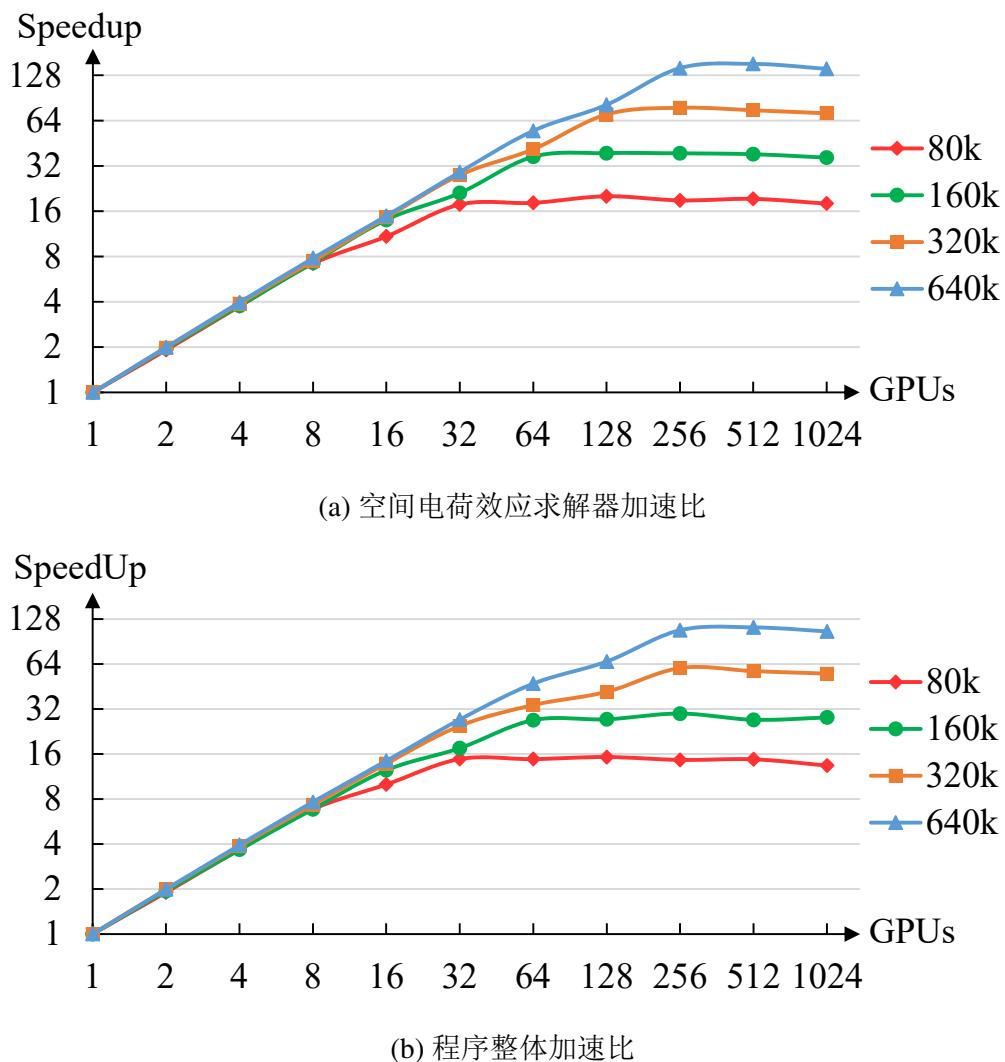


图 5.21: Symplectic 算法的多GPU加速比随着分解阶数和粒子数目的变化

使用了 $64 \times 2688 = 172032$ 个内核，即使用的核心数大于粒子数。在这种情况下，我们无法通过简单地使用更多的GPU来获得进一步加速。而粒子数目增加时，比如320k个粒子或640k个粒子，所能够使用的GPU数目也会随之增加，最大加速比和线性范围也就会增加。

图(5.21b)是程序整体总时间的加速比，其中外部传输矩阵，坐标变换，粒子信息统计这些部分也是并行化的。因为它们的计算量较低，很难取得很高的并行度。所以加速比略微下降。但是因为其他部分在程序整体耗时中所占比重很小，所以下降幅度很小。

5.3 小结

通过对并行程序的性能测试可以看出，对于PIC算法，在单个普通家用GPU（GTX 1060）上相对于单核CPU实现了超过50倍的加速。当模拟使用的粒子数较大时，PIC算法在GPU集群上也显示出良好的可扩展性；而当粒子数目较小时其可扩展性较差。

对于Symplectic算法，在一个普通家用GPU上相对于单核CPU实现了超过450倍的加速比。同时，我们在GPU集群Titan上的测试还显示出这种算法有良好的可扩展性，程序的加速比随着GPU数目几乎线性增加。在未来的研究中，我们将继续对这个算法进行扩展，并在不同架构的计算机上比较Symplectic算法的效率。

第六章 束流动力学相关问题研究和模拟

在本章中，我们使用束流模拟程序针对几个空间电荷相关的问题进行了详细研究。首先，节(6.1)我们使用P-TOPO程序对C-ADS的注入器I进行了模拟研究。之后，节(6.2)我们使用Symplectic算法对空间电荷导致的三阶共振进行了研究。最后，在节(6.3)我们对加速器中的共振穿越现象进行了研究。

6.1 C-ADS注入器I模拟

C-ADS是由中国科学院高能物理研究所和近代物理研究所共同研究的一个加速器项目[61–63]。ADS是一种新型的能源装置，其基本原理是利用经过加速器产生的高能质子撞击重靶核发生反应，一个质子引起的散裂反应可产生几十个中子，再使用散裂反应产生的中子作为中子源来驱动次临界核能系统，使次临界核能系统维持链式反应以便得到能量并利用多余的中子增殖核材料和嬗变核废物。

接下来，节6.1.1对C-ADS的整体加速器结构以及关键技术问题进行了简要介绍，并介绍了使用程序模拟对C-ADS的意义；节6.1.2中我们对其中的RFQ进行了模拟；节6.1.2中我们对超导段进行了模拟；最后，节6.1.4给出了总结。

6.1.1 C-ADS简介

C-ADS加速器的主要参数如表(6.1)所示[61]，可以看出C-ADS主要有高流强高功率、CW工作模式、严格的束损控制、以及高稳定性等技术难点和要求。为了达到高流强高功率，C-ADS采用分段设计和建造，由注入器和主加速段两部分组成。C-ADS在设计中尽量采用超导加速结构，以获得更好的加速效率和更大的束流孔径，从而满足CW工作模式和束损控制的要求。为了维持裂变反应堆的进行，ADS对于束流的中断次数要求非常严格，采用超导加速结构也提高了整个加速器的可靠性。而且C-ADS加速器使用了容错设计，即使某些元件失效了，我们也可以通过对其它元件进行一定的补偿，以保证机器按照设计的参数正常运行。

图(6.1)是C-ADS的整体加速器结构示意图[61, 64]。C-ADS采用了备份设计，其中在束流能量低于 10MeV 时，设计采用两条并行的线路，即使用双注入器来保证其可靠性。两台注入器分别命名为注入器I和注入器II，分别由高能物理研究所和近代物理研究所设计和建造。当束流能量大于 10MeV 时，由于其束流能量较高，与低能段相比，单腔能量增益对束流运行状态的影响要弱很多。在这部分，

表 6.1: C-ADS 加速器主要参数

粒子类型	质子
能量 (GeV)	1.5
流强 (mA)	10
束流功率 (MW)	15
频率 (MHz)	162.5/325/650
占空比 (%)	100
束损 (W/m)	<1
故障次数 (每年)	
$1s < t < 10s$	<25000 次
$10s < t < 5m$	<2500 次
$t > 5m$	<25 次

C-ADS 主要通过串联的方式进行备份，即主加速器只有一条线路，但是其工作参数低于正常运行参数，以留出调整空间。当主加速器中某一个元件失效时，我们可以通过调整其他元件的参数来使整个加速器的输出保持不变。

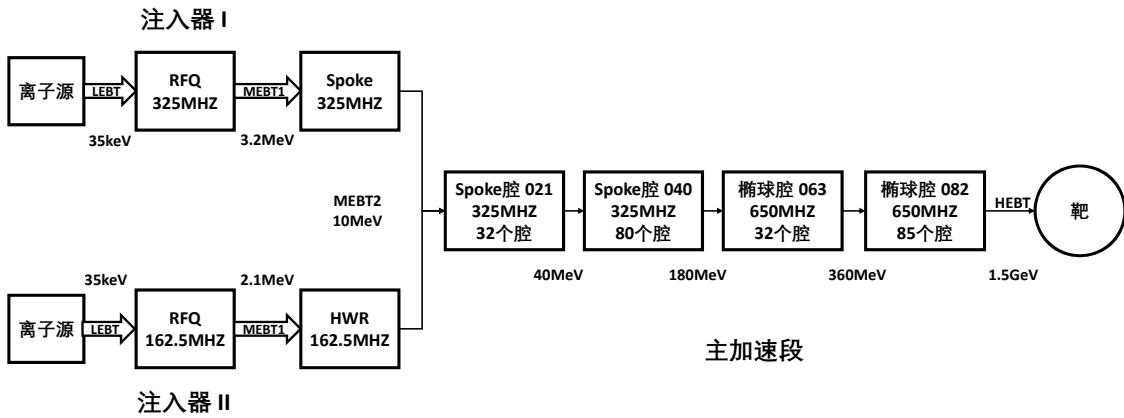


图 6.1: C-ADS 加速器整体布局示意图

C-ADS 的束流动力学设计要求非常严格。由于峰值流强较高，而且由于超导加速结构的特点以及聚焦结构的特殊设计，加速器中的空间电荷效应十分显著。在加速器参数设计不当时，束流会出现由空间电荷力驱动的不同自由度之间的发射度交换。为了保证加速器的稳定运行，C-ADS 的设计必须选择合适的工作点，以避免横纵向耦合导致的共振和束流损失；并尽可能使横纵向的零流强相移小于 90° ，以避免空间电荷效应导致的束流包络不稳定性[47]。

C-ADS的注入器设计也是整个加速器设计的难点之一，其中注入器I由高能物理研究所设计和制造，着眼于基本物理问题和关键技术的发展，如超导Spoke腔，低温技术等等。由于采用与主加速器相同的频率可以使主加速器的束团电荷密度最小，因此高能物理研究所提出了基于325MHz的注入器I设计方案，并采用了工作在 $3\beta\gamma/2$ 模式下的高 β 腔代替低 β 腔，解决了加工上的困难。而且此方案只需开发325MHz轮辐超导腔，能够有效地降低研发时间。

C-ADS注入器I的基本参数如表(6.2)所示。C-ADS的注入器I由离子源 (ECRIS)、低能传输线、RFQ、中能传输线、两个超导加速模块、以及最后的垃圾桶组成，其结构如图(6.2)所示。其中RFQ使用PARMTEQM进行设计 [65, 66]，频率为325MHz。RFQ将流强为10mA的质子束从35keV加速到3.2MeV，未来流强还计划升级到15mA。因为采用较低的注入能量 (35keV)，RFQ的聚束段 (buncher) 和成型段 (shaper) 使用绝热设计以降低空间电荷效应的影响。这较好地控制了发射度的增长，达到了最终较小的纵向发射度。RFQ之后是14个超导加速腔，将束流加速到最终的10MeV。

表 6.2: C-ADS注入器I基本参数

粒子种类	质子
频率 (MHz)	325
注入能量 (MeV)	0.035
输出能量 (MeV)	10
流强 (mA)	10
占空比	100%
X方向归一化发射度 (π mm mrad)	0.2
Y方向归一化发射度 (π mm mrad)	0.2

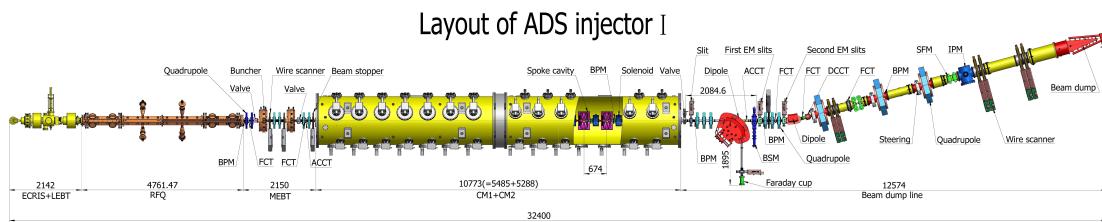


图 6.2: C-ADS注入器I结构示意图

C-ADS注入器I是一个复杂的装置，由众多元件组成，而且技术要求很高，我们在正式建设之前必须对其设计进行检验。如果其中存在不合理之处，将会导致

巨大的损失。但是由于C-ADS注入器I是一个复杂的整体，耗资巨大，我们不能通过建设实物的方式对其设计进行检验，所以采用数值程序进行模拟验证。为了验证C-ADS注入器I设计的合理性和可行性，我们使用P-TOPO程序对其进行模拟。因为C-ADS注入器I中的束流受空间电荷影响较大，束流演化不容易预测，所以模拟过程必须正确处理空间电荷效应，才能得到准确的模拟结果，从而有效地对其进行验证。

