



北京大学

本科生毕业论文

题目: GPU 并行加速在
Herschel 数据处理系统
的应用研究

姓 名: 金逸飞

学 号: 00904173

院 系: 物理学院

专 业: 天文学

研究方向: 天文高新技术与应用

导师姓名: 黄茂海 研究员

二〇一三年六月

版权声明

任何收存和保管本论文各种版本的单位和个人，未经本论文作者同意，不得将本论文转借他人，亦不得随意复制、抄录、拍照或以任何方式传播。否则，引起有碍作者著作权之问题，将可能承担法律责任。

GPU 并行加速在 Herschel 数据处理系统的应用研究

金逸飞 天文学

导师姓名：黄茂海 研究员

摘要

Herschel空间望远镜是目前最大口径的红外空间望远镜。它由欧洲空间局主要研制，于2009年5月14日发射升空，2013年4月29日结束在轨运行阶段。望远镜口径3.5m，装载了三个后段设备（PACS，SPIRE，HIFI），观测波段覆盖 $55 - 672 \mu m$ 。约2300升液氮支持望远镜稳定工作了近4年，积累了超过25 000个小时的科学数据，期间完成超过600个观测项目，观测了超过35 000个天体源。此外还有超过2000个小时的定标源数据。可以说，虽然Herschel的观测任务已经结束，但是对观测数据的科学分析还远远没有结束。

由于 Herschel 需要制冷剂运行，所以寿命有限。为此，在赫歇尔在轨运行期间，整个赫歇尔工作群体，包括科学家、工程师、管理人员等，他们的工作重点在快速定标、成功观测、尽量快速分析出初步结果以指导下一步观测。虽然 Herschel 第一年正式运行就发表了130多篇SCI论文，但深入挖掘赫歇尔数据的工作随着在轨运行阶段的结束才真正开始。天文界未来将有多年深入挖掘赫歇尔远红外数据的任务，所以如何提高赫歇尔数据处理系统的执行效率是一个具有紧迫性和重要性的课题。

Herschel 数据处理系统采用 java 语言编写内部类，并用 jython 编写了友好的图形界面，实现交互式数据处理过程。java 和 jython 的结合，一方面使得软件系统方便的实现了一体化的设计思想，从卫星发射，到卫星观测任务规划，数据

下载，数据处理等，另一方面也使得科学家编程效率都得到了很大的提高。但是 java 和 jython 都是面向对象的语言，虽然很大程度上方便了软件的组织构架，却一定程度上牺牲了一些机器效率，致使一些很好的数据处理算法由于效率问题没有办法得到实施。高效的数据处理系统，是将观测数据变现为科学成果的关键一步，所以在不破坏系统基本构架的情况下，如何最大限度的提高系统效率，成为一个十分严峻的问题。

另一方面，近年来 GPU 在加速计算领域扮演着越来越重要的角色。GPU 区别于传统的 CPU，它简化了复杂的控制系统，缩减了运算单元的指令集，但大大增加了逻辑运算单元的数量。这种设计的改进，使得它在并行运算上比传统的 CPU 有了极大的提速。

本文基于HCSS 10.0版本 (Herschel Common Science System)，讨论了将 GPU 引入 Herschel 数据处理系统的可能性。基本思想是探究主要的处理脚本是不是最后能把耗时的运算归约到几个主要的基本类上，然后再讨论如何能将这几个基本类用GPU进行加速。

本文构建了一套比较完整的方法来分析一个处理脚本花时间最多的类有哪些，并具体分析这些类花时间占整个脚本总时间的百分比。作为例子，本文处理了PACS的深场点源成像数据流水线脚本， SPIRE 的扫描成像数据流水线和 SPIRE 的傅里叶光谱处理流水线，得到的基本结论是： Double1d, Complex1d 和 xform 是主要的三个运算耗时类。

本文进一步探索了用 GPU 加速这些目标类的可能性。首先 GPU 的基本编程语言是 CUDA，以 nvcc 为编译器，类似 C 语言，并不能直接在 java 语言中运行。本文介绍了用 jni (java native interface) 技术实现 java 语言对 GPU 的调用的方法。并作为例子，改进了两个主要的函数方法，分别是快速傅里叶变换和矩阵乘法，并比较了CPU 和 GPU 对这种基本运算的运行效率差别。

最后本文还讨论了一下GPU加速会取得较好或较差效果的情况，作为例子，本文用GPU加速的类代替原来系统的类，处理了 SPIRE 光谱和 SPIRE 扫描成像， PACS 扫描成像脚本。对比了 CPU 和 GPU 处理的时间效，在 FTS (傅里叶光谱) 光谱处理中取得了较好的效果，时间效率提高了近一倍。

关键词： GPU， 并行， 赫歇尔， 数据处理

GPU Parallel and Herschel Data Processing System

Yifei. Jin Astronomy

Directed by Prof. Maohai Huang

Abstract

The Herschel Space Observatory was built and operated by European Space Agency (ESA), which active from May 14th,09 to May 29th, 13. Herschel is the largest infrared telescope which carry a 3.5 meter primary mirror and three instruments — PACS,SPIRE and HIFI — which are sensitive to the far infrared and submillimetre wavebands ($55 - 672 \mu m$). The 2300 liters of liquid helium kept Herschel working well for almost 4 years. Herschel has made over 35 000 scientific observations, amassing more than 25 000 hours of science data from about 600 observing programmes. A further 2000 hours of calibration observations also contribute to the rich dataset. Although the telescope has relaxed, it will keep astronomers busy for many years to come.

Limited by the volume of liquid helium, Herschel has finished its on-orbit phase. Because of its limited life, during its in-orbit phase, all people including scientists, engineers, and managers, focus on calibration, observation, and how to analyse the data as soon as possible to get primary results. Although there are lots of SCI papers owing to Herschel, deep mining of the Hershcel data hasn't begun until the in-orbit phase finished. It will be a big challenge to deep analyze Herschel data in the

following years. So it is significant to improve the performance of data processing software, especially its efficiency.

In order to process the data, ESA also developed a strong data processing software — Herschel Common Science System (HCSS) — which used to download data and process. The system, whose inner class is based on java and user interface is based on jython, is very convinence for astronomers. But the efficiency of java and jython which are object-based language is low. How to improve the performance of the system become more and more important.

On the other hand, the graph processing unit(GPU) plays a important role in today Scientific Computing field, along with the evolution of the programming language CUDA. GPU, which contains less controllers and more Arithmetic Logic Unit(ALU), is different from Central Processing Unit(CPU).

This paper talk about whether and how we can import GPU in HCSS to improve its performance. The essential idea is to find the most time cost classes and methods in the pipeline. And then try to improve the class using GPU.

This paper construct an feasible methods to find the most time cost class in the pipeline. As examples, the paper test the deep survey miniscan pointsource pipeline of PACS, and small map pipeline, large map pipeline and spectrometer mapping pipeline of SPIRE. We find that the most time cost classes are Double1d, Complex1d and xform.

Further, we use GPU to improve these classes and test whether GPU can improve the performance of HCSS. As an example, the spectrometer mapping pipeline takes most of time to calculate FFT and we use CudaFFT to instead of FFT, the time cost of the pipeline decline half.

Keywords: GPU, CUDA, parallel, Herschel, HCSS, Data Processing

目录

引言	1
0.1 大数据时代的天文学	1
0.2 Herschel 数据处理系统加速的重大意义	2
0.3 GPU 并行处理和 Herschel 数据处理系统	3
第一章 Herschel空间天文台及HCSS数据处理系统	5
1.1 Herschel红外空间天文台简介	5
1.1.1 Herschel小史	5
1.1.2 望远镜主要参数	6
1.1.3 后端设备	8
1.1.4 观测和研究任务	9
1.2 Herschel通用科学系统	9
1.2.1 软件构架	9
1.2.2 HDPS软件模块	10
1.2.3 HCSS开发简介	13
第二章 GPU并行编程简介	17
2.1 GPU与高效能计算	17
2.1.1 CPU和GPU的性能对比	17
2.1.2 GPU简介	19
2.2 GPU编程简介	20
2.2.1 CUDA编程方法	20
2.2.2 CUDA的常用类库	26

2.2.3 GPU的拓展应用	27
2.3 GPU的发展前景	28
第三章 HCSS脚本效率分析方法	29
3.1 HCSS中运算基础类	29
3.2 脚本运行效率分析方法	31
3.2.1 脚本分析思路和分析方法概貌	31
3.2.2 计数模块	33
3.2.3 计时模块	34
3.3 时间统计的预处理和后续处理过程	36
3.3.1 预处理过程	39
3.3.2 后续处理过程	40
第四章 HCSS 系统函数的 GPU 实现方式	45
4.1 JNI技术实现java语言对CUDA的调用	45
4.2 JCUDA库简介	50
4.3 典型的 HCSS 基础类在 CPU 与 GPU 上运行的效率对比	51
第五章 分析和加速HIPE数据流水线：SPIRE 光谱处理	55
5.1 HIPE数据流水线	55
5.2 分析和加速 SPIRE 光谱处理流水线	57
5.2.1 SPIRE傅里叶光谱数据流水线	57
5.2.2 SPIRE光谱处理流水线分析	59
5.2.3 GPU对SPIRE光谱流水线的加速效果	62
5.3 分析和加速SPIRE 光谱处理流水线小结	65
第六章 分析和加速HIPE数据流水线：SPIRE和PACS扫描成像数据处理	67
6.1 SPIRE 扫描成像数据处理	67
6.1.1 SPIRE 扫描成像流水线	68
6.1.2 SPIRE 扫描成像流水线分析	69
6.1.3 GPU对SPIRE扫描成像流水线的加速效果	71
6.2 PACS 扫描成像数据处理	75

6.2.1 PACS 扫描成像数据流水线	75
6.2.2 PACS 成像流水线分析	76
6.2.3 PACS 成像数据流水线GPU加速效果	80
6.3 分析和加速 SPIRE 和 PACS 扫描成像流水线小结	80
6.3.1 SPIRE 扫描成像小结	80
6.3.2 PACS 扫描成像小结	83
第七章 总结与展望	85
7.1 总结	85
7.2 本工作对HCSS系统设计的指导意义	86
7.3 展望	86
参考文献	89
附录 A java 内核代码	95
A.1 HCSS矩阵乘法和快速傅里叶变换的GPU实现	95
A.1.1 矩阵乘法的GPU实现	95
A.1.2 FFT/IFFT的GPU实现	100
A.1.3 矩阵乘法 CPU和GPU效率对比	102
A.1.4 FFT CPU和GPU效率对比	106
A.2 HCSS脚本分析中的计数辅助类和计时辅助类	110
A.2.1 FFT 计数辅助类	110
A.2.2 FFT 计时辅助类	117
附录 B jython 脚本代码	121
B.1 GPU 和 CPU 效率对比	121
B.1.1 矩阵乘法	121
B.1.2 快速傅里叶变换	123
B.2 HCSS 脚本分析框架	126
B.3 Complex1d 计数和时间花费分析	128
B.3.1 Complex1d 计数分析	128
B.3.2 Complex1d 时间花费分析	129

B.4 Double1d 计数和时间花费分析	133
B.4.1 Double1d 计数分析	133
B.4.2 Double1d 时间花费分析	134
B.5 Double2d 计数和时间花费分析	139
B.5.1 Double2d 计数分析	139
B.5.2 Double2d 时间花费分析	139
B.6 FFT 计数和时间花费分析	140
B.6.1 FFT 计数分析	140
B.6.2 FFT 时间花费分析	141
B.7 MatrixMultiply 计数和时间花费分析	141
B.7.1 MatrixMultiply 计数分析	141
B.7.2 MatrixMulyiply 时间花费分析	143
B.8 SingularValueDecomposition 计数和时间花费分析	144
B.8.1 SVD 计数分析	144
B.8.2 SVD 时间花费分析	146
致谢	147

引言

虽然看似与人们的生产生活不直接挂钩的天文学，却一次次的从现代技术中汲取营养，并刺激技术的一次次革新，可以说天文学一直走在各种技术的最前沿。现代天文学从古代占星学中脱胎而来，见证了望远镜的出现，见证了射电天线的应用，见证了电荷耦合器件（CCD，Charge Coupled Device）代替传统的感光底片，见证了辐射热测定器（bolometer）等一系列红外探测设备开辟了红外天文学，见证了现场可编程门阵列（FPGA）在系统实时控制中展露头角，见证了主动光学和自适应光学在反馈控制系统中的发展。天文学是一门古老而新的科学，是人们研究领域的一个极端代表。人们作为宇宙渺小的一瞬，要对浩瀚的宇宙在极大的空间和时间尺度上有所了解，自然也涉及到对人类技术的极端追求。可以说天文学不仅从通讯，自动化，材料，计算机等诸多领域吸收最前沿的技术，还推动这些领域的发展。当然，进入21世纪，天文学和天文技术也面临着新的问题和挑战。

0.1 大数据时代的天文学

进入信息时代，海量数据处理成为天文学的又一大趋势。随着大型的天文仪器和设备不断投入使用，天文观测对象不断向高红移，低亮度方向发展，积分时间加长，分辨率增高，数据量也越来越大。一方面空前的观测结果令人兴奋，另一方面海量的天文数据如何处理也是一个不小的难题。

21世纪的天文学早已经不是感光底片的时代了。以SDSS数字巡天(Sloan Digital Sky Survey SDSS; <http://www.sdss.org>)为例，它的第9次数据发布（DR9）已经覆盖天区达到14 555平方度，观测源达932 891 133个，其中有光谱的星系，类星体和恒星数量分别达到1 457 002，228 468，668 054个。^[1]而下一代的天文望

远镜数据产生量更为尤甚，以三十米望远镜（TMT，Thirty Meter Telescope）为例，预计其平均数据流量可达0.02 Gbit/s，高峰可达0.10 Gbit/s。^[2]

传统的数据处理速度很大程度上已经赶不上数据的膨胀速度。虽然根据摩尔定律，每18个月硬件设备的计算性能可以翻一番已上。我们可以等CPU性能的提高来处理我们的数据。但是硬件设备的资源总是有限的，而且随着集成电路工艺越来越接近极限，坐等CPU技术的发展显然有守株待兔，不可强求的意味。另一方面我们就需要改进我们的算法，用更快的方法来解决的我们的问题。但是天文学属于极端科学，人们对运算精度的追求总是无止境的，这使很多运算效率更高的近似算法并不被采用。事实上，与快速近似算法恰恰相反，天文学家为了追求精度更高的结果，更细致的物理内容，很多很复杂的算法被设计出来。所以没有好的数据处理手段，这些设计都是徒劳无功的。

0.2 Herschel 数据处理系统加速的重大意义

Herschel红外空间望远镜于 2009 年 5 月 14 日发射升空，2013 年 4 月 29 日结束在轨运行阶段。望远镜主镜口径3.5m，搭载三个后端设备（PACS, SPIRE, HIFI），观测波段从55 到 $672\mu m$ ，装载了约 2300 升液氮，支持望远镜稳定工作近4 年，积累了超过 25 000 个小时的科学数据，期间完成超过 600 个观测项目，观测了超过35 000 个天体源。此外还有超过 2000 个小时的定标源数据。

由于赫歇尔需要制冷剂运行，所以寿命有限。为此，在赫歇尔在轨运行期间，整个赫歇尔工作群体，包括科学家、工程师、管理人员等，他们的工作重点在快速定标、成功观测、尽量快速分析出初步结果以指导下一步观测。虽然赫歇尔第一年正式运行就发表了130多篇SCI论文，但深入挖掘赫歇尔数据的工作随着在轨运行阶段的结束才真正开始。

展望未来10-20年，大型红外空间设备只有JWST和日本的SPICA。JWST工作波段截止到50多微米，SPICA截止到200多微米，并不能代替赫歇尔的远红外数据，而且SPICA 是否能成功立项还有巨大变数。

天文界未来将有多年深入挖掘赫歇尔远红外数据的任务，所以如何提高赫歇尔数据处理系统的执行效率是一个具有紧迫性和重要性的课题。

0.3 GPU 并行处理和 Herschel 数据处理系统

Herschel 的设计体现了一体化的设计思想，除了强大的硬件系统外，还专门设计了强大的软件支撑 HCSS (Herschel Common Science System). 该系统支持 Herschel 科学数据中心和三个仪器控制中心的工作，包括信息提供，提案的提交和处理，观测调度，观测产品的生成和质量控制，存储，访问和检索数据产品，校准和交叉定标等工作。对于用户来说还提供交互式数据处理体统 HIPE (Herschel Interface Processing Environment)。该系统的内核部分由 java 编写，交互式界面由 jython 编写，这样系统能比较好的实现内部计算效率和编程效率的兼容。不过虽然 java 语言便于系统构建，但是还是会损失一些效率。尤其对于大数据量时，效率就很重要了。

另一方面，GPU (Graph Processing Units) 在科学计算领域开始扮演越来越重要的角色。GPU是计算机的图形运算单元，与CPU (Central processing unit) 不同的是，GPU简化了控制系统，缩减了指令集，而大量增加了逻辑运算单元 (ALU, Arithmetic logic unit)。以 Nvidia 公司出品的新一代 Kepler 构架的 GPU —— GTX 680，它有 1536 个核心。这使得 GPU 在并行运算上有很大优势。

经过多年的发展，GPU 已经不再是纯粹图形处理的硬件，对GPU的操作已经脱离了直接处理图形处理 API(Application Processing Interface)的阶段，基于CUDA (Compute Unified Device Architecture)并行运算平台和编程模型的 GPU 已经有很好的编译语言支持。 CUDA提供类C语言编译器nvcc，用户可以方便的用类似 C 语言高级语言编写自己的 GPU 应用，这使得 CUDA 很容易被掌握运用，而且 CUDA 把 CPU 和 GPU 的功能很好的融合起来，很方便的实现了对 CPU 和 GPU 运算的切换。

如今 GPU 已经在天文学的多个领域取得了成就，典型的如 N-Body 模拟^[3]，实时天文图象处理，计算射电信号的相关性，计算宇宙学常数^[4]等。但是目前这些应用主要集中在改进具体的、单一的天文应用程序上，而且由于目前CUDA良好的支持的语言只有C/C++、fortran语言，所以这些应用还主要集中在修改以前的C/C++、fortran程序上。

但像HCSS这样的通用科学系统，需要处理的应用复杂多样，即使是相同的应用程序，由于要处理的数据源巨大，数据源的大小也有很大的变化范围，要想对这些应用一一进行GPU加速，编程的工作量是不可想象的。另外 HCSS 是基于

纯 java 的系统，目前并没有支持 CUDA 的 java 语言编译器，这也大大增加了在 HCSS 引入 GPU 应用的难度。

本工作就是在此背景下，希望在保持 HCSS 系统基本构架不变的情况下，讨论将 GPU 并行计算引入 Herschel 系统加速运算的可能性。并测试引入 GPU 对 HCSS 系统效率的提高能有多少。本工作设计了一套完整的方法来找到每个处理脚本的主要耗时类，基于 HCSS 系统，在保持外部接口不变的情况下，修改内部类代码，引入 GPU。最后还讨论了一下 GPU 可以实现加速的情况。论文构架为：第一章我们介绍一下 Herschel 卫星和数据处理系统，对软件构架和数据处理过程做简单说明。第二章介绍一下 GPU 和如何使用 CUDA 实现 GPU 编程，并介绍一下如何用 jni (java native interface) 方法，实现 java 代码对 GPU 的调用。第三章介绍如何分析一个 HCSS 脚本的运行效率，找出其中主要耗费时间的基础类的方法。第四章介绍一下典型的 HCSS 基类函数如何用 GPU 重新实现，并对比相同的函数在 CPU 和 GPU 上的实现效率。第五章介绍一下利用我们的脚本分析方法分析 SPIRE 光谱处理流水线的运行效率，找出耗时基类，利用 GPU 加速技术代替其中主要的函数，并分析加速效果和数据处理结果。第六章我们利用同样的方法处理了 SPIRE 和 PACS 的扫描成像数据处理流水线，并分析了 GPU 加速的效果，根据加速结果，对系统设计提出了一些建议。最后第七章为全文总结。

另外需要注明的是，本工作所做的分析和统计的硬件平台为 Dell Inspiron R14。CPU 型号为 Inter Core i5-3210M，GPU 型号为 Geforce GT 640M（具体参数可见第 2.2.1 节）。HCSS 软件版本为 10.0。基于不同平台属性，部分具体结果，如编译指令、时间统计、加速效果等可能略有差异。

第一章 Herschel空间天文台 及HCSS数据处理系统

1.1 Herschel红外空间天文台简介

1.1.1 Herschel小史

当地时间2009年5月14日10点12分，在南美洲法属圭亚那东北部海岸近库鲁（Kourou）火箭发射中心，亚利安五号运载火箭（Ariane 5 ECA）搭载着两颗极具历史意义的天文空间望远镜发射升空，它们中的一颗就是Herschel。（另一颗为Planck）

其实Herschel项目早在1982年就被提出来，当时项目命名为远红外和亚毫米波望远镜（FIRST, Far InfraRed and Submillimetre Space Telescope）经过82-83年的一个可行性研究报告，该项目被列为欧洲航天局（ESA, European Space Agency）“Horizon 2000”长期支持项目当中，经过之后几年的对任务设计、航天器配置、望远镜、科学负载等项目的研究，在1993年ESA科学规划委员会终于决定FIRST项目为其实施的第四个基石项目，此前该级别的项目分别是Rosetta, Planck和Gaia。

FIRST项目在参考了之前ISO（Infrared Space Observation）望远镜的利弊得失之后，于1997年10月基本确定了其科学负载，于次年5月选定了三个后端设备，于1999年通过ESA科学项目委员会审核，2000年9月开始为项目航天器投标，同年10月为纪念William Herschel发现红外线200周年，FIRST项目正式更名为Herschel。次年Herschel项目开始正式的实体生产。

Herschel望远镜从09年发射升空后，观测了近四年的时间，由于装载的2367公

航天器	
规格高/宽	7.4/4.0 m
质量毛重（含液氦）/净重	3400 (335) / 2800 kg
功率全部/科学仪器	1200/506w
科学数据流量	130 kbps
卫星太阳方位角	60° 110°
绝对指向	~ 2"

望远镜	
主镜口径物理/有效	3.5m / 3.28m
副镜口径	30.8cm
系统/主镜焦比数	8.70/0.5
最佳对焦波前误差（中心/边缘）	4.8/5.5 μ m
角分辨率	~ 7"($\lambda_{obs}/100\mu$ m)
工作温度	~ 85K

表 1.1: Herschel航天器和望远镜主要参数

升液氦耗尽，于2013年4月29日正式退役。望远镜积累了超过25 000个小时的科学数据，期间完成超过600个观测项目，观测了超过35 000个天体源。此外还有超过2000个小时的定标源数据。

项目并没有因为望远镜停止观测而完全结束，预计项目后续处理阶段会持续到2017年。

1.1.2 望远镜主要参数

Herschel望远镜的基本结构如图1.1。Herschel属于经典的卡塞格林式设计，由12段拼接成单面主镜，主镜口径3.5m，主要材质为碳化硅(SiC)，小副镜，有效的避免驻波和那喀索斯(Narcissus)效应^{1[5]}。主镜和副镜表面镀高反射(低吸收)的铝层。卫星运行在第二拉格朗日点(L2)上，具有稳定的观测环境并较少的受到地球的干扰。

航天器和望远镜的主要参数见表1.1^[6]

¹ 那喀索斯(Narcissus)是希腊神话中著名的美丽少年，因深深被自己的美貌所打动，竟把泉中自己的倩影误认为仙女而投入水中，最后淹死在那里。这里是指红外天文观测中常见的一种效应，因为副镜自身也辐射红外射线，而使得副镜在观测平面上成像，干扰观测

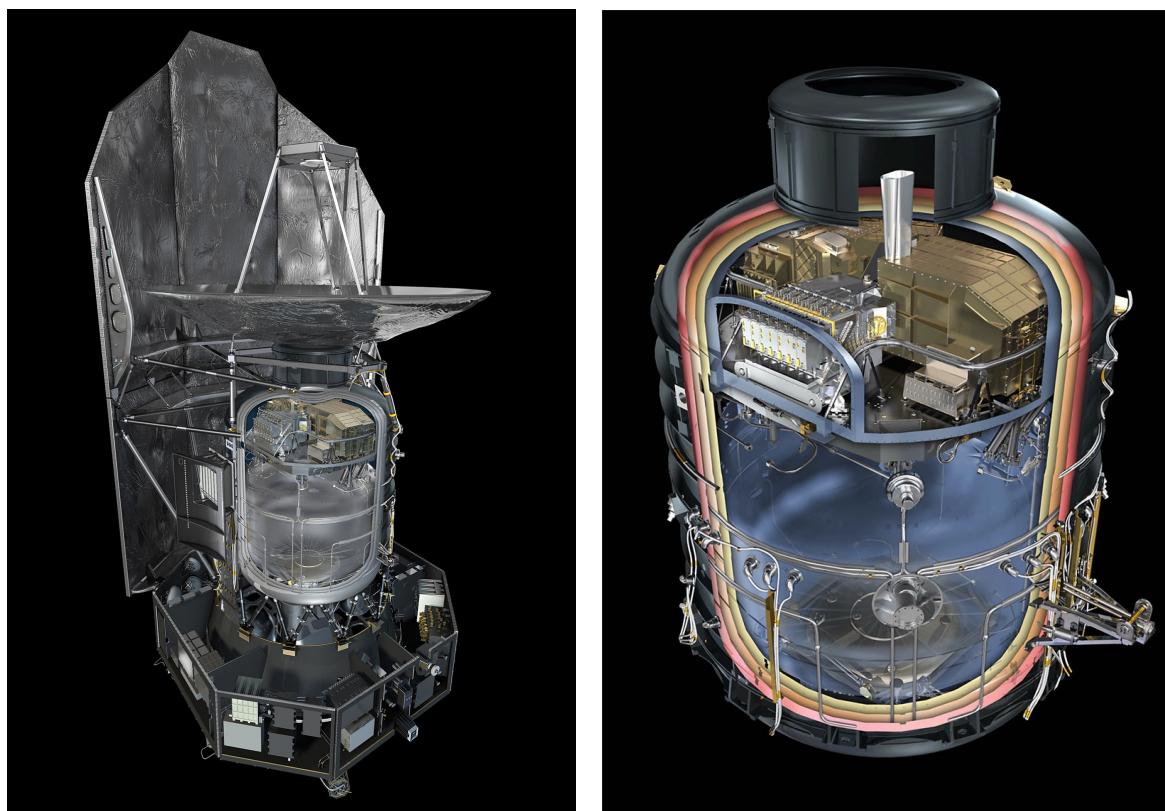


图 1.1: 左: Herschel概貌。右: 光学平台, 仪器设备和液氦瓶

HIFI	外差光谱仪
波长范围	$157 - 212 \mu m$
视场 (FOV)	天区中单个像素
探测器	5x2 SIS & 2x2 HEB mixers
光谱仪	auto-correlator & acousto-optical
谱分辨能力	典型值为 10^6
PACS	双瓣成像光度计
波长范围	$60 - 85 \mu m$ or $85 - 130, 130 - 210 \mu m$
视场 (FOV)	$0.5F\lambda$ sampled $1.75' \times 3.5'$
探测器	64x32 & 32x16 pixel bol. arrays
PACS	积分场光谱仪
波长范围	$55 - 210 \mu m$
视场 (FOV)	$(5 \times 5 \text{ pixel}) 47'' \times 47''$
探测器	two 25x16 pixel Ge:Ga arrays
谱分辨能力	1000 – 4000
SPIRE	三瓣成像光度计
波长范围	$250, 350, 500 \mu m$
视场 (FOV)	$2F\lambda$ sampled $4' \times 8'$
探测器	139, 88 & 43 pixel NTD bol. arrays
SPIRE	傅立叶变换光谱仪
波长范围	$194 - 24 \mu m$ & $316 - 671 \mu m$
视场 (FOV)	$(2F\lambda)$ sampled circular $2.6'$
探测器	two 37 & 19 pixel NTD bol. arrays
谱分辨能力	370-1300 (high) / 20-60 (low)

表 1.2: Herschel后端设备参数

1.1.3 后端设备

Herschel卫星装载了三个后端设备。分别是：PACS (The Photodetector Array Camera and Spectrometer)、SPIRE (The Spectral and Photometric Imaging Receiver)、HIFI (The Heterodyne Instrument for the Far Infrared)。这三个设备使得Herschel在6个波段内都具有良好的成像能力。这6个波段的中心波长为： $70 \mu m, 100 \mu m, 160 \mu m, 250 \mu m, 350 \mu m, 500 \mu m$ 。在这些测光波段内都可以做光谱。三个设备不仅使得Herschel具有很宽的观测波段，而且使得Herschel具有灵活多变的观测模式：如单源、小天区、大天区成像；单源单条谱线和多频段范围的谱线（多个仪器并发）；单源或多源的谱线观测。关于后端设备的具体参数可见表1.2。具体可参考文献 [7](#) [8](#) [9](#)

1.1.4 观测和研究任务

Herschel的工程花费超过10亿欧元，这样大型的科学投资，其目标也指向最前沿的科学问题。大口径，高分辨率，远红外观测波段使得Herschel在观测上有着得天独厚的优势，它基本的科学目标如下：

- 早期宇宙的星系形成和星系演化。
- 宇宙分子化学。
- 恒星形成及恒星与星际介质的相互作用。
- 研究宇宙时间尺度上恒星形成速率的变化。
- 研究银河系及邻近星系中的大气体和尘埃团。
- 太阳系天体（包括行星，彗星，柯伊伯带天体，月亮等）的表面大气的化学组成。

1.2 Herschel通用科学系统

1.2.1 软件构架

Herschel通用科学系统（HCSS，Herschel Common Science System）被用于支持Herschel科学中心（HSC，Herschel Science Center）和三个仪器控制中心（ICCs，Instrument Control Centres）的日常工作。系统提供如下基本功能：

- 信息提供
- 提案提交和处理
- 观测调度
- 观测产品的生成和质量控制
- 数据，产品和软件的存储，访问和检索
- 支持校准和交叉校准

系统支持Herschel 卫星在仪器测试阶段、综合系统测试阶段、在轨操作阶段和后期任务归档等多个阶段的任务。以单一系统支持多阶段的任务，可以使每个阶段相似功能的代码段得到最大程度的重用，而且在后一阶段可以实现对前一阶段的数据操作访问，更重要的是软件系统方便实现提前开发，让卫星在地面测试时就能同时测试软件的实用性，方便修改调试。

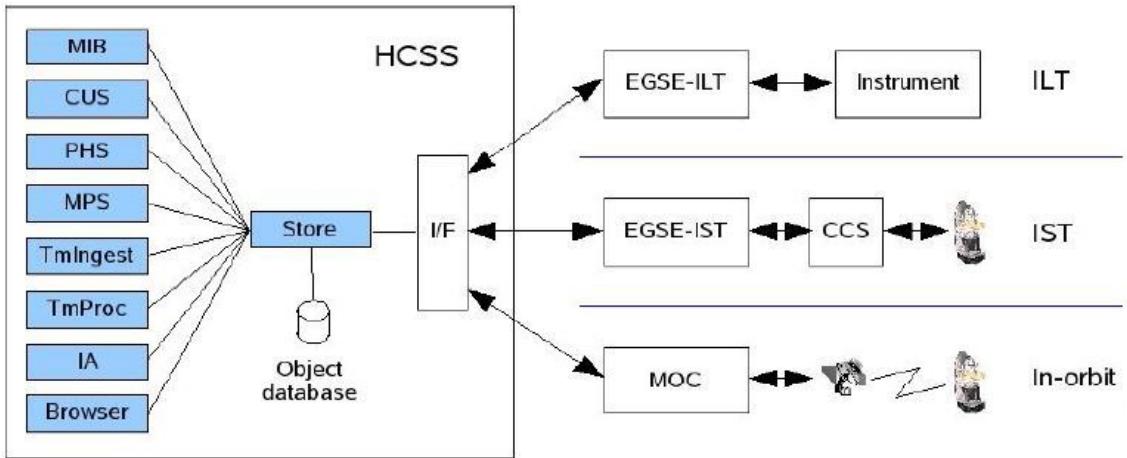


图 1.2: HCSS 主要模块及其与不同任务阶段的关系

软件由java语言编写，遵循面向对象的方法。软件具有很好的可移植性，支持Window 7, Mac OS和Linux等常用操作系统。软件基本架构包括MIB, CUS, PHS, MPS, MOC, Tmingest, TMproc, IA, Browser等模块。²HCSS主要模块及其与不同任务阶段的关系见图1.2。

目前HCSS的稳定版本是10.0版，测试版为11.0, 12.0版正在开发当中。用户可以到<http://herschel.esac.esa.int/hcss/build.php>看HCSS的开发现状，并下载需要的版本。

1.2.2 HDPS软件模块

上面提到的HCSS通用软件系统包括关于望远镜操作等所用功能，作为天文学家直接关心的还是数据处理部分，我们这里重点介绍一下Herschel数据处理系统(HDPS, Herschel Data Processing System)。^{[10][11]}

HDPS系统是开源软件，旨在为天文学家提供一个界面友好的，方便使用的，经过严格测试并且文档完备的数据处理系统。软件将数据检索，流水线执行和科学分析集成到一个系统中，关于数据处理的所有工具，如仪器定标，趋势分析和质量控制系统都被包含在HDPS中。系统有java/jython语言编写，方便移植到各个系统当中。

² 这些模块的具体含义是：MIB：仪器指令定义；CUS：通用上载系统；PHS：提案提交和控制子系统；MPS：科学任务规划系统；MOC：任务操作中心；Tmingest：遥感数据读取；TMproc：遥感数据分析；IA：互动分析系统。

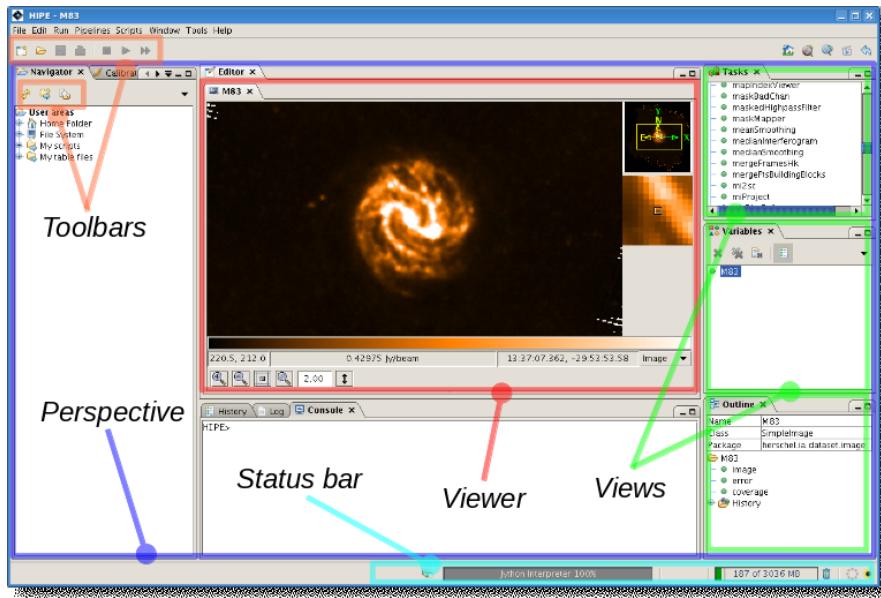


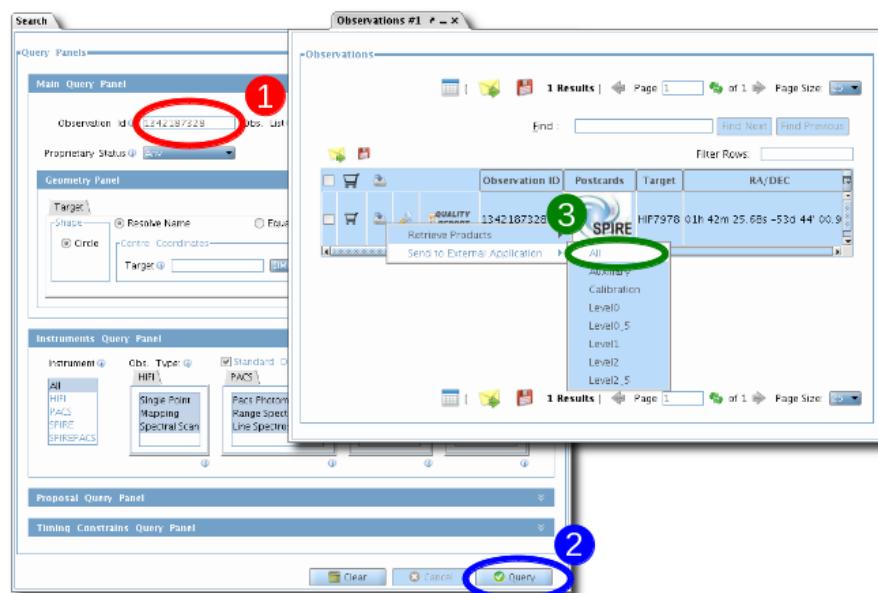
图 1.3: HIPE界面及基本功能

系统为用户设计了友好的用户界面HIPE (Herschel Interactive Processing Environment)。(见图1.3)。一方面该环境提供了脚本运行环境和命令行执行环境，适合专家和开发者开发新的算法脚本的需要。另一方面环境还是以数据为基础的，有强大的菜单栏、工具条和右键菜单，窗口菜单，对于不熟悉java/jython语言的用户，也可以很好的完成自己的工作。在同一个框架下，用户可以实现数据下载，再处理，数据分析和对比等工作。除此之外，系统还提供大量关于Herschel仪器的数据处理流水线，方便用户根据需要修改使用。

系统的另一个特点是对数据的访问下载十分方便。系统专门设计了产品访问层 (PAL, Product Access Layer)。可以通过它直接从Herschel 科学档案库 (HSA, Herschel Science Archive) (见图1.4) 中访问，下载，存储数据。

系统的脚本运行环境使用的语言是jython。jython语言是java python的缩写，它的语言结构与python基本相同，但是是用java编写的解释器，可以很方便调用java的类库，而且jython 是解释型语言，适合前期的尝试开发，可以数倍的提升编码效率，不到C++/Java一半的代码行将大幅度减少开发过程和维护阶段的工作量。

强大的javadoc使得系统的帮助功能完备，你可以在菜单栏 Help→Working in Hipe 中找到详细的帮助。



1. Enter the observation ID
2. Click Query
3. Select All from the Send to External Application menu

图 1.4: HSA数据访问界面

1.2.3 HCSS开发简介

HCSS有超过200个开发人员，可以说是一个巨大的工程，不过值得庆幸的是HCSS的源代码是开源的，在GNU 通用公共许可框架下可免费获得和使用。如果用户对java 语言有一定了解，可以在HCSS 系统修改和开发自己想要的功能。本文的工作也是在这个框架下开发的。这里简单介绍一下HCSS开发的基础知识。

HCSS的源代码在你下载软件的时候可以一并获得。代码的基本模块分为ext和hc_{ss}两部分， ext是HCSS用到的第三方支持库，包括jython, jama, junit等。 hc_{ss}中的代码属于HCSS系统自主开发，主要包含 hcss.dp.core, hcss.dp.hifi, hcss.dp.spire, hcss.dp.pacs, hcss.common, hcss.odb, hcss.apps, hcss.hscops, hcss.services, hcss.icc.hifi, hcss.icc.spire, hcss.icc.pacs 等模块，每个模块下又分有很多几个子模块，本文主要关心的是 hcss.dp.core 模块，是数据处理的核心部分。

编写java代码首先要在系统中装好Java开发工具包（JDK），设置好系统路径。Java代码可以直接用文本编辑器编写，也可以用一些通用的代码编辑环境编写，如emacs, vim等。然后在命令行中³用javac指令编译代码，用java指令运行即可。值得注意的是java 是基于对象的语言，用户修改结果并不作为一个独立软件运行，而是生成一个*.jar的软件包供HCSS系统调用使用。

当然，对于这样大的项目，直接用如emacs这样的编辑环境在调试，编译，版本控制等方面都不是很方便。目前有很多针对java的集成编译环境对于编写这种大型项目很有帮助，如eclipse, NetBeans等。笔者习惯使用的eclipse，本文的工作也基于eclipse版本： Juno Sr1⁴。下面简单介绍一下eclipse。

Eclipse 是一个开放源代码的、基于Java的可扩展开发平台。它提供了一个java程序开发的集成开发环境，通过插件，它亦可作为其他计算机语言（比如C++ 和Python）的开发工具。eclipse基本界面如图1.5。eclipse非常适合HCSS的软件开发，原因如下：

- 作为可扩展的平台，它非常容易集成HCSS软件开发的各种工具，如： jake, emma, javacc, pydev等。

³ linux 用户即在shell 中， windows 用户在命令提示符cmd 中

⁴ Juno (朱诺)：天后，主神朱庇特之妻，妇女及婚姻的保护人，相当于希腊神话中的赫拉

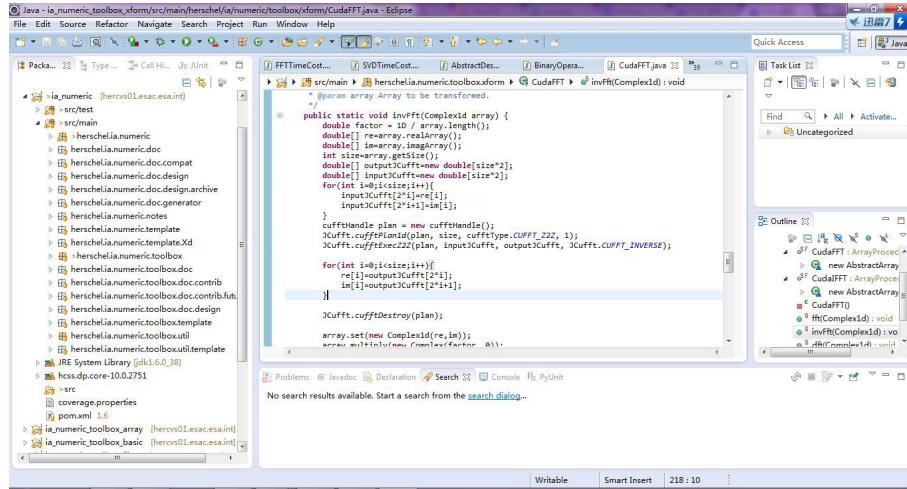


图 1.5: eclipse界面

- 作为集成开发环境（IDE），eclipse除了很方便的开发java软件，另外通过插件，还可以开发C++和python软件，这和HCSS正好珠联璧合，HCSS作为综合的科研软件，其使用内部java类和外部jython接口的方式，很容易在eclipse中一并开发，另外本文的工作，基于CUDA的GPU加速，更是用到了一定量的c \ c++代码，这在eclipse里并没有增加过多的复杂性。
- 如HCSS这样的大型软件要方便多人一起开发，版本控制系统就显得格外重要，eclipse支持CVS（Concurrent Versions System）软件开发模式，方便多人共同开发软件。
- 在软件开发过程中，对代码添加外部连接库的路径，添加外部jar支持包都很方便；eclipse可以嵌入junit，对代码测试和调试都很方便；另外最终产品生成*.jar 软件包在eclipse实现起来也很方便。

eclipse的官网上<http://www.eclipse.org/>上提供完整的使用说明，这里就不再累述。我们介绍一下如何利用eclipse开发和扩展HCSS系统。

首先我们要得到需要的源代码。HCSS系统的代码根据不同类库分别打包，用户不需要通过重新生成整个HCSS系统来实现局部应用的变更。得到所需部分的源代码有两个途径，一是在下载后我们同时可以得到HCSS的源代码，源代码是按照功能分别打包的，即每个**.jar 软件包的代码打包成一个**_src.gz 压缩包。这种方式一般是用户查看代码功能时使用。为了修改代码功能，更方便的是通过eclipse里新建一个CVS项目，在如下地址中可直接下载所需部分的代码，

建成一个CVS项目。

```
hercvs01.esac.esa.int:2401/services/repositories/HERSCHEL_CVS
```

根据在eclipse中生成一个项目，我们可以根据需求修改这个项目的代码，重新打包成**.jar文件，代替原有的**.jar包，即可实现修改的功能。需要注意的一点是，如果需要新加功能，需要在__init__.py中声明才行，因为HIPE的接口是python 的，只有在__init__.py声明的方法才会在系统初始化的时候调入HIPE执行环境中，才能在HIPE 界面中直接调用。

第二章 GPU并行编程简介

2.1 GPU与高效能计算

2.1.1 CPU和GPU的性能对比

中央处理单元（CPU，Central Processing Unit）近20年来的迅速发展，极大程度的推动了计算机应用程序的性能提升和成本降低，正是这些微处理器，使得计算机的浮点运算可以达到每秒十亿次，而集群服务器甚至可以达到每秒数百亿次。在这种硬件发展环境下，很多软件开发人员依赖底层硬件的性能提升来提升程序的性能。然而近年来，由于CPU功耗随主频增加而急剧增加，能耗和散热限制了时钟频率的提高，而且随着单CPU的芯片集成度不断提高，已经逐渐接近半导体工艺的极限，单CPU的计算能力受到很大限制。单靠CPU运算能力的提升来提高软件性能显然已经略显疲态。

在这种情况下，人们似乎已经看到了依靠主频来提高CPU性能的途径已经走到了尽头。硬件开发厂商开始转变思想，多核并行成为一种趋势。

一方面是CPU的多核发展。2005年英特尔推出简单封装双核的奔腾D和奔腾四至尊版840。次年7月23日英特尔基于酷睿(Core)架构的处理器正式发布，真正的“双核时代”开始了。但是目前来看，CPU的核心集成度还比较有限，随着核心数量的增多，由于核心结构过于复杂，不仅不能提升性能，还会带来线延迟增加和功耗变大等问题。另一方面，作为计算机的中枢神经，CPU并行直接涉及操作系统的并行，随着核心数量不断增加，基于串行思想发展起来的计算机系统如何有效调度资源也是需要考虑的问题。

另一方面，GPU（Graphic Processing Unit）在高性能科学计算领域开始崭露头角。GPU是计算机的图形运算单元，与CPU（Central processing unit）不

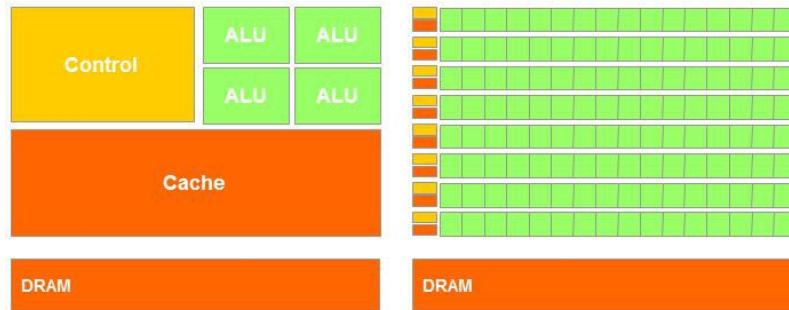


图 2.1: CPU和GPU不同的设计架构

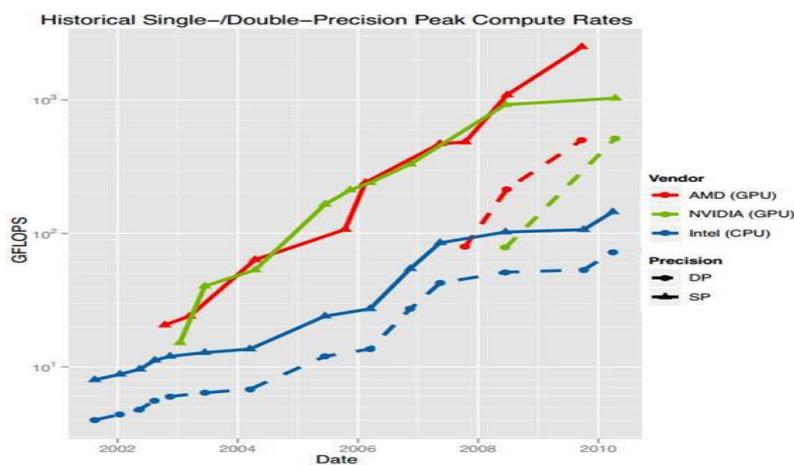


图 2.2: GPU和CPU运算性能对比

同的是，GPU 简化了控制系统，缩减了指令集，而大量增加了逻辑运算单元 (ALU, Arithmetic logic unit)。GPU的众核架构可以集成更多的核，与多核CPU相比，并行度和数据吞吐量都得到了极大的提高。以Nvidia Geforce GTX 680为例，它的核心数可达到1536个，浮点运算执行速度可以达到每秒3090Gflops (Giga Floating-point Operations Per Second)。而且GPU的发展速度也异常迅猛，显卡的核心产品平均6个月就会更新一次。GPU和CPU架构的区别见图2.1 典型的GPU 和CPU 单精度和双精度浮点运算速度对比见图2.2^[12]。

可见GPU在高效能计算中还是很有潜力的，目前采用CPU高速串行运算和GPU高速并行运算混合模式已经成为高效能计算的一个趋势。这种模式也已经在天文学的多个领域取得了成就，典型的如N-Body模拟^[3]，实时天文图象处理，计算射电信号的相关性^[13]，计算宇宙学常数^[4]等。

2.1.2 GPU简介

GPU的前身是图形运算协处理器，GPU这个词第一次被使用是在1999年，Nvidia公司为其生产一款显卡Geforce 256，开始推广这个名词，Nvidia公司也从此时开始，逐渐在显卡领域占据霸主地位。20世纪八九十年代，为了加速计算机的图像显示速度和效果，人们开始开发各类图形加速器。显卡的图形处理功能不断得到加强，如画线、区域填充、加速运动、操纵和组合多个任意的位图，2D 加速，纹理处理，图像渲染，平滑消锯齿，3D 图形加速等都添加到GPU中。于此同时，主要图形应用程序API库¹也开始流行。如微软提供的多媒体专用API——DirectX和Silicon Graphics公司开发的OpenGL。人们逐渐发现GPU的众核架构非常适合于并行运算，一些人开始把自己要处理的问题转化为图像运算问题，利用通用的图形处理API来处理这些问题。GPU开始在高性能运算中展露头角。通用目的的GPU（GPGPU，General Purpose GPU）运动开始萌芽，但是这个时期GPU仍然是那些能记住图形处理API的人的专利。直到2003年，斯坦福大学Ian Buck教授领导的一个小组扩展了C语言，设计了新的编译器Brook，使得新的语言在类C语言的框架下有了数据并行处理的架构，Nvidia公司邀请Ian Buck加盟，终于在2006年发布了第一个GPGPU的解决方案CUDA（Compute Unified Device Architecture）有效结合了GPU的硬件性能和良好的软件开发环境。

目前流行的独立显卡厂家有Nvidia和AMD（ATI）两家，两家显卡在计算性能上处于伯仲之间，但是目前支持CUDA的只有Nvidia一家，这也使得Nvidia公司的显卡在科学计算领域内应用更广。我们下面主要介绍一下Nvidia公司的产品现状，让读者对目前的GPU在科学计算领域的情况有个基本的了解。

Nvidia主要有三个系列的产品，分别是Tesla，Quadro和Geforce。Quadro是专业的图形卡，在各种图形处理上做了很多优化，一般用于专业的CAD（Computer - Aided Design）设计。GeForce系列是面向一般用户推出的显卡，兼顾图像和计算，多用于台式机和笔记本。Tesla已经不是传统意义上的显卡，它去除了图像输出功能，而专业做高性能计算，在服务器上使用广泛。Nvidia的显卡架构历经三代——Tesla²、Fermi、Kepler。目前最新一代的架构是Kepler，对应的产品一般为Geforce 600/700系列，Tesla K10/K20等。下一代Maxwell预计会于2014年发

¹ 即Application Programming Interface，API是一种标准化的软件层（即库函数的集合），它支持应用程序使用软件或硬件的服务和功能

² 这里的Tesla是指硬件架构，与前面的产品系列有区别

布。

为了更好的理解并行结构程序设计，优化并行程序效率，需要熟悉内部结构。GPU内部已多核流处理器（SM，Streaming Multiprocessor）为单位组织众多核心。整块GPU贡献一块显存，每个SM又有自己的共享存储器、常数存储器和纹理存储器，见图2.3。以最新的Kepler架构Geforce GTX 680为例，显存可达4GB，一片芯片有8个SM，每个SM有192个核心。³输入到GPU的线程以块（block）的形式分配到每个SM，每个SM以warp的方式组织每个块中的线程（Threads），一般一个warp包括32个线程，即32个一组，执行相同的操作⁴。GPU最终能实现高速并行，warp这种执行方式立下了汗马功劳。每个指令都是有指令周期的，一般上会占用几个到几十个时钟周期。当每个SM中存在大量的warp是，这些warp根据自己占用资源的不同（如计算单位ALU，I/O读取等），可以有效调度，峰值可以达到每个warp的每条指令平均只用了一个时钟周期。所以在编程的时候，一定要增加程序的并行度，一般来说，并行的线程数越多，平均速度会越快。

另一点需要提到的是，GPU的浮点运算能力对于单精度和双精度是有差别的。因为GPU的基本功能是图形计算，所以精度不需要太高，所以单精度浮点运算的效率会远高于双精度。但是这种现状在考虑到GPGPU的思想后，已经有很大改观，双精度的计算速度不断被提高。在最新的GK110架构中，双精度浮点运算速度已经接近单精度的一半。

2.2 GPU编程简介

2.2.1 CUDA编程方法

CUDA（Compute Unified Device Architecture）是Nvidia发布的并行计算平台和编程模型。它的出现使得GPU在高效能计算中的调度效率有了巨大的提高。真正一个程序得益运行的时间包括程序员的编程时间和程序运行时间，在CUDA出现之前，程序员为了利用GPU，需要很长一段时间的学习周期。要对底层图形处理的API很熟悉，所以虽然程序的运行效率可以得到提高，但是程序员的

³ 读者要想知道自己设备的硬件资源，可以在装上驱动后直接从Nvidia控制面板得到，当然在下文介绍完CUDA后，还有更直接专业的办法

⁴ 这里相同操作是SIMD的概念，即单指令多数据形式，32个线程执行相同的指令，但是各自有各自的数据

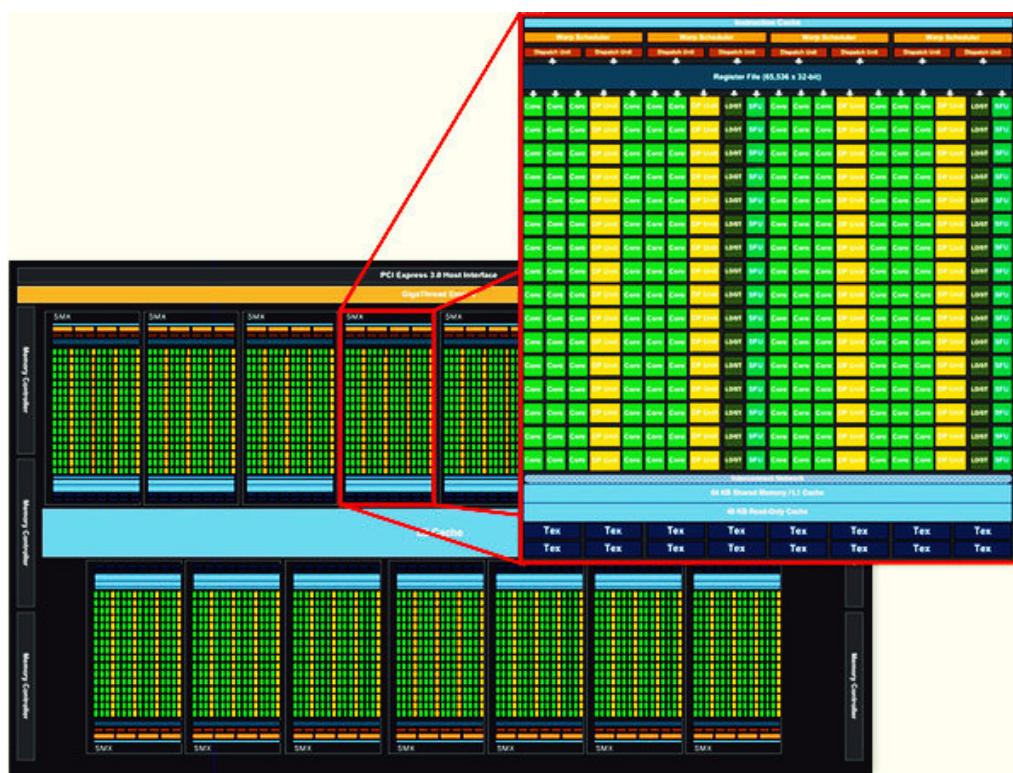


图 2.3: GPU 内部构架

编程时间却大大的浪费了。而CUDA的出现，使得习惯高级语言的程序员，不再需要去了解底层汇编级别的操作。大大提高了编程效率。

另外CUDA对Nvidia不同代的产品有很好的兼容性，不同代的产品在硬件能力上有很大差别，比方说是否支持原子操作，是否支持双精度，CUDA根据不同代划分不同的计算能力，目前是从1.0到3.5不同，同样的程序在新的硬件上不需要修改仍可以运行。对不同代的产品到底支持那些操作，可以参考Nvidia网站<https://developer.nvidia.com/cuda-gpus>和参考文献 14。

目前最新的CUDA版本是5.0v，支持Windows、Mac和大部分Linux版本的操作系统。比起以前的版本，CUDA 5.0 增加了一些新的功能，如动态并行（Dynamic Parallelism）等功能。CUDA安装包包含Nvidia Driver，CUDA Toolkit，SDK code samples三部分。Nvidia Driver是CUDA 设备的驱动程序，里面提供直接与硬件打交道的API，用户可以不用直接与之打交道。CUDA Toolkit包含CUDA程序的编译器nvcc和一些Nvidia官方提供的CUDA调用库，如cublas（线性代数库），cufft（快速傅里叶变换库）等。SDK code samples是一些例子程序，其中包含很多实用的程序，即使很好的学习教程又可以是自己工作的一个很好的蓝本。在CUDA 5.0 中这三部分已经不用分别安装，是整体一步安装的。

部分系统CUDA类库和编译器路径是默认添加到系统的环境变量中的，为保险起见，读者需要检查自己系统的环境变量。例如Ubuntu用户，需要在 .bashrc 文件中加如下代码：

```
export PATH=$PATH:/usr/local/cuda/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib
                  :/usr/local/cuda/lib64
export CUDA_SDK=~/NVIDIA_GPU_Computing_SDK/
```

Window用户也需要在系统环境中做相应修改。

下面我们就一个具体例子，来介绍一下CUDA编程的主要思想。例子实现的是简单的矩阵相乘，代码如下：

```
1 # include <iostream>
2 # include <cuda.h>
```

```
3 # include <cuda_runtime.h>
4
5 #define TILE_WIDTH 5
6 using namespace std;
7
8 __global__ void MatrixMulKernel(float* Md, float* Nd, float* Pd, int Width)
9 {
10     __shared__ float Mds[TILE_WIDTH][TILE_WIDTH];
11     __shared__ float Nds[TILE_WIDTH][TILE_WIDTH];
12
13     int bx= blockIdx.x; int by=blockIdx.y;
14     int tx= threadIdx.x; int ty=threadIdx.y;
15
16     int Row=by*TILE_WIDTH+ty;
17     int Col=bx*TILE_WIDTH+tx;
18
19
20     float Pvalue=0;
21
22     for(int m=0;m<Width/TILE_WIDTH;++m){
23         Mds[ty][tx]=Md[Row*Width+(m*TILE_WIDTH+tx)];
24         Nds[ty][tx]=Nd[(m*TILE_WIDTH+ty)*Width+Col];
25         __syncthreads();
26
27         for(int k=0;k<TILE_WIDTH;k++)
28             Pvalue+=Mds[ty][k]*Nds[k][tx];
29         __syncthreads();
30     }
31
32     Pd[Row*Width+Col]=Pvalue;
33 }
34
35 void MatrixMultiplication(float* M, float* N, float* P, int Width)
36 {
37     int size=Width*Width*sizeof(float);
38     float* Md,*Nd,*Pd;
39
40     cudaMalloc((void**)&Md,size);
41     cudaMemcpy(Md,M,size,cudaMemcpyHostToDevice);
```

```
42     cudaMalloc((void**)&Nd,size);
43     cudaMemcpy(Nd,N,size,cudaMemcpyHostToDevice);
44
45     cudaMalloc((void**)&Pd,size);
46
47     dim3 dimBlock(Width,Width);
48     dim3 dimGrid(1,1);
49
50     MatrixMulKernel<<<dimGrid,dimBlock>>>(Md,Nd,Pd,Width);
51
52     cudaMemcpy(P,Pd,size,cudaMemcpyDeviceToHost);
53     cudaFree(Md);cudaFree(Nd);cudaFree(Pd);
54 }
55
56 int main()
57 {
58
59     float M[25]={1,2,3,4,5,1,2,3,4,5,1,2,3,4,5,1,2,3,4,5,1,2,3,4,5};
60     float N[25]={1,2,3,4,5,1,2,3,4,5,1,2,3,4,5,1,2,3,4,5,1,2,3,4,5};
61     float P[25]={};
62     int Width=5;
63     MatrixMultiplication(M,N,P,Width);
64
65     for(int i=0;i<Width*Width;i++) cout << P[i]<<endl;
66
67 }
```

首先CUDA编程至少要涉及两个硬件，一个是host指主机CPU，一个是device，指的是GPU设备。编程的一般流程如图2.4，具体分三步：

- 把host要并行的数据导入到device中，在例子中如MatrixMultiplication函数中40到45行。
- 在GPU中运算，如例子中的MatrixMulKernel函数。
- 将运算结果从device中拷贝回host。在例子中如MatrixMultiplication函数中52行，然后释放GPU中的资源，见53行。

在host代码中要调用的device函数要加上`__global__`关键词，成为kernel函数。在host代码中把并行的线程划分为网格（grid），每个网格中包含若干个块

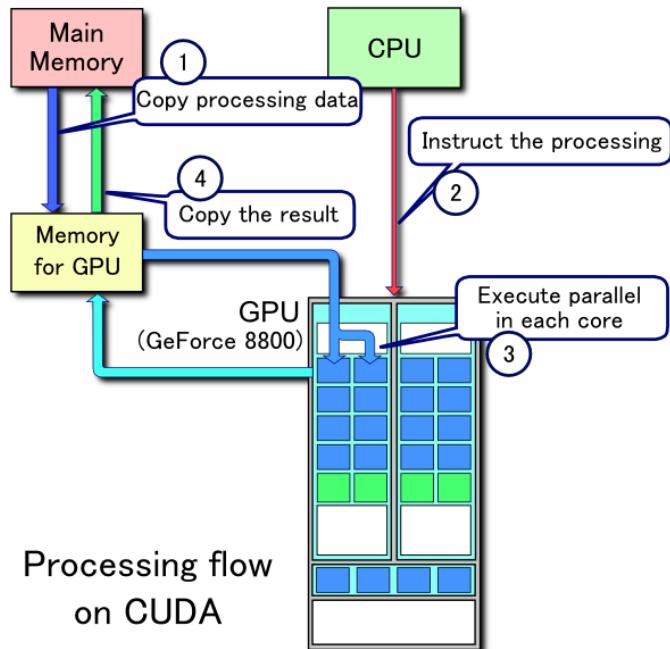


图 2.4: CUDA程序处理一般流程

(block)，每个块中划分成若干个线程（threads）。GPU函数的调用比普通C函数调用多一项，就是要说明网格和块是怎么划分的。如代码50 行，即如下格式：

```
KernelFunc <<<dimGrid , dimBlock>>> (prm1,prm2,prms...)
```

在device代码编写的时候要注意，直接从host拷贝到device上的数据是在全局存储器中的，全局存储器容量大，但速度慢。而每个块会划分自己的共享存储器，共享存储器容量小，但速度快，这时可以把一个块频繁调用的数据读取到共享存储器中。如代码的10 11行。最后要注意虽然线程在不同计算单元计算，每个计算单元的计算速度基本相同，但是仍然会有先后的可能，所以要注意线程同步，用`__syncthreads()`使得线程同步。

这样，在例子程序中基本把一个CUDA程序设计的元素都包含了，另外CUDA还有一些功能，如用CUDA stream来利用I/O和CPU同步加速程序，利用GPU纹理寄存器来提高I/O的速度等。关于CUDA编程的具体介绍，可以参考文献[15] [16]。

CUDA除了提供这些关键词以外，还提供大量的API供用户调用，关于 CUDA 的 API 可以参考<http://docs.nvidia.com/cuda/cuda-runtime-api/>。我们

这里介绍CUDA内置的一个关于设备属性的结构体：cudaDeviceProp。CUDA SDK Code Samples 中有一个例子deviceQuery.cu程序，就是调用这个结构体来确定设备属性。例如本文工作基于的平台Geforce GT 640M的参数，可由deviceQuery.cu得到如下：

```
deviceQuery.exe Starting...

CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "GeForce GT 640M"
  CUDA Driver Version / Runtime Version      5.0 / 5.0
  CUDA Capability Major/Minor version number: 3.0
  Total amount of global memory:            2048 MBytes (2147483648 bytes)
    ( 2) Multiprocessors x (192) CUDA Cores/MP: 384 CUDA Cores
    GPU Clock rate:                         709 MHz (0.71 GHz)
    Memory Clock rate:                      900 Mhz
    Memory Bus Width:                       128-bit
    L2 Cache Size:                          262144 bytes
    Max Texture Dimension Size (x,y,z):     1D=(65536), 2D=(65536,65536), 3D=(4096,4096,4096)
    Max Layered Texture Size (dim) x layers: 1D=(16384) x 2048, 2D=(16384,16384) x 2048
    Total amount of constant memory:          65536 bytes
    Total amount of shared memory per block:   49152 bytes
    Total number of registers available per block: 65536
    Warp size:                             32
    Maximum number of threads per multiprocessor: 2048
    Maximum number of threads per block:       1024
    Maximum sizes of each dimension of a block: 1024 x 1024 x 64
    Maximum sizes of each dimension of a grid: 2147483647 x 65535 x 65535
    Maximum memory pitch:                    2147483647 bytes
    Texture alignment:                      512 bytes
    Concurrent copy and kernel execution:    Yes with 1 copy engine(s)
    Run time limit on kernels:              Yes
    Integrated GPU sharing Host Memory:     No
    Support host page-locked memory mapping: Yes
    Alignment requirement for Surfaces:      Yes
    Device has ECC support:                 Disabled
    CUDA Device Driver Mode (TCC or WDDM):   WDDM (Windows Display Driver Model)
    Device supports Unified Addressing (UVA): Yes
    Device PCI Bus ID / PCI location ID:    1 / 0
    Compute Mode:
      < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 5.0, CUDA Runtime Version = 5.0, NumDevs = 1, Device0 = GeForce GT 640M
```

2.2.2 CUDA的常用类库

直接编写CUDA kernel函数一般是根据具体应用作具体优化，并行加速效率也最高。但是对于通用应用来说，还有一种更简单的办法，就是利用Nvidia提供的CUDA函数库，一般来说这些函数库的调用方式和普通C函数库的调用方式相同，并且可以让你的应用轻松提速2到10倍。下面我们简单介绍一下这些类库。

CUBLAS 是线性代数BLAS(Basic Linear Algebra Subprograms)的GPU实现。应用CUBLAS库，用户可以轻松的分配向量和矩阵空间，传输这种类型的数据，调用诸如向量加减乘除、点乘叉乘，矩阵加减乘除和类似 $C = C + A * B$ 这种类型的矩阵运算。