我们对C-ADS注入器I中的RFQ和超导段进行了分段模拟，使用的宏粒子数目为20000个，横向初始分布使用KV分布，纵向初始分布为均匀分布。我们使用实际的加速器结构来决定模拟中的束流丢失，其中对RFQ更为严格，其丢失判据的孔径为当前cell的最小半径。空间电荷效应的网格数为 $64 \times 64 \times 64$ 。在RFQ段，我们使用P-TOPO程序，同时也使用Track程序[12]以相同的初始条件进行模拟；在超导段，除了P-TOPO之外，我们还使用TraceWin[9]来进行研究和对比。下面，我们对RFQ和超导段分别进行讨论。

6.1.2 RFQ模拟

在RFQ模拟中，RFQ中的场可由傅里叶贝塞尔方程的八项式得到：

$$\begin{aligned} U_{ex}(r, \theta, z) = & \frac{V}{2} [A_{01}(r/r_0)^2 \cos(2\theta) + A_{10}I_0(kr) \cos(kz) \\ & + A_{03}(r/r_0)^6 \cos(6\theta) + A_{21}I_2(2kr) \cos(2\theta) \cos(2kz) \\ & + A_{12}I_4(kr) \cos(4\theta) \cos(kz) + A_{03}I_0(3kr) \cos(3kz) \\ & + A_{23}I_6(2kr) \cos(6\theta) \cos(2kz) + A_{32}I_4(3kr) \cos(4\theta) \cos(3kz)], \end{aligned} \quad (6.1)$$

上式中的 I_n 为第n阶修正贝塞尔方程， $k = \pi/2\beta\gamma$ ，其中 γ 为洛伦兹因子， A_{mn} 系数由RFQ设计程序PARMTEQM给出。

图(6.3)和图(6.4)分别是P-TOPO和TRACK在0mA下和15mA下对RFQ模拟得到的横向和纵向发射度演化，其中红色实线为P-TOPO的结果而绿色虚线为TRACK的结果，两个程序的结果基本吻合。但是在横向和纵向两个方向上，P-TOPO的发射度演变结果都更加光滑。尤其是在RFQ的前端，束流纵向相空间丝化形成束团。

图(6.5)是使用P-TOPO和TRACK在15mA下对RFQ模拟得到的横向束团尺寸演化，图(6.6)是使用P-TOPO和TRACK在15mA下对RFQ模拟得到的纵向束团尺寸和能散演化，其中红色实线为P-TOPO的结果而绿色虚线为TRACK的结果。两个程序在束团均方根尺寸上吻合得很好，其差别在合理范围内。通过P-TOPO模拟计算，我们认为C-ADS注入器I的RFQ设计能够有效的控制束团的发射度和尺寸。

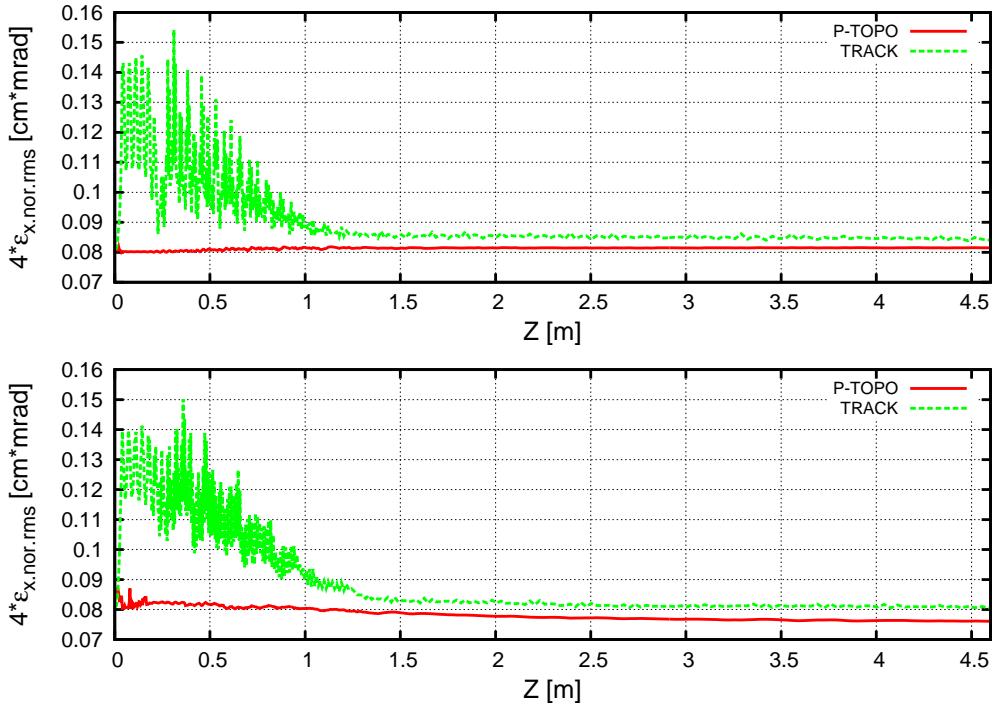


图 6.3: 0mA (上) 和 15mA (下) 的C-ADS注入器I的RFQ中的横向发射度变化

6.1.3 超导段模拟

在超导段中，聚束腔和加速腔的场由文件导入，P-TOPO使用数值插值来得到粒子所受到的场强。在设计的15mA流强下，RFQ出口处3.2MeV的质子束流经过聚束，通过MEBT进入超导段，逐渐加速到10MeV。在超导段中我们使用P-TOPO和TraceWin来进行模拟。

图(6.7)为P-TOPO和TraceWin在15mA流强下对RFQ模拟得到的横向和纵向发射度演化，其中红色实线为P-TOPO的结果，而绿色虚线为TraceWin的结果。两个程序得到的结果相一致。其中横向发射度除了在螺线管处有一个峰值外，基本保持不变。螺线管处的峰值是一方面是因为发射度需要在共轭坐标下讨论，而 PTOPO 和 TraceWin 在螺线管中的统计量并不是共轭量；另一方面是因为在束团进入螺线管的时候相空间发生扭曲。束团的纵向发射度有略微增长，其增长幅度在20%以内。发射度增长是由空间电荷力、磁铁边缘场的非线性效应、以及超导腔内的非线性纵向力导致，这几个来源共同驱动发射度增长，并使粒子相空间扭曲，最终会导致束晕的产生。P-TOPO的模拟中，出口能量为10.01MeV，而TraceWin的模拟出口能量为10.06MeV。

图(6.8)是P-TOPO和TraceWin在15mA下对超导段模拟得到的横纵向束团尺寸和能散，其中红色实线为P-TOPO的结果而绿色虚线为TraceWin的结果。两个程序

模拟结果的细微差别来自于两个程序获取同步相位的方法不同。TraceWin使用时间漂移法获得同步相位，而P-TOPO使用扫相获得同步相位。束团的横向尺寸相比管道半径都很小，而纵向尺寸也被聚束腔和接下来的各个加速腔有效地压缩，在这种情况下束损通常不会大量形成。

6.1.4 总结

通过使用P-TOPO对C-ADS注入器I进行模拟，我们一方面验证了注入器I设计的合理性，另一方面也通过和其他模拟程序的对比验证了P-TOPO的正确性。

模拟结果表明，C-ADS注入器I能够有效的对粒子束团进行加速。无论是在零流强还是在15mA流强下，粒子在RFQ和超导段中的传输效率都在99.9%以上，而且束团尺寸和束损都得到了有效控制，横向和纵向的发射度都保持良好。