CUFFT 是GPU的快速傅里叶变换库(FFT, Fast Fourier Transform)。支持单/双精度；支持1D, 2D, 3D的FFT；支持实数和复数格式；可以批量执行FFT；线程安全，可以支持多个独立的host线程调用；对输入数据规模可以分解为2、3、5、7为因子的情况做了特别优化；

CURAND 支持简单有效的生成高质量的伪随机数和准随机数。整个库分为两部分，一部分可以实现在host上直接生成随机数，另一部分是在device上生成随机数，然后从device上拷贝回host。

CUSPARSE CUSPARSE库是对CUBLAS库的一种加强。这个库主要用于处理稀疏矩阵，包括稀疏矩阵元素的存储和稀疏矩阵的运算。主要用来节省存储空间。

Thrust 是一个为CUDA编程设计的C++的模板库，基于标准的模板库STL(Standard Template Library)。Thrust提供了丰富的数据收集并行原语，如扫描，排序，归约等。方便用户用简洁易读的代码实现复杂的算法。

2.2.3 GPU的拓展应用

CUDA目前的标准支持是C/C++语言。2009年6月,PortlandGroup (PGI) 联合NVIDIA共同推出了新版Fortran编译器，可支持CUDA架构，在高性能计算技术方面的应用主要体现在充分利用NVIDIA GPU在CUDA架构下的处理能力上。在2013年3月的GTC(GPU Technology Conference)会议上，Nvidia宣布可以通过Continuum Analytics公司全新Anaconda Accelerate 产品中的一款Python编译器NumbaPro实现python语言对CUDA并行编程的实现。另外，通过第三方的软件支持，CUDA可以在java, Perl, Ruby, Lua, Haskell, MATLAB, IDL等语言中应用。

2.3 GPU的发展前景

近年来GPU的发展速度非常迅猛，一方面是计算速度和并行度的提高，另一方面是新的构架不断被提出，GPU不断增加新的功能。从双精度被在GPU中实现到现在最新的Kepler 构架 GK110支持动态并行技术⁵，hyper-Q 技术⁶，GPUDirect⁷ 只历时不到4 年。这方面的硬件功能的高速发展要求软件工作者也不要固步自封，要时刻保持对新的知识和技术好奇，运用新的成果来提高软件的性能。

⁵ 传统的GPU 只能由host来扩展调用GPU，而GPU不支持递归调用，不能自行扩展线程。而动态并行技术可以使得GPU自行扩展，有效减少了host和device上不必要的信息交流，实现更广的GPU应用和速度的提高

⁶ 实现多CPU同时向单GPU提交任务，大大减少了CPU 的空闲时间

⁷ 实现GPU 间自行通讯，而不需要CPU 干涉

第三章 HCSS脚本效率分析方法

HCSS提供了良好的用户界面交互系统HIPE，用户可以方便的编写自己的jython 脚本，调用HCSS已有的java 类库，处理自己的具体应用，此外，HCSS系统提供了大量的数据流水线处理 (pipeline) 脚本¹供用户使用。但是这些脚本的功能复杂多样，调用的函数灵活多变，脚本的运行效率参差不齐，如果能提供一个简便的方法，从这些具象的应用中，抽象出共同可加速的部分，以最小的代价来实现脚本最大的加速，是一个听起来不错的事情。但这个思想在HCSS中是否可行，还有待研究。

HCSS是以面向对象为基础的系统，所以主要运算部分都很可能继承自相同的基类，这给我们想法的实现提供了很大便利。我们的方法就是要找到这些主要调用的基类，统计这些基类运行时间占总运行时间的百分比，如果这些基类的耗时很长，下一步就利用GPU，看能不能加速这些基类，从而以最小的代码修改代价，实现最大程度的加速。本章的总结了一套普遍适用的，简便容易实现，而且对系统的额外负担很小的方法，利用这个方法我们抽象出一些可能的主要耗时部分的基类，分析这些积累在脚本运行时间中占的时间比。第五章，我们应用这种方法处理了HCSS中主要的数据流水线脚本，并尝试用第三章提到的GPU 加速办法加速这些基类，分析了GPU 对HCSS 系统的加速效果。

3.1 HCSS中运算基础类

要利用基类的加速来加速整个脚本，首先要清楚有哪些主要的基类。HCSS中关于数据运算最基础的基类包含在 ia.numeric 中，主要定义了Herschel数据运算的基本数据类型，如Bool, Byte, Short, Int, Long, Float, Double, Complex. HCSS把

¹ 具体介绍见第五章

自己的数据组织成多维数组，包括一到五维，根据数据类型和数据维度打包成单独的一个类，如Double1d, Double2d ... Double5d。将数组的运算包括加减乘除，模，绝对值，相反数，幂运算，点乘和叉乘运算等都定义为这些数据类的基本方法，以方便调用。

根据这些新定义的基本类型，根据不同功能向下继承还具体分为²：

- ia.numeric.toolbox.basic：基础数学函数，其中包括绝对值，三角函数，指数对数函数，取整等函数。
- ia.numeric.toolbox.filter：卷积和高斯滤波器
- ia.numeric.toolbox.fit：拟合，包括线性拟合，多项式拟合，高斯型拟合等。
- ia.numeric.toolbox.integr：积分运算包，用于算定积分，包括高斯厄米、高斯雅可比等积分方法。
- ia.numeric.toolbox.interp：差值运算包，包括一维、二维线性差值，最近邻差值和三次样条差值。
- ia.numeric.toolbox.matrix：矩阵运算类，其中包括如矩阵乘法，矩阵特征值求解，矩阵求，矩阵奇异值分解，矩阵LU分解，矩阵QR分解等。其中部分代码直接用了开源矩阵运算库jama。
- ia.numeric.toolbox.random：生成随机数，可以产生平均分布，高斯分布，指数分布，泊松分布等常用分布的随机数。
- ia.numeric.toolbox.stat：统计分析包，可以算数组的方差，几何平均值， χ^2 等。
- ia.numeric.toolbox.wavelet：小波变换包。
- ia.numeric.toolbox.xform：傅里叶变换包。其中主要有 FFT, FFT_PACK, FFT_PACK_ODD, FFT_PACK_EVEN 等类，这些类基本上涵盖了最常用的快速傅里叶变换方法。

根据天文数据处理最常出现的数据类型为双精度实数和复数，运算最常出现的是数组基本运算（加减乘除乘方，点乘等），傅里叶变换，数据拟合等。我们主要关注其中有Double1d, Double2d, Complex1d, Complex2d, xform, matrix等几个类的时间花费。

² 另外还有ia.numeric.toolbox.mask, ia.numeric.toolbox.array等，主要提供一些辅助功能

3.2 脚本运行效率分析方法

因为要统计的类 (class) 较多，每一个类又有自己的方法 (method)，这样要精确统计一个脚本中每一个方法的时间花费，并不是件容易的事情。如果对于不同的类，不同的方法都抽象出一种相同方法策略进行时间统计，就可以大大提高代码的重用度，将编码的时间成本大大降低。下面就来介绍一下如何实现对每个类的时间花费统计。³

3.2.1 脚本分析思路和分析方法概貌

每个HIPE脚本的计算部分归根结底会调用运算基础类，我们针对每个要统计的基础类（以下用“目标类”指代）配置两个辅助类，分别抽象的命名为NumCount 和TimeCost，这两个类的作用如下：

NumCount: 这个类用于统计目标类和目标类中的方法被调用的次数，因为函数运行时间与输入函数的数据规模有关，所以这个类还要具有按照输入数据的规模进行分别统计的功能

TimeCost: 这个类用于计算脚本里目标类总共的时间花费。实现此功能的基本思想是用统计平均和线性拟合的方法计算出对应数据规模一次调用的时间花费，再乘以NumCount中统计的调用次数，即可得到时间。

本质上我们是通过两步走的方式实现HIPE脚本目标类时间花费的统计的。处理流程可见图3.1

这里用两步走得策略是有其合理性的，有读者可能想能不能直接用一个类直接统计时间花费。其实仔细分析，用一个类一次性完成不仅实现上有困难，而且统计上也不会十分准确。一方面记录一个类的时间花费，我们需要一个停表机制，这个停表要在目标类被调用的那一刻开始计时（准确的来说是在目标类被调用的指令的前一个指令调用计时指令）。如果我们不通读所有代码，然后一一标记的话⁴，目标类被调用可以看作是随机出现的，我们无法预测下一条指令是不是调用目标类，所以也就没有好的办法来准确的“启动计时”。另外即使我们能很好的

³ 注意这一节的代码都是java代码，即对原代码的改动都是在内核实现的。这样做是有道理的，这涉及到时间统计的准确性问题，HIPE 中jython脚本只是外壳，是解释性语言，要编译器解释运行，这样解释语句是有时间花费的，而我们关注的时间，是内核代码真正的运行时间，也是真实可提速的部分。

⁴ 这样做显然是不可行的，一是代码的阅读量太大，而是对代码的改动也很多，工作量巨大

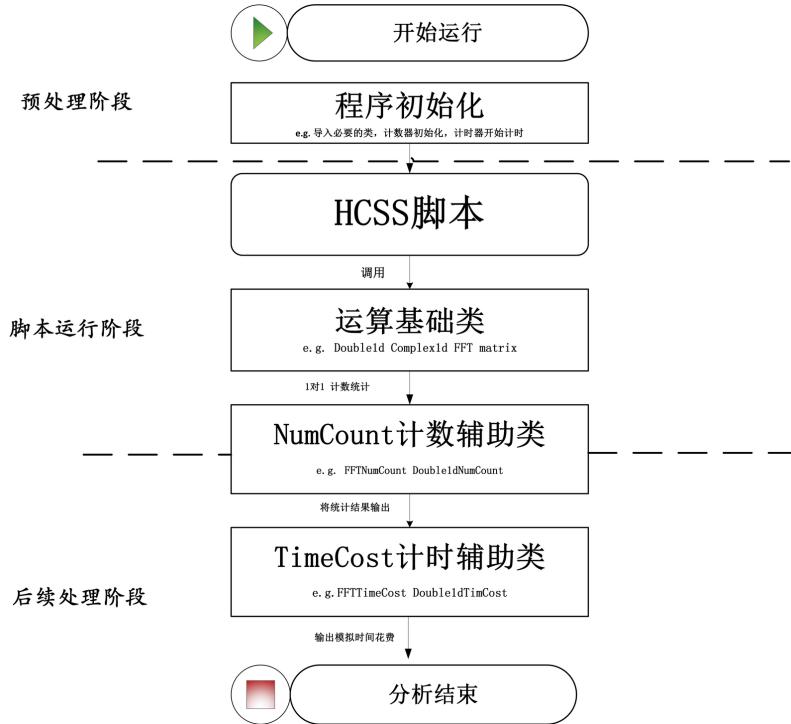


图 3.1: HIPE脚本目标类时间花费分析流程

预计目标类被调用的时刻，从而在前一条指令开始计时，这种停表机制也不会太准确，我们知道目标类的一次调用时间花费是很小的，很可能小到只有时钟周期的几倍。这样由于系统噪声——如java的垃圾收集等⁵——可能使得这一次计时与目标类的本征时间花费有较大的出入，而且即使在这次的指令调用中没有插入其他进程，时钟周期量级的时间间隔也给停表的时间精度带来很大压力，而使得最后时间统计有较大误差。

而我们采用的两步走的策略很好的规避了这两个问题。首先我们第一步只是统计目标类被调用的次数，所以我们完全可以在目标类每次调用后进行计数，解决了预测问题。第二我们用后期模拟的办法进行时间统计，我们知道目标类和目标函数的时间花费与数据规模有关而和数据的具体取值关系不大。这样我们后期用随机数的办法，多次求平均数。这样一是可以把系统噪声平均掉，而是多次调用可以让停表的时间精度得以保证。

本工作修改标记的目标类包括 Double1d, Double2d, Complex1d, Complexd, xform, matrix。修改和新建的java代码见表3.1。当然，相同的思路和方法很容易

⁵ 可以视为随机出现，但是一旦出现就会类似 δ 函数一样，花费大量时间

软件包	目标类	分析辅助类
herschel.ia.numeric	Complex1d	Complex1dNumCount Complex1dTimeCost
	Double1d	Double1dNumCount Double1dTimeCost
	Double2d	Double2dDim Double2dNumCount Double2dTimeCost
herschel.ia.numeric.toolbox.matrix	MatrixMultiply	MatrixDim MatrixNumCount MatrixTimeCost
	SingularValueDecomposition	SVDNumCount SVDTIMECost
herschel.ia.numeric.toolbox.xform	FFT	FFTNUMCount FFTTimeCost

i:具体的java文件即类名加上 .java。部分代码可见附录A

表 3.1: 本文关注的目标类和新建的辅助类

被扩展到其他的类库上。

下面我们以Double1d类型为例子⁶，分析它两个辅助类——Double1dNumCount.java 和 Double1dTimeCost.java ——的具体实现。我们主要考察Double1d 中关于数学运算的函数：加（add），减（subtract），乘（multiply），除（divide），幂（power），模（modulo），绝对值（abs），相反数（negate），点乘（dotproduct），叉乘（outproduct）的时间花费情况。这些运算种类虽然很多，但是每一种运算都是基于相同的数据类型（Double1d），所以统计起来并不会增加太多复杂性。

3.2.2 计数模块

Double1dNumCount.java的代码可见附录A，其中主要包含如下功能：

- 针对每一个Double1d中关心的方法建立一个静态对象。子类的类名为 Double1dPara，生成的对象分别为Pow，Mod，Abs等
- 统计这个方法被调用的次数和每次调用的数据维数。主要调用 Double1dCount()，和 Double1dDtlCount()。

⁶ 完整代码可见附录A

- 查看目前已调用过的次数和不同数据规模的调用次数。调用函数是 Double1dGetCount()，和 Double1dGetDtl()。
- 组织结果输出到文件或者窗口。主要调用 Double1dNcOutput()。

这里要注意的是每个类的具体实现在 Double1dPara 里，Double1dNumCount 类只提供接口。因为每一个方法很可能被调用很多次，而且数据规模变化浮动可能较大，这样直接分配一个大空间的一维静态数组可能会比较浪费空间，所以才用一个二维数组 _Detail[2][dim]，一维记录调用时的数据规模，另一维记录被调用次数。

设计好 Double1dNumCount 类了之后，只需要在原来 Double1d 每个要统计的方法里加入 Double1dDount()，Double1dDtlCount() 即可，例如对于abs 函数有：

```
public Double1d abs() {
    ABS.unary(_array,dim(0));
    Double1dNumCount.Double1dCount(Abs);
    Double1dNumCount.Double1dDtlCount(Abs, dim(0));
    return this;
}
```

3.2.3 计时模块

Double1dTimeCost.java 的代码可见附录 A，其主要用于统计后的时间花费计算。包含功能比较简洁，如下：

- 读入Double1dNumCount输出的统计结果。即 TcInput() 函数。
- 计算目标函数在被用到的数据规模上，一次运算的时间花费（通过多次求平均值），即 TimeCal() 函数
- 根据一次运算的花费和被调用的次数可以得到总的时间花费，即 TimeCost() 函数。

需要注意的是 TimeCost 这一部分数据后期模拟部分，只要知道不同数据规模函数的调用次数即可利用这个类来分析时间花费。即这个类与数据处理脚本并不直接相关，而是从 NumCount 中读取数据。所以这个类写成 jython 脚本更简

洁方便，但是由于这里涉及到如何计时的问题，下面的实验结果表明，必须写成java内核代码，才能更好的模拟脚本中目标类的真实时间花费。

jython 中计时模块 time 提供两个计时函数time.clock(), time.time()可以用来计时。而 HCSS 内核使用java 编写的。java 中也提供了两个计时函数 System.currentTimeMillis(), System.nanoTime() 。到底用那个函数会比较合适呢？这里对这四个函数作一点讨论。

jython中的两个计时函数函数返回值是不同的为⁷:

- time.clock(): 返回的是系统给定的一个进程时间，单调递增，精度可以达到 $10^{-6}s$ 但是与物理时间没有直接关系。
- time.time(): 返回以秒为单位的儒略日，与物理时间直接挂钩，但是精度较低，只有 $10^{-2}s$

java中的两个计时函数函数返回值于此相似：

- System.currentTimeMillis():与time.time()相似，返回的是以毫秒为单位的儒略日，精度可达 $1ms$ 返回的不是实型而是长整型数据。
- System.nanoTime():与time.clock()相似，返回的是一个以纳秒为单位的系统时间，与物理时间不直接对应，返回值为长整型，精度可达 $1ms$ 。

因为我们需要设计的是一个停表，所以并不需要和物理时间对应，而是要精度更高的计时手段，所以显然要选择 time.clock() 或者 System.nanoTime() 。但是究竟是选择那个函数呢？这直接关系到在什么环境下编程。我们的结论是这里需要使用System.nanoTime()。虽然 time.clock() 可以在 jython 环境下写脚本，显然更方便一些。但是进一步的研究表明在 jython 环境下程序是解释运行的，计时器记下的除了函数运算的时间还包括编译器解释程序的时间。而我们的数据处理脚本，虽然是在 jython 环境下运行的，但是其中的运算部分都是调用 java 内核完成的。显然编写一个java的计时辅助类，调用 System.nanoTime() 能更接近脚本中运算的真实时间花费。

为了说明jython和java计时上差异。我们以一维数据加法为例子来说明。图3.2 表示jython中一维数据加法的时间花费随数据规模增大的变化。其中红色曲线表示直接统计加法的时间花费，品红色表示空操作for循环的时间花费，绿色曲线是这两者的差。我们认为空for循环的时间花费是jython 编译器解释程序的时间

⁷ 注意：这里给出的是Windows环境下的函数定义，linux环境下这两个函数的定义正好于此相反

花费。所以用加法的时间减去空for循环的时间花费后就比较接近真实的加法时间花费了。但是即时是这样，这个时间还是与真实的时间花费有差异，因为加法运算的解释时间没办法扣掉。图3.3中，我们对比了 `nanoTime()` 和 `time.clock()` 的计时差异。

上面实验结果中，很清楚的表明jython解释程序的时间花费甚至远远大于加法运算本身的时间花费，这严重影响了用jython脚本设计计时辅助类的精度。所以我们的计时辅助类还必须用编写在java内核中。

实验结果还对我们编写HIPE脚本有一定的指导意义。从另一个角度看，这指导我们编写脚本程序的时候，对时间花费大的运算部分，要尽量调用java内核去处理，而不是直接用jython 还做运算。

根据上面的对计数模块和计时模块的介绍，时间统计的java内核代码就完成了。我们主要统计了FFT, SVD, Double1d, Double2d, Complex1d, MatrixMultiply 等类型，代码见附件A。要真正在HCSS数据处理脚本中使用这些类，还需要对HCSS 脚本做一点预处理。下面我们如何用jython脚本实现预处理。

3.3 时间统计的预处理和后续处理过程

在真正统计一个数据处理脚本的时间花费时，要对脚本做一些预处理，如：新建一个NumCount对象，并对参数进行初始化等。在一个脚本运行完后，要统计时间花费，也需要做一些后续处理，比方说调用TimeCost()函数，或者为了进一步得到更准确的时间，还可以进入对时间花费曲线做拟合等更复杂的技术。这些就是后续处理。这一节我们简单介绍一下如何实现预处理和后续处理。

一般对于一个完整的jython脚本应用，比方说HCSS数据处理流水线等，我们希望它正常运行的情况下，不要对它做任何改动。这一是比较符合功能分区的规范，减少不必要的编码花费，提高代码的可读性，另一方面也提高我们处理手段的通用性，而且便于后期扩展我们方法的应用。所以我们编写一个主脚本文件（本工作中定义为 `Main.py` 文件，代码见附录B），在这个脚本文件里组织其他脚本文件依次运行，子脚本不互相干扰。主脚本基本架构如下：

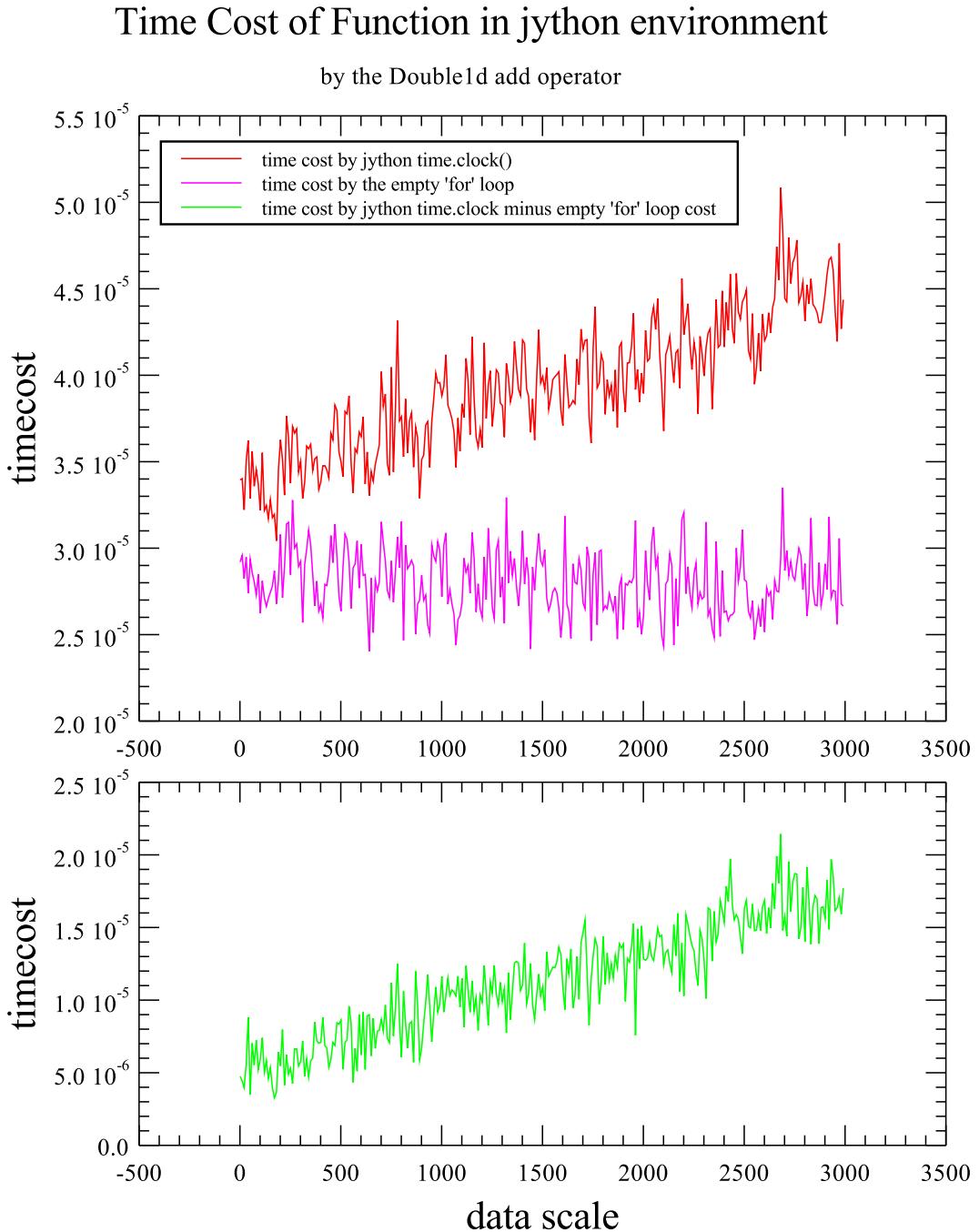


图 3.2: jython中数组加法时间花费——其中红色曲线表示直接统计加法的时间花费，品红色表示空操作for循环的时间花费，绿色曲线是这两者的差。

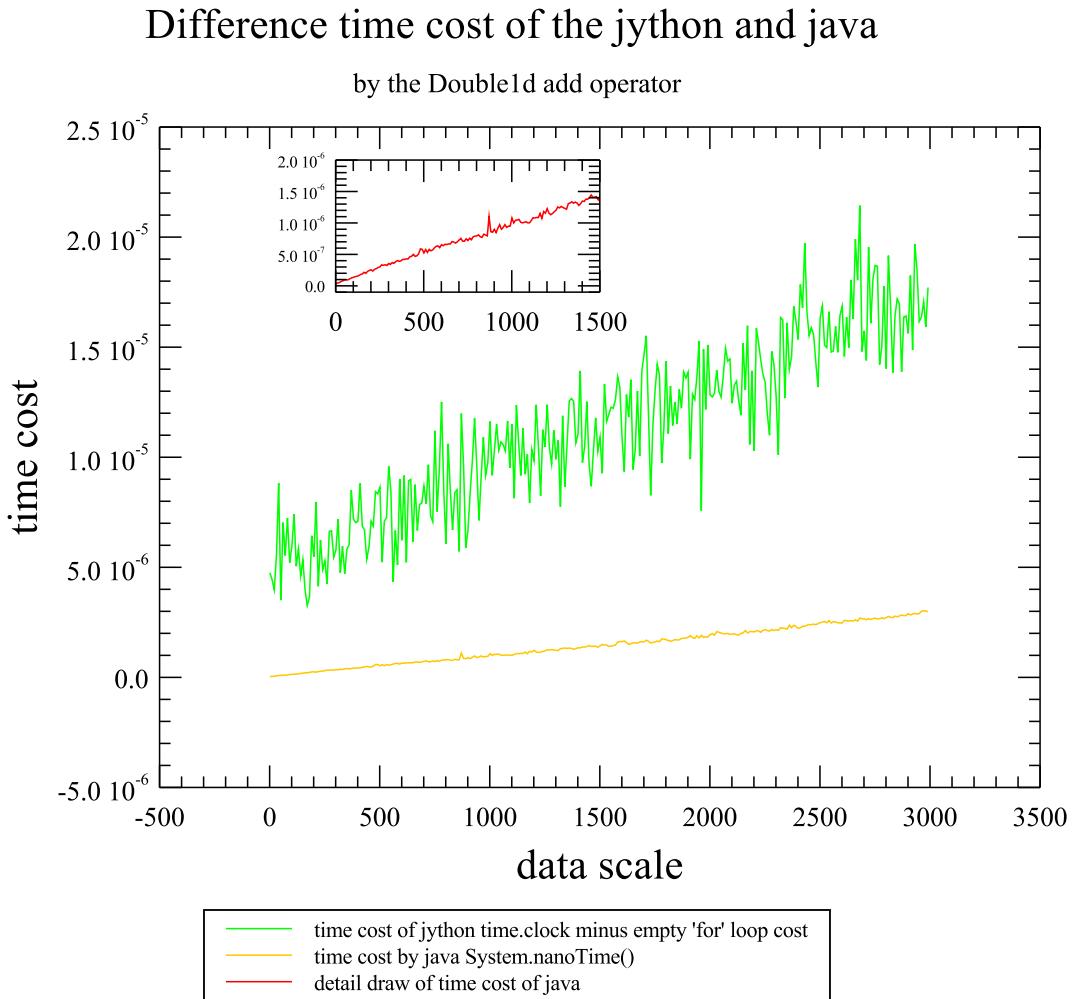


图 3.3: jython 和 java 在统计时间上的差异。绿线是扣除空 for 循环时间花费后的 jython 调用 Double1d 加法运算的时间花费，橘黄色线是 java 调用 Double1d 加法运算的时间花费。小图是 java 调用的细节放大。

文件名	功能
Main.py	时间花费统计处理主脚本
Complex1dNumCount.py	一维复类的调用次数统计
Complex1dFitTime.py	拟合法求出一维复类的时间花费
Double1dFitTime.py	拟合法求一维实数类的时间花费
Double1dNumCount.py	一维实数类的调用次数统计
Double2dNumCount.py	二维实数类调用次数统计
DOuble2dTimeCost	二维实数类时间花费计算
FFTNumCount.py	快速傅里叶变换调用次数统计
FFTTimeCost.py	快速傅里叶变换时间花费计算
MatrixNumCount.py	矩阵乘法调用次数统计
MatrixTimeCost.py	矩阵乘法时间花费计算
SVDNumCount.py	矩阵奇异值分解调用次数统计
SVDTimeCost.py	矩阵奇异值分解时间花费统计

表 3.2: 预处理和后续处理脚本总结

```

pre-processing
execute pipeline
post-processing

```

其中第二步就是执行数据处理脚本，我们不对它作任何修改。调用以下语句即可执行⁸:

```
execfile(path+pipeline+".py")
```

下面我们分别介绍预处理过程和后续处理过程。本工作编写的预处理和后续处理脚本见表3.2完整的主脚本代码可见附录B。

3.3.1 预处理过程

预处理过程其实比较简单。主要需要做三件事:

1. 声明进入脚本需要的模块。这其实是每一个jython脚本都应该做的事情，这里需要进入time, os, sys三个模块，另外主要要加上from herschel.ia.all import *。因为这里要在脚本里运行其他脚本，HIPE里默认一级脚本默认

⁸ 注意其中的path指脚本路径，pipeline为所处理的脚本文件名

引入所有的HCSS模块，但是我们需要在脚本里运行其他脚本，所以加上这一句，会避免一些解释运行的错误。

2. 指定工作路径，这里主要是我们这些处理脚本的工作空间路径。
3. 计数器初始化，因为对于一个要处理的脚本，要把我们的计数器预先清零。

这里列出作者的预处理过程，读者可根据自身需要添加删除一些内容。

```

4 #////////////----- preprocess -----
5
6 # import the necessary module
7 import sys
8 import time
9 import os
10 from herschel.ia.all import *
11
12 # define the path of workspace
13 sys.path.append("C:\\\\Users\\\\yfjin\\\\hcsp\\\\workspace\\\\")
14 path="C:\\\\Users\\\\yfjin\\\\hcsp\\\\workspace\\\\"
15 #pipeline="Photometer_Small_Map_Pipeline"
16 #pipeline="Spectrometer_Mapping_Pipeline"
17 #pipeline="Photometer_Large_Map_Pipeline"
18 pipeline="scanmap_Deep_Survey_miniscan_Pointsource"
19 # Reset the Counter
20 Double1dNumCount.ReSet()
21 Double2dNumCount.ReSet()
22 Complex1dNumCount.ReSet()
23 MatrixNumCount.ReSet()
24 FFTNumCount.ReSet()

```

3.3.2 后续处理过程

后续处理过程比预处理过程要复杂一些，可扩展性也更强。主要也分为三个部分：

1. 显示统计结果。注意这里统计结果可能输出在命令行窗口中⁹
2. 将次数统计结果输出到外存储器上。

⁹ windows下为cmd窗口，linux 下为shell窗口

3. 读取次数统计结果，计算时间花费。

前两步的实质其实就是调用上一节提到的一些设计好的函数即可，另外为了更直观的展示结果，可以加入一些绘图函数。为了方便组织，也可以根据不同的目标类，编写相应的脚本，在主脚本中调用这些子脚本即可。这里列一个可能的后续处理过程，这里调用到的处理子脚本都可以在附录B 找到。

```
30
31 execfile(path+pipeline+".py")
32
33 #####----- post-processing -----#####
34
35 # output the NumCount results. the result output in the Dos Windows
36
37 endPipeline=time.clock()
38 TPipeline=endPipeline-beginPipeline
39 print TPipeline
40 Double1dNumCount.Double1dGetALL()
41 Double2dNumCount.Double2dGetALL()
42 Complex1dNumCount.Complex1dGetALL()
43 MatrixNumCount.MatrixGetALL()
44 FFTNumCount.FFTGetALL()
45 SVDNumCount.SVDGetALL()

46
47 # execute the post-processing about NumCount.
48 # do something like draw the result and
49 # output the result to memory
50 execfile(path+"FFTNumCount.py")
51 execfile(path+"Double1dNumCount.py")
52 execfile(path+"Complex1dNumCount.py")
53 execfile(path+"Double2dNumCount.py")
54 execfile(path+"SVDNumCount.py")
55 execfile(path+"MatrixNumCount.py")

56
57 # execute the post-processing about TimeCost.
58 # each meshod's time cost
59 execfile(path+"FFTTTimeCost.py")
60 execfile(path+"Double1dFitTime.py")
61 execfile(path+"Complex1dFitTime.py")
```

这里主要讲一下计算时间花费上的可扩展性。我们之前提到为了更准确的计算每一次调用的时间花费，我们采用了多次累计运算求平均值的办法，但是实验表明，像垃圾回收等运算时间花给远远超过函数本身的时间花费，用统计平均虽然减小了这种偶然噪声的影响¹⁰，但是并没有完全去除它，仍然会引入了误差。最好的办法是进行拟合，然后剔除这些奇异点。在图3.4中，我们可以明显的看到有这种奇异点存在¹¹。但是一般情况下时间花费曲线并不好拟合，我们这里对最简单的情况——线性和二次——的情况进行拟合，其他情况仍然用统计求平均的办法。当然如何改进时间统计的方法，以尽量减小误差也是今后工作需要改进的一个地方¹²。

Double1d, Complex1d中的大部分运算处理的数据都是一维的，理论时间曲线也是线性的。我们就以Double1d为例，来看一下如何进行拟合，剔除奇异点。具体代码可见附录B中B.4.2Double1dFitTime.py。

HCSS中提供拟合包，可以进行线性拟合，多项式拟合和高斯拟合等常用的函数拟合，但是直接进行线性拟合，并不能很好的去除奇异点的影响¹³，我们还需要另外的方法去除奇异点后再拟合，这里用到的技术是clip。简单来说，对于运行时间随数据规模增大线性增加的函数。我们通过模拟可以得到一条曲线，这个曲线近似为一条直线叠加一些奇异点（少量）。我们对于每个点比较这个点的值和这个点前后10个点（不包括该点）值得平均值，如果该点的值远大于前后十点的平均值，则认为这个点为其一点，舍弃之。这个阈值可以用初步线性拟合的 $n\sigma$ 的值表示。一般奇异点比较少，clip一次即可，奇异点比较多的情况需要clip多次。图3.4中给了两个例子，来说明通过clip来消除模拟运算花费时出现的毛刺现象。其中红线是没有用clip的拟合，绿线是用clip技术去除奇异点的拟合，可以看出绿线还是很好的拟合了基线的，比较能代表运算的真实时间花费。

¹⁰ 下一章具体结果中，读者能有更直观的感受

¹¹ 这里毛刺的产生的原因并没有定论，我们猜测是系统在做垃圾收集时的时间消耗。实验表明，并不是所有的运算都会有这么明显的毛刺，是在add, sub, div和mul, pow函数运算参数为double是会明显产生，有理由相信是double变量一次运算被扩展成Double1d，运算完成后会释放这个空间消耗，这样就产生了垃圾回收的工作。不过也有猜测可能这种毛刺来自系统在多线程运算时，遇到静态的NumCount运算，而导致线程等待，突然的线程调度也会花费时间。不过这个时间消耗随数据规模扩大，时间花费增大，这样看来更有可能源自垃圾收集

¹² 虽然本文目前采用曲线拟合办法计算时间花费的只有线性拟合和多项式拟合两种形式，对于像快速傅里叶变换这种计算复杂性大的函数，我们还不能很好的拟合。不过实验表明在这种时间花费大的函数中，毛刺的影响已经不严重了（相对误差不严重），仅统计平均的办法也能达到较好的实验效果。

¹³ 事实上，如果直接线性拟合和统计求平均是等价的

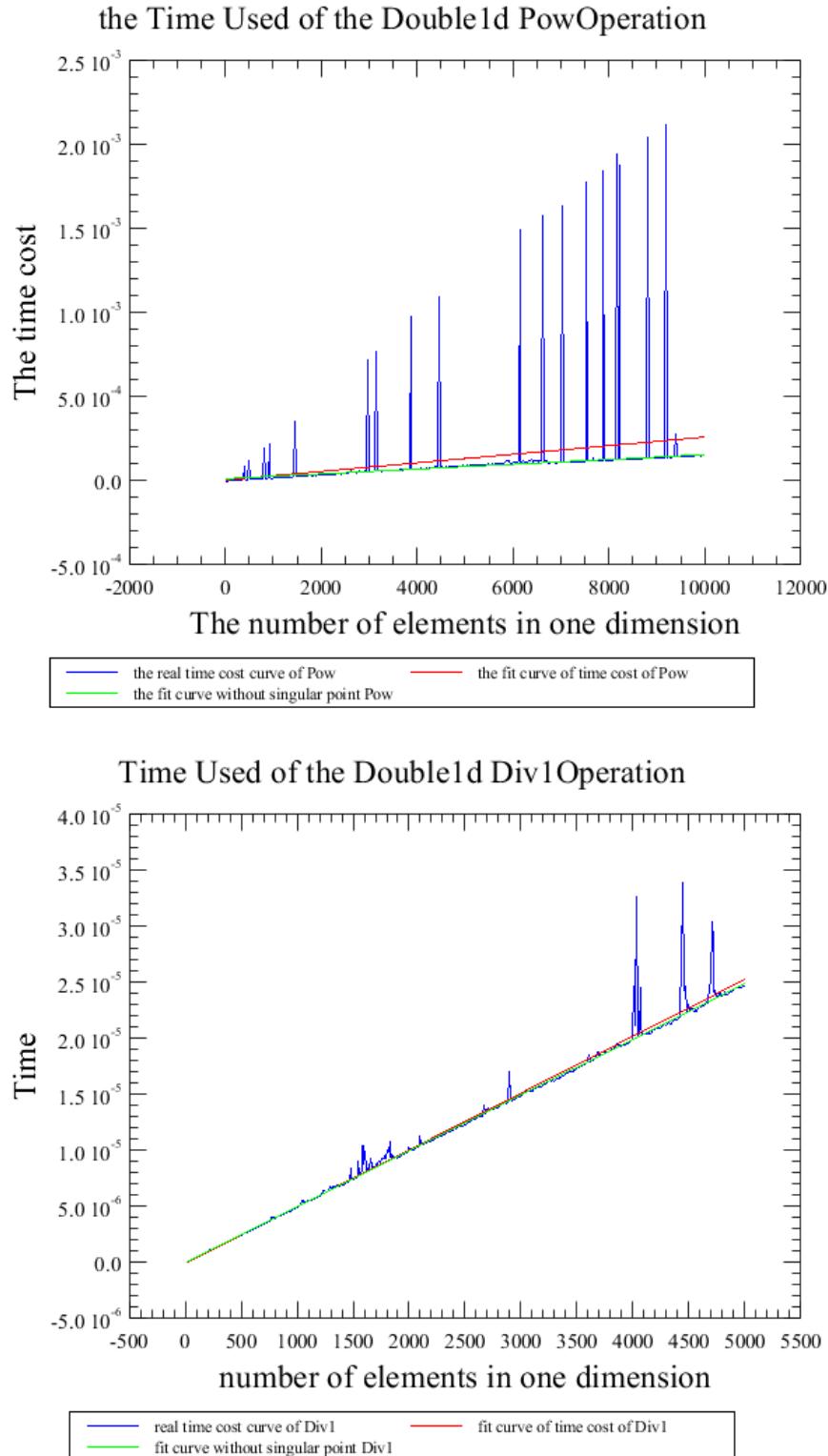


图 3.4: 统计运算时间中的毛刺分析。上: Double1d power函数, 下: Double1d divide函数。其中蓝色线为模拟时间的统计平均值, 红色线为没有用clip 时的曲线拟合, 绿色线为用了clip的曲线拟合

这样我们就完整介绍了在HIPE环境下，对脚本的预处理和后续处理过程了。通过java和jython脚本的设计，我们最终完成了一套比较方便，便于实施的方法来对HCSS数据处理脚本的时间花费做统计。

第四章 HCSS 系统函数的 GPU 实现 方式

CUDA虽然有很广泛的应用，但目前其良好支持的语言能只有 C/C++ 和 fortran. HCSS 是基于 java 语言的，CUDA 本身并不支持java 语言。java语言拥有高效的编程效率和良好的软件组织构架，但是软件运行效率却稍逊一筹。如何利用GPU来弥补java软件系统的短板是件看似诱人，但实际操作并不轻松的工作。我们期待今后会出现直接支持CUDA的java语言编译器，不过目前的技术手段还不允许。如果不能在java语言中调用GPU，那么本工作之前的一切分析都将成为空谈，无法实施。幸运的是，java 提供一种方式调用本地动态链接库，我们可以将自己的 GPU 应用编写成本地链接库的方式让 java 程序调用。这种调用本地链接库的方式就是下面要介绍的JNI(Java Native Interface) 技术

4.1 JNI技术实现java语言对CUDA的调用

JNI (Java Native Interface) 可以确保运行在java 虚拟机上的java 代码调用本地用其他语言编写的程序或者类库。一开始JNI是为了本地已编译语言，尤其是C 和C++而设计的，但是它并不妨碍你使用其他语言，只要调用约定受支持就可以了。

我们仍以2.2.1节的例子为基础，按照JNI要求的格式改写成MatrixJ.cu 代码，看看如何让java 程序调用cuda 代码。首先我们看一下基本流程。如下列表。

- 首先如一切常规的java代码一样，我们将要用本地函数实现的功能用关键词native声明，用 `System.loadlibrary()` 加载含此功能的本地类库¹，

¹ 注意这里的加载只是声明性质的，本地类库现在还可以不存在，之后再编译生成

用javac 编译java 代码。

- 用 `javadoc -jni` 生成定义的本地函数的头文件，生成的头文件主要定义了java可认得函数名定义。
- 将生成的头文件include到我们的本地函数文件中，编/改写我们的cuda程序，主要修改一下函数定义就行了。
- 编译成动态链接库。用nvcc编译器就行了，编译选项选择动态链接库而不是可执行程序。
- 用 `java **` 运行程序即可。

下面我们逐条进行说明，`MatrixJ.java`代码如下：

```
1 public class MatrixJ
2 {
3     private native void MatrixMultiplication(float[] M,float[] N,float[] P,int Width);
4
5     static
6     {
7         System.loadLibrary("MatrixMultiplication");
8     }
9
10    public static void main(String[] args)
11    {
12
13        float [] M={1,2,3,4,5,1,2,3,4,5,1,2,3,4,5,1,2,3,4,5,1,2,3,4,5};
14        float [] N={1,2,3,4,5,1,2,3,4,5,1,2,3,4,5,1,2,3,4,5,1,2,3,4,5};
15        float [] P={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
16        int Width=5;
17
18        new MatrixJ().MatrixMultiplication(M,N,P,Width);
19
20        for(int i=0;i<25;i++){
21            System.out.println(P[i]);
22        }
23    }
24 }
```

代码第3行将原来Matrix.cu程序中的函数 MatrixMultiplication 声明为这个 java 程序的 native 函数，第7行，将包含 MatrixMultiplication 的本地链接库作为静态方法引入 java 程序，这里名称不必须用 MatrixMultiplication，但是要和后面编译的动态链接库同名。当然这里静态的引入动态链接库是比较简单的一种方法，也可以在java 程序的构造函数中引入这个链接库。编写好这个程序后，用javac MatrixJ.java编译生成MatrixJ.class文件²。

用 javah -jni MatrixJ 生成MatrixJ.h文件，打开可见如下代码：

```

1  /* DO NOT EDIT THIS FILE - it is machine generated */
2  #include <jni.h>
3  /* Header for class Matrix */
4
5  #ifndef _Included_MatrixJ
6  #define _Included_MatrixJ
7  #ifdef __cplusplus
8  extern "C" {
9  #endif
10 /* 
11  * Class:      MatrixJ
12  * Method:     MatrixMultiplication
13  * Signature:  ([F[F[FI)V
14  */
15 JNIEXPORT void JNICALL Java_MatrixJ_MatrixMultiplication
16   (JNIEnv *, jobject, jfloatArray, jfloatArray, jfloatArray, jint);
17
18 #ifdef __cplusplus
19 }
20 #endif
21 #endif

```

主要是对引入的本地函数作符合JNI调用格式的函数定义声明。见代码第15行。

之后就可以根据这个声明修改我们的Matrix.cu代码了，我们把新的程序命名为MatrixJ.cu。MatrixMulKernel函数不变³，MatrixMultiplication根据刚

² 这里用命令行的方式编译，便于直观呈现，也可以用集成编译环境IDE编译

³ 这里值得一提的是，这个函数时CUDA的kernel函数，包含关键词 __global__，__share__，JNI不能良好支持，所以把这个方法隐藏在MatrixMultiplication函数中执行，是一种讨巧而实用的办法

才MatrixJ.h中的定义修改，如代码37到63行。

```
1 #include <cuda.h>
2 #include <stdio.h>
3 #include <cuda_runtime.h>
4 #include <time.h>
5 #include <iostream>
6 #include "MatrixJ.h"
7 #define TILE_WIDTH 4
8 using namespace std;
9
10 __global__ void MatrixMulKernel(float* Md, float* Nd, float* Pd, int Width)
11 {
12     __shared__ float Mds[TILE_WIDTH][TILE_WIDTH];
13     __shared__ float Nds[TILE_WIDTH][TILE_WIDTH];
14
15     int bx=blockIdx.x; int by=blockIdx.y;
16     int tx= threadIdx.x; int ty=threadIdx.y;
17
18     int Row=by*TILE_WIDTH+ty;
19     int Col=bx*TILE_WIDTH+tx;
20
21
22     float Pvalue=0;
23
24     for(int m=0;m<Width/TILE_WIDTH;++m){
25         Mds[ty][tx]=Md[Row*Width+(m*TILE_WIDTH+tx)];
26         Nds[ty][tx]=Nd[(m*TILE_WIDTH+ty)*Width+Col];
27         __syncthreads();
28
29         for(int k=0;k<TILE_WIDTH;k++)
30             Pvalue+=Mds[ty][k]*Nds[k][tx];
31         __syncthreads();
32     }
33
34     Pd[Row*Width+Col]=Pvalue;
35 }
36
37 JNIEXPORT void JNICALL Java_MatrixJ_MatrixMultiplication
```

```

38 (JNIEnv *env, jobject obj, jfloatArray jM, jfloatArray jN, jfloatArray jP, jint Width)
39 {
40     int size=Width*Width*sizeof(float);
41     jfloat* M = env->GetFloatArrayElements(jM,0);
42     jfloat* N = env->GetFloatArrayElements(jN,0);
43     jfloat* P = env->GetFloatArrayElements(jP,0);
44
45     jfloat *Md,*Nd,*Pd;
46     cudaMalloc((void**)&Md,size);
47     cudaMemcpy(Md,M,size,cudaMemcpyHostToDevice);
48     cudaMalloc((void**)&Nd,size);
49     cudaMemcpy(Nd,N,size,cudaMemcpyHostToDevice);
50
51     cudaMalloc((void**)&Pd,size);
52
53     dim3 dimBlock(TILE_WIDTH,TILE_WIDTH);
54     dim3 dimGrid(Width/TILE_WIDTH,Width/TILE_WIDTH);
55
56     MatrixMulKernel<<<dimGrid, dimBlock>>>(Md,Nd,Pd,Width);
57
58     cudaMemcpy(P,Pd,size,cudaMemcpyDeviceToHost);
59     cudaFree(Md);cudaFree(Nd);cudaFree(Pd);
60
61     env->ReleaseFloatArrayElements(jM,M,0);
62     env->ReleaseFloatArrayElements(jN,N,0);
63     env->ReleaseFloatArrayElements(jP,P,0);
64
65 }

```

这样需要修改的cuda程序就完成了，用编译器编译成动态链接库即可，作者用的指令为：

```

nvcc --compiler-options '-fPIC' -o libMatrixMultiplication.so --shared
MatrixJ.cu -I/usr/lib/jvm/java-1.6.0-openjdk-amd64/include
-I/usr/lib/jvm/java-1.6.0-openjdk-amd64/include/linux
-L/usr/local/cuda/lib64/java

```

注意这里可以用 -I 指明java路径， -L指明java类库路径。linux下动态链接库的命名规则为lib***.so， windows为***.dll。

最后我们用 `java MatrixJ` 运行即可。如果动态链接库的路径不在同一个文件夹下，还需要声明链接库的路径。

这样我们就完成了java语言通过JNI对cuda程序的调用，但是有一点是需要注意的，java程序的一个特点就是其对平台的移植性强。如果java 代码调用本地程序，必然会一定程度上破坏平台的移植性。所以如果不能大规模的软件提高性能，并不推荐使用JNI调用本地程序。

4.2 JCUDA库简介

读者也发现，在没有支持CUDA的java编译器的情况下，编写一个能让java调用的动态链接函数并不方便。而且上面的例子左右文件都在一个文件夹中，所有路径都用的是当前路径，但是对于HCSS这样大型的软件，几乎每一步都涉及到路径声明的问题，稍不留神就会出错。例如为了编译 `MatrixMultiply.java` 这个类，要指明它依赖库的路径，要复杂的写成：

```
javah -classpath ".:/home/users/woekspace/ia_numeric/out/main/lib"  
herschel.ia.numeric.toolbox.matrix.MatrixMultiply
```

在生成动态链接库后，还要把动态链接库的路径加入系统路径。如果没修改一个函数就要进行如此繁琐的操作，就会极大的限制编程效率。那么将常用的功能打成动态链接函数将是一个不错的想法。JCUDA 就在这种情况下应运而生。

JCUDA属于第三方开发的软件应用，其基本思想就是把Nvidia官方开发的CUDA函数库⁴，改写成java可直接调用的动态链接库。目前JCUDA 已经支持Windows，MAC OS和大部分Linux操作系统。我们可以在<http://www.jcuda.org/>下载软件包。包括`**.jar`的函数接口，和`**.dll`（linux 下为`**.so`）的动态链接库。

在 HCSS 中使用 JCUDA 的方法很简单，JCUDA的下载包里都是库函数，不需要安装，将 `**.dll`（linux中为`**.so`）放到系统目录下。将`**.jar` 包放到 java 的 classpath 下。例如在linux系统中就可以直接在 `.bashrc` 文件中添加如下代码。Window下类似的需要在系统环境变量中添加需要的路径设置。

⁴ 见2.2.2 节

```

export JCuda_HOME =/home/users/JCUda-All-0.5.0-bin-linux-x86_64
export PATH=$PATH:$JCuda_HOME
export CLASSPATH= $JCuda_HOME/jcuda-0.5.0.jar:$JCuda_HOME
/jcublas-0.5.0.jar:$JCuda_HOME/jcufft-0.5.0.jar:$JCuda_HOME
/jcurand-0.5.0.jar:$JCuda_HOME/jcusparse-0.5.0.jar

```

以最新的jcuda 0.5.0a（支持CUDA 5.0，最新更新为2013-03-20）为例，其中包含如下几类应用：

jcuda 其中包含CUDA驱动接口（CUDA driver API）和CUDA运行接口（CUDA runtime API）。支持CUDA的基本操作，如存储、拷贝、传输、释放等功能。

jcublas 对应cublas库，是线性代数库。功能和调用方式和cublas相似，接口和普通java函数一致。

jcufft 对应cufft库，支持1D, 2D, 3D的快速傅里叶变换和反变换。使用方法与cufft类似，可以像普通java函数一样被调用。

jcurand 对应curand库，用于由GPU产生伪随机数。

jcusparse 对应cusparse库，用于稀疏矩阵的存储和运算。

然后就可以利用第二章的方法在 HCSS 的代码中使用 JCUDA 的类库了。

4.3 典型的 HCSS 基础类在 CPU 与 GPU 上运行的效率对比

下面我们就JCUDA中的两个典型应用——矩阵乘法和快速傅里叶变换——来说明一下如何把HCSS基础函数用GPU实现，并对比这些函数在CPU和GPU上的运行效率。

矩阵乘法：

矩阵乘法在 HCSS 中对应 MatrixMultiply 类。在 herschel.ia.numeric.toolbox.matrix 包下。其中包括 bool, byte, short, int, long, float, double 等数据类型的矩阵乘法。这里我们主要关心double和float型的矩阵乘法，因为这是科学计算里最常见的数

据类型。

如前所述我们用jcublas中的函数 JCublas.cublasSgemm 和 JCublas.cublasDgemm 来计算单/双精度的矩阵相乘。代码见附录A。时间对比结果如图
快速傅里叶变换:

HCSS中傅里叶变换的模块在 herschel.ia.numeric.toolbox.xform 中，有 FFT, IFFT 两个基本函数，另外还有 FFT_PACK, FFT_PACK_ODD, FFT_PACK_EVEN等。FFT 函数中主要基于两种方式做傅里叶变换，如果数组维数是2的整数次幂，则用CooleyTukey 方法（基2的傅里叶变换），如果数组维数不是2的整数次幂，则采用chirpZ 傅里叶变换（自由基的傅里叶变换）。JCuda中利用cufft来计算FFT，函数 Jcufft.cufftExecZ2Z()可以用来做复数到复数的傅里叶变换。

我们对两种方法的傅里叶变换（CooleyTukey, chirpZ）分别作比较，结果如图4.2从图中可以看出显然数组维数越大，GPU的优势也越明显。

从上面的结果可以看出对于矩阵相乘，到 133×133 矩阵规模时，GPU 的运行效率已经超过CPU。而对于FFT来说，CooleyTukey方法在 2^{14} 数据规模，ChirpZ 方法在4000数据规模时，GPU 的运行效率已经超过CPU。这个结果是比较符合直观的，在数据规模小的时候，从CPU到GPU的数据传输占用了大量的时间，所以CUDA话的时间比java 多。但是当数据规模大的时候GPU 的运算优势就体现出来了。对于FFT运算GPU超过CPU的数据规模大于矩阵相乘，这也是容易理解的。我们知道对于串行系统，方阵乘法的时间复杂度是 $O(n^3)$ ，FFT的时间复杂度是 $O(n \cdot \log(n))$ 。所以矩阵乘法的数据规模在很小的时候，GPU 的运算优势就体现出来了。

我们希望优化HCSS软件系统，就是希望找到那些数据传输不频繁，计算复杂度高，可并行度高的函数。但是并不是找到这种函数就一定能优化系统性能的。要知道如果这些复杂的运算只算了很少的几次，而我们的应用大量时间花费在计算复杂度低的地方，优化这些复杂度高的类仍然不能显著提高系统性能。下一章我们就来讨论如何找到系统性能提升的瓶颈，找到主要耗时的应用部分。

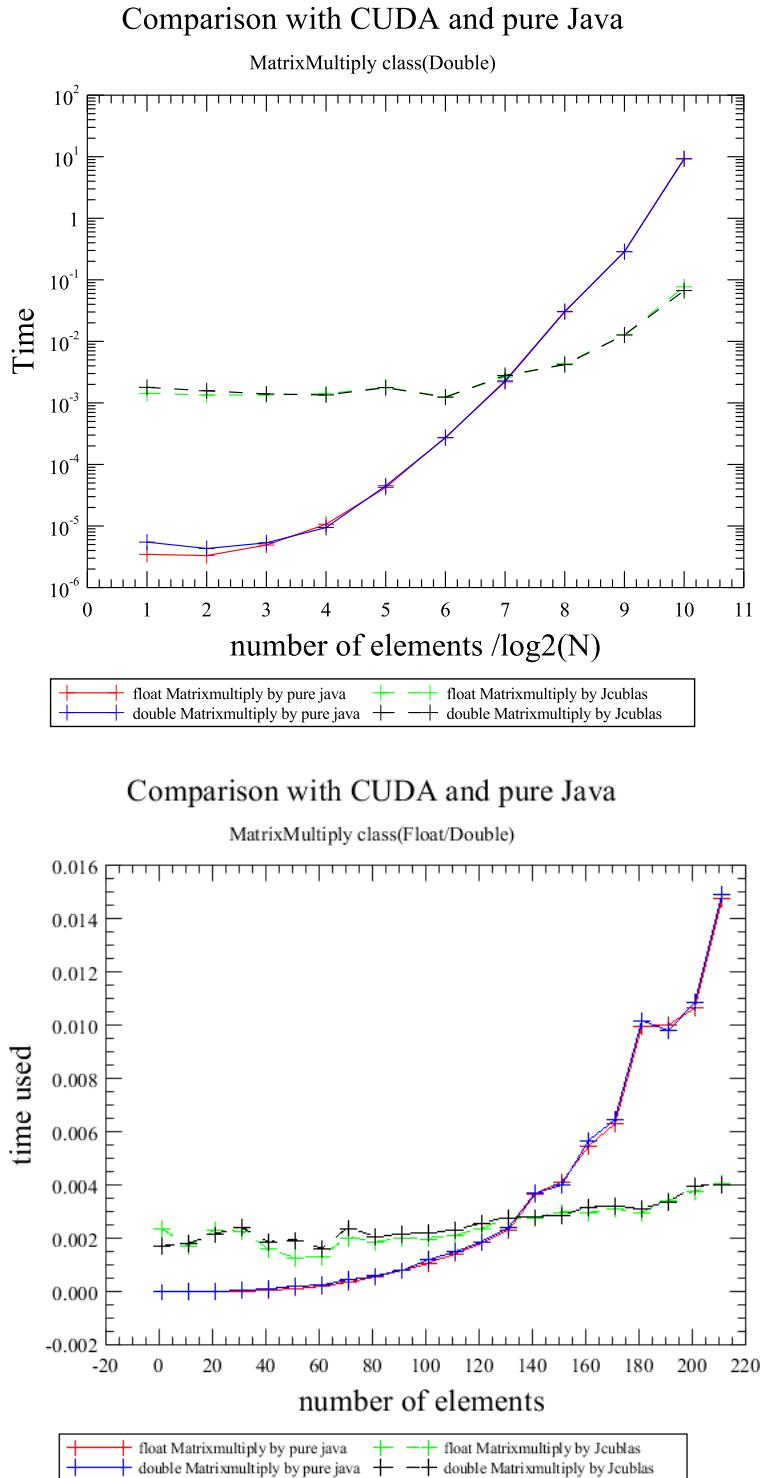


图 4.1: 矩阵相乘在Java和JCuda上运行时间效率对比, 上图为取对数图, 下图为线性图

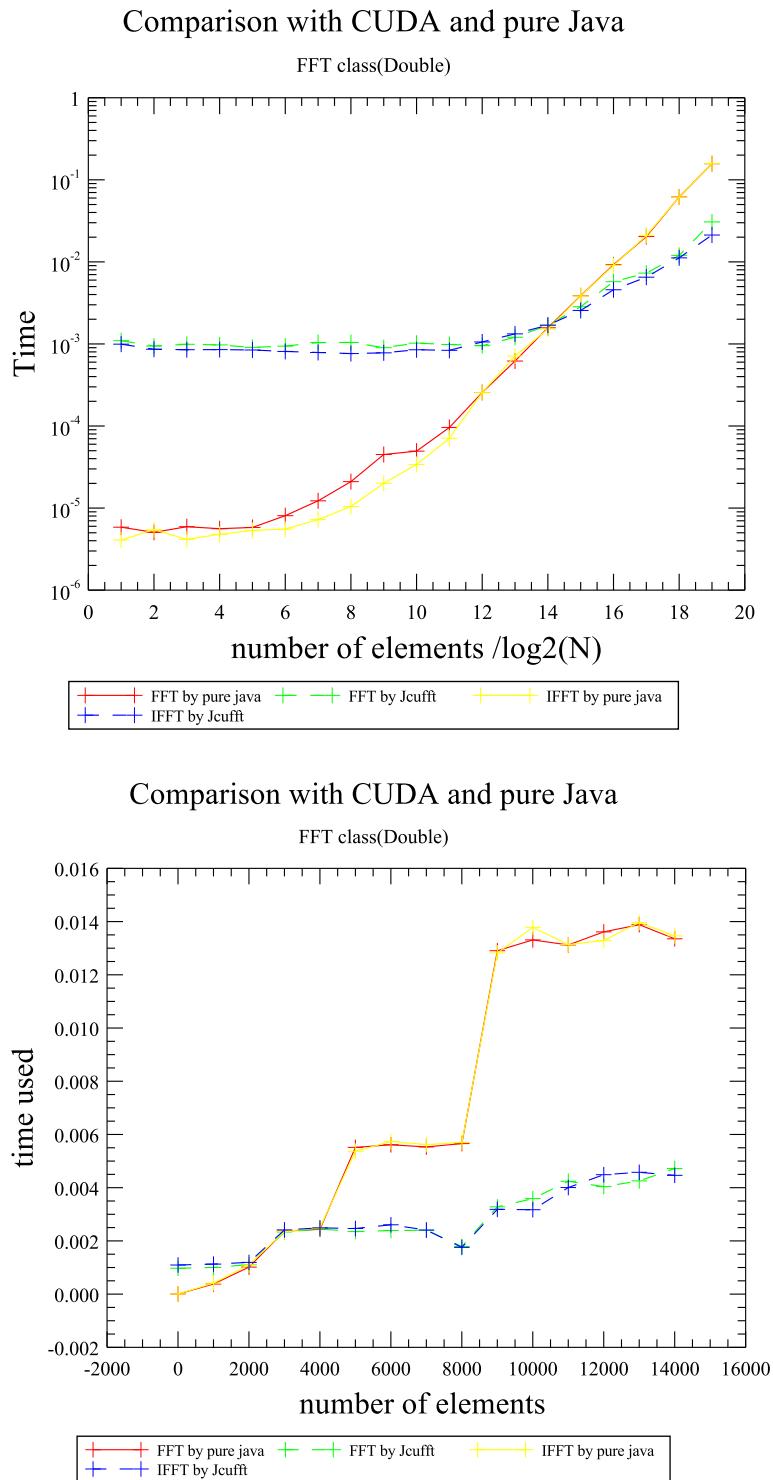


图 4.2: FFT/IFFT 在 java 和 CUDA 下运行时间对比，上图是 CooleyTukey，下图是 ChirpZ

第五章 分析和加速HIPE数据流水线： SPIRE 光谱处理

经过第三和四章的介绍，我们已经掌握了分析和加速HCSS数据处理脚本的基本方法。HCSS 提供常用的数据处理流水线，典型的天文数据处理莫过于光谱处理和测光成像。这一章我们主要关注 SPIRE 光谱处理脚本，来看我们的分析方法的实现效果，找出主要耗时的目标类，并尝试用 GPU 实现其加速，测试用 GPU 加速的加速效果。最后根据加速结果，总结 GPU 加速 SPIRE 光谱处理的特点。

5.1 HIPE数据流水线

Herschel望远镜一共携带三个后端设备：SPIRE, PACS 和 HIFI。其中 SPIRE 和 PACS 可以做测光观测也可做光谱观测，HIFI 可以做光谱观测。这些后端设备从观测到数据到处理成天文学家可以直接利用的 fits 文件之间还有很大的距离。HCSS的一大作用就是提供了完整的从原始数据 (level-0) 到科学可利用的成品数据 (level-2) 的数据处理脚本，成为数据流水线 (pipeline). 这些脚本用jython编写，可以在HIPE 交互界面中直接运行，而且用户可以根据自己观测源的特点进行有针对的修改。

下面在说明基本的数据处理流程前，我们先介绍一下 Herschel 数据分级。

Herschel根据数据处理过程的不同阶段和数据的不同用途，给数据分了级别和种类，简单列表如下：

History: 记录对原始数据做过操作，包括对数据施加的任务 (task)，任务参数和其他操作。

Auxiliary context: 包含直接或者间接的Herschel科学数据处理中用到的非科学性的航天器数据。

Calibration context: 描述卫星和后端设备的仪器参数。

Level-0 context: 原始数据 (raw data)。观测数据根据观测的时间顺序组织成一维时间线 (timeline) 的形式。

Level-0.5 context: 做了一些初步的器件参数的处理，这个阶段数据还是以时间线的形式存在。

Level-1 context: 做过了仪器定标并将数据转换到天球坐标中，原则上这个阶段的观测数据已经和仪器无关了。

Level-2 context: 可直接用于科学分析，达到虚拟天文台数据访问的质量级别。

此外还有Level-2.5 context, Level-3 context, Observation log context, Quality context, Trend analysis context, Telemetry context等级别的数据。有兴趣的读者可以参见HIPE中的帮助，这里我们主要关注原始数据到level-2的处理过程，这也是HIPE数据流水线处理的内容。

需要明确的一点是虽然根据天体源的特点（点源，面源），观测的内容（测光，光谱）的不同，HIPE数据流水线也会有很大的不同。但是根据我们对数据等级的定义，无论什么样的数据流水线脚本，最终都是为了处理到指定质量要求的数据结果。例如，对于点源测光成像或者面源测光成像，抑或是光谱图处理，SPIRE的数据流水线都可以表示成如图5.1。用户可以控制和修改的是从level-0到level-2的部分。

HCSS根据后端设备的不同和光测内容的不同有大量的数据处理流水线，我们并不打算对这些数据流水线做一一介绍，有兴趣的读者可以参考HCSS帮助和参考文献 [17](#) [17](#) [18](#) [19](#) 等。这里主要对SPIRE 的光谱处理，SPIRE的成像处理，PACS的成像处理做分析。根据前文的介绍，我们的分析方法是普遍使用的，读者只要稍加修改，就可以扩展到自己的应用上去。

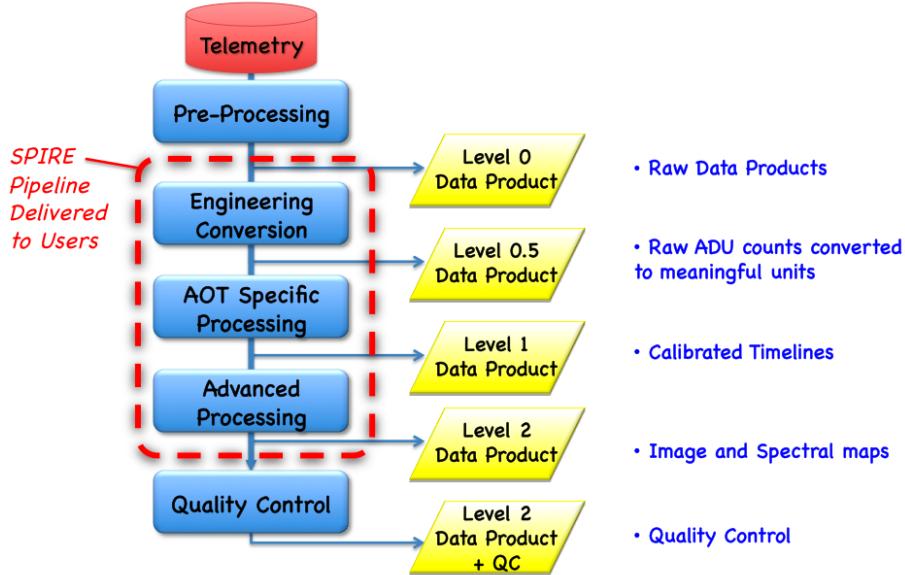


图 5.1: SPIRE数据处理阶段和流程

5.2 分析和加速 SPIRE 光谱处理流水线

5.2.1 SPIRE傅里叶光谱数据流水线

SPIRE的采用傅里叶光谱仪（FTS），可以将level-0的原始数据处理生成level-2的科学数据。包括点源光谱和光谱图（成像图上多点的光谱可以同时得到）两种。脚本分别为：

- Spectrometer_Point_Pipeline.py
- Spectrometer_Mapping_Pipeline.py

SPIRE光谱处理第一步从level-0到level-0.5，叫做工程转化（Engineering Conversion），这一步无论测光观测和光谱观测都相同的，其中主要包含的部分包括从原始时间线中分离数据，数据格式转换，掩盖坏通道，掩盖坏的遥感参数，时间转换和重新排序，计算结型场效应管电压，计算热耦合探测器的电压和电阻的均方根值，最后才能生成level-0.5的数据，具体流程图可见图5.2。

第二步是从level-0.5到level-2.0。其主要处理流程分为几个阶段^{[18][19]}：修改时间线，创建干涉图，修改干涉图，转换干涉图，修改光谱。处理的内容包括：一级毛刺去除，非线性校正，温度校正，滤波，Clipping校正，校正时域相位，