我们还将P-TOPO的模拟结果与其他模拟程序的结果进行了对比，不同程序的结果基本吻合，其差别在合理范围之内，P-TOPO的正确性得到了验证。

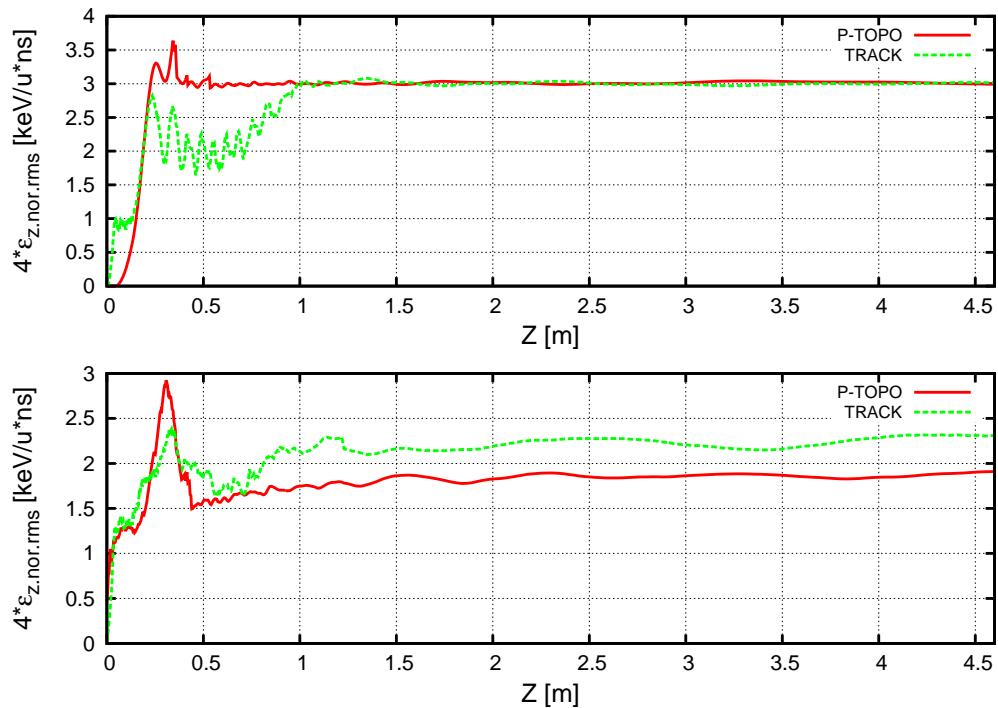


图 6.4: 0mA (上) 和15mA (下) 的C-ADS注入器I的RFQ中的纵向发射度变化

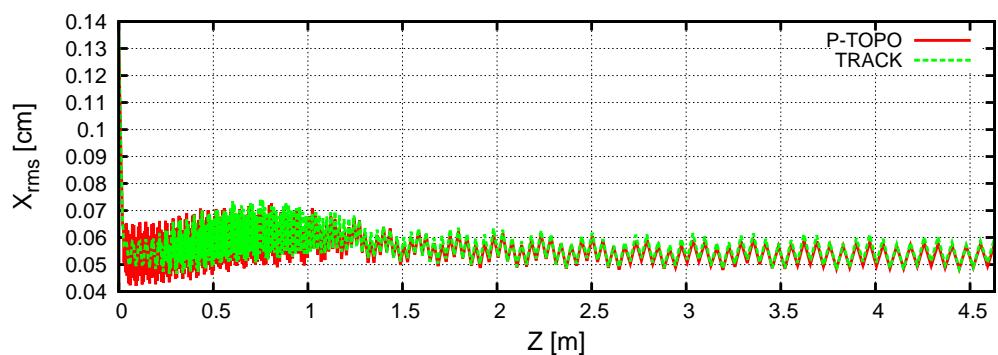


图 6.5: C-ADS注入器I的RFQ中束团横向均方根尺寸变化

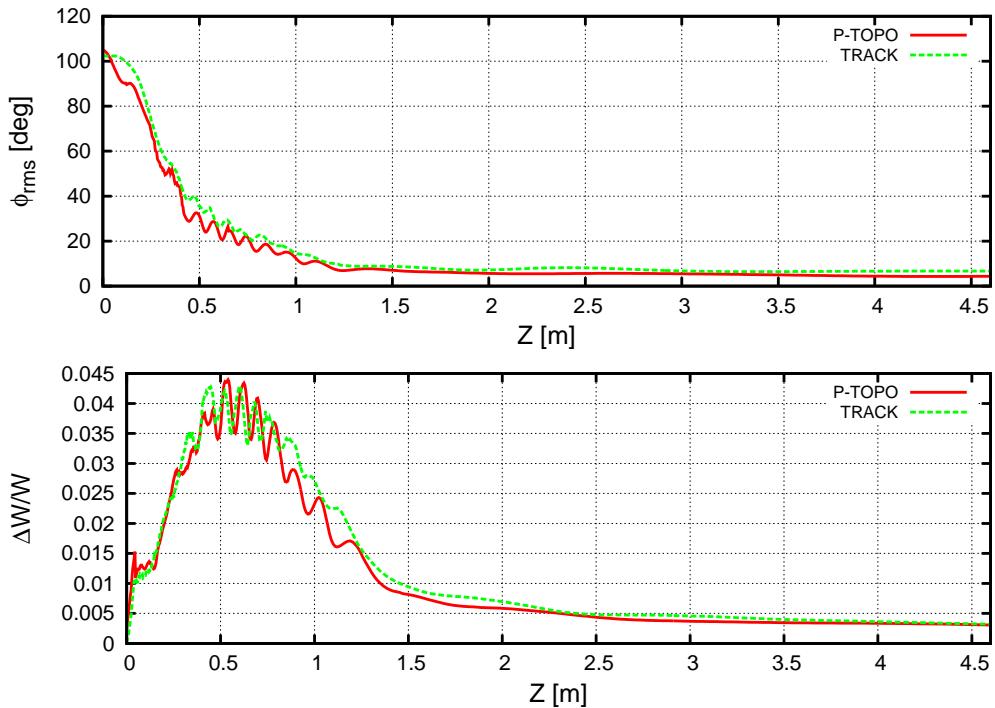


图 6.6: C-ADS注入器I的RFQ中束团纵向均方根尺寸和能散变化

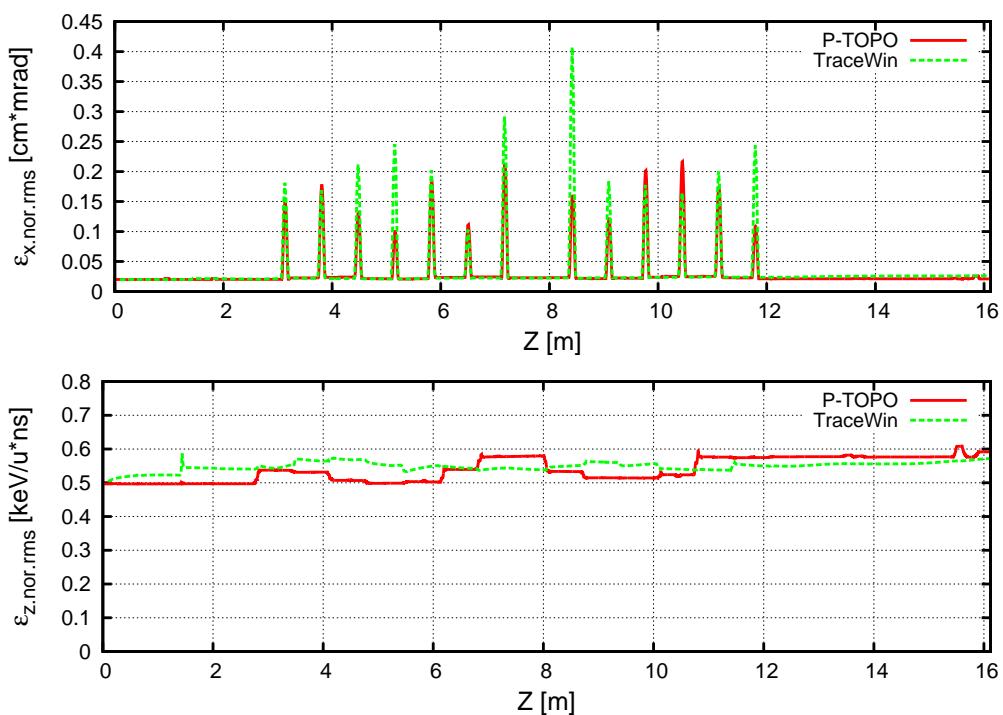


图 6.7: C-ADS注入器I的超导段横向发射度和纵向发射度变化

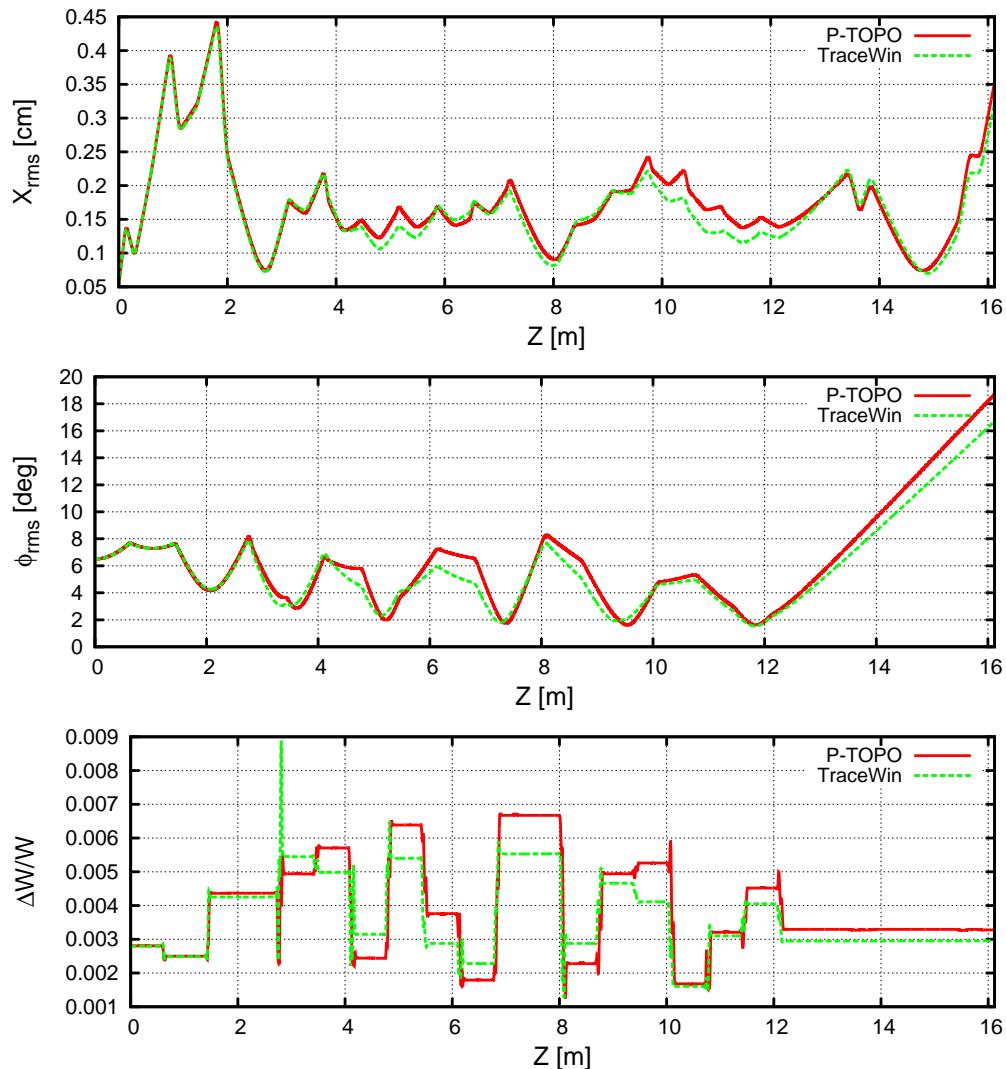


图 6.8: C-ADS注入器I的超导段束团横纵向尺寸以及能散变化

6.2 三阶共振模拟

我们使用Symplectic算法在周期聚焦结构中进行粒子跟踪和模拟，来研究束流中的三阶共振现象。如果使零流强的周期相移为86.3259度，并将10个周期视为一个环，则环的工作点为2.3979。随着流强的增加，相移会被压缩，工作点会在0.6A左右穿过2.3333的三阶共振线。如果环中存在一个六极磁铁，则束流会出现明显的三阶共振现象。

图(6.9)为六极铁 $K = 1.0$ 时不同流强下的束流发射度增长变化。可以看到，束流发射度在流强为0.1A和0.2A时基本保持不变，这时的工作点约为2.40。但是发射度在流强为0.4A, 0.6A, 和0.8A时持续增长，这时的工作点在2.3333附近，可知其发射度增长是由于三阶共振导致。

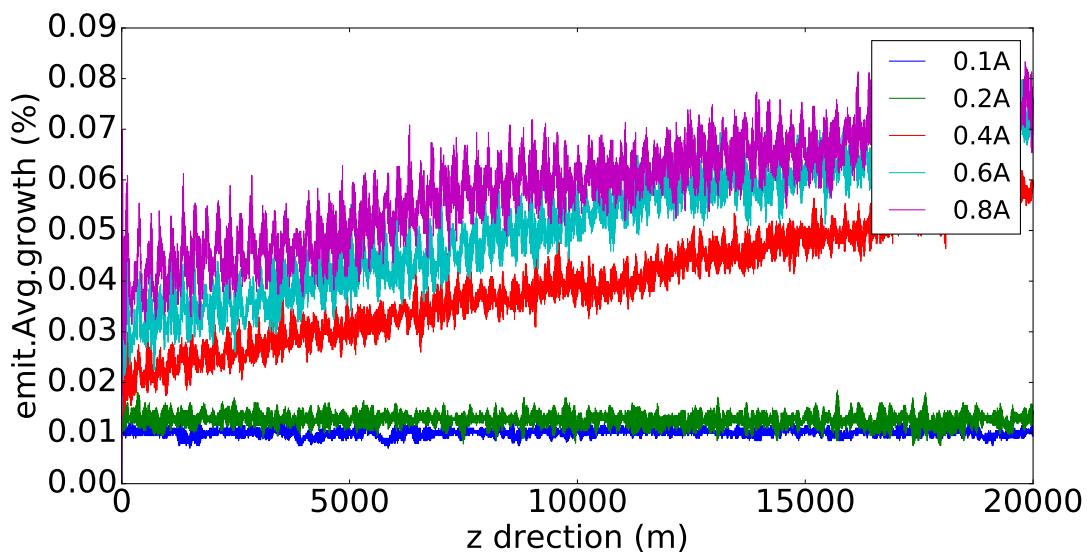


图 6.9: 不同流强下的发射度增长

图(6.10)是工作点为2.3333附近时的粒子坐标的庞加莱截面，其中颜色越暗表示粒子密度越大，而不同的图片代表粒子处于不同的初始位置。受空间电荷效应驱动，庞加莱截面会被扭曲，并塑造成三角形形状。受到三阶共振影响的粒子的横向位置会逐渐变大。

本次模拟的三阶共振效应小，因此导致的发射度增长很小。这个小的增长效应需要较高精度的模拟。正是因为Symplectic算法在长程研究中几乎不出现非自然的发射度增长，其精确度较高，所以本次模拟中微小的发射度增长才能较为明显地通过模拟得到。