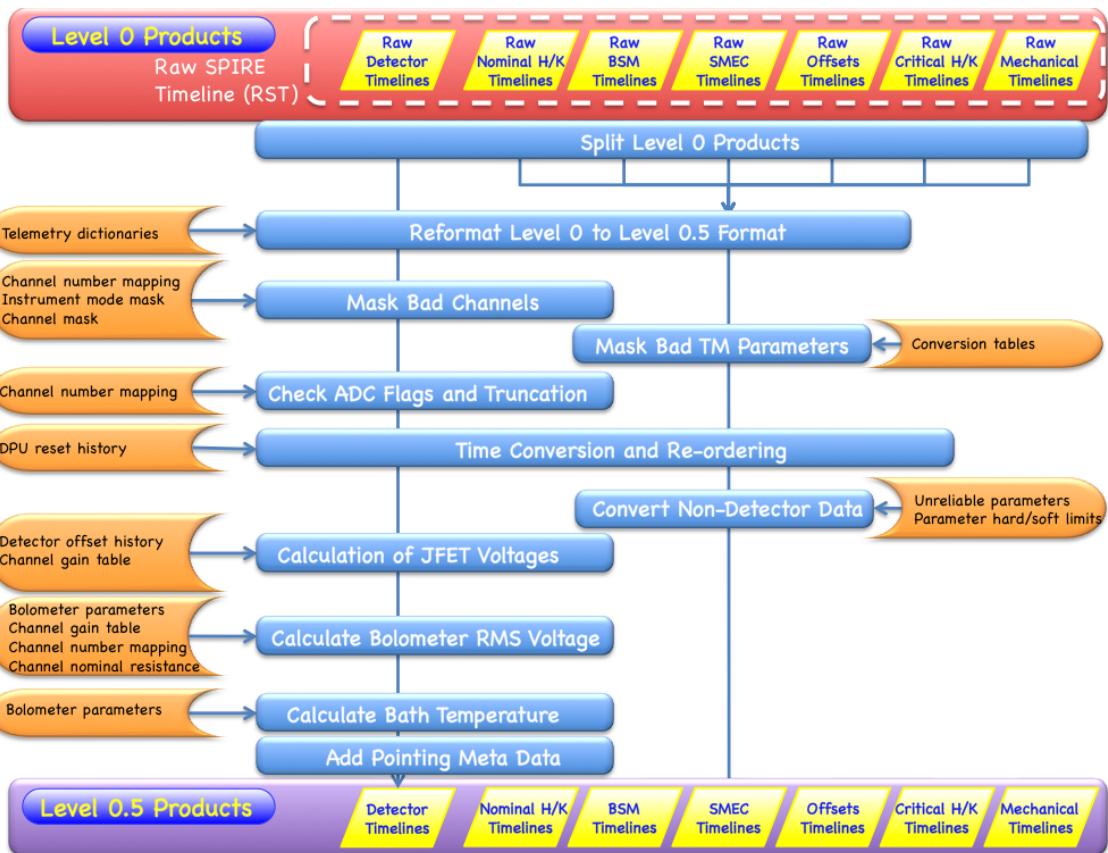


图 5.2: SPIRE从level-0到level-0.5的数据处理流程

观测ID	1342208388	观测源	m82
标准产品代 (SPG) ID	SPG v9.1.0	观测开始时间	2010-11-08 01:58:26.0
仪器	SPIRE	观测结束时间	2010-11-08 06:55:49.0
观测模式	SpireSpectroPoint	观测周期	17843.0s
赤经	148°57'53.82"	观测者	cwilso01
赤纬	+69°40'47.32"	提案ID	KPGT_cwilso01_1
PA	110.445	观测数	543

表 5.1: Herschel M82 观测数据, 观测参数

基线校正, 二级毛刺去除, 相位校正, 傅里叶变换, 删带外的功率, 仪器校正, 流量校正, 望远镜校正, 流量转换, 生成光谱或光谱cube 等。流程图可见图5.3。

5.2.2 SPIRE光谱处理流水线分析

我们采用的数据是M82星系 (NGC 3034、雪茄星系) 的SPIRE红外光谱。M82是一个位于大熊星座, 距离地球1200万光年的不规则星系¹, 视星等为8.41等。M82中心是直径约500 pc 的活动星爆区, 为研究大质量星团和超新星提供大量数据样本。我们采用是Herschel 2010年拍摄的数据, 数据的具体参数如下5.1表:

因为观测数据是光谱图(scan map)的形式, 我们使用如下脚本:

- Spectrometer_Mapping_Pipeline.py

下面我们用第三章的方法, 分析这个脚本, 主要统计Double1d, Double2d, Complex1d, Complex2d, matrix, xform等类里的各个方法(函数)的调用总次数, 针对不同数据规模的调用次数, 最终时间花费等。我们先把结果列于表5.2, 再来一一分析结果中每一项的意义。

从上表中可以看出最主要的时间依次是xform.fft, Double1d.power, Complex1d.power等。更直观的比例结果可见图5.4

考虑一个函数时间花费多少, 主要考虑三方面的因素, 一是函数的运算复杂度, 这一点上像 MatrixMultiply, SVD, FFT, DotProduct 的花费会比较大。二是函数的被调用的次数, 从表上看 DotProduct 等 Double1d 的函数调用次数都很多; 三是数据规模, 这一点没有在表中显示, 我们的方法统计了每次函数调用的数据规模大小, 我们就fft, Pow等时间花费大的类, 看一下它们的数据规模大

¹ 之前一直认为M82 是不规则星系, 但是在2005 年, 人们在近红外波段发现它是有两条旋臂的, 具体参见^[20]

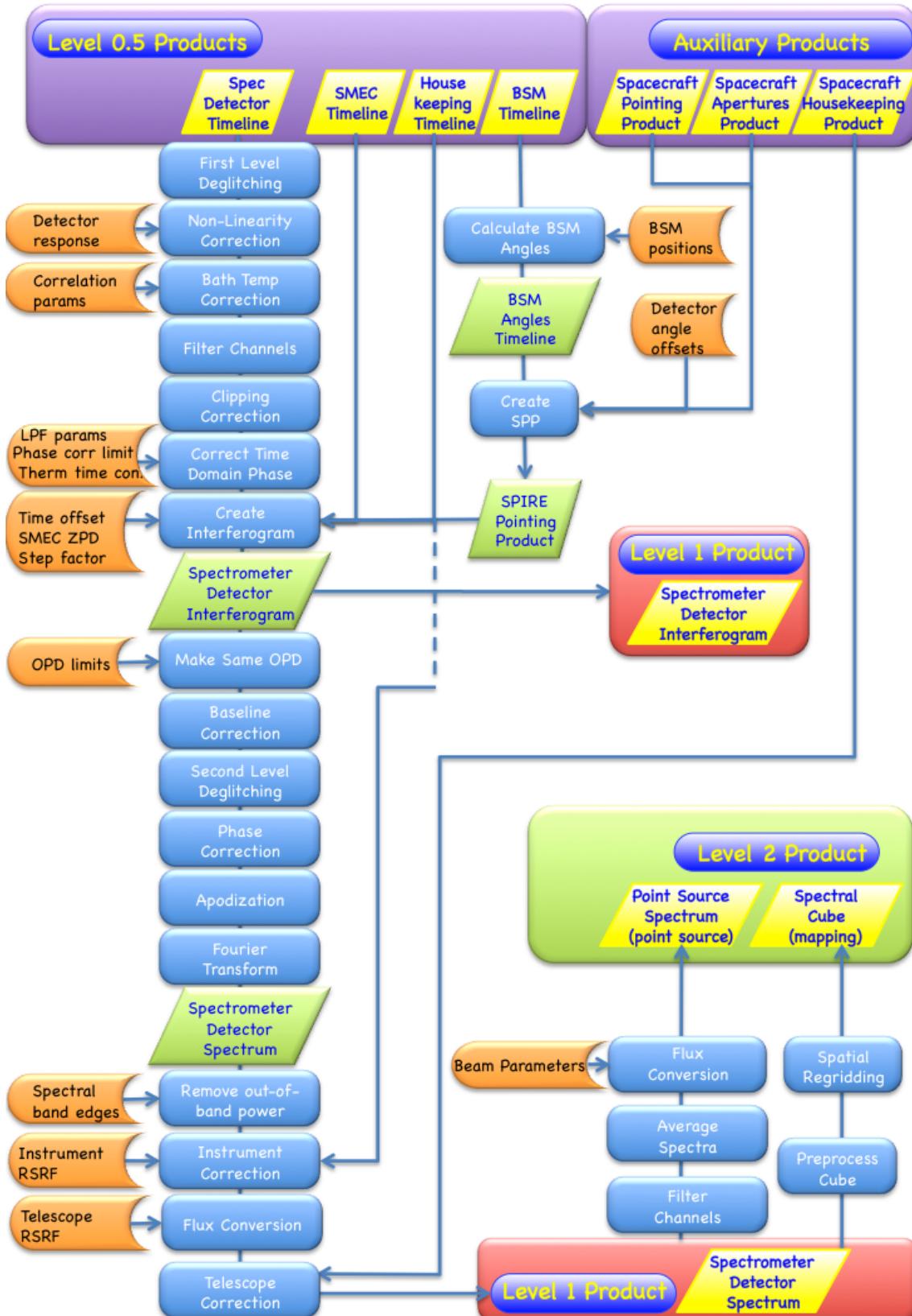


图 5.3: SPIRE FTS光谱处理流程

M82 光谱图处理时间总花费 (CPU) :2238.32s						
目标类	目标函数	计数	时间花费	总时间	百分比	总百分比
Complex1d	Add	6656	0.03s	8.38s	< 1%	0.37%
	Sub	0	0.0s		0.0%	
	Mul	124989	6.07s		0.27%	
	Div	832	0.04s		< 1%	
	Pow	1664	2.24s		1.00%	
	Abs	0	0.0s		0%	
Double1d	Add	568399	4.21s	111.71s	0.18%	4.97%
	Sub	9418995	0.91s		< 1%	
	Mul	7011313	3.83s		0.17%	
	Div	153939	7.25s		0.324%	
	Pow	11247621	88.54s		4%	
	Abs	71167	0.28s		< 1%	
	Dot	141147028	6.71s		0.3%	
xform	FFT	13884	909.05s	1308.35s	40.6%	58.4%
	IFFT	6143	399.30s		17.8%	
matrix	MatrixMultiply	328098	0.48s	0.48s	< 1%	< 1%
	SVD	0	0.0s		0	

- i 此外还有一些method统计后发现没有被调用，如Double2d等，数据为零，未列于表内。
ii 这里一些方法，如Add, Sub等根据输入参数不同，实现了函数重载，在统计时间时已经考虑了这个因素，而分别统计计算加和。

表 5.2: M82 光谱数据处理时间花费分析

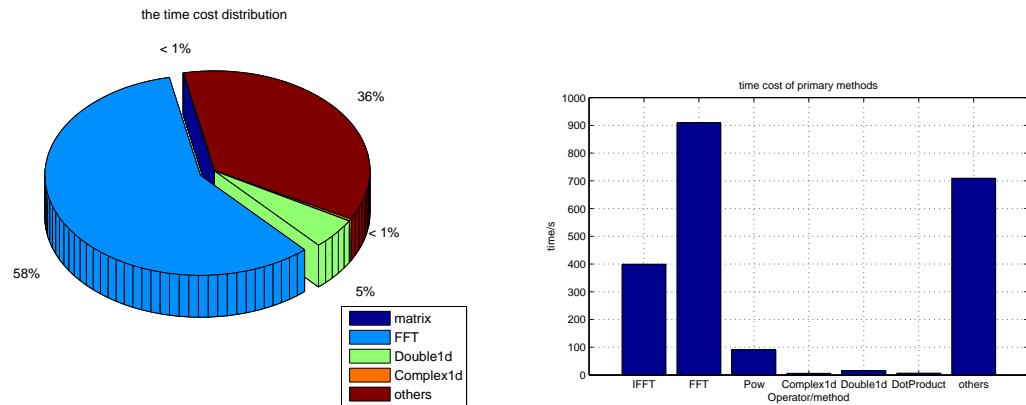


图 5.4: SPIRE光谱处理脚本时间分析 (数据 M82光谱图)。左: 主要函数用时百分比。右: 主要函数时间花费

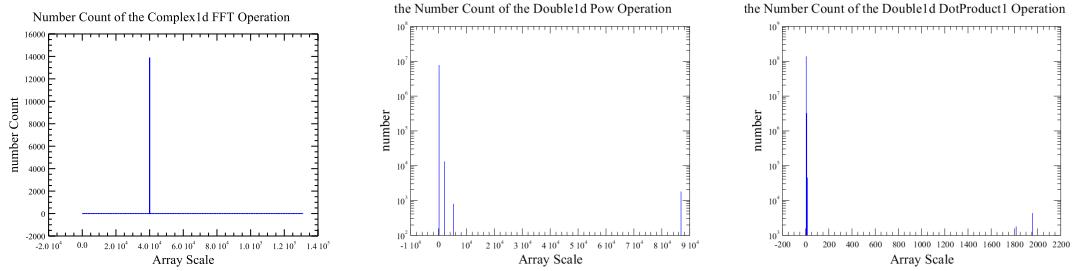


图 5.5: 左: FFT函数调用的数据规模。中: Pow函数调用数据规模, 右: DotProduct函数调用的数据规模

M82 SPIRE 光谱图处理流水线脚本				
运行设备	FFT/IFFT时间花费	加速比例	脚本时间花费	加速比例
CPU	1308.35s		2238.32s	
CPU&GPU	74.24s	17.6x	1218.37s	1.83x

表 5.3: GPU加速SPIRE光谱处理结果

小, 再就 MatrixMultiply, DotProduct 等虽然实际时间花费少, 但理论上在其他两方面都很可能是大时间花费项的函数, 看看它们的数据规模。见图5.5。从图中也就能比较好的理解为什么 fft 和 pow 的时间花费大了。

我们再回顾一下什么情况会适合 GPU 加速: 一是数据运算的可并行度大, 二是每个线程的运算量和数据传输比大, 即并行节省的时间能够补偿数据传输花费的时间, 计算的代价大于 I/O 的代价。三是可并行运算成分的时间花费在整个脚本中的时间花费比例大。这样看了 FFT 和 Pow 还是比较符合的。但是是否能加速含需要实验数据的支持。

5.2.3 GPU对SPIRE光谱流水线的加速效果

经过上一节的分析, 我们发现SPIRE光谱数据流水线是最适合GPU加速的数据处理脚本。我们这里将其中的FFT 使用GPU实现, 再重新运行脚本, 统计前后的时间花费对比, 见表5.3

CPU和GPU运行相同的数据处理流水线, 我们期待最终结果是相同的。那么下面看看CPU和GPU的处理结果。

SPIRE光谱扫描图 (scan map) 处理结果为低频段 (SLW) 和高频段 (SSW) 的光谱cube。CPU和GPU的光谱cube 在几何空间的投影对比如图5.6。从图中结

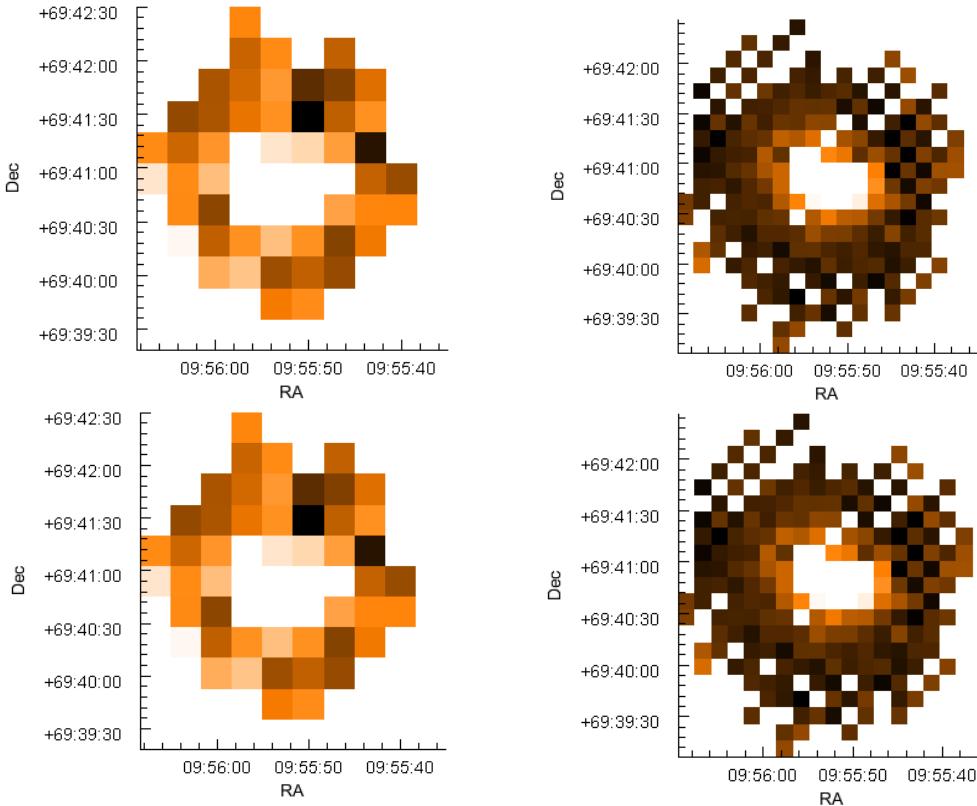


图 5.6: CPU与GPU处理SPIRE光谱数据流水线结果对比。左上为CPU处理的SLW光谱cube投影，右上为GPU处理的SLW光谱cube投影。左下为CPU处理的SSW光谱cube 投影，右下为GPU处理的SSW光谱cube投影

果看是完全相同的，当然对于光谱cube来说这还不能说明CPU和GPU处理结果完全相同。还要说明每条光谱是完全相同的，我们以cube中 $RA : 09^{\circ}55'47''$ $Dec : 69^{\circ}41'25''$ 点的光谱为例，说明低频和高频段的CPU和GPU光谱处理结果，见图5.7

从以上结果容易看出，CPU和GPU的结果是相同的，所以用GPU处理的结果是可以得到保证的。从加速效果看，FFT的加速效果很明显，整个脚本也有接近2倍的加速。但是如果单看FFT的加速的话，应该可以节省约1200s，但是整个脚本看只节约了1000s。还剩的300s的时间我们并没有具体标定出这个时间花费的原因。不过我们在计算FFT和CuFFT的时间效率的时候是直接对比他们一次（或者连续n次）的时间花费。但是我们知道一个脚本在运行过程中调用FFT是分散的，穿插在各中指令中间。所以我们猜测这个时间花费可能好在CPU和GPU中的通讯上。每次调用GPU，CPU要发出指令看GPU是否空闲，如果空闲才能使用GPU，如果不是CPU就会等待GPU。所以这个实验上的200s差异，很可能花费

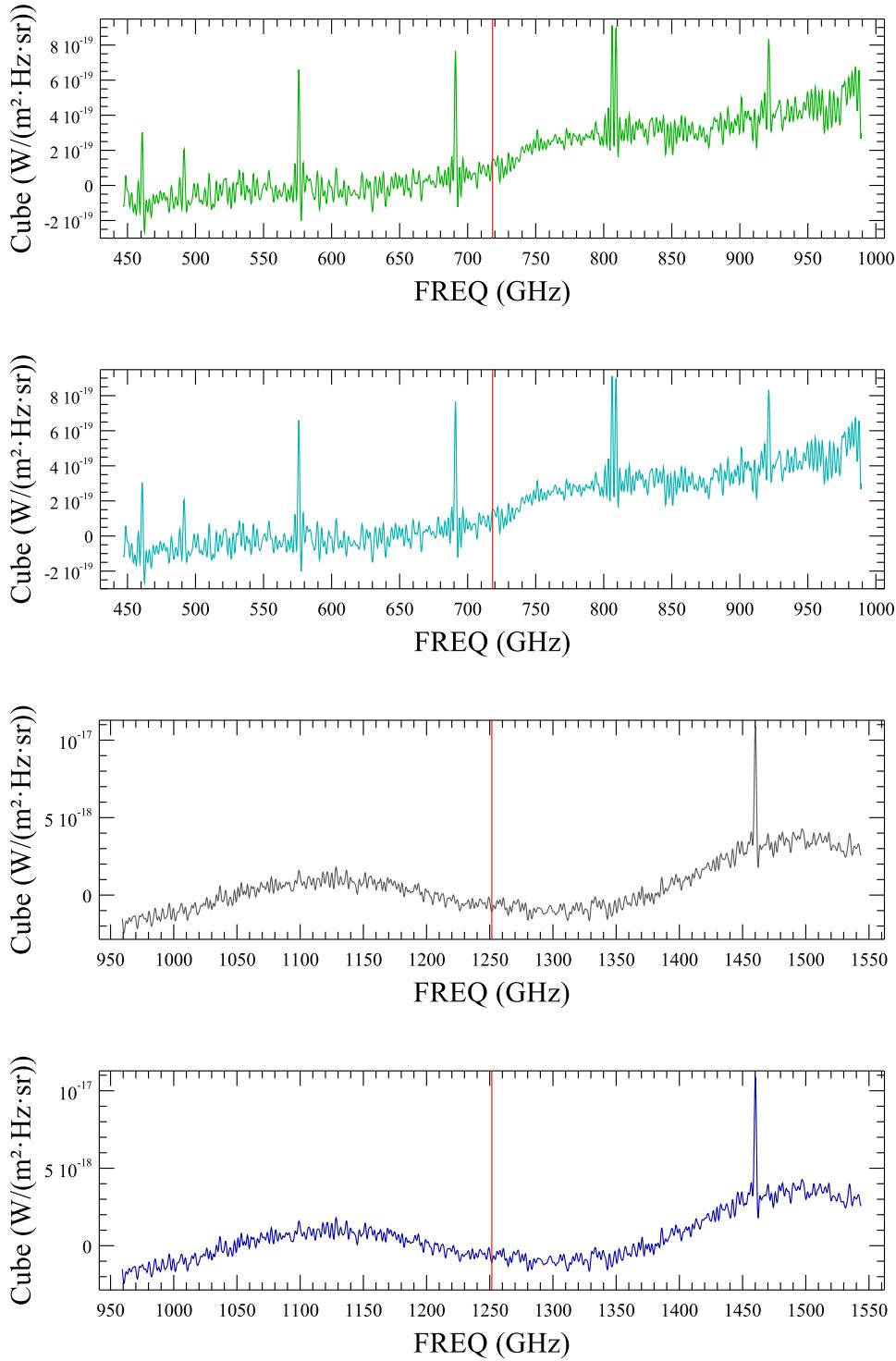


图 5.7: CPU与GPU处理SPIRE光谱数据流水线结果对比。观测点坐标为：
 $RA : 09^{\circ}55'47''$ $Dec : 69^{\circ}41'25''$ 。从上到下依次是CPU SLW光谱，GPU SLW光谱，CPU SSW光谱，GPU SSW光谱。

在进程等待GPU资源释放上。

实验结果一方面说明了GPU有很好的加速效果，另一方面也说明GPU的实际加速效果可能略次于理论加速效果，这可能是因为实际调用时资源分配需要一定时间的等待。具体应用能否加速，还要依靠实验结果。

5.3 分析和加速SPIRE 光谱处理流水线小结

首先从5.2.2节，我们可以看到SPIRE光谱处理流水线具备许多可以利用GPU加速的特点，如：

- 目标类的时间花费占到整个脚本时间花费的近70%。这给我们用 GPU 加速脚本提供了基础保证。如果目标类的时间花费所占比例很小的话，即使可以用 GPU 加速目标类对整个脚本的效率提高也意义不大。
- SPIRE 的主要时间花费不仅集中在目标类上，准确的说是主要集中在FFT上，这对编写 GPU 代码提供了极大的方便。试想如果主要花费的时间在多个基础运算中平摊分布，那么每个基础运算都需要编写相应的 GPU 代码，代码编写的工作量就会比较大。另外，平摊分布最大的危险是，在大数据规模调用中穿插了小数据规模的函数调用。我们知道，对于 GPU 而言，数据规模大时有加速效果，而数据规模小时，反而比 CPU 花费的时间还多。这样，一来一去，最后的加速效果就不明显了。所以在快速傅里叶变换上时间约60%，对最终 GPU 加速脚本提供了很大的优势。
- FFT 是比较容易用GPU加速的应用之一，另外从图5.5中可得，FFT函数调用的数据规模是40000，这时HIPE系统采用ChirpZ 方法实现傅里叶变换。图4.2可以看出，在这个数据规模下，GPU做快速傅里叶变换的运行效率是远高于CPU 的。
- SPIRE FTS 光谱处理的总时间花费比较大，这使得用 GPU 加速的意义更大，因为如果整个脚本的总时间花费很少的话，引不引入 GPU 作用也不明显。

SPIRE的光谱仪为FTS系统，即基于傅里叶变换的光谱仪，所以需要大量傅里叶变换也是理所应当的。而且这里SPIRE处理的是光谱cube，即一次给出一张图上多个点的光谱，所以运算量自然而然也比较大。

从5.2.3的加速结果看，加速效果还是不错的，简单的加速了FFT一个类后，整个脚本的运行效率提高了近一倍。而且从图5.6和图5.7看，处理结果完全正确。

第六章 分析和加速HIPE数据流水线： SPIRE和PACS扫描成像数据处理

这一章我们分析和加速另一类常见的天文数据流水线：测光成像数据处理。一般来说测光成像单个像素点的运算量远小于光谱处理，但是成像源的特点更多，根据源的特点，扫描天空区域的大小不同，仪器不同，都有不同的数据流水线脚本。所以分析起来更复杂，加速的难度也更大。这一章我们以 SPIRE 和 PACS 扫描成像流水线为例，分析和加速Herschel测光数据流水线。

6.1 SPIRE 扫描成像数据处理

这一节我们给出SPIRE扫描数据处理脚本的时间花费。SPIRE测光分为点源测光和面源测光。主要的数据处理流水线脚本有：

- Photometer_Point_Source_Pipeline.py
- Photometer_Large_Map_Pipeline.py
- Photometer_Small_Map_Pipeline.py

我们主要关心面源测光。实验结果表明SPIRE的测光成像数据处理分析不像光谱处理那样，可以比较明显的把大量的时间花费规约到较简单的几个类和方法上，我们这一节分别对小视场图和大视场图进行分析，看看数据规模的提高是不是有利于程序并行。

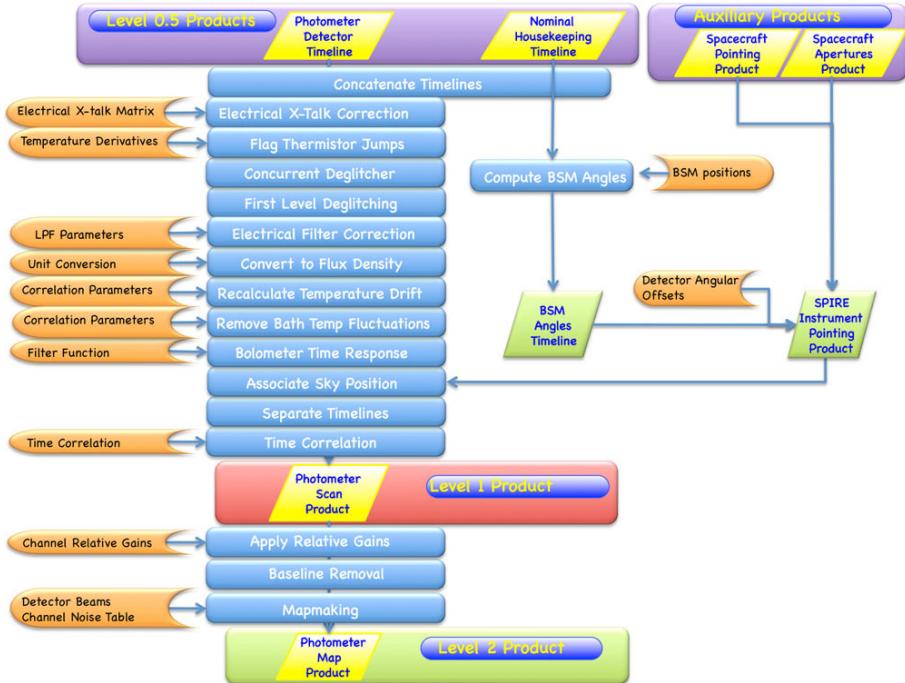


图 6.1: SPIRE测光扫描成像流程图

6.1.1 SPIRE 扫描成像流水线

SPIRE扫描测光（Photometer Scan Map）同样是处理level-0到level-2的过程。在HIPE中的脚本为：

- Photometer_Small_Map_Pipeline.py
- Photometer_Large_Map_Pipeline.py

level-0到level0.5的处理流程和SPIRE光谱流水线的相同。见图5.2。

level-0.5到level2.0的主要处理流程^[17] 包括：光束转向镜时间线向角度转化，创建SPIRE点源数据产品（product），并发去毛刺，电气串扰校正，信号跳变检测，小波变换去毛刺，电气滤波器响应校正，流量定标，重新计算冷却器温度漂移，温度漂移校正，热耦合探测器时间校正，光学串扰校正，天空坐标关联，时间校正，相关增益校正，基线漂移去除，生成图像等部分。基本的流程图可见图6.1^[17]。

(a) GRB110422		(b) M82	
观测ID	1342220542	观测ID	1342185537
观测设备	SPIRE	观测设备	SPIRE
观测模式	SpirePhotoSmallScan	观测模式	SpirePhotoLargeScan
赤经	112°02'08.42"	赤经	148°58'31.81"
赤纬	75°06'10.34"	赤纬	69°40'27.26"
目标源	GRB110422A	目标源	m82
观测周期	1135.0s	观测周期	2418.0s
观测者	mhuang01	观测者	cwilso01

表 6.1: SPIRE 测光数据观测参数——左: GRB110422A, 右: M82

6.1.2 SPIRE 扫描成像流水线分析

本节使用的小图数据为: γ 爆GRB110422A。使用的大图数据是: M82星系的SPIRE测光数据。两个源的观测参数可见表6.1。GRB110422的视场约 $0.3^\circ \times 0.3^\circ$ 。M82的视场约为 $0.7^\circ \times 0.7^\circ$ 。

γ 爆(Gamma-Ray Burst)是在遥远星系中观测到的极高能得 γ 射线爆发现象。这种爆发的时间尺度在10毫秒到几分钟不等。在最初的强射线爆发后, γ 爆还伴随着更长寿命的“余辉”, “余辉”的波段更高, 频段也很宽, 跨越X射线, 紫外线, 光学, 红外, 微波和射电波段。目前已经观测到的 γ 爆大部分发生在距地球十亿光年以上的遥远星系, 这意味着 γ 爆的能量极高并且发生几率很小。典型的 γ 爆几秒钟爆发的能量相当于太阳一生辐射的能量, 而它发生的频率大约是每个星系每百万年一次。本节使用的数据源GRB110422A是极高亮度的持续时间较长的 γ 射线爆, 也是观测远红外波段 γ 爆“余辉”的第一次尝试。

m82星系所用数据与上节不同, 是SPIRE的扫描测光数据。

这两个数据流水线的处理结果为level-2.0的科学数据。包括三个Fits文件。分别是“短波”(PSW, Photometer Short WaveLength $250\mu m$)、“中波”(PMW, Photometer Middle WaveLength $350\mu m$)、“长波”(PLW, Photometer Long WaveLength $500\mu m$)。这里列出两个脚本的PSW如图6.2

与上一节类似, 我们可以得到这两个数据处理流水线的时间花费情况, 因为各个部分时间花费都较少, 为了使得显示结果更明显, 我们将Double1d(or Complex1d) 的加减乘除四则运算时间合并¹, 统一用“Arithmetic”表示。实验结

¹ 这是合理的, 因为我们的目的是统计基类的时间花费, 并对花费大的用GPU进行加速, 而加减乘除四

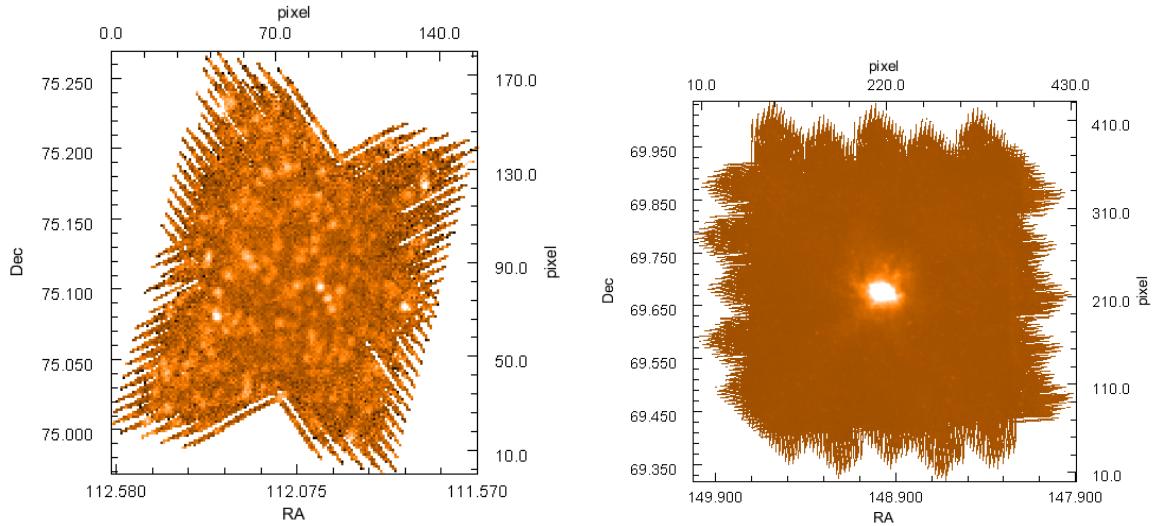


图 6.2: SPIRE 扫描成像流水线处理结果。左: GRB110422A, 右M82。

果见表6.2，在图6.3中我们更清楚的表明了它们的这种差异。

对于小图来说，我们考察的主要函数所占时间并不占绝对优势，还有大量的时间没有被分析出来。这可能是有些运算分散在数据处理流水线的其他部分，没有被归约到最基本的运算上，所以我们不能统计出来。但是从大图的情况看，效果稍微好一些，虽然还有一部分未知，但主要的占用时间已经被分析出来。所以更可能存在的原因是无论是小图还是大图，都需要占用一些非计算花费时间，如调用I/O，生成图像等。这些操作的时间花费比较固定，基本不随数据规模的增长而增大。从图6.3中也可以看出，大图相对于小图，整个脚本的处理时间增加超过200s，但是除了这些基础类以外其他部分的时间增加不足70s。

另外为了说明大图时间花费的增加主要来自基础计算的时间代价的增加，我们可以提取小图运算和大图运算的数据规模做对比。作为例子，我们比较一下两图中FFT和Pow调用次数和每次调用数据规模的差异。在小图中FFT共调8929次，pow调用了297480次。而在大图中这个次数扩大了近1.5倍，分别为11160 次和524403 次。除了调用次数增加外，调用的数据规模也扩大了。图6.4 中列出了两图FFT 和Pow 运算的数据规模。可以看出大图调用相同函数是数据规模通常也比较大。通常来说，并行度正比于数据规模的大小。所以视场越大的图，越容易并行。

则运算在GPU上加速的模式相近，且运行时间差别不大

根据已上分析，小图来说数据计算部分占总的处理时间的百分比小于大图中数据计算占的百分比。所以一般来说视场越大，数据量越大，数据处理中计算的部分的时间花费会越多。也就是说数据量越大，计算的时间花费越多，越适合采用并行。这这结果与实际需要也正好吻合，因为数据量越大，总的花费时间越大，我们越需要并行处理来减小时间花费，而数据量小的时候时间花费本来就不多，即使不用并行加速，也可以在较短的时间内运行完。

当然，我们结合上一节介绍的光谱处理，还需要注意，无论是小图还是大图，测光处理的时间都是远远小于光谱处理的时间的。而且傅里叶变换的时间花费已经不占主导地位，而幂运算的时间花费开始凸显，所以要想加速测光数据处理脚本，不能简单的从傅里叶变换下手，而且要考虑更基本的一些运算。虽然计算部分的时间花费还占到约 $1/2$ ，但是计算分散，大部分运算加速的效率并不高，而且很多情况并不好直接加速，要想加速需要修改的类变多，编程代价也更高。

6.1.3 GPU对SPIRE扫描成像流水线的加速效果

从已上分析结果可以看出，SPIRE扫描成像流水线需要修改多个基类才能最终实现较好的加速效果。应该说我们这一节的加速版本并不完善，我们只修改了快速傅里叶变换和矩阵乘法两类，但是不是修改了所有的基类函数加速效果就一定会好呢，也不尽然，我们可以看采用快速傅里叶变换和矩阵乘法的GPU 实现和单单只有FFT用GPU实现的加速效果，具体的加速结果见表6.3和表6.3。显然简单使用FFT GPU加速的脚本运行效率更高，而同时使用 FFT 和 MatrixMultiply, GPU加速的脚本运行效率反而慢了。

这一结果提醒我们，能用GPU加速的情况是比较局限的，一般只有数据规模大，并行度高，I/O操作少的操作才能用GPU加速。我们虽然分析除了SPIRE扫描成像流水线的主要时间花费，把时间规约到几个简单的基类上，但是绝大部分基类不容易加速，即使可以用GPU实现其功能，但是是否有加速效果还需要根据调用函数的数据规模确定。

随后需要一提的是，我们讨论的效率问题，无论CPU还是GPU，程序的正确性都是可以得到保证的。例如我们用GPU的处理结果见图6.5，和图6.2对比，是相同的。

进一步我们可以将图6.5和图6.2做差，得到残差图，统计残差分布。具体结果

GRB110422A 测光处理时间总花费 (CPU) :162.15s					
目标类	目标函数	时间花费	总时间	百分比	总百分比
Complex1d	Arithmetic	1.71s	23.14s	1.05%	14.25%
	Pow	21.43s		13.2%	
	Abs	0.0s		0.0%	
Double1d	Arithmetic	0.924s	31.77	0.6%	19.7%
	Pow	29.78s		18.4%	
	Abs	0.28s		0.2%	
	Dot	0.786s		0.5%	
xform	FFT	19.22s	37.45s	11.9%	23.01%
	IFFT	18.23s		11.2%	
matrix	MatrixMultiply	0.032s	2.48s	< 1%	1.5%
	SVD	2.45s		1.5%	

M82 测光处理时间总花费 (CPU) :367.86s					
目标类	目标函数	时间花费	总时间	百分比	总百分比
Complex1d	Arithmetic	2.18s	35.712s	0.59%	9.69%
	Pow	33.50s		9.1%	
	Abs	0.032s		< 1%	
Double1d	Arithmetic	2.29s	141.88s	0.62%	38.6%
	Pow	136.11s		37.00%	
	Dot	3.47s		0.94%	
	Abs	0.032s		< 1%	
xform	FFT	26.96s	52.93s	7.32%	14.37%
	IFFT	25.97s		7.05%	
matrix	MatrixMultiply	0.109s	0.48s	< 1%	2.92%
	SVD	10.77s		2.92%	

- i 此外还有一些method统计后发现没有被调用，如Double2d等，数据为零，未列于表内。
ii Arithmetic 是指所有加减乘除四则运算。

表 6.2: SPIRE 扫描成像数据处理时间花费分析

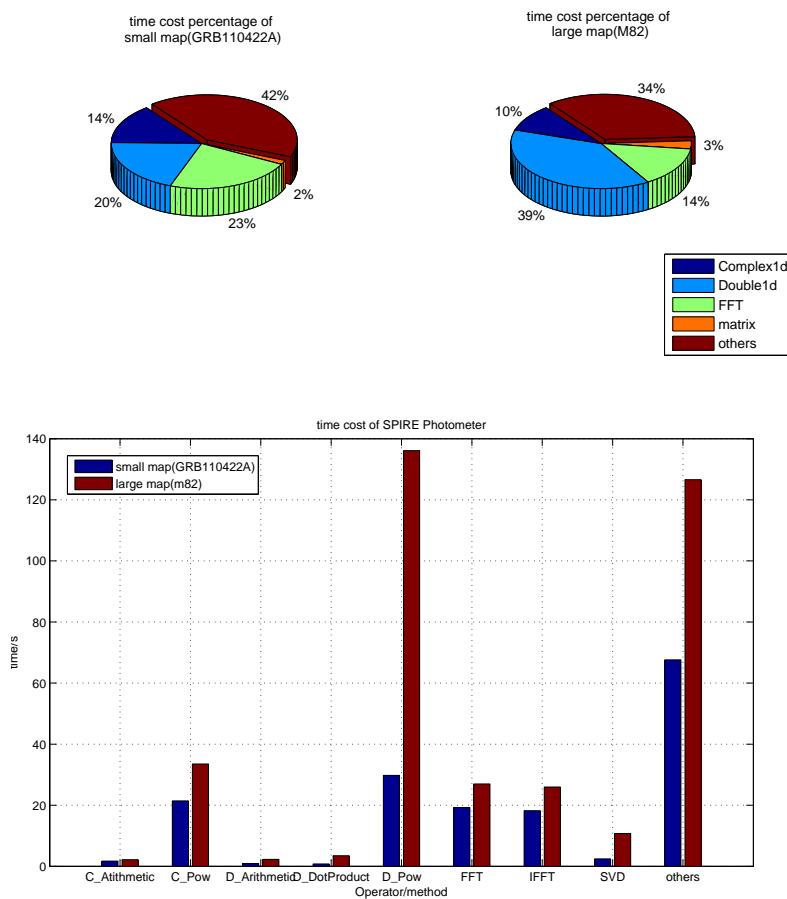


图 6.3: SPIRE测光处理脚本时间分析。上: 主要函数用时百分比。下: 主要函数时间花费

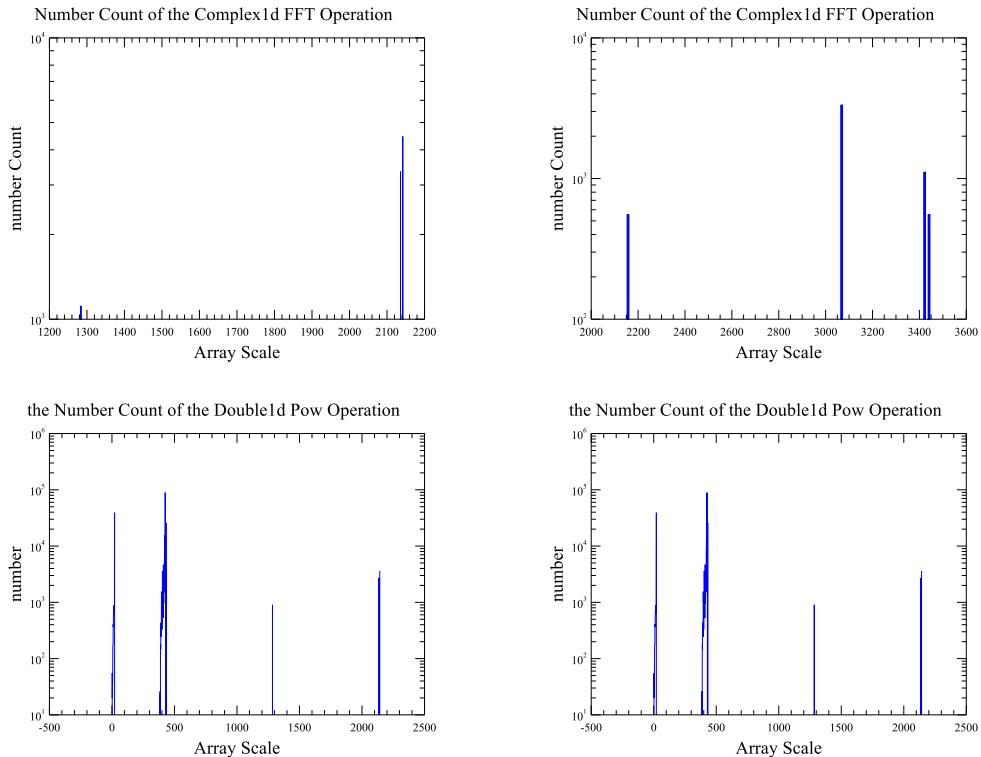


图 6.4: SPIRE测光处理，大图和小图相同函数调用数据规模对比。上：快速傅里叶变换（fft）。下：幂运算（power）。左侧为小图数据，右侧为大图数据

SPIRE small map流水线			
	CPU	CPU&GPU	加速比例
FFT/IFFT	37.45s	15.40s	2.43x
MatMul	0.032s	60.29	5.3×10^{-4} x
Small Map	162.15s	224.53	0.722x

SPIRE large map流水线			
	CPU	CPU&GPU	加速比例
FFT/IFFT	54.93s	23.24s	2.36x
MatMul	0.48s	76.83s	6.2×10^{-3} x
Large Map	367.86s	454.54s	0.8x

表 6.3: GPU 加速 SPIRE 扫描成像结果。用 GPU 替换的类包括 FFT 和 Matrix-Multiply

SPIRE small map流水线			
	CPU	CPU&GPU	加速比例
FFT/IFFT	37.45s	15.71s	2.38x
Small Map	162.15s	152.66s	1.06x

SPIRE large map流水线			
	CPU	CPU&GPU	加速比例
FFT/IFFT	52.93s	22.70s	2.33x
Large Map	367.89s	334.76s	1.09x

表 6.4: GPU 加速 SPIRE 扫描成像结果。用 GPU 替换的类仅包括 FFT

见图6.6和6.7。分别对应GRB110422A和m82的 CPU、GPU 运算差异分布。

6.2 PACS 扫描成像数据处理

PACS 和 SPIRE 一样，可以做测光成像和光谱，但 PACS 的数据流水线与 SPIRE 的处理流程并不完全相同，甚至调用的上层类都是分别设计的，关联度不大。但是我们设计的方法是从底层基类分析，是对一切脚本普遍适用的。本文作为例子，我们对 PACS 测光进行分析。

6.2.1 PACS 扫描成像数据流水线

PACS 扫描成像数据流水线的处理流程可见图6.8。 [21]

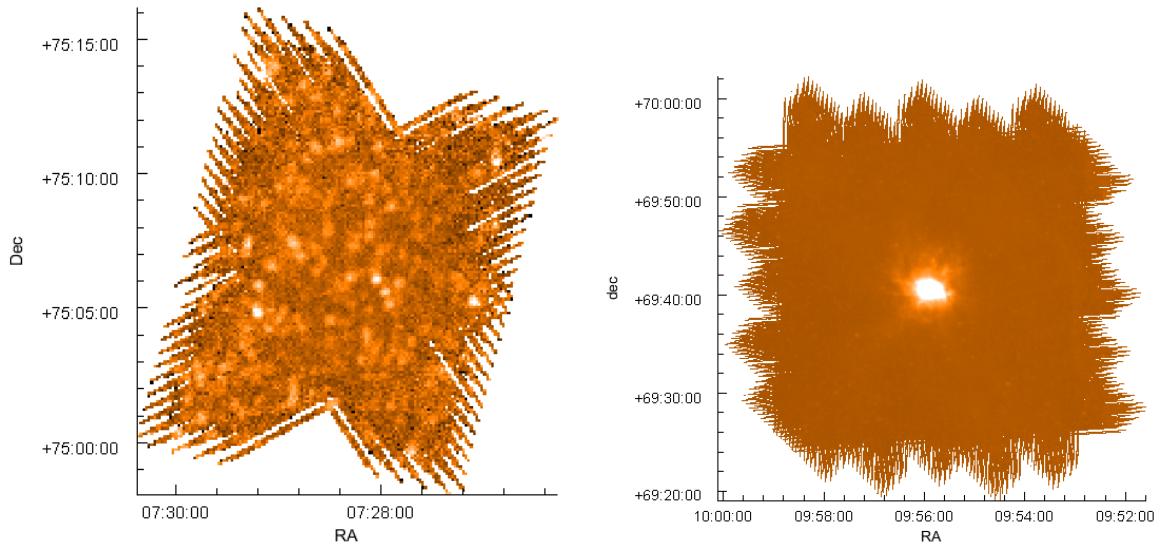


图 6.5: SPIRE 扫描成像流水线的GPU处理结果(PSW)。左: GRB110422A, 右M82。

6.2.2 PACS 成像流水线分析

我们采用的观测源仍是 γ 爆GRB110422A。上一节的 SPIRE 数据观测与2011年5月3日，而本节的数据观测于2011年4月30日。观测设备是 PACS，观测波段波长更短。具体观测参数可见表6.5。我们使用的 HIPE 数据处理脚本为：

- `scanmap_Deep_Survey_miniscan_Pointsource.py`

脚本的运行结果见图6.9左图。

类似前两节的分析。我们可以给出这个脚本中各个目标类的调用次数和时间花费。见表6.6。从表中可以看出来PACS的测光数据处理和SPIRE的测光数据处理有很大区别。如果只通过设置 `Double1d`, `Double2d`, `Complex1d`, `xform`, `matrix` 等类为目标类进行统计，基本没有找到PACS 测光处理的耗时所在。对于这种情况，我们有以下可能的猜测：

- PACS测光数据处理不能规约到几个简单的基类。可能PACS的数据流水线在设计的时候并没有想到利用基类运算，而是把运算单元重新写了代码，而导致运算分散。不容易被我们标记的基类捕捉。
- PACS测光的数据处理脚本本省时间花费就很少，基本运算很难体现出来。
- PACS测光的数据流水线主要部分并没有用到大量运算，而时间主要花费在如I/O处理，图像生成处理，时间线转化等部分。

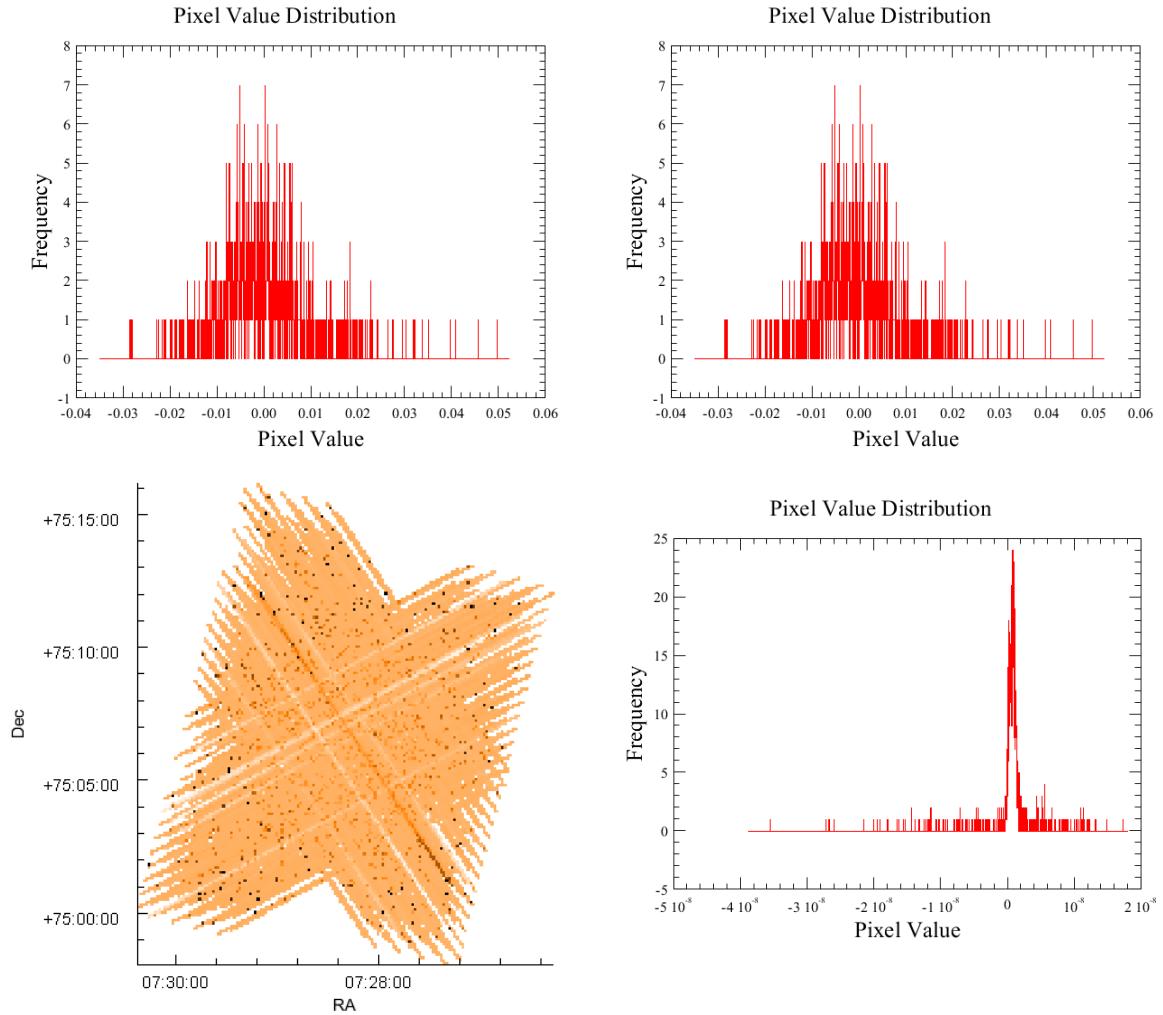


图 6.6: SPIRE 扫描成像流水线的CPU和GPU处理结果对比。左上: GRB110422A CPU处理结果分布, 右上: GRB110422A GPU处理结果分布。左下: GRB110422A 残差图。右下: GRB110422A 残差分布。残差控制在 10^{-9} 量级, 相对误差在 10^{-7}

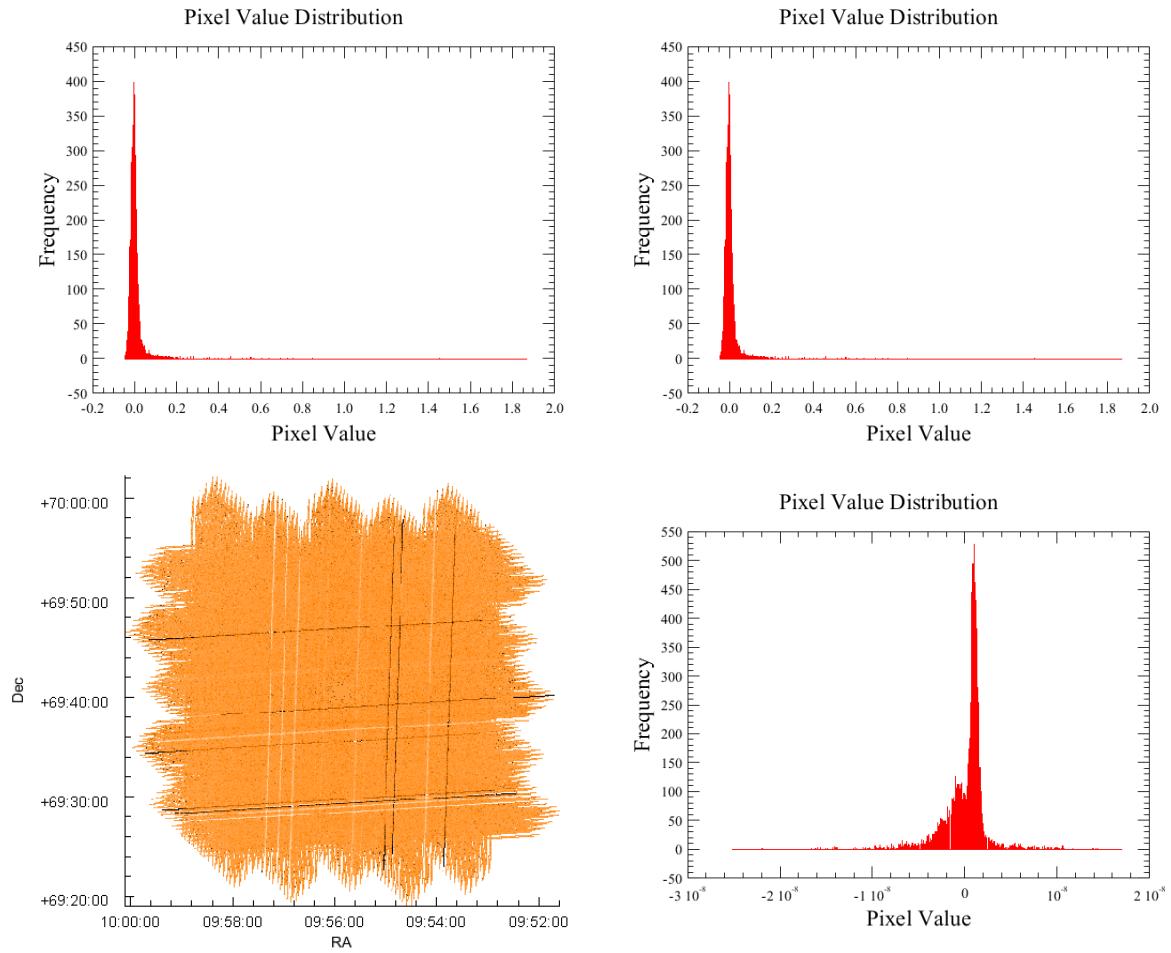


图 6.7: SPIRE 扫描成像流水线的CPU和GPU处理结果对比。左上: M82 CPU 处理结果分布, 右上: M82 GPU 处理结果分布。左下: M82 残差图。右下: M82 残差分布。残差控制在 10^{-9} 量级, 相对误差在 10^{-7}

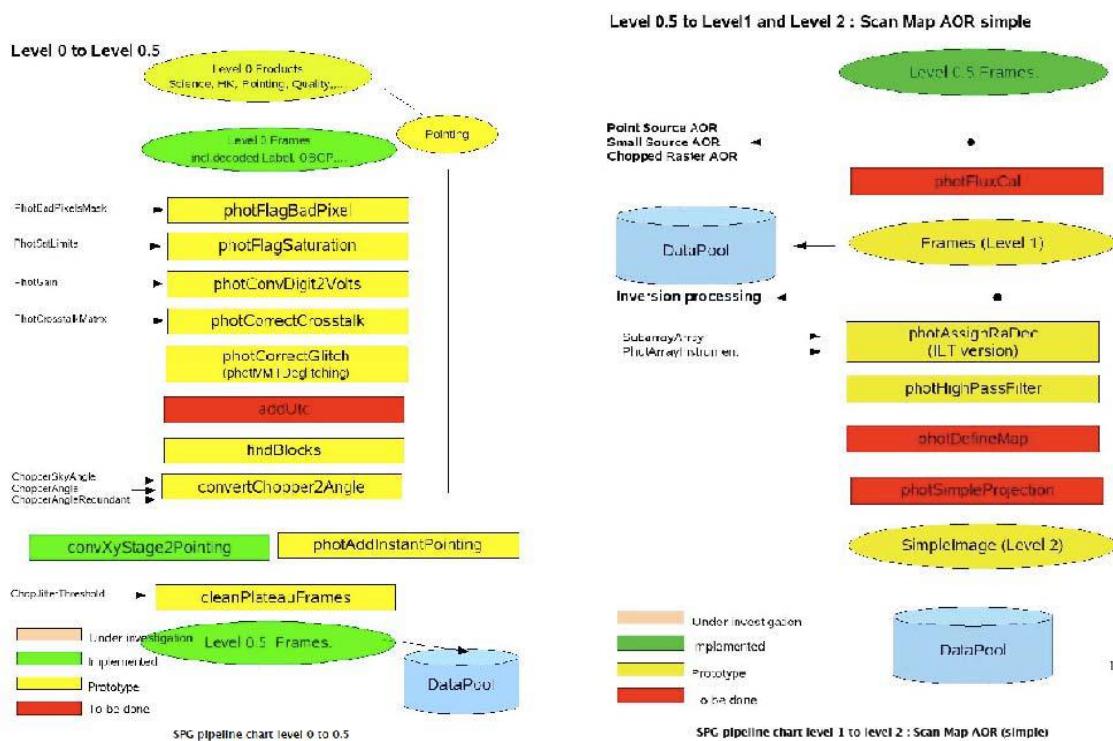


图 6.8: PACS成像数据流水线。左: level-0到level-0.5, 右:level-0.5到level-2.0。

观测ID	1342220914
观测设备	PACS
观测模式	PacsPhoto
赤经	112°02'49.70"
赤纬	75°06'25.93"
目标源	GRB110422A
观测起始时间	2011-4-30 13:59:28.0
观测结束时间	2011-4-30 14:14:26.0
观测周期	898.0s
观测者	mhuang01

表 6.5: Pacs 测光观测源GRB110422A的观测参数

- 并不是PACS的运算不能规约到几个简单的基类中，而是PACS的主要耗时基类并不在我们标记的目标类里。

对于上面的猜想，我们认为第1点和第4点的可能性比较大。一个数据处理流水线没有大量计算应该不太可能。而我们的统计中，除了Double1d, Double2d 中设计一些运算，其他在数据处理中很必要的操作都没有被调用，究竟如何去分析PACS的脚本还有待进一步的研究。

6.2.3 PACS 成像数据流水线GPU加速效果

根据上面的分析可以看出PACS成像数据流水线基本没有调用基类函数，我们将 FFT 和 MatrixMultiply 用 GPU 实现应该不影响整个脚本的时间运行。结果见表6.7，同样运行结果的差别应该在误差以内。见图6.9。

6.3 分析和加速 SPIRE 和 PACS 扫描成像流水线 小结

6.3.1 SPIRE 扫描成像小结

从6.1.3节中可以看出 SPIRE扫描成像的加速效果并不好，提高不到10%。这里浅析一下 SPIRE 脚本加速的难度所在。

- 无论大图和小图，虽然目标类的总时间花费所占比例不小，接近或超过60%。这给我们加速脚本提供了基础。但是这些时间花费在各个目标类中

GRB110422A PACS测光成像数据处理 (CPU) :121.19s						
目标类	目标函数	计数	时间花费	总时间	百分比	总百分比
Complex1d	ALL	0	0.s	0.0s	0.0%	0.0%
Double1d	Add	17640	0.0025s	0.0293s	< 1‰	< 1‰
	Sub	26462	0.016s		< 1‰	
	Mul	26462	0.0108s		< 1‰	
	others	0	0.0s		0.0%	
Double2d	Add	16	≈ 0	≈ 0	< 1‰	< 1‰
	Div	16	≈ 0		< 1‰	
	others	0	0.0s		0.0%	
xform	ALL	0	0.0s	0.0s	0.0%	0.0%
matrix	ALL	0	0.0s	0.0s	0.0%	0.0%

i 表中一些参数说明：

(1) Double1d中others表示出了列出的目标函数外的其他函数，包括Div, Dotproduct, Abs, Pow等。

(2) xform中的ALL包括fft和ifft两个目标函数。

(3) matrix中的ALL包括矩阵乘法和SVD分解两个函数。

ii 这里一些方法，如Add, Sub等根据输入参数不同，实现了函数重载，在统计时间时已经考虑了这个因素，而分别统计计算加和。所以Double1d中sub的时间比mul时间略多，因为sub 里有8822次调用的参数为double，而其余17640次调用参数为Double1d

表 6.6: GRB110422A 测光成像数据处理时间花费分析

PACS photometer 流水线			
	CPU	CPU&GPU	加速比例
FFT/IFFT	0.0s	0.0s	null
MatMul	0.0s	0.0s	null
Small Map	121.19s	128.34s	0.94x

表 6.7: GPU 加速 PACS 扫描成像结果。用 GPU 替换的类包括 FFT 和 Matrix-Multiply

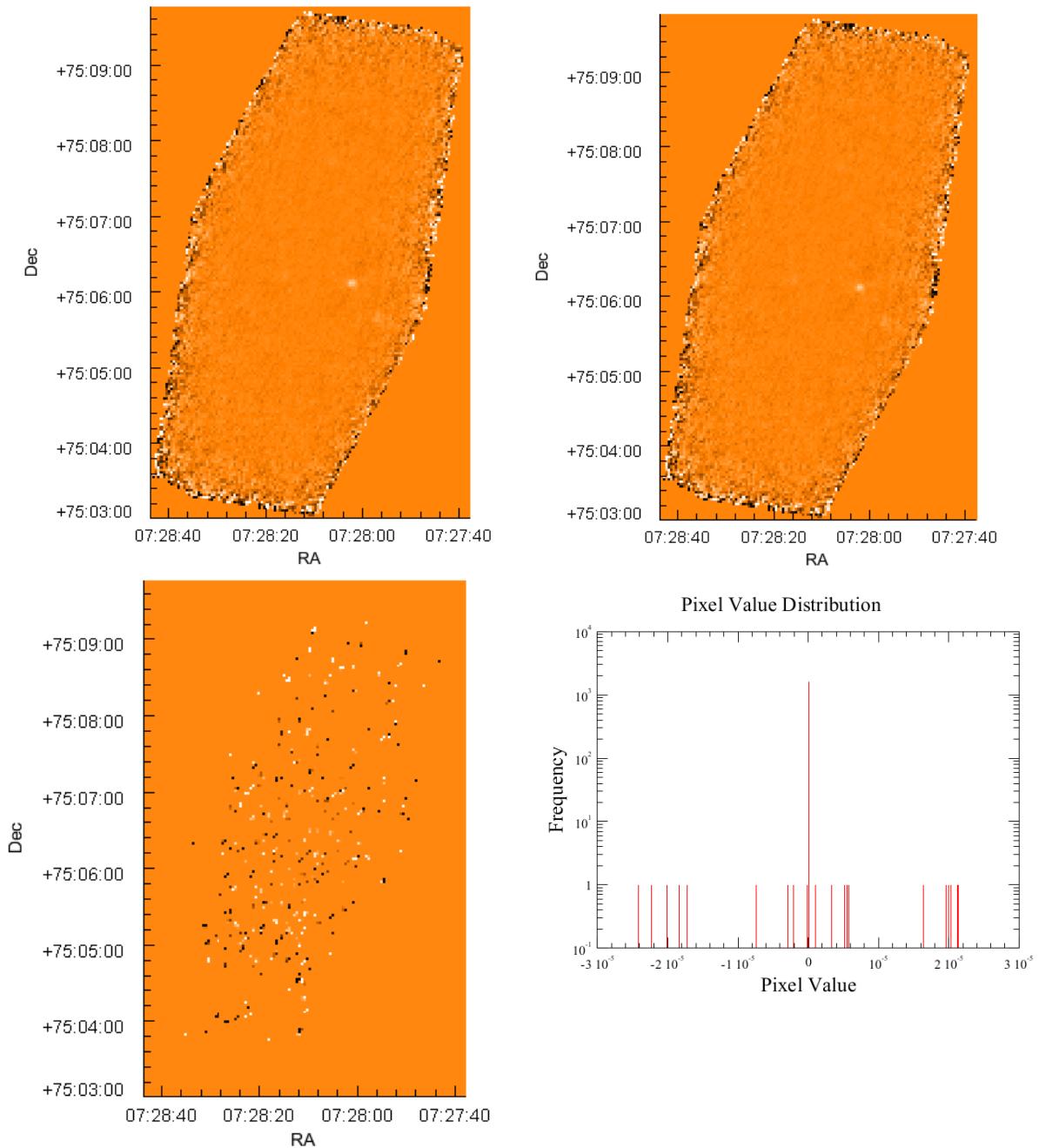


图 6.9: PACS 扫描成像流水线处理结果，数据源为GRB110422A，左上：CPU 处理结果，右上：GPU 处理结果。左下为残差图。右下为残差分布。实验表明即使两次都用CPU计算仍会有此残差出现。这也与理论相符，理论上这个脚本并没有调用所修改的 GPU 函数，所以出现残差也应与 GPU 无关

分布分散，这对GPU并行带来了一定的困难，因为我们要对整个基类做较大的改动，对大量的基类运算方法引入GPU模式。但是目前并没有实现对每个目标类的GPU 加速，所以实际能加速的部分还远不到60%。