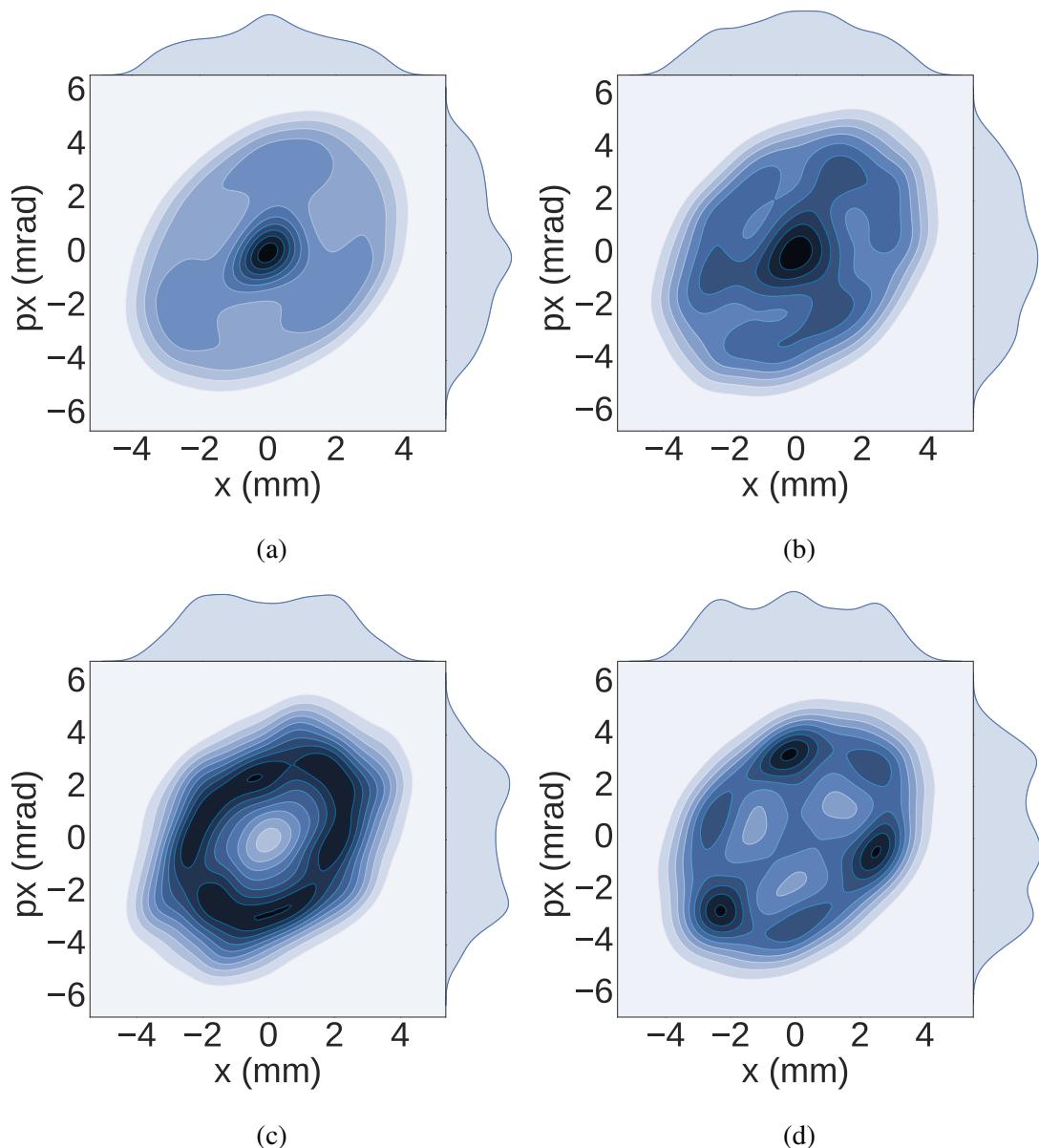


图 6.10: 三阶共振附近的庞加莱截面

6.3 共振穿越研究

在一个实际的加速器中，特别是直线加速器中，外部元件的聚焦强度一般随着束流被加速而逐渐变化，束流的周期相移也随之变化。在这个过程中，束流有可能穿越结构共振禁带，即“共振穿越”。

加速中由各种非线性驱动的共振被认为是导致束流恶化的主要来源，例如束流发射度增长、束晕、束损等。除了加速器元件产生的非线性效应之外，束团内部的空间电荷效应也是非线性共振的一个来源。在强流质子加速器中，空间电荷效应会在不同自由度之间产生耦合，相关学者已经在数值和实验上对此进行了很多

研究 [16, 17, 67]。但是对于束流在穿越过程中具体如何演化，以及不同穿越速度对于束流有什么影响，人们并没有一个统一的结论。如果共振穿越无法避免，目前人们对于如何处理共振穿越仍然存在这样的争议：是应该尽量绝热变化以达到更好的匹配，还是应该尽快完成穿越以减少束流在共振区的时间？

这些问题在加速器设计中非常重要。为了探寻这些问题，我们基于前人对结构共振的研究，对束流穿过结构共振禁带时的一些现象进行了模拟研究，比如束流和粒子的相干和非相干特性将如何演化、束流如何自发地受到结构共振的影响、以及不同穿越速度对于束流发射度的影响等。

接下来，节(6.3.1)将简要介绍了结构共振的模型，并定义了用来描述相干和非相干效应的一些术语；节(6.3.2)中给出了禁带的方程，使用多粒子PIC追踪程序模拟了束流穿越相干结构共振时的现象，并在暂态意义上进行了讨论；节(6.3.3)讨论了非相干共振的粒子特征；最后节(6.3.4)进行了总结。

6.3.1 物理模型

束流运动的表述有两个层面，每个层面分别都有对应的模型来描述。其非线性共振的两个层面的模型分别为单粒子动力学（非相干效应）和RMS束流动力学（相干效应）。单粒子动力学描述的一个典型例子是共振线图中由于加速器中多级铁和Lattice的误差导致的各阶共振线 $nv_x + mv_y$ 。共振线图被广泛应用在加速器的工作点选择和实际运行中。考虑到非线性效应，加速器的工作点会产生偏移 [68]，所以当束流工作点靠近共振线时就会发生单粒子周期共振穿越 [69]。对于RMS束流动力学描述，外部元件和内部空间电荷效应产生的非线性效应在束流集体不稳定性中起到了非常关键的作用 [70, 71]，其可以通过以自治的方式求解扰动的弗拉索夫方程来描述[72–74]。然而，虽然人们付出了极大地努力来扩展这些模型以涵盖各种束流条件，但是可解决的问题只是局限于某些特定的情况。

对于强流离子束流中的空间电荷效应导致的集体不稳定性，其研究一方面从RMS包络方程出发 [75]，其中二阶不稳定性，即束流包络不稳定性，得到了人们广泛而深入的研究 [49, 51, 76–79]。而另一方面，研究从自治的求解弗拉索夫-泊松方程出发 [80–83]。研究表明，求解弗拉索夫-泊松方程得到的二阶偶共振正是包络方程的得到的相干共振。最近研究提出了一种空间电荷物理中的弗拉索夫-泊松模型的通用理论 [80, 81]，预测了在Lattice参数没有被精心优化的情况下，结构共振会伴随构造本征模式发生，另外，低阶的结构共振禁带可以自然的作为高阶结构共振禁带的一部分¹。

¹数学证明详见参考文献 [81]

作为对弗拉索夫-泊松模型的理论研究的延伸，我们选取了90°相移附近的二阶共振禁带进行研究，并使用PIC模拟来验证了理论预测的有效性。这些研究可以很容易地扩展到更高阶的模式，比如三阶（60°）和六阶结构共振（120°）[84, 85]。需要注意的是，束流和结构共振之间的相互作用必须在瞬态状态下进行研究。而低阶模是高阶模的组成部分这一概念是理解模拟和实验结果的关键[86, 87]。

本节简要介绍了结构共振的物理模型和基本的处理方法，其中可解的耦合弗拉索夫-泊松方程仅限于4D KV分布[88]。一般来说，对束流动力学行为的研究是在哈密顿系统的框架中进行的，而粒子哈密顿系统中不可积的部分可以被视为微扰，这样共振条件就可以使用微扰理论来得到。接下来，我们使用周期性的FODO结构来模拟加速器中的连续束的演变，考虑到周期聚焦结构中完全匹配的束流的哈密顿量是恒定的，平衡分布函数和对应的哈密顿量可以表示为：

$$f_0(x, p_x, y, p_y) = f(H_0), \quad H_0 = k_x(s)x^2 + p_x^2 + k_y(s)y^2 + p_y^2 + V_{sc}(x, y). \quad (6.2)$$

其中 $k_x(s)$ 和 $k_y(s)$ 外部四极铁的聚焦强度， $V_{sc}(x, y)$ 是空间电荷势。分布函数 f_0 必须同时满足Vlasov方程和Poisson方程：

$$\frac{\partial f_0}{\partial s} + [f_0, H_0] = 0, \quad \Delta V_{sc}(x, y) = \frac{1}{\epsilon_0} \int \int f_0 dx dy, \quad (6.3)$$

其中 $[,]$ 为泊松括号。

如果粒子分布函数存在一个扰动 f_1 ，其将会导致空间电荷势受到扰动 $V_1 = H_1$ 。扰动后的弗拉索夫方程和泊松方程可以表示为：

$$\frac{\partial f_1}{\partial s} + [f_1, H_0] + [f_0, V_1] = 0, \quad \Delta V_1(x, y) = \frac{1}{\epsilon_0} \int \int f_1 dx dy. \quad (6.4)$$

目前认为，这组方程只有在粒子分布为理想的KV分布 $f_0 = \delta(H_0)$ 的时候才可解。一般情况下，束流内受扰动的空间电荷势可以表示为多项式的形式：

$$V_1 = \sum_{m=0}^n A_m(s)x^{n-m}y^m + \sum_{m=0}^{n-2} A_m^{(1)}(s)x^{n-m}y^m + \dots. \quad (6.5)$$

集体模 $I_{j;k,l}(s)$ 可以在适当的边界条件下得到。对于式(6.5)中的 n 阶偶数模和奇数模可以根据 m 是偶数或奇数来分开处理，其直接代表了粒子分布在扰动后在实空间中的倾角。设 S 为一个聚焦周期的长度， $I_{j;k,l}(s)$ 将满足 $\mathbf{I}'(s) = \mathbf{M}(S)\mathbf{I}(s)$ ，这正是 Mathieu 方程。而这个系统的稳定性将由其雅克比矩阵 $M(S)$ 的本征值 λ 决定，详细的数学证明见参考文献[80–83]。

根据之前的研究[80, 81]，结构共振条件可以显式的表达为 $\Phi_{j;k,l} + \Phi_{j;k,l} = n \times 360^\circ$ 和 $\Phi_{j;k,l}/\Phi_e = n/m$ ，其中 $\Phi_{j;k,l}$ 为集体模 $I_{j;k,l}$ 的本征相位； Φ_e 是一个周期中的包络振荡相移，恒为 360° 。结构共振会随着本征相位锁定的现象发生，而发生结构共振的参数空间被称为不稳定禁带。

6.3.2 结构共振穿越 – 相干效应

在加速器设计与运行中有一条经验准则：“如果共振穿越不可避免，则穿越速度应该越快越好。”比如在一些直线加速器的设计中，一开始的零流强相移大于90°，之后在几个周期之内迅速降到90°以下。接下来，我们使用90°相移附近的二阶结构共振来研究当穿越共振禁带时的束流和结构共振之间的相互作用²。为了简化，我们使束流在两个方向上的发射度相等，并使用对称的周期FODO聚焦结构 ($|k_x| = |k_y|$)。

6.3.2.1 二阶相干效应

对于二阶偶数模，束流内部的扰动的空间电荷势为 $V_{2e} = A_0(s)x^2 + A_2(s)y^2$ ；对于二阶奇数模，其为 $V_{2o} = A_1(s)xy$ 。通过动力学系统 $\mathbf{I}'(s) = \mathbf{M}(s)\mathbf{I}(s)$ 可以得到 $(I_{0;2,0}, I_{2;0,2})$ 和 $(I_{1;1,1}, I_{1;1,-1})$ ，分别代表二阶偶数模和奇数模 [80–82]，其雅克比矩阵 $\mathbf{M}(s)$ 可以表示为：

$$\mathbf{M}(s) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ J_{21}(s) & J_{22}(s) & J_{23}(s) & 0 \\ 0 & 0 & 0 & 1 \\ J_{41}(s) & 0 & J_{43}(s) & J_{44}(s) \end{pmatrix}, \quad (6.6)$$

上式中每个元素 $J_{ij}(s)$ 的显式表达见附录(B)。

从包络方程出发的包络不稳定性已经得到了大量的研究，其描述了和二阶偶数模相同的物理机制 [80–82]。最近，人们使用Chernin模型研究了“和共振”与“差共振”不稳定性 [78, 79]，研究给出了与从二阶偶模结构共振出发相同的结果。GSI和CERN的研究人员也在实验上对 $I_{0;2,0}$ 和 $I_{2;0,2}$ 进行了测量 [89, 90]。图 (6.11) 是在流强不变而考虑空间电荷效应后的相移 σ 从 95° 变化到 75° 时的二阶偶数模与奇数模的本征值 $Abs[\lambda_{j;k,l}]$ 和本征相位 $\Phi_{j;k,l}$ 的变化³，其对应的零流强相移 σ_0 从 110° 变化到 80°。如图所示，对于二阶偶数模，每当本征相位合流时，比如 84° ~ 89° (本征相位发生锁定 $\Phi_{2;0,2} = \Phi_{0;2,0}$ ，也被叫做合流共振 [81])，本征值 $Abs[\lambda]$ 就会离开单位圆，其绝对值会离开 1.0，这代表了集体不稳定性，会导致发射度增长。因为这里选择了对称束流条件，二阶奇数模不会导致任何不稳定性。下面，我们在PIC模拟 [91, 92] 中使用由400个聚焦强度线性变化的FODO周期构成的 Lattice，以研究结构共振穿越。初始分布采用水袋分布，宏粒子数为50000。

²如上文所述，二阶集体模实际上是四阶集体模的一部分 [81]，所以在下文中二阶模指的是二阶/四阶混合禁带。

³我们使用了对称束流条件，因此无论考虑空间电荷效应与否，两个自由度上的相移都相等。

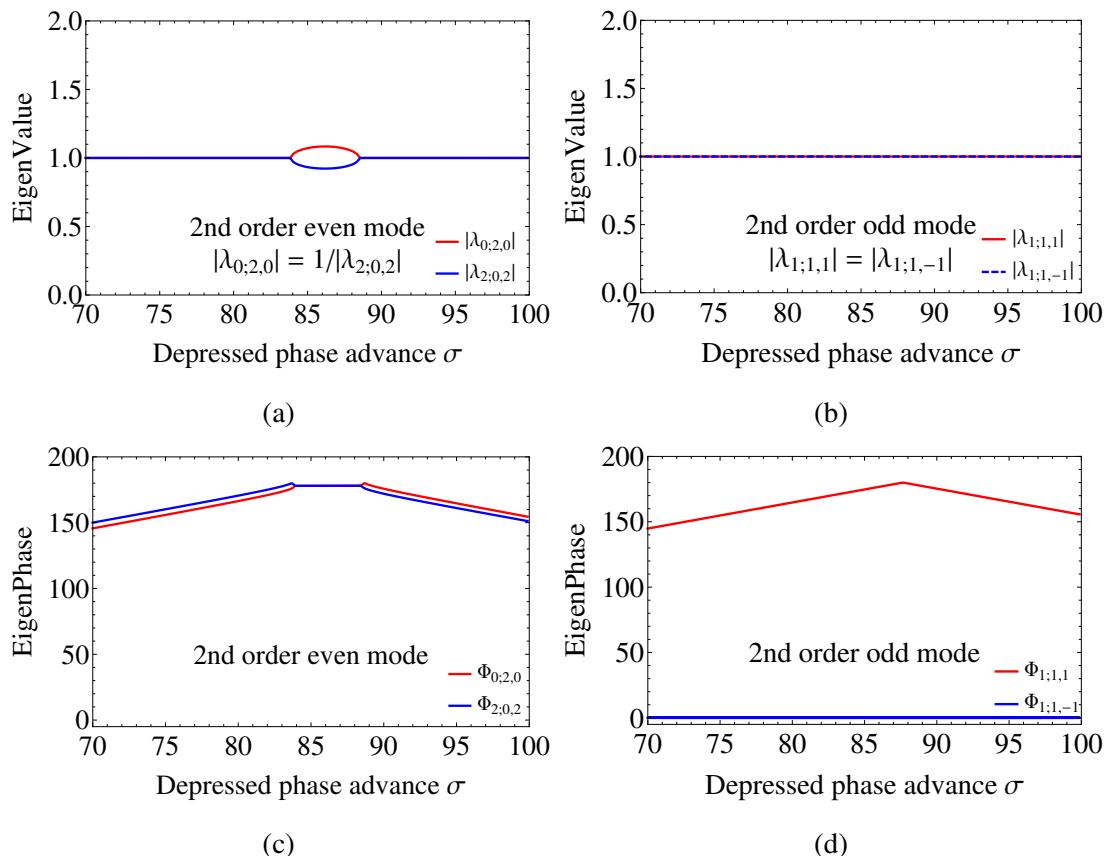


图 6.11: 二阶偶数模和奇数模的本征值 $Abs[\lambda_{j;k,l}]$ 和本征相位 $\Phi_{j;k,l}$ 随考虑空间电荷效应后的相移 σ 的变化

6.3.2.2 二阶结构共振穿越研究

在束流流强固定，而外部四极铁的聚焦强度逐渐变化的条件下，我们对束流穿越二阶结构共振禁带从两个方面进行了研究。一方面使束流周期相移从小到大变化（从下向上穿越，即周期相移 σ 在 400 个周期里线性地从 75° 增加到 95° ）；另一方面正相反，使束流周期相移从大到小变化（从上向下穿越）。

图 (6.12) 显示了束流相位从下方和从上方分别穿越结构共振禁带的情况，其中灰色区域为共振禁带，大约为 $84^\circ < \sigma < 89^\circ$ 。蓝色和红色曲线分别代表模拟得到的 X 和 Y 方向的周期相移演化，而黑色曲线代表平滑处理后的平均周期相移。原则上，模拟的周期相移被限制在黄色和绿色线所界定的区域内，黄色和绿色直线分别是预期的零流强周期相移和考虑空间电荷效应后的周期相移。垂直的黑色直线为束流实际进入和穿出共振禁带的位置，绿色直线和黑色曲线与禁带边界之间的交叉点分别表示束流在设计上和实际模拟中进入和穿出共振禁带的位置。图 (6.13) 为相应的 rms 发射度的变化，图 (6.12) 相同，垂直的黑色直线表示束流进入和穿出共振禁带时的位置。

在图 (6.12a) 中当束流从下方穿越共振区时，模拟得到的周期相移一开始与预期值相符并单调增加，但是当束流在大约 200 周期进入禁带后，束流周期相移穿越的速度开始变得比预期要快（如绿色直线和黑色曲线分别与禁带边界之间的交叉点所示），束流在大约 250 个周期时就穿出了共振禁带，并到达了一个局部平衡态。图 (6.13a) 显示束流一旦进入结构共振禁带，其 rms 发射度就开始增长，而在束流穿越禁带之后，发射度就停止增长，其中物理机制会在下文中详细讨论。

相反的，图 (6.12b) 和图 (6.13b) 显示了从上方穿越共振禁带时的周期相移和发射度的变化。在图 (6.12b) 中，可以明显看出束流比预期在禁带里停留的时间更长，即束流穿越共振带的速度更慢，在大约 370 个周期后才离开共振带。而且同从下方穿越的情况相比，束流的发射度增长更多。

共振穿越必须在瞬态的意义上进行研究 [81]。实际上，共振禁带的上边界和下边界随着穿越中的束流发射度增长也发生了变化。在从下方穿越的情况下，禁带的上边界移动到了大约 90° 附近，束流也在禁带里多停留了 15 个周期，直到 265 个周期才穿出禁带。同样的，对于从上方穿越的情况，禁带的下边界也略微向上移动，束流比预期中更早地（大约在 300 个周期左右）就穿出了禁带。这两种情况的一个特性就是共振穿越的速度不同。当束流从下向上穿越时，共振禁带会有一个“吸引”效应；而当束流从上向下穿越时，共振禁带会有一个“排斥”效应。其原因在于，在一般意义上，在外部场和内部空间电荷场的综合影响下，如果束流发生任何“不稳定”，比如 rms 发射度增长或者产生束晕，束流总是自发去

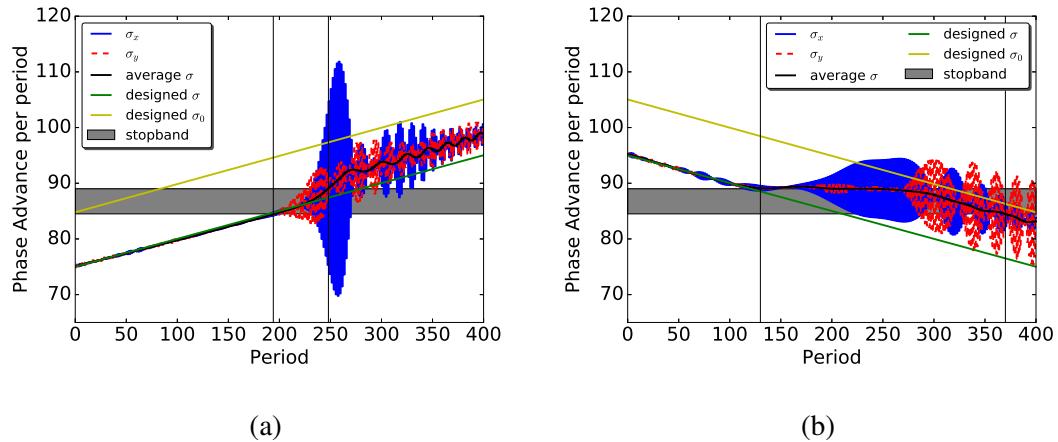


图 6.12: 从下方穿越共振区（左）与从上方穿越共振区（右）的周期相移比较

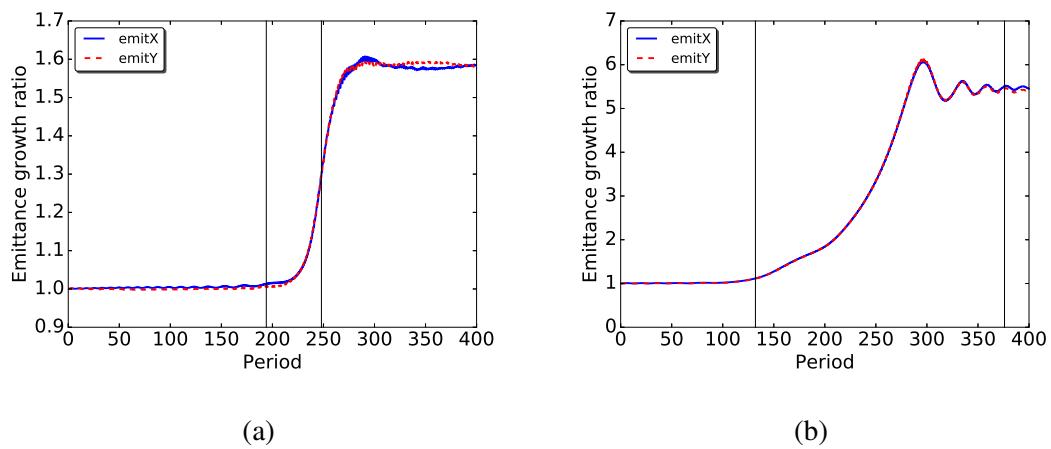


图 6.13: 从下方穿越共振区（左）与从上方穿越共振区（右）的发射度增长率(ϵ_f/ϵ_i)比较

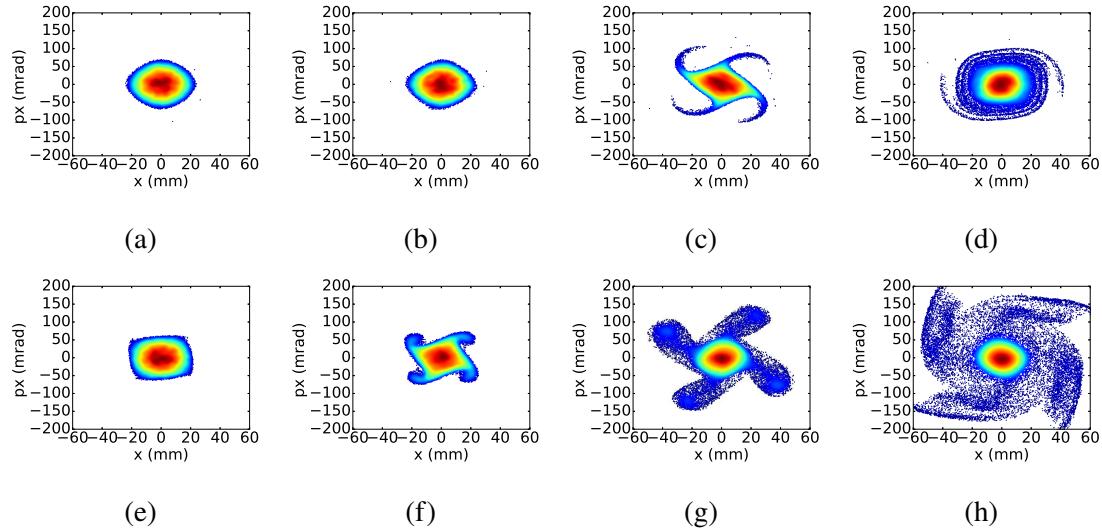


图 6.14: 从下方穿越共振区 (上) 与从上方穿越共振区 (下) 的穿越过程中的粒子相空间形状($x - p_x$)

摆脱这种不平衡。从能量的观点来看，束流总是倾向于“释放”能量，以到达一个能量最低的平衡态。因此很自然的，束流总是倾向于使空间电荷势减弱 [49]。研究表明一旦束流受到结构共振的影响，瞬态束流周期相移压缩因子 $\eta = \sigma/\sigma_0$ 会变得大于设计值，这一结果符合上文的理论。

图 (6.14)显示了从下方穿越共振区 (上) 与从上方穿越共振区 (下) 的穿越过程中在不同的FODO周期位置处的粒子分布相空间，图中的位置分别在第50、150、250、350个周期处。从图中可以清楚地看到，无论是从下方还是从上方穿越结构共振禁带，束流一进入共振禁带，相空间就出现了四个共振岛的结构，这正是四阶结构共振的现象。四阶结构共振会出现在二阶结构共振禁带的原因为低阶禁带是高阶结构共振禁带的一部分 [80, 81]。一般来说，不同阶的结构共振的出现需要合适的驱动力 (式 (6.5))。从WB初始分布开始，二阶和四阶扰动势从一开始就存在，并不断演变。因此，在二阶结构共振禁带中的相空间本就应该既出现双重结构又有四重结构 (我们将会在第 (6.3.3) 节中给出一个例子)。最后，由于这些混合的结构共振效应，束流产生了束晕。