- 对于能加速的部分，函数调用规模参差不齐，有小数据规模，也有大数据规模。一般来说小数据规模CPU 的时间花费会比较小，而大数据规模，GPU的效率优势才会体现出来。而我们目前的办法还比较简单，在系统运行前，就指定好了这些应用到底用CPU还是GPU，这使得GPU在大数据规模上节省的时间很大程度都耗费在小数据规模上了，例如MatrixMultiply 在GPU反而比CPU花费了大量的时间。
- 运算量大的类花费时间并不大。我们知道，对于GPU来说，传输数据要花费大量时间，所以计算复杂性高的类，数据传输在整个时间花费中占的比例较小，而目标类中运算复杂性比较高的类，如快速傅里叶变换和矩阵乘法的时间花费并不高，而且随着数据量的增大（大图相对于小图），这两个类的时间花费并没有明显增加。

目前我们对 SPIRE 的加速效果还仅有10s的量级，不到整个脚本的10%。加速 SPIRE 扫描成像脚本并非易事。接下来工作的方向一是想办法实现所有基础类的GPU 加速，另一方面是去实现CPU 和GPU对不同数据规模的自主调用，让程序根据数据规模的大小，自动选择硬件接口。

6.3.2 PACS 扫描成像小结

PACS的光度成像处理有点出乎意料，因为本工作的初衷是希望找到一个普适的方法去分析HIPE中的一切脚本，但是，显然只通过 xform, matrix, Double1d 等目标类的标记，基本上没有对PACS的脚本做出很好的分析。我们猜测 PACS 脚本设计时没有充分利用基类而导致运算分散²等。如果是这样，那么想通过加速基础运算类来加速 PACS 扫描成像流水线基本是不可行的了。但是也可能PACS是没有调用我们标记的目标类，但是它的运算部分被集中在另一些主要的上层类上，这样的话想加速PACS 扫描成像脚本可能可以从更高级的一些函数下手。对这些猜想的最终验证工作还没有完成，这也是今后工作的一个方向。

² 这是有理由相信的，因为HIPE 中有些数学运算也直接调用了第三方的软件，而没有设计自己的基类，比方说第三方的矩阵运算库jama

第七章 总结与展望

7.1 总结

本文通过对Herschel数据处理系统的研究，分析了GPU引入HCSS的方法途径、可行性、适用条件和最终效果。

首先本文设计了一套可行的方法，通过对HCSS基础框架的研究，抽象出几个可能的基类为目标类，把这些基类中的主要计算函数设置为目标函数。通过对每个目标类设计一个计数辅助类和计时辅助类的办法，得出每个目标类在整个脚本中被调用的次数，及每次调用时的数据规模，并最终通过计时辅助类得到每个目标类，每个目标函数在整个脚本运行时的时间花费和占整个脚本时间花费的百分比。通过对目标类的研究，我们最终可以在复杂的脚本中抽象出较为简单的且时间花费大的一些基类，通过对这些基类的加速而实现整个脚本的加速。作为例子，本文分析了SPIRE光谱数据处理流水线，SPIRE光度成像数据处理流水线和PACS测光数据处理流水线。分析了这些脚本中目标基类的时间花费。

其次，由于HCSS是纯java的系统，目前对支持GPU的主流平台是CUDA，但是目前还没有可以直接支持CUDA的java编译器，这使得想利用GPU提高HCSS系统性能，产生了一定难度。本文调研了将GPU引入HCSS的方法，并最终通过JNI技术和调用JCUDA库的方式，实现HCSS系统对GPU的调用。作为例子，本文给出了HCSS系统中快速傅里叶变换和矩阵乘法的GPU实现，并对比了GPU实现和CPU实现中的效率差别。对于较小数据规模，CPU仍占优势，但是随着数据规模增大，GPU优势会迅速凸显。

最后，本文结合上两步的方法，针对最适合GPU加速的SPIRE光谱数据处理流水线（Spectrometer Mapping Pipeline）做了GPU加速处理，最终实现加速类17倍的加速和整个SPIRE光谱处理脚本近2倍的加速。说明GPU对这些计算密

集型的脚本还是有较好的加速作用的。

7.2 本工作对HCSS系统设计的指导意义

本文工作对今后HCSS脚本编写具有一定的指导意义。脚本分析中发现，虽然HCSS设计了一到五维的数据类型，但是只有一维数据类被频繁使用，二维及更高维的数据基本没有被使用。这与Herschel原始数据以时间线的方式存储有关。这种情况对串行系统的效率影响不大，但是对于脚本利用GPU加速的效果影响却是巨大的。我们知道二维数据运算，每个数据点与这个二维数组中的其他元素交互更自然，更方便。这时运算更容易被组织成线程。而且从GPU编程角度看，矩阵形式将比嵌套多层的循环语句更自然。而且一次数据传输能完成的任务更多，提高了运算量与数据传输量的比例，GPU加速效果会更好。

另外，实验通过PACS测光处理等脚本的分析表明，很多HCSS脚本的计算部分较为分散，调用了很多java的基础类，而不是从HCSS定义的基类出发。我们对代码的修改加速层级最底层就是HCSS的基类。修改java编译器提供的基类，一是极不方便，二是会使系统不稳定。所以，对于运算复杂，时间花费大的HCSS脚本编程应该从HCSS基类出发。这样比较方便GPU加速。

Herschel系统基本趋于成型，虽然目前还在不断完善中，但是基本构架已经不会改变。但是科学是不断前进和完善的，本工作的另一个意义是对今后的望远镜软件系统的编写提供一点借鉴，在设计初期就考虑到使用GPU的可能，而留出更多的方便GPU引入的系统接口。

7.3 展望

当然，限于作者的能力水平，本文工作还有一些欠缺不足。首先对于那些计算密集，但是分散在较多的基类的脚本，需要修改较多的基类函数，以分别引入GPU加速，本工作虽涉及并设计了一些类的GPU 实现，但并没有完全实现所有基类函数的GPU调用，这是今后工作可以开发的一个地方，这也是实现像SPIRE光谱处理流水线这种脚本进一步加速的有效途径。另外对脚本的分析还不够彻底，PACS 测光脚本的分析中发现其没有调用很多次基类，对于这种情况

如何利用GPU 加速，本文并未给出一个可行的解决方案。最后本文工作还可以拓展到HIFI 数据处理脚本上，HIFI作为一个纯测光谱的后端设备，其运算量更大，时间花费也更多，更需要利用GPU实现加速。

参考文献

- [1] C. P. Ahn, R. Alexandroff, C. Allende Prieto, S. F. Anderson, T. Anderton, B. H. Andrews, É. Aubourg, S. Bailey, E. Balbinot, R. Barnes, et al. The Ninth Data Release of the Sloan Digital Sky Survey: First Spectroscopic Data from the SDSS-III Baryon Oscillation Spectroscopic Survey[J]. *ApJS*. Dec. 2012, **203**:21. [1207.7137](#)
- [2] D. R. Silva, G. Angeli, C. Boyer, M. Sirota, T. Trinh. Thirty Meter Telescope: observatory software requirements, architecture, and preliminary implementation strategies[C]. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series. 2008, vol. 7019 of Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series
- [3] R. Yokota, L. A. Barba. Hierarchical N-body simulations with auto-tuning for heterogeneous systems[J]. ArXiv e-prints. Aug. 2011. [1108.5815](#)
- [4] D. Bard, M. Bellis, M.T. Allen, H. Yeremyan, J.M. Kratochvil. Cosmological calculations on the {GPU}[J]. *Astronomy and Computing*. 2013, **1**(0):17 – 22
- [5] A. S. Lau. The Narcissus effect in infrared optical scanning systems[C]. J. D. Lytle, H. Morrow, (Editors) Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series. 1977, vol. 107 of Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, 57–62
- [6] G. L. Pilbratt, J. R. Riedinger, T. Passvogel, G. Crone, D. Doyle, U. Gageur, A. M. Heras, C. Jewell, L. Metcalfe, S. Ott, M. Schmidt. Herschel Space Observatory. An ESA facility for far-infrared and submillimetre astronomy[J]. *A&A*. Jul. 2010, **518**:L1. [1005.5331](#)

- [7] A. Poglitsch, C. Waelkens, N. Geis, H. Feuchtgruber, B. Vandenbussche, L. Rodriguez, O. Krause, E. Renotte, C. van Hoof, P. Saraceno, J. Cepa, F. Kerschbaum, P. Agnèse, B. Ali, B. Altieri, P. Andreani, J.-L. Augueres, Z. Balog, L. Barl, O. H. Bauer, N. Belbachir, M. Benedettini, N. Billot, O. Boulade, H. Bischof, J. Blommaert, E. Callut, C. Cara, R. Cerulli, D. Cesarsky, A. Contursi, Y. Creten, W. De Meester, V. Doublier, E. Doumayrou, L. Duband, K. Exter, R. Genzel, J.-M. Gillis, U. Grözinger, T. Henning, J. Herreros, R. Huygen, M. Inguscio, G. Jakob, C. Jamar, C. Jean, J. de Jong, R. Katterloher, C. Kiss, U. Klaas, D. Lemke, D. Lutz, S. Madden, B. Marquet, J. Martignac, A. Mazy, P. Merken, F. Montfort, L. Morbidelli, T. Müller, M. Nielbock, K. Okumura, R. Orfei, R. Ottensamer, S. Pezzuto, P. Popesso, J. Putzeys, S. Regibo, V. Reveret, P. Royer, M. Sauvage, J. Schreiber, J. Stegmaier, D. Schmitt, J. Schubert, E. Sturm, M. Thiel, G. Tofani, R. Vavrek, M. Wetzstein, E. Wieprecht, E. Wiezorrek. The Photodetector Array Camera and Spectrometer (PACS) on the Herschel Space Observatory[J]. *A&A*. Jul. 2010, **518**:L2. [1005.1487](#)
- [8] M. J. Griffin, A. Abergel, A. Abreu, P. A. R. Ade, P. André, J.-L. Augueres, T. Babbedge, Y. Bae, T. Baillie, J.-P. Baluteau, M. J. Barlow, G. Bendo, D. Benielli, J. J. Bock, P. Bonhomme, D. Brisbin, C. Brockley-Blatt, M. Caldwell, C. Cara, N. Castro-Rodriguez, R. Cerulli, P. Chanial, S. Chen, E. Clark, D. L. Clements, L. Clerc, J. Coker, D. Communal, L. Conversi, P. Cox, D. Crumb, C. Cunningham, F. Daly, G. R. Davis, P. de Antoni, J. Delderfield, N. Devin, A. di Giorgio, I. Diduschuns, K. Dohlen, M. Donati, A. Dowell, C. D. Dowell, L. Duband, L. Dumaye, R. J. Emery, M. Ferlet, D. Ferrand, J. Fontignie, M. Fox, A. Franceschini, M. Frerking, T. Fulton, J. Garcia, R. Gastaud, W. K. Gear, J. Glenn, A. Goizel, D. K. Griffin, T. Grundy, S. Guest, L. Guillemet, P. C. Hargrave, M. Harwit, P. Hastings, E. Hatziminaoglou, M. Herman, B. Hinde, V. Hristov, M. Huang, P. Imhof, K. J. Isaak, U. Israelsson, R. J. Ivison, D. Jennings, B. Kiernan, K. J. King, A. E. Lange, W. Latter, G. Laurent, P. Laurent, S. J. Leeks, E. Lellouch, L. Levenson, B. Li, J. Li, J. Lilenthal, T. Lim, S. J. Liu, N. Lu, S. Madden, G. Mainetti, P. Marliani, D. McKay, K. Mercier, S.

Molinari, H. Morris, H. Moseley, J. Mulder, M. Mur, D. A. Naylor, H. Nguyen, B. O'Halloran, S. Oliver, G. Olofsson, H.-G. Olofsson, R. Orfei, M. J. Page, I. Pain, P. Panuzzo, A. Papageorgiou, G. Parks, P. Parr-Burman, A. Pearce, C. Pearson, I. Pérez-Fournon, F. Pinsard, G. Pisano, J. Podosek, M. Pohlen, E. T. Polehampton, D. Pouliquen, D. Rigopoulou, D. Rizzo, I. G. Roseboom, H. Roussel, M. Rowan-Robinson, B. Rownd, P. Saraceno, M. Sauvage, R. Savage, G. Savini, E. Sawyer, C. Scharmberg, D. Schmitt, N. Schneider, B. Schulz, A. Schwartz, R. Shafer, D. L. Shupe, B. Sibthorpe, S. Sidher, A. Smith, A. J. Smith, D. Smith, L. Spencer, B. Stobie, R. Sudiwala, K. Sukhatme, C. Surace, J. A. Stevens, B. M. Swinyard, M. Trichas, T. Tourette, H. Triou, S. Tseng, C. Tucker, A. Turner, M. Vaccari, I. Valtchanov, L. Vigroux, E. Virique, G. Voellmer, H. Walker, R. Ward, T. Waskett, M. Weilert, R. Wesson, G. J. White, N. Whitehouse, C. D. Wilson, B. Winter, A. L. Woodcraft, G. S. Wright, C. K. Xu, A. Zavagno, M. Zemcov, L. Zhang, E. Zonca. The Herschel-SPIRE instrument and its in-flight performance[J]. *A&A*. Jul. 2010, **518**:L3. **1005. 5123**

- [9] T. de Graauw, F. P. Helmich, T. G. Phillips, J. Stutzki, E. Caux, N. D. Whyborn, P. Dieleman, P. R. Roelfsema, H. Aarts, R. Assendorp, R. Bachiller, W. Baechtold, A. Barcia, D. A. Beintema, V. Belitsky, A. O. Benz, R. Bieber, A. Boogert, C. Borys, B. Bumble, P. Caïs, M. Caris, P. Cerulli-Irelli, G. Chattopadhyay, S. Cherednichenko, M. Ciechanowicz, O. Coeur-Joly, C. Comito, A. Cros, A. de Jonge, G. de Lange, B. Delforges, Y. Delorme, T. den Boggende, J.-M. Desbat, C. Diez-González, A. M. di Giorgio, L. Dubbeldam, K. Edwards, M. Eggens, N. Erickson, J. Evers, M. Fich, T. Finn, B. Franke, T. Gaier, C. Gal, J. R. Gao, J.-D. Gallego, S. Gauffre, J. J. Gill, S. Glenz, H. Golstein, H. Goulooze, T. Gunsing, R. Güsten, P. Hartogh, W. A. Hatch, R. Higgins, E. C. Honingh, R. Huisman, B. D. Jackson, H. Jacobs, K. Jacobs, C. Jarchow, H. Javadi, W. Jellema, M. Justen, A. Karpov, C. Kasemann, J. Kawamura, G. Keizer, D. Kester, T. M. Klapwijk, T. Klein, E. Kollberg, J. Kooi, P.-P. Kooiman, B. Kopf, M. Krause, J.-M. Krieg, C. Kramer, B. Kruizenga, T. Kuhn, W. Laauwen, R. Lai,

- B. Larsson, H. G. Leduc, C. Leinz, R. H. Lin, R. Liseau, G. S. Liu, A. Loose, I. López-Fernandez, S. Lord, W. Luinge, A. Marston, J. Martín-Pintado, A. Maestrini, F. W. Maiwald, C. McCoey, I. Mehdi, A. Megej, M. Melchior, L. Meinsma, H. Merkel, M. Michalska, C. Monstein, D. Moratschke, P. Morris, H. Muller, J. A. Murphy, A. Naber, E. Natale, W. Nowosielski, F. Nuzzolo, M. Olberg, M. Olbrich, R. Orfei, P. Orleanski, V. Ossenkopf, T. Peacock, J. C. Pearson, I. Peron, S. Phillip-May, L. Piazzo, P. Planesas, M. Rataj, L. Ravera, C. Risacher, M. Salez, L. A. Samoska, P. Saraceno, R. Schieder, E. Schlecht, F. Schlöder, F. Schmülling, M. Schultz, K. Schuster, O. Siebertz, H. Smit, R. Szczerba, R. Shipman, E. Steinmetz, J. A. Stern, M. Stokroos, R. Teipen, D. Teyssier, T. Tils, N. Trappe, C. van Baaren, B.-J. van Leeuwen, H. van de Stadt, H. Visser, K. J. Wildeman, C. K. Wafelbakker, J. S. Ward, P. Wesselius, W. Wild, S. Wulff, H.-J. Wunsch, X. Tielens, P. Zaal, H. Zirath, J. Zmuidzinis, F. Zwart. The Herschel-Heterodyne Instrument for the Far-Infrared (HIFI)[J]. *A&A*. Jul. 2010, **518**:L6
- [10] S. Ott. The Herschel Data Processing System – HIPE and Pipelines – Up and Running Since the Start of the Mission[C]. Y. Mizumoto, K.-I. Morita, M. Ohishi, (Editors) Astronomical Data Analysis Software and Systems XIX. 2010, vol. 434 of Astronomical Society of the Pacific Conference Series, 139. [1011.1209](#)
- [11] Science Ground Segment Consortium, P. Hifi, Spire Consortia. HIPE: Herschel Interactive Processing Environment[Z], Nov. 2011. Astrophysics Source Code Library, [1111.001](#)
- [12] John Owens. GPUs for Scientific Computing and Visualization[Z]
- [13] M. A. Clark, P. C. La Plante, L. J. Greenhill. Accelerating Radio Astronomy Cross-Correlation with Graphics Processing Units[J]. ArXiv e-prints. Jul. 2011. [1107.4264](#)
- [14] Nvidia Inc. CUDA C PROGRAMMING GUIDE[J]. Oct. 2012. [PG-02829-001_v5.0](#)

- [15] Wen-mei W. Hwu David B.Kirk. Programming Massively Parallel Processors: A Hands-on Approach[M]. 2010
- [16] Edward Kandrot Jasom Sanders. CUDA by example[M]. 2010
- [17] Matt Griffin, C. Darren Dowell, Tanya Lim, George Bendo, Jamie Bock, Christophe Cara, Nieves Castro-Rodriguez, Pierre Chanial, Dave Clements, Rene Gastaud, Steve Guest, Jason Glenn, Victor Hristov, Ken King, Glenn Laurent, Nanyao Lu, Gabrielle Mainetti, Huw Morris, Hien Nguyen, Pasquale Panuzzo, Chris Pearson, Frederic Pinsard, Michael Pohlen, Edward Polehampton, Davide Rizzo, Bernhard Schulz, Arnold Schwartz, Bruce Sibthorpe, Bruce Swinyard, Kevin Xu, Lijun Zhang. The Herschel-SPIRE photometer data processing pipeline[J]. 2008:70102Q–70102Q–12
- [18] Trevor R. Fulton, David A. Naylor, Jean-Paul Baluteau, Matt Griffin, Peter Davis-Imhof, Bruce M. Swinyard, Tanya L. Lim, Christian Surace, Dave Clements, Pasquale Panuzzo, Rene Gastaud, Edward Polehampton, Steve Guest, Nanyao Lu, Arnold Schwartz, Kevin Xu. The data processing pipeline for the Herschel/SPIRE imaging Fourier Transform Spectrometer[J]. 2008:70102T–70102T–12
- [19] Trevor R. Fulton, Jean-Paul Baluteau, George Bendo, Dominique Benelli, Rene Gastaud, Matt Griffin, Steve Guest, Peter Imhof, Tanya L. Lim, Nanyao Lu, David A. Naylor, Pasquale Panuzzo, Edward Polehampton, Arnold Schwartz, Christian Surace, Bruce M. Swinyard, Kevin Xu. The data processing pipelines for the Herschel/SPIRE imaging Fourier transform spectrometer[J]. 2010:773134–773134–12
- [20] Y. D. Mayya, L. Carrasco, A. Luna. The Discovery of Spiral Arms in the Starburst Galaxy M82[J]. ApJ. Jul. 2005, **628**:L33–L36. [arXiv:astro-ph/0506275](https://arxiv.org/abs/astro-ph/0506275)
- [21] MPIA Heidelberg Jürgen Schreiber, [Z]

附录 A java 内核代码

A.1 HCSS矩阵乘法和快速傅里叶变换的GPU实现

A.1.1 矩阵乘法的GPU实现

完整代码见MatrixMultiply1.java。因为矩阵乘法的GPU实现中，类的生成函数和调用接口与原程序相同，这里只给出关键的GPU函数实现步骤，即代码311行到506行，单精度浮点型和双精度浮点型的矩阵乘法GPU实现。

```
311  /**
312   * Computes the matrix multiplication of specified input array with
313   * the array passed during construction fo this class.
314   * @return an array of type Float1d.
315   */
316  public ArrayData of(Float1d lhs) {
317      return to2d(lhs).apply(this).apply(Basic.CONCATENATE);
318  }
319
320  /**
321   * Computes the matrix multiplication of specified input array with
322   * the array passed during construction fo this class.
323   * @return an array of type Float2d.
324   */
325  public ArrayData of(Float2d lhs) {
326      validate(lhs);
327      Float2d rhs=(Float2d)_rhs;
328
329      // create the resulting array and some short-cuts:
330      Float2d y=new Float2d(_lhsM,_rhsN);
```

```
331 // do the multiplication
332 float alpha=1f;
333 float beta=0f;
334
335
336 final float[][] B=rhs.getArray();
337 final float[][] A=lhs.getArray();
338 float[][] C=y.getArray();
339
340
341
342 int m=_lhsM;
343 int n=_rhsN;
344 int l=_lhsN;
345
346
347 // Create a CUBLAS handle
348 JCublas.cublasInit();
349 // Allocate memory on the device
350 Pointer d_B = new Pointer();
351 Pointer d_C = new Pointer();
352 Pointer d_A = new Pointer();
353 JCublas.cublasAlloc(m*l, Sizeof.FLOAT, d_A);
354 JCublas.cublasAlloc(l*n, Sizeof.FLOAT, d_B);
355 JCublas.cublasAlloc(m*n, Sizeof.FLOAT, d_C);
356
357 // Copy the memory from the host to the device
358 float AA[]=new float[m*l];
359 float BB[]=new float[l*n];
360 float CC[]=new float[m*n];
361
362 for(int i=0;i<m;i++){
363     for(int j=0;j<l;j++){
364         AA[j*m+i]=A[i][j];
365     }
366 }
367 for(int i=0;i<l;i++){
368     for(int j=0;j<n;j++){
369         BB[j*l+i]=B[i][j];
```

```
370     }
371 }
372 for(int i=0;i<m;i++){
373     for(int j=0;j<n;j++){
374         CC[j*m+i]=C[i][j];
375     }
376 }
377
378 JCublas.cublasSetMatrix(m,l, Sizeof.FLOAT, Pointer.to(AA), m, d_A, m);
379 JCublas.cublasSetMatrix(l,n, Sizeof.FLOAT, Pointer.to(BB), l, d_B, l);
380 JCublas.cublasSetMatrix(m,n, Sizeof.FLOAT, Pointer.to(CC), m, d_C, m);
381 JCublas.cublasSgemm('n', 'n', m, n, l, alpha, d_A, m, d_B, l, beta, d_C, m);
382
383 // Copy the result from the device to the host
384 JCublas.cublasGetMatrix(m,n, Sizeof.FLOAT, d_C, m, Pointer.to(CC), m);
385
386
387 // Clean up
388
389
390 for(int i=0;i<_lhsM;i++){
391     for (int j=0;j<_rhsN;j++){
392         C[i][j]=CC[j*_lhsM+i];
393     }
394 }
395
396 JCublas.cublasFree(d_A);
397 JCublas.cublasFree(d_B);
398 JCublas.cublasFree(d_C);
399 JCublas.cublasShutdown();
400
401 // and pass it back:
402 return _rhsIsRankOneArray?y.apply(Basic.CONCATENATE):y;
403 }
404
405 /**
406 * Computes the matrix multiplication of specified input array with
407 * the array passed during construction fo this class.
408 * @return an array of type Double1d.
```

```
409     */
410     public ArrayData of(Double1d lhs) {
411         return to2d(lhs).apply(this).apply(Basic.CONCATENATE);
412     }
413
414     /**
415      * Computes the matrix multiplication of specified input array with
416      * the array passed during construction fo this class.
417      * @return an array of type Double2d.
418      */
419     public ArrayData of(Double2d lhs) {
420         validate(lhs);
421         Double2d rhs=(Double2d)_rhs;
422
423         Double2d y=new Double2d(_lhsM,_rhsN);
424
425         // create the resulting array and some short-cuts:
426         final double[][] B=rhs.getArray();
427         final double[][] A=lhs.getArray();
428         double[][] C=y.getArray();
429
430
431         int m=_lhsM;
432         int n=_rhsN;
433         int l=_rhsN;
434
435         // Create a CUBLAS handle
436         JCublas.cublasInit();
437         // Allocate memory on the device
438         Pointer d_A = new Pointer();
439         Pointer d_B = new Pointer();
440         Pointer d_C = new Pointer();
441         JCublas.cublasAlloc(m*l, Sizeof.DOUBLE, d_A);
442         JCublas.cublasAlloc(l*n, Sizeof.DOUBLE, d_B);
443         JCublas.cublasAlloc(m*n, Sizeof.DOUBLE, d_C);
444
445         // Copy the memory from the host to the device
446         double AA[]=new double[m*l];
447         double BB[]=new double[l*n];
```

```
448     double CC[] = new double[m*n];  
449  
450  
451  
452     for(int i=0;i<m;i++){  
453         for(int j=0;j<l;j++){  
454             AA[j*m+i]=A[i][j];  
455         }  
456     }  
457     for(int i=0;i<l;i++){  
458         for(int j=0;j<n;j++){  
459             BB[j*l+i]=B[i][j];  
460         }  
461     }  
462     for(int i=0;i<m;i++){  
463         for(int j=0;j<n;j++){  
464             //CC[j*m+i]=C[i][j];  
465             CC[j*m+i]=11;  
466         }  
467     }  
468  
469  
470  
471     JCublas.cublasSetMatrix(m,l, Sizeof.DOUBLE, Pointer.to(AA), m, d_A, m);  
472     JCublas.cublasSetMatrix(l,n, Sizeof.DOUBLE, Pointer.to(BB), l, d_B, l);  
473     JCublas.cublasSetMatrix(m,n, Sizeof.DOUBLE, Pointer.to(CC), m, d_C, m);  
474  
475     double alpha=1;  
476     double beta=0;  
477  
478  
479     // Execute sgemm  
480  
481     JCublas.cublasDgemm('n', 'n', m, n, l, alpha, d_A, m, d_B, l, beta, d_C, m);  
482     //JCublas.cublasSgemm('n', 'n', m, n, l, alpha, d_A, m, d_B, l, beta, d_C, m);  
483     // Copy the result from the device to the host  
484     JCublas.cublasGetMatrix(m,n, Sizeof.DOUBLE, d_C, m, Pointer.to(CC), m);  
485  
486
```

```

487     // Clean up
488
489
490     for(int i=0;i<m;i++){
491         for (int j=0;j<n;j++){
492             C[i][j]=CC[j*m+i];
493
494         }
495     }
496
497
498     JCublas.cublasFree(d_A);
499     JCublas.cublasFree(d_B);
500     JCublas.cublasFree(d_C);
501     JCublas.cublasShutdown();
502
503     // and pass it back:
504     return _rhsIsRankOneArray?y.apply(Basic.CONCATENATE):y;
505 }
506

```

A.1.2 FFT/IFFT的GPU实现

完整代码见CudaFFT.java。因为FFT的GPU实现中，类的生成函数和调用接口与原程序相同，这里只给出关键的GPU函数实现步骤，即代码172行到227行，单精度浮点型和双精度浮点型的矩阵乘法GPU实现。

```

172     public static void fft(Complex1d array) {
173         double[] re=array.realArray();
174         double[] im=array.imagArray();
175         int size=array.getSize();
176         double[] outputJCufft=new double[2*size];
177         double[] inputJCufft=new double[2*size];
178         for(int i=0;i<size;i++){
179             inputJCufft[2*i]=re[i];
180             inputJCufft[2*i+1]=im[i];
181         }
182         cufftHandle plan = new cufftHandle();

```

```
183     JCufft.cufftPlan1d(plan, size, cufftType.CUFFT_Z2Z, 1);
184     JCufft.cufftExecZ2Z(plan, inputJCufft, outputJCufft, JCufft.CUFFT_FORWARD);
185
186
187     for(int i=0;i<size;i++){
188         re[i]=outputJCufft[2*i];
189         im[i]=outputJCufft[2*i+1];
190     }
191     JCufft.cufftDestroy(plan);
192     array.set(new Complex1d(re,im));
193
194 }
195
196
197 /**
198 * Perform in-place inverse FFT.
199 *
200 * @param array Array to be transformed.
201 */
202 public static void invFft(Complex1d array) {
203     double factor = 1D / array.length();
204     double[] re=array.realArray();
205     double[] im=array.imagArray();
206     int size=array.getSize();
207     double[] outputJCufft=new double[size*2];
208     double[] inputJCufft=new double[size*2];
209     for(int i=0;i<size;i++){
210         inputJCufft[2*i]=re[i];
211         inputJCufft[2*i+1]=im[i];
212     }
213     cufftHandle plan = new cufftHandle();
214     JCufft.cufftPlan1d(plan, size, cufftType.CUFFT_Z2Z, 1);
215     JCufft.cufftExecZ2Z(plan, inputJCufft, outputJCufft, JCufft.CUFFT_INVERSE);
216
217     for(int i=0;i<size;i++){
218         re[i]=outputJCufft[2*i];
219         im[i]=outputJCufft[2*i+1];
220     }
221 }
```

```
222     JCufft.cufftDestroy(plan);  
223  
224     array.set(new Complex1d(re,im));  
225     array.multiply(new Complex(factor, 0));  
226 }  
227
```

A.1.3 矩阵乘法 CPU和GPU效率对比

```
1 package herschel.ia.numeric.toolbox.matrix;  
2 import herschel.ia.numeric.*;  
3 import java.util.Random;  
4  
5  
6 /**  
7 *  
8 * @author Yifei jin<m-astro@163.com>  
9 *  
10 */  
11  
12 public class MatrixMulCmp {  
13  
14  
15  
16     public final int LOGSCALE=0;  
17     public final int LINESCALE=1;  
18     public final int FLOAT=0;  
19     public final int DOUBLE=1;  
20  
21     int begin;  
22     int end;  
23     int step;  
24     int num;  
25     int type;  
26  
27     public MatrixMulCmp(){  
28         begin=1;  
29         end=1001;
```

```
30     step=20;
31     num=50;
32     type=DOUBLE;
33 }
34
35 public MatrixMulCmp(int _begin, int _end, int _step,int _num, int _type) {
36     begin=_begin;
37     end=_end;
38     step=_step;
39     num=_num;
40     type=_type;
41 }
42
43 public void SetPara(int _begin, int _end, int _step,int _num, int _type) {
44     this.begin=_begin;
45     this.end=_end;
46     this.step=_step;
47     this.num=_num;
48     this.type=_type;
49 }
50
51 public double[][] MatrixMultime(int scaleMode){
52     if(scaleMode==this.LINESCALE){
53         double[][] timelist= new double[2][(int)((end-begin)/step)];
54         double[] timeonce ={0,0};
55         int j=0;
56         for(int i=begin;i<end;i+=step){
57             timeonce=ComPare(i,this.num,this.type);
58             timelist[0][j]=timeonce[0];
59             timelist[1][j]=timeonce[1];
60             j=j+1;
61         }
62         return timelist;
63     }else if(scaleMode==this.LOGSCALE){
64         double[][] timelist= new double[2][(int)((end-begin)/step)];
65         double[] timeonce ={0,0};
66         int j=0;
67         for(int i=begin;i<end;i+=step){
68             timeonce=ComPare((int)java.lang.Math.pow(2, i),this.num,this.type);
```

```
69         timelist[0][j]=timeonce[0];
70         timelist[1][j]=timeonce[1];
71         j=j+1;
72     }
73     return timelist;
74
75 }else{
76     throw new IllegalArgumentException("NO SUCH SCALE MODE");
77 }
78
79 }
80
81 public double[] ComPare(int dim, int num,int type) {
82
83
84     Float2d Xf=null,Yf=null;
85     Double2d Xd=null,Yd=null;
86     double[] timelist =null;
87     double timeBegin=0,timeEnd=0;
88     double cuTime=0,jaTime=0;
89     if(type==this.FLOAT){
90         for(int i=0;i<num;i++){
91             Xf=GenerateFloatArray(dim,dim);
92             Yf=GenerateFloatArray(dim,dim);
93             MatrixMultiply Majava= new MatrixMultiply(Xf);
94             timeBegin=(double)System.nanoTime();
95             Yf.apply(Majava);
96             timeEnd =(double)System.nanoTime() - timeBegin;
97             jaTime+=timeEnd;
98
99             MatrixMultiply1 Macuda= new MatrixMultiply1(Xf);
100            timeBegin=(double)System.nanoTime();
101            Yf.apply(Macuda);
102            timeEnd =(double)System.nanoTime() - timeBegin;
103            cuTime+=timeEnd;
104        }
105    }
106    else if(type==this.DOUBLE){
107        for(int i=0;i<num;i++){
```

```
108     Xd=GenerateDoubleArray(dim,dim);
109     Yd=GenerateDoubleArray(dim,dim);
110     MatrixMultiply Majava= new MatrixMultiply(Xd);
111     timeBegin=(double)System.nanoTime();
112     Yd.apply(Majava);
113     timeEnd =(double)System.nanoTime() - timeBegin;
114     jaTime+=timeEnd;
115
116     MatrixMultiply1 Macuda= new MatrixMultiply1(Xd);
117     timeBegin=(double)System.nanoTime();
118     Yd.apply(Macuda);
119     timeEnd =(double)System.nanoTime() - timeBegin;
120     cuTime+=timeEnd;
121 }
122 }else {
123     throw new IllegalArgumentException("the data type is error");
124 }
125
126 jaTime = jaTime/(1000000000.0*num);
127 cuTime = cuTime/(1000000000.0*num);
128 timelist=new double[]{jaTime,cuTime};
129 return timelist;
130
131 }
132
133 public Double2d GenerateDoubleArray(int col,int row){
134     Random rdm= new Random();
135     double[][] X = new double[col][row];
136     for(int i=0;i<col;i++){
137         for(int j=0;j<row;j++){
138             X[i][j]=rdm.nextDouble();
139         }
140     }
141     Double2d Array = new Double2d(X);
142     return Array;
143 }
144
145
146 public Float2d GenerateFloatArray(int col,int row){
```

```
147     Random rdm= new Random();
148     float[][] X = new float[col][row];
149     for(int i=0;i<col;i++){
150         for(int j=0;j<row;j++){
151             X[i][j]=rdm.nextFloat();
152         }
153     }
154     Float2d Array = new Float2d(X);
155     return Array;
156 }
157 }
158
159
160 }
161
```