6.3.2.3 结构共振穿越速度和rms发射度增长

为了研究如何处理共振穿越这个问题，这里我们主要关注束流的均方根发射度与结构共振穿越速度之间的关系。与上一小节中处理束流周期相移的方法相似，在这里，我们一使用400个FODO周期，其中前 N 个周期($N=10, 20, 50, 100, 200, 400$)的束流周期相移线性的从 75° 增加到 95° (从下方穿越) 或者从 95° 减小到 75°

(从上方穿越), 并使用400个周期后最终的rms发射度增长作为评价束流品质的标准。

图(6.15a)为最终发射度增长和渐变周期数 N 的关系。在相同的共振穿越速度下, 从上方穿越的发射度增长远大于从下方穿越的发射度增长, 这表明了和我们在图(6.13)中解释的相同的物理机制。对于不同的共振穿越速度, 从上方穿越和从下方穿越都表明束流应尽可能快地穿越结构共振禁带, 以避免显著的发射度增长。图(6.15b)为发射度增长和束流在共振禁带内的周期数的关系, 发射度增长与束流停留在结构共振禁带中的时间几乎线性相关。

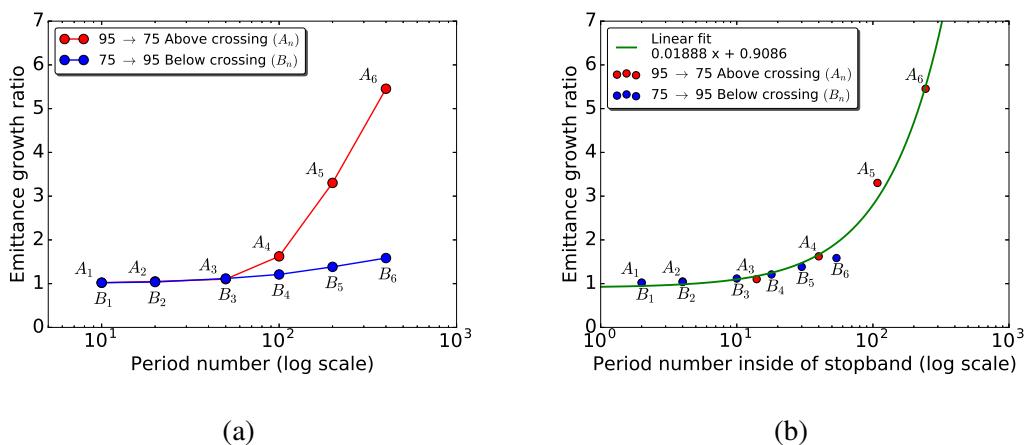


图 6.15: a): 400个FODO周期后的最终发射度增长率与用于共振跨越控制的FODO周期数的关系; b): 400个FODO周期后的最终发射度增长率与束流在结构共振禁带中的有效周期数的关系。

6.3.3 单粒子动力学 – 非相干效应

从非相干单粒子动力学的角度来看, 因为之前所讨论的周期相移主要在 $\sigma \sim 90^\circ$ 附近, 人们很直观的会将4重结构共振归因于4阶particle-core共振($90^\circ / 360^\circ = 1 : 4$)⁴。因此, 结构共振穿越过程中的particle-core共振岛向核心运动或向外运动的方向被用来解释所形成的四重相空间结构以及当结构共振从下方和上方穿越时束晕的严重差异 [93]。

图(6.16)为使用particle-core共振模型的单粒子在相空间中的Poincaré截面 [51, 94]。很显然, 当从上方穿越结构共振禁带时, 通过分叉过程 [48, 95]在原点产生四个particle-core 共振岛, 并在穿越过程中逐渐向外移动(图(6.16a) → 图(6.16d));

⁴360°在这里用来表示束核的振荡频率, 即在一个FODO周期中完全匹配束流的束核振荡一次。

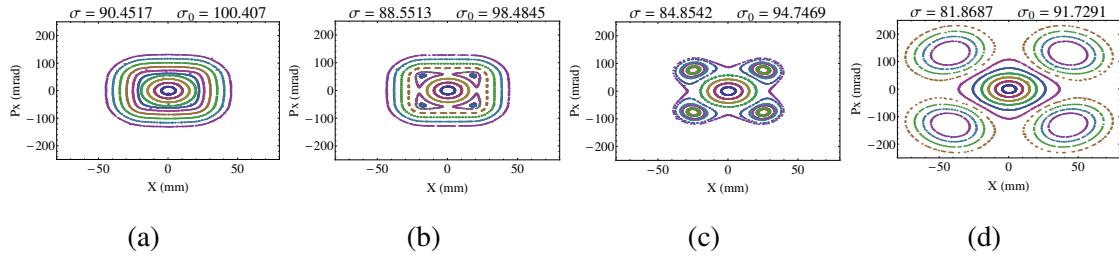
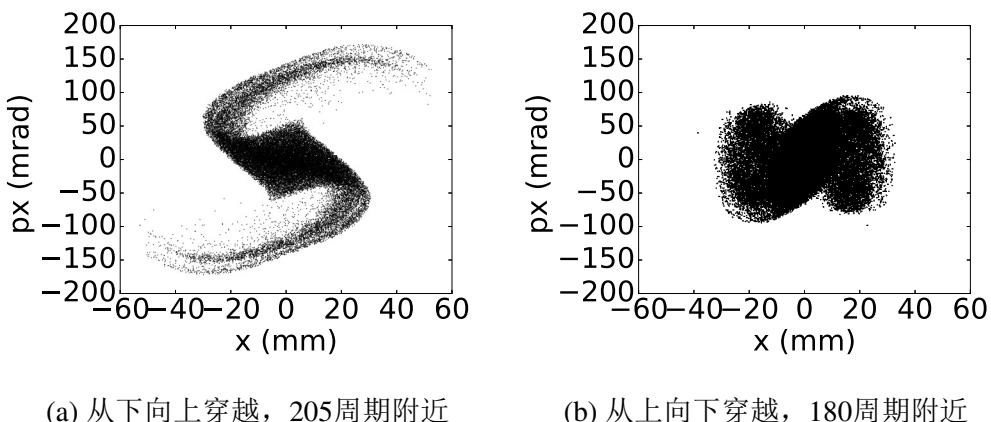


图 6.16: 使用粒子束核模型得到的单粒子Poincaré截面

相反的，当从下方穿越时，particle-core 共振岛从外向核内移动 (图 (6.16d) → 图 (6.16a))。人们直观上会认为相空间中的四重结构与这种particle-core共振有关。一方面因为这种particle-core 共振在这种共振图像中可以逐渐将核心中的粒子带到外部并形成稳定的四重相空间结构；另一方面因为如果particle-core 共振岛从外部朝向内核移动，由于最初外部没有粒子，较少的粒子会占据四重共振岛并且生成更少的束晕。在目前的理解下这个图像可能会产生束晕，但是我们的研究认为这个图像并不对束晕的产生起主导作用。然而，这是不对的，因为如果空间电荷效应是束流系统中唯一的非线性效应，则 N 重共振岛结构主要取决于相干集体效应，而不是非相干 particle-core共振。例如，非相干的四阶particle-core共振不能预测模拟中从下方穿越和从上方穿越都出现的相空间中的两个共振岛结构，如图 (6.17) 所示，而这正是二阶结构共振（相干效应）的证据。正如之前的研究 [49] 所指出的，我们认为，非相干 particle-core共振仅在长时间尺度上可能导致四重相空间结构。在这里，我们认为在模拟中形成的四重相空间结构对应于二阶/四阶混合相干结构共振。



(a) 从下向上穿越，205周期附近

(b) 从上向下穿越，180周期附近

图 6.17: 相空间分布上的两个共振岛结构示意图

6.3.4 总结

我们的研究清楚地揭示了束流穿越结构共振禁带时的瞬态行为。二阶/四阶混合相干结构共振对PIC模拟得到的结果给出了相当合理的解释。必须再次强调的是，束流和结构共振之间的相互作用需要在瞬态的意义上进行研究。由于束流的均方根特性被结构共振所改变，共振禁带的“吸引”和“排斥”效应是束流的一种自发反应。研究还发现，束流的发射度增长几乎与束流受到结构共振影响的时间正相关。束流的最终平衡态是结构共振和非线性阻尼的综合结果。非相干particle-core共振可能造成粒子相位空间扭曲、发射度增长、或形成束晕，但其需要一个较长的时间尺度。

另一方面，我们的模拟清楚地证明了之前的理论预测，即低阶禁带自然地包含在高阶禁带中。相同的理解还可以扩展到关于高阶结构共振的研究。例如在 60° 周期相移附近($3 \times 60^\circ = 180^\circ, 6 \times 60^\circ = 360^\circ$) [51]和 120° 周期相移附近($3 \times 120^\circ = 360^\circ, 6 \times 120^\circ = 720^\circ$) [80, 84]的三阶/六阶混合结构共振。而对与空间电荷相关的相干和非相干共振的理解能够让我们更好地理解最近的实验结果 [87, 96, 97]。然而，尽管对非线性现象的解释有了一定进展，但是理论、数值模拟、和实验结果之间仍然存在着很大的距离。在今后的研究中，我们还需要对实际的6D粒子动力学做进一步的研究。

6.4 小结

首先，我们使用P-TOPO程序对C-ADS注入器I的RFQ和超导段分别进行了模拟，并且与其他程序进行了比较。模拟结果证明了现有设计的合理性，束团的尺寸和发射度都得到了有效控制，束损以及能散也在合理范围之内，完全满足需求。P-TOPO模拟结果其他程序结果的比较也验证了P-TOPO的正确性。

之后，我们在周期性聚焦结构中使用Symplectic算法进行了一个由于加速器元件和空间电荷效应导致的三阶共振模拟。当工作点远离共振线时，束流不会出现发射度增长，而当其接近共振线时发射度会持续增长。

最后，我们使用P-TOPO程序对加速器中的共振穿越进行了研究。研究清楚地揭示了束流穿越结构共振禁带时的瞬态行为。研究也表明束流的发射度增长与束流受到结构共振影响的时间正相关。

将来我们将继续对P-TOPO进行拓展并加入更多新的功能，以满足强流加速器的各种需求，并实际的粒子动力学的研究做进一步的研究。

第七章 论文总结与课题研究展望

通过本论文的工作，我们对空间电荷效应以及其引起束晕现象有了较为深刻的理解，并且根据PIC算法编写出了一个新的粒子模拟程序P-TOPO来处理强流加速器中的非线性效应。在程序开发的过程中，我们了解并掌握束流模拟程序底层的计算过程。本论文还讨论了一种计算空间电荷效应的新算法。为了优化程序效率，我们在GPU上并行实现了PIC算法和symplectic算法，在CPU集群上实现了PIC算法，均取得了令人满意的加速比。

本论文在GPU上实现了基于PIC方法的多粒子模拟程序，并对如何避免线程竞争、目前采用的GPU代码结构、以及对应的并行策略进行了介绍。在单个GPU卡上，我们使用一个普通的家用GPU（GTX 1060）能够实现超过50倍的加速。当粒子数较大时，程序在GPU集群上也显示出良好的可扩展性。我们也在基于最新CPU架构的计算平台KNL上实现了PIC程序，并探索了其最佳性能的并行线程配置。通过PIC程序在CPU与GPU上的运算时间比较，可以看出单GPU运行的程序和使用4或8个节点的程序性能相当。

本文我们还研究了一种求解空间电荷效应的新算法，Symplectic算法。这个算法能够保证辛条件，同PIC算法相比，Symplectic算法以增加运算量为代价效降低了由于网格热效应带来的发射度增长。Symplectic算法非常适合GPU并行化。我们在GPU上实现了这种新算法，在一个普通家用GPU上，Symplectic程序获得了超过450倍的加速比。同时，我们在GPU集群Titan上的测试还显示出这种算法有良好的可扩展性，程序的加速比随着GPU数目几乎线性增加。

在P-TOPO程序的正确性得到验证之后，我们研究了几个空间电荷相关的束流物理问题。P-TOPO程序被应用在了C-ADS注入器I的RFQ和超导段中，我们使用P-TOPO对C-ADS注入器I进行了模拟，并且与其他程序进行了比较。模拟结果显示束团在ADS注入器I的传输过程中，束团的尺寸和发射度都得到了有效控制，束损以及能散也在合理范围之内。结果表明整体加速器的设计是合理的。我们还在周期性聚焦结构中使用Symplectic程序进行了三阶共振模拟，当工作点远离共振线时，束流不会出现发射度增长；而当其接近共振线时发射度会持续增长。之后，我们使用P-TOPO对加速器中的共振穿越问题进行了模拟研究，讨论了束流在穿越相干结构共振禁带时的瞬态行为，并且揭示了束流发射度增长与其受到结构共振影响的时间的关系。

在之后的研究和工作中，我们将继续研究空间电荷效应对强流加速器的影响，

同时对P-TOPO进行拓展并加入更多新的功能，以满足强流束流模拟和动力学研究的各种需求。我们还将继续提高束流模拟程序效率，研究程序在新型计算机架构上的优化策略，不断对P-TOPO进行更新。除此之外，我们还希望不断探索新的物理模型和数值算法，考虑更多的物理效应，并将其包括进模拟程序中。

附录 A 粒子模拟程序用户界面

为了提高模拟程序的易用性，我们基于Impact[36]开发了图形化的用户界面（GUI）。GUI使用python内置的tkinter模块进行开发，并依赖matplotlib, numpy, scipy三个库。GUI可以跨平台使用，我们在Windows, Linux, MacOS均进行了测试。GUI主要分为主界面，预处理，后处理三部分。用户可在github上自由下载本界面：

<https://github.com/zhicongliu/ImpactGUI.git>

GUI主界面如图(A.1)所示，菜单栏包括“存档”、“读取”、“切换内核”、以及“帮助”等功能。用户可以在主界面内设置Impact运行所需要的参数，比如初始时间步长，步数，束流能量，流强，空间电荷网格数，粒子分布，Lattice结构等。除此之外，用户还可以通过高级设置（Advanced Setting）对输入参数进行更加详细的设置。在用户输入相应的参数后，点击运行（Run）按钮，GUI会自动调取Impact内核，使用所给定参数进行模拟。GUI还包括数据预处理（Pre-Process）和后处理（Post-Process），预处理主要用于计算腔体的同步相位。GUI后处理主要用于数据可视化，根据Impact内核产生的输出文件作图。

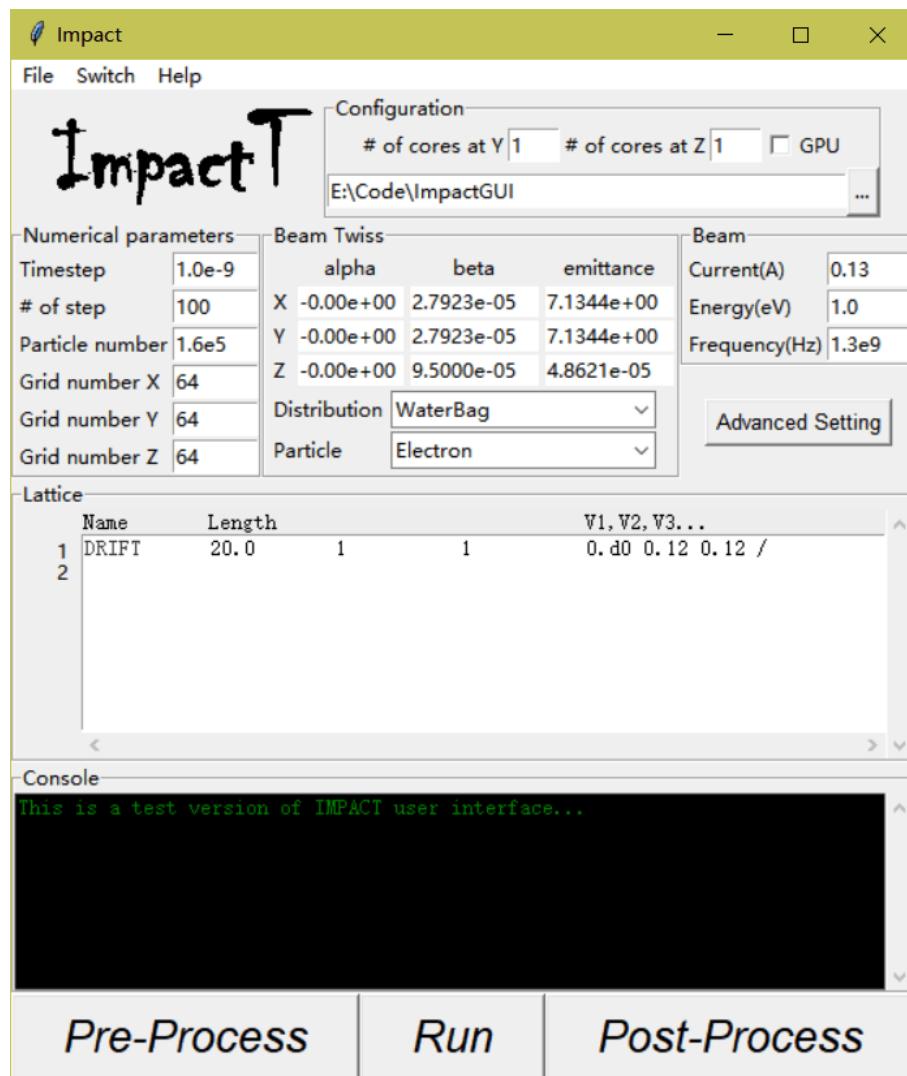


图 A.1: Impact用户界面

附录 B 二阶集体模的雅克比矩阵

记 $C_{i,j} = i/\beta_x + j/\beta_y$, 则对于二阶偶数模:

$$\begin{aligned}
 J_{21} &= -C_{2,0}^2 - C_{2,0} \frac{2K}{\epsilon_x} \frac{a(2a+b)}{2(a+b)^2} \\
 J_{22} &= \frac{C'_{2,0}}{C_{2,0}} \\
 J_{23} &= -C_{2,0} \frac{2K}{\epsilon_x} \frac{ab}{2(a+b)^2} \\
 J_{41} &= -C_{0,2} \frac{2K}{\epsilon_x} \frac{ab}{2\Gamma(a+b)^2} \\
 J_{43} &= -C_{0,2}^2 - C_{0,2} \frac{2K}{\epsilon_x} \frac{b(a+b)}{2\Gamma(a+b)^2} \\
 J_{44} &= \frac{C'_{0,2}}{C_{0,2}};
 \end{aligned} \tag{B.1}$$

则对于二阶奇数模:

$$\begin{aligned}
 J_{21} &= -C_{1,1}^2 - C_{1,1} \frac{2K}{\epsilon_x} \frac{ab(1+\Gamma)}{2\Gamma(a+b)^2} \\
 J_{22} &= \frac{C'_{1,1}}{C_{1,1}} \\
 J_{23} &= -C_{1,1} \frac{2K}{\epsilon_x} \frac{ab(-1+\Gamma)}{2\Gamma(a+b)^2} \\
 J_{41} &= -C_{1,-1} \frac{2K}{\epsilon_x} \frac{ab(1+\Gamma)}{2\Gamma(a+b)^2} \\
 J_{43} &= -C_{1,-1}^2 - C_{1,-1} \frac{2K}{\epsilon_x} \frac{ab(-1+\Gamma)}{2\Gamma(a+b)^2} \\
 J_{44} &= -\frac{C'_{1,-1}}{C_{1,-1}}.
 \end{aligned} \tag{B.2}$$

其中, K 为广义导流系数, a 和 b 为一个周期内的束流尺寸, $\Gamma = \epsilon_x/\epsilon_y$ 为两个自由度发射度的比值。在本文中, 我们使用对称束流, 即 $\Gamma = 1$ 。

参考文献

- [1] WEI J. Synchrotrons and accumulators for high-intensity proton beams[J]. *Reviews of Modern Physics*. 2003, 75 (4): 1383.
- [2] CHOU W. Synchrotron based proton drivers[C]//AIP, AIP Conference Proceedings: volume 642: AIP, 2002: 29–37.
- [3] OWENS J D, HOUSTON M, LUEBKE D, et al. Gpu computing[J]. *Proceedings of the IEEE*. 2008, 96 (5): 879–899.
- [4] NVIDIA C. Programming guide[M]. [S.l.]: [s.n.], 2010.
- [5] Accelerator code list[M/OL]. https://oraweb.cern.ch/pls/hhh/code_website.disp_allcat.
- [6] RYNE R D, HABIB S. Beam dynamics calculations and particle tracking using massively parallel processors[J]. *Part. Accel.* 1996, 55: 119–128.
- [7] TAKEDA H, BILLEN J. Recent Developments in the Accelerator Design Code PARMILA[R]. [S.l.]: [s.n.], 1998.
- [8] QIANG J, RYNE R D, HABIB S, et al. An object-oriented parallel particle-in-cell code for beam dynamics simulation in linear accelerators[C]//ACM, Proceedings of the 1999 ACM/IEEE conference on Supercomputing: ACM, 1999: 55.
- [9] URIOT D, PICHOFF N. Tracewin[J]. CEA Saclay, June. 2014.
- [10] TANKE E, VALERO S, OTHERS. Dynac: A multi-particle beam dynamics code for leptons and hadrons[J]. LINAC02, Gyeongju, TH429. 2002, 667.
- [11] SHISHLO A, COUSINEAU S, DANILOV V, et al. The orbit simulation code: benchmarking and applications[C]//Proceedings of ICAP. 2006.
- [12] ASEEV V, OSTROUMOV P, LESSNER E, et al. Track: The new beam dynamics code[C]//IEEE, Particle Accelerator Conference, 2005. PAC 2005. Proceedings of the: IEEE, 2005: 2053–2055.
- [13] JACKSON J D. Electrodynamics[M]: Wiley Online Library, 1975.
- [14] 郭硕鸿. 电动力学 (第三版)[M]: 高等教育出版社, 1979.
- [15] 秦庆. 高能环形加速器物理[R]. 北京: 中国科学院大学加速器物理讲义, 2011: 6-7.
- [16] LEE S Y. Accelerator physics[M]: World Scientific Publishing Co Inc, 2004.
- [17] CHAO A W, MESS K H, TIGNER M, et al. Handbook of accelerator physics and engineering[M]: World scientific, 2013.

- [18] ABRAMOWITZ M, STEGUN I A. Handbook of mathematical functions: with formulas, graphs, and mathematical tables: volume 55[M]: Courier Corporation, 1964.
- [19] 王书鸿. 质子直线加速器原理[M]: 原子能出版社, 1986.
- [20] 姚充国. 电子直线加速器原理[M]: 科学出版社, 1986.
- [21] WANGER T P. Principles of RF linear accelerator[M]: [s.n.], 1998.
- [22] 吕建钦. 带电粒子束光学[M]: 高等教育出版社, 2004.
- [23] QIANG J. Symplectic multiparticle tracking model for self-consistent space-charge simulation[J/OL]. Phys. Rev. Accel. Beams. Jan 2017, 20: 014203. <http://link.aps.org/doi/10.1103/PhysRevAccelBeams.20.014203>. DOI: 10.1103/PhysRevAccelBeams.20.014203.
- [24] 唐靖宇. 质子加速器基础[R]. 北京: 讲义, 2014: 11-12.
- [25] GLUCKSTERN R L. Analytic model for halo formation in high current ion linacs[J]. Physical review letters. 1994, 73 (9): 1247.
- [26] WANGER T, CRANDALL K, RYNE R, et al. Particle-core model for transverse dynamics of beam halo[J]. Physical review special topics-accelerators and beams. 1998, 1 (8): 084201.
- [27] CHEN C, JAMESON R. Self-consistent simulation studies of periodically focused intense charged-particle beams[J]. Physical Review E. 1995, 52 (3): 3074.
- [28] OKAMOTO H, IKEGAMI M. Simulation study of halo formation in breathing round beams[J]. Physical Review E. 1997, 55 (4): 4694.
- [29] GLUCKSTERN R, FEDOTOV A, KURENNOY S, et al. Halo formation in three-dimensional bunches[J]. Physical Review E. 1998, 58 (4): 4977.
- [30] FEDOTOV A, GLUCKSTERN R, KURENNOY S, et al. Halo formation in three-dimensional bunches with various phase space distributions[J]. Physical Review Special Topics-Accelerators and Beams. 1999, 2 (1): 014201.
- [31] QIANG J, RYNE R D. Beam halo studies using a three-dimensional particle-core model[J]. Physical Review Special Topics-Accelerators and Beams. 2000, 3 (6): 064201.
- [32] HOCKNEY R W, EASTWOOD J W. Computer simulation using particles[M]: crc Press, 1988.
- [33] LUCCIO A, D' IMPERIO N, BEEBE-WANG J. Space charge dynamics simulated in 3-d in the code orbit “[C]//Proceedings of EPAC2002, Paris, France. 2002.

- [34] BIRDSALL C K. Particle-in-cell charged-particle simulations, plus monte carlo collisions with neutral atoms, pic-mcc[J]. IEEE Transactions on Plasma Science. Apr 1991, 19 (2): 65-85. DOI: 10.1109/27.106800.
- [35] FRIEDMAN A, GROTE D P, HABER I. Three-dimensional particle simulation of heavy-ion fusion beams[J]. Physics of Fluids B: Plasma Physics. 1992, 4 (7): 2203–2210.
- [36] QIANG J, RYNE R D, HABIB S, et al. An object-oriented parallel particle-in-cell code for beam dynamics simulation in linear accelerators[J/OL]. Journal of Computational Physics. 2000, 163 (2): 434 - 451. <http://www.sciencedirect.com/science/article/pii/S0021999100965707>. DOI: <http://dx.doi.org/10.1006/jcph.2000.6570>.
- [37] QIANG J, FURMAN M A, RYNE R D. A parallel particle-in-cell model for beam-beam interaction in high energy ring colliders[J]. Journal of Computational Physics. 2004, 198 (1): 278–294.
- [38] AMUNDSON J, SPENTZOURIS P, QIANG J, et al. Synergia: An accelerator modeling tool with 3-d space charge[J/OL]. Journal of Computational Physics. 2006, 211 (1): 229 - 248. <http://www.sciencedirect.com/science/article/pii/S0021999105002718>. DOI: <http://dx.doi.org/10.1016/j.jcp.2005.05.024>.
- [39] URIOT D, PICHOFF N. Tracewin[J]. CEA Saclay, June. 2014.
- [40] BATYGIN Y K. Particle-in-cell code beampath for beam dynamics simulations in linear accelerators and beamlines[J]. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment. 2005, 539 (3): 455–489.
- [41] BIRDSALL C K, LANGDON A B. Plasma physics via computer simulation[M]: CRC press, 2004.
- [42] CHANNELL P, SCOVEL C. Symplectic integration of hamiltonian systems[J]. Nonlinearity. 1990, 3 (2): 231.
- [43] YOSHIDA H. Construction of higher order symplectic integrators[J]. Physics Letters A. 1990, 150 (5-7): 262–268.
- [44] LI Z, CHENG P, GENG H, et al. Physics design of an accelerator for an accelerator-driven subcritical system[J]. Physical Review Special Topics-Accelerators and Beams. 2013, 16 (8): 080101.
- [45] HENDERSON S, ABRAHAM W, ALEKSANDROV A, et al. The spallation neutron source accelerator system design[J]. Nuclear Instruments and Methods in

- Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment. 2014, 763: 610–673.
- [46] ESHRAQI M, DANARED H, DE PRISCO R, et al. Ess linac beam physics design update[J]. MOPOY045, IPAC16, Busan (these proceedings). 2016.
- [47] 李超. TOPOPIC程序开发以及对FODO结构中束晕抑制问题的研究[D]. 兰州: 中国科学院研究生院(近代物理研究所), 2014.
- [48] LI C, LIU Z, ZHAO Y, et al. Nonlinear resonance and envelope instability of intense beam in axial symmetric periodic channel[J]. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment. 2016, 813: 13–18.
- [49] LI C, ZHAO Y L. Envelope instability and the fourth order resonance[J]. Physical Review Special Topics-Accelerators and Beams. 2014, 17 (12): 124202.
- [50] LI C, QIN Q. Collective beam instability and beam halo due to space charge[C]//JACoW, Proc. of ICFA Advanced Beam Dynamics Workshop on High-Intensity and High-Brightness Hadron Beams (HB'16): JACoW, .
- [51] LI C, QIN Q. Space charge induced beam instability in periodic focusing channel[J]. Physics of Plasmas. 2015, 22 (2): 023108.
- [52] STUDIO M. Cst-computer simulation technology[J]. Bad Nuheimer Str. 2008, 19: 64289.
- [53] CHEN C, DAVIDSON R C. Nonlinear properties of the kapchinskij-vladimirskij equilibrium and envelope equation for an intense charged-particle beam in a periodic focusing field[J]. Physical Review E. 1994, 49 (6): 5679.
- [54] NVIDIA C. CUFFT library[M]. [S.l.]: [s.n.], 2010.
- [55] TIWARI D, GUPTA S, GALLARNO G, et al. Reliability lessons learned from gpu experience with the titan supercomputer at oak ridge leadership computing facility[C]//ACM, Proceedings of the international conference for high performance computing, networking, storage and analysis: ACM, 2015: 38.
- [56] WELLS J, BLAND B, NICHOLS J, et al. Announcing Supercomputer Summit[R]. [S.l.]: [s.n.], 2016.
- [57] HE Y, COOK B, DESLIPPE J, et al. Preparing nersc users for cori, a cray xc40 system with intel many integrated cores[J]. Concurrency and Computation: Practice and Experience. 2018, 30 (1).
- [58] Titan[M/OL]. <https://www.olcf.ornl.gov/computing-resources/titan-cray-xk7/>.

- [59] Summit[M/OL]. <https://www.olcf.ornl.gov/summit/>.
- [60] Cori Knight Landing[M/OL]. <http://www.nersc.gov/users/computational-systems/cori/configuration/cori-intel-xeon-phi-nodes/>.
- [61] 李智慧, 闫芳, 耿会平, 等. 中国ads质子加速器物理设计[J]. 中国核科学技术进展报告(第二卷)——中国核学会2011年学术年会论文集第6册(核物理分卷, 计算物理分卷, 粒子加速器分卷). 2011.
- [62] FANG Y, ZHI-HUI L, CAI M, et al. Physics design for the c-ads main linac based on two different injector design schemes[J]. Chinese physics C. 2014, 38 (2): 027004.
- [63] FANG Y, MENG C, GENG H P, et al. Instability investigation of china-ads injector-isc section design[C].
- [64] 孟才. 中国ADS加速器误差分析和束流损失机制的研究[D]. 北京: 中国科学院大学, 4 2014.
- [65] CRANDALL K, WANGLER T. Parmteq (phase and radial motion in transverse electric quadrupole linacs): A beam dynamics code for the rfq (radio-frequency quadrupole)[M]: IA-UR-88-1546: [s.n.], Jan 1987.
- [66] CRANDALL K R, WANGLER T P, YOUNG L M, et al. RFQ design codes[R]. [S.l.]: [s.n.], 1998.
- [67] REISER M. Theory and design of charged particle beams[M]: John Wiley & Sons, 2008.
- [68] FEDOTOV A, PAPAPHILIPPOU Y, WEI J, et al. Space-charge and resonance considerations for choosing the SNS ring working points[R]. [S.l.]: [s.n.], 2001.
- [69] FRANCHETTI G, HOFMANN I. Particle trapping by nonlinear resonances and space charge[J]. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment. 2006, 561 (2): 195–202.
- [70] SACHERER F J. Transverse space-charge effects in circular accelerators[M]: [s.n.], 1968.
- [71] SACHERER F. A longitudinal stability criterion for bunched beams[J]. IEEE Transactions on Nuclear Science. 1973, 20 (3): 825–829.
- [72] CHAO A W. Physics of collective beam instabilities in high energy accelerators[M]: Wiley, 1993.
- [73] GLUCKSTERN R L. Oscillation modes in two dimensional beams.[C]//Conf. Proc.: volume 700928. 1970: 811.

- [74] GLUCKSTERN R L, CHASMAN R, CRANDALL K. Stability of phase space distributions in two dimensional beams[C]//Conf. Proc.: volume 700928. 1970: H-6.
- [75] SACHERER F J. Rms envelope equations with space charge[J]. IEEE Transactions on Nuclear Science. 1971, 18 (3): 1105–1107.
- [76] STRUCKMEIER J, REISER M. Theoretical studies of envelope oscillations and instabilities of mismatched intense charged particle beams in periodic focusing channels[J]. Part. Accel. Apr 1983, 14 (GSI-83-11. GSI-PRE-83-11): 227-260. 44 p.
- [77] LUND S M, BUKH B. Stability properties of the transverse envelope equations describing intense ion beam transport[J]. Physical Review Special Topics-Accelerators and Beams. 2004, 7 (2): 024801.
- [78] YUAN Y S, BOINE-FRANKENHEIM O, HOFMANN I. Modeling of second order space charge driven coherent sum and difference instabilities[J/OL]. Phys. Rev. Accel. Beams. Oct 2017, 20: 104201. <https://link.aps.org/doi/10.1103/PhysRevAccelBeams.20.104201>. DOI: 10.1103/PhysRevAccelBeams.20.104201.
- [79] CHERNIN D. Evolution of rms beam envelopes in transport systems with linear xy coupling[J]. Part. Accel. 1988, 24: 29–44.
- [80] LI C, JAMESON R A, QING Q. Collective beam instability and beam halo due to space charge[C]//in 57th ICFA beam dynamics workshop HB2016, Maloe, Sweden 2016, MOPL006; in 7th International Particle Accelerator Conference IPAC2016, Busan, Korea, "THOBA02". .
- [81] LI C, JAMESON R. Structure resonances due to space charge in periodic focusing channels[J]. Physical Review Accelerators and Beams. 2018, 21 (2): 024204.
- [82] HOFMANN I, LASLETT L J, SMITH L, et al. Stability of the kapchinskij-vladimirskij (kv) distribution in long periodic transport systems[J]. Particle Accelerators. 1983, 13: 145.
- [83] HOFMANN I. Stability of anisotropic beams with space charge[J]. Physical Review E. 1998, 57 (4): 4713.
- [84] JEON D O, HWANG K R, JANG J H, et al. Sixth-order resonance of high-intensity linear accelerators[J/OL]. Phys. Rev. Lett. May 2015, 114: 184802. <http://link.aps.org/doi/10.1103/PhysRevLett.114.184802>. DOI: 10.1103/PhysRevLett.114.184802.
- [85] HOFMANN I, BOINE-FRANKENHEIM O. Space-charge structural instabilities and resonances in high-intensity beams[J/OL]. Phys. Rev. Lett. Nov 2015, 115: 204802.

- <https://link.aps.org/doi/10.1103/PhysRevLett.115.204802>. DOI: 10.1103/PhysRevLett.115.204802.
- [86] GROENING L, HOFMANN I, BARTH W, et al. Experimental evidence of space charge driven emittance coupling in high intensity linear accelerators[J]. Physical review letters. 2009, 103 (22): 224801.
- [87] ITO K, OKAMOTO H, TOKASHIKI Y, et al. Coherent resonance stop bands in alternating gradient beam transport[J]. Physical Review Accelerators and Beams. 2017, 20 (6): 064201.
- [88] KAPCHINSKIJ I M, VLADIMIRSKIJ V V [J]. Proceedings of the 9th International Conference on High Energy Accelerators, (CERN, Geneva, 1959), P. 274. 1959.
- [89] SINGH R, FORCK P, KOWINA P, et al. Observations of the quadrupolar oscillations at gsi sis-18[J]. IBIC Proceedings. 2014.
- [90] OEFTIGER A. Quadrupolar Pick-ups to Measure Space Charge Tune Spreads of Bunched Beams[M]. [S.l.]: [s.n.], 2017.
- [91] LI C, ZHANG Z, QI X, et al. Beam dynamics study of rfq for c-ads with a 3d space-charge-effect[J]. Chinese physics C. 2014, 38 (3): 037005.
- [92] LIU Z, LI C, QIN Q, et al. Beam dynamics study of c-ads injector-i with developing p-topo code[C]//JACOW, Geneva, Switzerland, 57th ICFA Advanced Beam Dynamics Workshop on High-Intensity and High-Brightness Hadron Beams (HB'16), Malmö, Sweden, July 3-8, 2016: JACOW, Geneva, Switzerland, 2016: 195–198.
- [93] JEON D, GROENING L, FRANCHETTI G. Fourth order resonance of a high intensity linear accelerator[J]. Physical Review Special Topics-Accelerators and Beams. 2009, 12 (5): 054204.
- [94] JAMESON R A. Self-consistent beam halo studies & halo diagnostic development in a continuous linear focusing channel[C]: World Scientific, ISBN 981-02-2537-7, pp.530-560. LA-UR-94-3753, Los Alamos National Laboratory. AIP Proceedings of the 1994 Joint US-CERN-Japan International School on Frontiers of Accelerator Technology, Maui, Hawaii, USA, 3-9 November 1994, 1994.
- [95] DILÃO R. Nonlinear dynamics in particle accelerators: volume 23[M]: World Scientific, 1996.
- [96] GROENING L, HOFMANN I, BARTH W, et al. Experimental evidence of space charge driven emittance coupling in high intensity linear accelerators[J]. Physical review letters. 2009, 103 (22): 224801.

- [97] JEON D O. Experimental evidence of space charge driven resonances in high intensity linear accelerators[J]. Physical Review Accelerators and Beams. 2016, 19 (1): 010101.