A.1.4 FFT CPU和GPU效率对比

```
1 package herschel.ia.numeric.toolbox.xform;
2
3 import herschel.ia.numeric.*;
4 import java.util.Random;
5 import static herschel.ia.numeric.toolbox.xform.FFT.FFT;
6 import static herschel.ia.numeric.toolbox.xform.FFT.IFFT;
7 import static herschel.ia.numeric.toolbox.xform.CudaFFT.CudaFFT;
8 import static herschel.ia.numeric.toolbox.xform.CudaFFT.CudaIFFT;
9
10 public class FFTCmp {
11
12     public final int FORWARD=-1;
13     public final int REVERSE=1;
14     public final int LOGSCALE=0;
15     public final int LINESCALE=1;
16
17     int begin;
18     int end;
19     int step;
20     int direction;
```

```
21     int num;  
22  
23     public FFTCmp(){  
24         begin=1;  
25         end=1001;  
26         step=20;  
27         num=50;  
28         direction=this.FORWARD;  
29     }  
30     public FFTCmp(int _begin, int _end, int _step,int _num, int _direction) {  
31         begin=_begin;  
32         end=_end;  
33         step=_step;  
34         num=_num;  
35         direction=_direction;  
36     }  
37  
38     public void SetPara(int _begin, int _end, int _step,int _num, int _direction) {  
39         this.begin=_begin;  
40         this.end=_end;  
41         this.step=_step;  
42         this.num=_num;  
43         this.direction=_direction;  
44     }  
45  
46     public double[][] FFTtime(int scaleMode){  
47         if(scaleMode==this.LINESCALE){  
48             double[][] timelist= new double[2][(int)((end-begin)/step)];  
49             double[] timeonce ={0,0};  
50             int j=0;  
51             for(int i=begin;i<end;i+=step){  
52                 timeonce=ComPare(i,this.num,this.direction);  
53                 timelist[0][j]=timeonce[0];  
54                 timelist[1][j]=timeonce[1];  
55                 j=j+1;  
56             }  
57             return timelist;  
58         }else if(scaleMode==this.LOGSCALE){  
59             double[][] timelist= new double[2][(int)((end-begin)/step)];
```

```
60     double[] timeonce ={0,0};
61     int j=0;
62     for(int i=begin;i<end;i+=step){
63         timeonce=ComPare((int)java.lang.Math.pow(2, i),this.num,this.direction);
64         timelist[0][j]=timeonce[0];
65         timelist[1][j]=timeonce[1];
66         j=j+1;
67     }
68     return timelist;
69
70 }else{
71     throw new IllegalArgumentException("NO SUCH SCALE MODE");
72 }
73
74 }
75
76
77 public double[] ComPare(int dim, int num, int direction) {
78
79     Complex1d X=null;
80 //     Complex1d Y1=null;
81 //     Complex1d Y2=null;
82     double[] timelist =null;
83     double timeBegin=0,timeEnd=0;
84     double cuTime=0,jaTime=0;
85     if(direction==this.FORWARD){
86         for(int i=0;i<num;i++){
87             X=GenerateArray(dim);
88             timeBegin=(double)System.nanoTime();
89             X.apply(FFT);
90             timeEnd =(double)System.nanoTime() - timeBegin;
91             jaTime+=timeEnd;
92             timeBegin=(double)System.nanoTime();
93             X.apply(CudaFFT);
94             timeEnd =(double)System.nanoTime() - timeBegin;
95             cuTime+=timeEnd;
96         }
97     }
98     else if(direction==this.REVERSE){
```

```
99     for(int i=0;i<num;i++){
100         X=GenerateArray(dim);
101         timeBegin=(double)System.nanoTime();
102         X.apply(IFT);
103         timeEnd =(double)System.nanoTime() - timeBegin;
104         jaTime+=timeEnd;
105
106         timeBegin=(double)System.nanoTime();
107         X.apply(CudaIFFT);
108         timeEnd =(double)System.nanoTime() - timeBegin;
109         cuTime+=timeEnd;
110     }
111 }
112 else {
113     throw new IllegalArgumentException("the direction is error");
114 }
115
116 jaTime = jaTime/(1000000000.0*num);
117 cuTime = cuTime/(1000000000.0*num);
118 timelist=new double[]{jaTime,cuTime};
119 return timelist;
120
121 }
122
123 public Complex1d GenerateArray(int dim){
124     Random rdm= new Random();
125     double[] re = new double[dim];
126     double[] im = new double[dim];
127     for(int i=0;i<dim;i++){
128         re[i]=rdm.nextDouble();
129         im[i]=rdm.nextDouble();
130     }
131     Complex1d Array = new Complex1d(re,im);
132     return Array;
133 }
134 public boolean isCorrectResult(Complex1d result, Complex1d reference)
135 {
136     double errorNorm = 0;
137     double refNorm = 0;
```

```

138     double para;
139     for (int i = 0; i < result.getSize(); i++)
140     {
141         Complex diff = result.get(i).divide(reference.get(i));
142         errorNorm += diff.abs()*diff.abs();
143         para=(reference.get(i).multiply(result.get(i).conjugate())).abs();
144         refNorm += para*para;
145     }
146     errorNorm = (double) Math.sqrt(errorNorm);
147     refNorm = (double) Math.sqrt(refNorm);
148     if (Math.abs(refNorm) < 1e-6)
149     {
150         return false;
151     }
152     return (errorNorm / refNorm < 1e-6f);
153 }
154
155
156
157 }
```

A.2 HCSS脚本分析中的计数辅助类和计时辅助类

本工作涉及到的计数辅助类和计时辅助类可见表3.1。限于篇幅，这里作为例子仅给出FFT的计数辅助类和计时辅助类。

A.2.1 FFT 计数辅助类

FFT计数辅助类，代码文件名为FFTNUMCOUNT.java。

```

1 package herschel.ia.numeric.toolbox.xform;
2
3
4 import java.io.DataInputStream;
5 import java.io.DataOutputStream;
6 import java.io.FileInputStream;
7 import java.io.FileOutputStream;
```

```
8 import java.io.IOException;
9 import java.io.InputStream;
10
11 class FFTPara{
12     int dim;
13     int num;
14     long _Count;
15     long[][] _Detail;
16     String _para;
17     FFTPara(int _Dim, String name){
18         num=0;
19         dim=_Dim;
20         _Count=0;
21         _Detail=new long[2][_Dim];
22         _para=name;
23     }
24
25     public void ReSet(){
26         num=0;
27         _Count=0;
28         for(int i=0;i<2;i++){
29             for(int j=0;j<dim;j++){
30                 _Detail[i][j] = 0;
31             }
32         }
33     }
34 }
35
36     public void NumCount(){
37         _Count+=1;
38     }
39     public long GetCount(){
40         System.out.println("the number of "+_para+": "+ _Count);
41         return _Count;
42     }
43
44
45
46 // this part is used for dimension count
```

```
47     public boolean search(int len){
48         boolean flag= false;
49         for(int i=0;i<num;i++){
50             if(_Detail[0][i]==len){
51                 flag=true;
52                 break;
53             }
54         }
55         return flag;
56     }
57
58     public void Calculate(int len){
59         int point=0;
60         for(int i=0;i<num;i++){
61             if(_Detail[0][i]==len){
62                 point=i;
63                 break;
64             }
65         }
66         _Detail[1][point] = _Detail[1][point]+1;
67
68     }
69
70     public int find(int len){
71         int point=0;
72         while(_Detail[0][point] < len && point < num){
73             point++;
74         }
75
76         return point;
77     }
78
79
80
81     public void insert(int point,int len){
82         try {
83
84             if(num == dim) throw new Exception("space is not enough");
85             for(int i= num;i>point;i--){

```

```
86         for(int j=0;j<2;j++){
87             _Detail[j][i]=_Detail[j][i-1];
88         }
89     }
90     _Detail[0][point]=len;
91     _Detail[1][point]=1;
92     num=num+1;
93
94 } catch (Exception e) {
95     e.printStackTrace();
96     System.out.println("The record space is not enough:"+e);
97 }
98
99 }
100
101 public void DtlCount(int len){
102     int point;
103     if(search(len)==true){
104         Calculate(len);
105     }
106     else{
107         point=find(len);
108         insert(point,len);
109     }
110
111 }
112
113 public long[][] GetDtl(){
114     long[][] FFTDim= null;
115     FFTDim=new long[2][num];
116     for (int i=0;i<num;i++){
117         for (int j=0;j<2;j++){
118             FFTDim[j][i]=_Detail[j][i];
119         }
120     }
121     return FFTDim;
122 }
123
124 public void NcOutput(String name,String Path) throws IOException{
```

```
125     FileOutputStream fos = null;
126     DataOutputStream dos = null;
127     long[][] dbuf = new long[2][num];
128     for(int i=0;i<2;i++){
129         for(int j=0;j<num;j++){
130             dbuf[i][j]=_Detail[i][j];
131         }
132     }
133     try{
134         fos = new FileOutputStream(Path+name);
135         dos = new DataOutputStream(fos);
136         dos.writeInt(num);
137         for(int i=0;i<dbuf.length;i++){
138             for (long d:dbuf[i]){
139                 dos.writeLong(d);
140             }
141         }
142         dos.flush();
143     }catch(Exception e){
144         e.printStackTrace();
145     }finally{
146         if(dos!=null)
147             dos.close();
148         if(fos!=null)
149             fos.close();
150     }
151 }
152 }
153
154 public long[][] NcInput(String name, String Path) throws IOException{
155     InputStream is = null;
156     DataInputStream dis = null;
157     long[][] dbuf=null ;
158     try{
159         is = new FileInputStream(Path+name);
160         dis = new DataInputStream(is);
161         int i=0;
162         int j=0;
163         int num=dis.readInt();
```

```
164
165    dbuf=new long[2][num];
166     while(dis.available()>0)
167     {
168         if(j==num){
169             i++;
170             j=0;
171         }
172        dbuf[i][j] = dis.readLong();
173         j++;
174     }
175     }catch(Exception e){
176     e.printStackTrace();
177     }finally{
178         if(is!=null)
179             is.close();
180         if(dis!=null)
181             dis.close();
182     }
183     return dbuf;
184 }
185 }
186
187
188
189 public class FFTNumCount {
190
191
192     final static int _Dim=1000;
193     public FFTNumCount(){
194     }
195
196
197     public static FFTPara NCfft= new FFTPara(_Dim,"NCfft");
198     public static FFTPara NCifft= new FFTPara(_Dim,"NCifft");
199
200     public static FFTPara[] list={NCfft,NCifft};
201
202     public static void ReSet(){
```

```
203     for(FFTPara Ops : list){  
204         Ops.ReSet();  
205     }  
206 }  
207  
208 public static void FFTGetALL(){  
209     for(FFTPara Ops : list){  
210         Ops.GetCount();  
211     }  
212 }  
213 public static void FFTNcOutputALL(String Path) throws IOException{  
214     for(FFTPara Ops : list){  
215         Ops.NcOutput(Ops._para,Path);  
216     }  
217 }  
218  
219 public static void FFTCount(FFTPara Ops){  
220     Ops.NumCount();  
221 }  
222 public static long FFTGetCount(FFTPara Ops){  
223     return Ops.GetCount();  
224 }  
225 public static void FFTDtlCount(FFTPara Ops,int len){  
226     Ops.DtlCount(len);  
227 }  
228 public static long[][] FFTGetDtl(FFTPara Ops){  
229     return Ops.GetDtl();  
230 }  
231 public static void FFTNcOutput(FFTPara Ops,String Path) throws IOException{  
232     Ops.NcOutput(Ops._para, Path);  
233 }  
234 public static long[][] FFTNcInput(FFTPara Ops,String Path) throws IOException{  
235     return Ops.NcInput(Ops._para, Path);  
236 }  
237  
238 public static long[] FFTGetDim(FFTPara Ops){  
239     long[][] dtl=FFTGetDtl(Ops);  
240     int nums = dtl[0].length;  
241     int dm= (int) dtl[0][nums-1];
```

```

242     long[] Dim = new long[dm+1];
243     for(int i=0;i<nums;i++){
244         dm=(int) dtl[0][i];
245         Dim[dm]=dtl[1][i];
246     }
247     return Dim;
248 }
249
250
251 }
```

A.2.2 FFT 计时辅助类

FFT计数辅助类，代码文件名为FFTTTimeCost.java

```

1 package herschel.ia.numeric.toolbox.xform;
2
3 import herschel.ia.numeric.*;
4
5 import java.io.DataInputStream;
6 import java.io.FileInputStream;
7 import java.io.IOException;
8 import java.io.InputStream;
9 import java.util.Random;
10
11 public class FFTTTimeCost {
12
13     public FFTTTimeCost() {
14         // TODO Auto-generated constructor stub
15     }
16
17     //----- time cost calculate -----
18     public static double TimeCost(String Ops, String Path) throws IOException{
19         long[][] numCount = TcInput(Ops, Path);
20         int len,nums;
21         int row=numCount.length;
22         if(row!=2) throw new IllegalArgumentException("the lead dimension unmatched");
23         int col=numCount[0].length;
```

```
24     double AllTime=0;
25     for(int i=0;i<col;i++){
26         len=(int)(numCount[0][i]);
27         nums=(int)(numCount[1][i]);
28         AllTime=AllTime+TimeCal(Ops,len)*nums;
29
30
31     }
32     return AllTime;
33 }
34
35
36     public static double TimeCal(String Ops,int len){
37         double timeused=0;
38         double timeBegin=0,timeEnd=0;
39         int its=200;
40         if(Ops=="NCfft"){
41             for(int i=0;i<its;i++){
42                 Complex1d X = GenerateArray(len);
43                 timeBegin=System.nanoTime();
44                 FFT.fft(X);
45                 timeEnd=(double) (System.nanoTime()-timeBegin)/1000000000.0;
46                 timeused+=timeEnd;
47             }
48
49 //      according the methods, the IFFT has already been calculate by the FFT methods
50 }else if(Ops=="NCifft"){
51     for(int i=0;i<its;i++){
52         Complex1d X = GenerateArray(len);
53         timeBegin=System.nanoTime();
54         FFT.invFft(X);
55         timeEnd=(double) (System.nanoTime()-timeBegin)/1000000000.0;
56         timeused+=timeEnd;
57     }
58
59 }else{
60     timeEnd=-1;
61     throw new IllegalArgumentException("these kind of operation is not exsit");
62 }
```

```
63     return (timeused/its);
64 }
65
66 //-----input the data from number count file -----
67
68
69
70 public static long[][] TcInput (String name, String Path) throws IOException{
71     InputStream is = null;
72     DataInputStream dis = null;
73     long[][] dbuf=null ;
74     try{
75         is = new FileInputStream(Path+name);
76         dis = new DataInputStream(is);
77         int i=0;
78         int j=0;
79         int num=dis.readInt();
80
81         dbuf=new long[2] [num];
82         while(dis.available()>0)
83         {
84             if(j==num){
85                 i++;
86                 j=0;
87             }
88             dbuf[i] [j] = dis.readLong();
89             j++;
90         }
91     }catch(Exception e){
92         e.printStackTrace();
93     }finally{
94         if(is!=null)
95             is.close();
96         if(dis!=null)
97             dis.close();
98     }
99     return dbuf;
100 }
101 // ----- Generate Array -----
```

```
102  
103     public static Complex1d GenerateArray(int length){  
104         Random rdm =new Random();  
105         double[] re= new double[length];  
106         double[] imag=new double[length];  
107         for(int i=0;i<length;i++){  
108             re[i]=rdm.nextDouble();  
109             imag[i]=rdm.nextDouble();  
110         }  
111         Complex1d X= new Complex1d(re,imag);  
112         return X;  
113     }  
114  
115  
116  
117  
118 }  
119
```

附录 B jython 脚本代码

B.1 GPU 和 CPU 效率对比

B.1.1 矩阵乘法

矩阵乘法的CPU实现和GPU实现效率对比，代码文件名为MatrixMulCMP.py。

```
1 # the parameters to build the class MatrixMulCmp
2 begin=1
3 end=11
4 step=1
5 IterationNum=50
6 type=0 #means float
7 Path="C:\\Users\\yfjin\\hcsl\\workresult\\Matrix\\"
8 ScaleMode=0 #log
9 CMP=MatrixMulCmp(begin,end,step,IterationNum,type)
10
11 timelist=CMP.MatrixMultime(ScaleMode)
12 Xd=range(begin,end,step)
13
14 myPlot = PlotXY()
15 myLayer1 = LayerXY(Double1d(Xd),Double1d(timelist[0]),color=java.awt.Color.red)
16 myLayer2 = LayerXY(Double1d(Xd),Double1d(timelist[1]),color=java.awt.Color.green)
17 myLayer1.line = Style.MARKED
18 myLayer2.line = Style.MARK_DASHED
19 myLayer1.name="by pure java FFT"
20 myLayer2.name="by Jcufft"
21 myPlot.legend.visible = 1
```

```
22 myPlot.addLayer(myLayer1)
23 myPlot.addLayer(myLayer2)
24
25 type=1
26 timelist=CMP.MatrixMultime(ScaleMode)
27 myLayer3 = LayerXY(Double1d(Xd),Double1d(timelist[0]),color=java.awt.Color.blue)
28 myLayer4 = LayerXY(Double1d(Xd),Double1d(timelist[1]),color=java.awt.Color.black)
29 myLayer3.line = Style.MARKED
30 myLayer4.line = Style.MARK_DASHED
31 myLayer3.name="double Matrixmultiply by pure java "
32 myLayer4.name="double Matrixmultiply by Jcublas"
33 myPlot.legend.visible = 1
34 myPlot.addLayer(myLayer3)
35 myPlot.addLayer(myLayer4)
36
37 myPlot.xaxis.type = Axis.LINEAR
38 myPlot.yaxis.type = Axis.LOG
39 myPlot.titleText = "Comparison with CUDA and pure Java"
40 myPlot.subtitleText = "FFT class(Double)"
41 myPlot.xaxis.titleText = "number of elements /log2(N)"
42 myPlot.yaxis.titleText = "Time"
43 myPlot.saveAsEPS(Path+ "cujaMatrixLog.eps") # Encapsulated PS
44
45 # rerun using the linear scale
46
47 begin=1
48 end=221
49 step=10
50 IterationNum=50
51 type=0
52 Path="C:\\\\Users\\\\yfjin\\\\hcsl\\\\workresult\\\\Matrix\\\\"
53 CMP.SetPara(begin,end,step,IterationNum,type)
54 ScaleMode=1
55
56 timelist=CMP.MatrixMultime(ScaleMode)
57 Xd=range(begin,end,step)
58
59 myPlot = PlotXY()
60 myLayer1 = LayerXY(Double1d(Xd),Double1d(timelist[0]),color=java.awt.Color.red)
```

```

61 myLayer2 = LayerXY(Double1d(Xd),Double1d(timelist[1]),color=java.awt.Color.green)
62 myLayer1.line = Style.MARKED
63 myLayer2.line = Style.MARK_DASHED
64 myLayer1.name="float Matrixmultiply by pure java "
65 myLayer2.name="float Matrixmultiply by Jcublas"
66 myPlot.legend.visible = 1
67 myPlot.addLayer(myLayer1)
68 myPlot.addLayer(myLayer2)
69
70
71 type=1
72 timelist=CMP.MatrixMultime(ScaleMode)
73 myLayer3 = LayerXY(Double1d(Xd),Double1d(timelist[0]),color=java.awt.Color.blue)
74 myLayer4 = LayerXY(Double1d(Xd),Double1d(timelist[1]),color=java.awt.Color.black)
75 myLayer3.line = Style.MARKED
76 myLayer4.line = Style.MARK_DASHED
77 myLayer3.name="double Matrixmultiply by pure java "
78 myLayer4.name="double Matrixmultiply by Jcublas"
79 myPlot.legend.visible = 1
80 myPlot.addLayer(myLayer3)
81 myPlot.addLayer(myLayer4)
82
83 #myPlot.xaxis.type = Axis.LOG
84 #myPlot.yaxis.type = Axis.LOG
85 myPlot.titleText = "Comparison with CUDA and pure Java"
86 myPlot.subtitleText = "MatrixMultiply class(Float/Double)"
87 myPlot.xaxis.titleText = "number of elements "
88 myPlot.yaxis.titleText = "time used"
89 myPlot.saveAsEPS(Path+"cujaMatrixLine.eps")
90

```

B.1.2 快速傅里叶变换

快速傅里叶变换的CPU实现和GPU实现效率对比，代码文件名为CuJaFFTCMP.py

```

1 # the Time Used Comparison about FFT and CudaFFT
2
3 begin = 1

```

```
4 end = 20
5 step = 1
6 IterationNum=50
7 Direction=-1
8 Path="C:\\\\Users\\\\yfjin\\\\hcss\\\\workresult\\\\FFT\\\\"
9 ScaleMode=0
10 CMP=FFTCmp(begin,end,step,IterationNum,Direction)
11 timelist=CMP.FFTtime(ScaleMode)
12 Xd=range(begin,end,step)
13
14 myPlot = PlotXY()
15 myLayer1 = LayerXY(Double1d(Xd),Double1d(timelist[0]),color=java.awt.Color.red)
16 myLayer2 = LayerXY(Double1d(Xd),Double1d(timelist[1]),color=java.awt.Color.green)
17 myLayer1.line = Style.MARKED
18 myLayer2.line = Style.MARK_DASHED
19 myLayer1.name="FFT by pure java"
20 myLayer2.name="FFT by Jcufft"
21 myPlot.legend.visible = 1
22 myPlot.addLayer(myLayer1)
23 myPlot.addLayer(myLayer2)
24
25 Direction=1
26 timelist=CMP.FFTtime(ScaleMode)
27 Xd=range(begin,end,step)
28
29 myLayer3 = LayerXY(Double1d(Xd),Double1d(timelist[0]),color=java.awt.Color.yellow)
30 myLayer4 = LayerXY(Double1d(Xd),Double1d(timelist[1]),color=java.awt.Color.blue)
31 myLayer3.line = Style.MARKED
32 myLayer4.line = Style.MARK_DASHED
33 myLayer3.name="IFFT by pure java"
34 myLayer4.name="IFFT by Jcufft"
35 myPlot.legend.visible = 1
36 myPlot.addLayer(myLayer3)
37 myPlot.addLayer(myLayer4)
38
39
40 myPlot.xaxis.type = Axis.LINEAR
41 myPlot.yaxis.type = Axis.LOG
42 myPlot.titleText = "Comparison with CUDA and pure Java"
```

```
43 myPlot.subtitleText = "FFT class(Double)"
44 myPlot.xaxis.titleText = "number of elements /log2(N)"
45 myPlot.yaxis.titleText = "Time"
46 myPlot.saveAsEPS(Path+"FFTCudaJava1.eps") # Encapsulated PS
47
48 ## the linear rerun
49 begin=1
50 end=15001
51 step=1000
52 IterationNum=50
53 Direction=-1
54 Path="C:\\\\Users\\\\yfjin\\\\hcss\\\\workresult\\\\FFT\\\\"
55 CMP.SetPara(begin,end,step,IterationNum,Direction)
56 ScaleMode=1
57
58
59 timelist=CMP.FFTtime(ScaleMode)
60 Xd=range(begin,end,step)
61
62
63 myPlot = PlotXY()
64 myLayer1 = LayerXY(Double1d(Xd),Double1d(timelist[0]),color=java.awt.Color.red)
65 myLayer2 = LayerXY(Double1d(Xd),Double1d(timelist[1]),color=java.awt.Color.green)
66 myLayer1.line = Style.MARKED
67 myLayer2.line = Style.MARK_DASHED
68 myLayer1.name="FFT by pure java"
69 myLayer2.name="FFT by Jcufft"
70 myPlot.legend.visible = 1
71 myPlot.addLayer(myLayer1)
72 myPlot.addLayer(myLayer2)
73
74 Direction=1
75 timelist=CMP.FFTtime(ScaleMode)
76 Xd=range(begin,end,step)
77
78 myLayer3 = LayerXY(Double1d(Xd),Double1d(timelist[0]),color=java.awt.Color.yellow)
79 myLayer4 = LayerXY(Double1d(Xd),Double1d(timelist[1]),color=java.awt.Color.blue)
80 myLayer3.line = Style.MARKED
81 myLayer4.line = Style.MARK_DASHED
```

```

82 myLayer3.name="IFFT by pure java"
83 myLayer4.name="IFFT by Jcufft"
84 myPlot.legend.visible = 1
85 myPlot.addLayer(myLayer3)
86 myPlot.addLayer(myLayer4)

87

88

89 #myPlot.xaxis.type = Axis.LOG
90 #myPlot.yaxis.type = Axis.LOG
91 myPlot.titleText = "Comparison with CUDA and pure Java"
92 myPlot.subtitleText = "FFT class(Double)"
93 myPlot.xaxis.titleText = "number of elements "
94 myPlot.yaxis.titleText = "time used"
95 myPlot.saveAsEPS(Path+"FFTCudaJava2.eps") # Encapsula
96

```

B.2 HCSS 脚本分析框架

HCSS脚本分析主文件，负责调用预处理文件，运行要分析处理的脚本，调用后续处理文件。代码文件名为Main.py

```

1 # the main Function File
2
3
4 #####----- preprocess -----#####
5
6 # import the necessary module
7 import sys
8 import time
9 import os
10 from herschel.ia.all import *
11
12 # define the path of workspace
13 sys.path.append("C:\\\\Users\\\\yfjin\\\\hcsm\\\\workspace\\\\")
14 path="C:\\\\Users\\\\yfjin\\\\hcsm\\\\workspace\\\\"
15 #pipeline="Photometer_Small_Map_Pipeline"
16 #pipeline="Spectrometer_Mapping_Pipeline"

```

```
17 #pipeline="Photometer_Large_Map_Pipeline"
18 pipeline="scanmap_Deep_Survey_miniscan_Pointsource"
19 # Reset the Counter
20 Double1dNumCount.ReSet()
21 Double2dNumCount.ReSet()
22 Complex1dNumCount.ReSet()
23 MatrixNumCount.ReSet()
24 FFTNumCount.ReSet()
25 SVDNumCount.ReSet()
26 # start the stopwatch
27 beginPipeline=time.clock()
28
29 #####----- execute the pipeline -----#####
30
31 execfile(path+pipeline+".py")
32
33 #####----- post-processing -----#####
34
35 # output the NumCount results. the result output in the Dos Windows
36
37 endPipeline=time.clock()
38 TPipeline=endPipeline-beginPipeline
39 print TPipeline
40 Double1dNumCount.Double1dGetALL()
41 Double2dNumCount.Double2dGetALL()
42 Complex1dNumCount.Complex1dGetALL()
43 MatrixNumCount.MatrixGetALL()
44 FFTNumCount.FFTGetALL()
45 SVDNumCount.SVDGetALL()
46
47 # execute the post-processing about NumCount.
48 # do something like draw the result and
49 # output the result to memory
50 execfile(path+"FFTNumCount.py")
51 execfile(path+"Double1dNumCount.py")
52 execfile(path+"Complex1dNumCount.py")
53 execfile(path+"Double2dNumCount.py")
54 execfile(path+"SVDNumCount.py")
55 execfile(path+"MatrixNumCount.py")
```

```

56
57 # execute the post-processing about TimeCost.
58 # each meshod's time cost
59 execfile(path+"FFTTimeCost.py")
60 execfile(path+"Double1dFitTime.py")
61 execfile(path+"Complex1dFitTime.py")
62 #execfile(path+"Double2dTimeCost.py")
63 execfile(path+"MatrixTimeCost.py")
64 execfile(path+"SVDTimeCost.py")
65
66

```

B.3 Complex1d 计数和时间花费分析

B.3.1 Complex1d 计数分析

Complex1d后续处理脚本，用于计数处理，代码文件名为Complex1dNumCount.py。

```

1 # the Complex1dNumCount Program analysis
2
3
4 OptArray=[Complex1dNumCount.Add1,Complex1dNumCount.Add2,Complex1dNumCount.Sub1,\n
5     Complex1dNumCount.Sub2,Complex1dNumCount.Mul1,Complex1dNumCount.Mul2,\n
6     Complex1dNumCount.Div1,Complex1dNumCount.Div2,Complex1dNumCount.Pow,\n
7     Complex1dNumCount.Neg,Complex1dNumCount.Mod,Complex1dNumCount.Abs]
8
9 Path="C:\\\\Users\\\\yfjin\\\\hcsl\\\\workresult\\\\Complex1d\\\\"
10 Complex1dNumCount.Complex1dNcOutputALL(Path)
11
12 for Opt in OptArray:
13     s=Complex1dNumCount.Complex1dGetDt1(Opt)
14     if(len(s[0])==0):
15         continue
16     ss=Complex1dNumCount.Complex1dGetDim(s)
17     Array=Long1d(ss)
18     Dim=len(Array)

```

```

19
20     myPlot=PlotXY()
21     myLayer1 = LayerXY(Double1d(range(Dim)),Double1d(Array),color=java.awt.Color.blue)
22     myPlot.addLayer(myLayer1)
23     myPlot.xaxis.type = Axis.LINEAR
24     myPlot.yaxis.type = Axis.LOG
25     myPlot.titleText = "Number Count of the Complex1d " + Opt.toString() + " Operation"
26     myPlot.xaxis.titleText = "Array Scale of the Complex1d"
27     myPlot.yaxis.titleText = "Number"
28     myPlot.saveAsPNG(Path+Opt.toString()+"file.png")
29     myPlot.saveAsEPS(Path+Opt.toString()+"file.eps")
30     myPlot.close()
31

```

B.3.2 Complex1d 时间花费分析

Complex1d后续处理脚本，用于模拟并拟合目标脚本的Complex1d类时间花费，代码文件名为Comolex1dFitTime.py。

```

1 # fit the time curve
2 # the function clip Used for clear the strange point
3 # fit the time curve
4 # the function clip Used for clear the strange point
5 def clip(ksigma,n,Xrg,Yrg,Wrg):
6     myModel = PolynomialModel(1)
7     myFitter = Fitter(Xrg, myModel)
8     fitresults = myFitter.fit(Yrg,Wrg)
9     Yrg1=myModel(Xrg)
10    Variance=0
11    N=0
12    for i in range(len(Xrg)):
13        if(Wrg[i]!=0):
14            Variance=Variance+POW(ABS(Yrg[i]-Yrg1[i]),2)
15            N=N+1
16    Variance=Variance/N
17    Sigma=SQRT(Variance)
18
19    if ABS(Yrg[n]-Yrg1[n]) > ksigma*Sigma:

```

```
20         return 0
21     else:
22         return 1
23
24
25 def cliper(ksigma,X,Y,W):
26     lenX=len(X)
27     for i in range(5):
28         rg=Range(i,i+10)
29         Xrg=X.get(rg)
30         Yrg=Y.get(rg)
31         Wrg=W.get(rg)
32         Wrg.set(0,0)
33         W[i]=clip(ksigma,0,Xrg,Yrg,Wrg)
34         for i in range(5,lenX-5):
35             rg=Range(i-5,i+5)
36             Xrg=X.get(rg)
37             Yrg=Y.get(rg)
38             Wrg=W.get(rg)
39             Wrg.set(5,0)
40             W[i]=clip(ksigma,5,Xrg,Yrg,Wrg)
41             for i in range(lenX-5,lenX):
42                 rg=Range(i-10,i)
43                 Xrg=X.get(rg)
44                 Yrg=Y.get(rg)
45                 Wrg=W.get(rg)
46                 Wrg.set(9,0)
47                 W[i]=clip(ksigma,9,Xrg,Yrg,Wrg)
48             return W
49
50 # the function fintime Used for fitting the time curve
51 def fitTime(Opt,begin,end,step):
52     Y=[]
53     for i in range(begin,end,step):
54         Y.append(Complex1dTimeCost.TimeCal(Opt,i))
55     Y=Double1d(Y)
56     X=Double1d(range(begin,end,step))
57     myModel = PolynomialModel(1)
58     myFitter = Fitter(X, myModel)
```

```
59     # result 1
60     fitresults1 = myFitter.fit(Y)
61
62     myPlot = PlotXY()
63     myLayer1 = LayerXY(Double1d(X),Double1d(Y),color=java.awt.Color.blue)
64     myLayer1.name="real time cost curve of " +Opt
65     myLayer2 = LayerXY(Double1d(X),myModel(X),color=java.awt.Color.red)
66     myLayer2.name="fit curve of time cost of " +Opt
67     myPlot.addLayer(myLayer1)
68     myPlot.addLayer(myLayer2)
69     myPlot.legend.visible =1
70     myPlot.titleText = "Time Used of the Complex1d " + Opt +"Operation"
71     myPlot.xaxis.titleText = "The number of elements in one dimension"
72     myPlot.yaxis.titleText = "The time cost"
73
74     W=[]
75     lenX=len(X)
76     for i in range(lenX):
77         W.append(1)
78         W=Double1d(W)
79         W=cliper(5,X,Y,W)
80         W=Double1d(W)
81         yWeight=cliper(3,X,Y,W)
82
83         yWeights=Double1d(yWeight)
84         #print yWeights
85         fitresults2 = myFitter.fit(Y, yWeights)
86
87         myLayer3 = LayerXY(Double1d(X),myModel(X),color=java.awt.Color.green)
88         myLayer3.name="fit curve without singular point "+Opt
89         myPlot.addLayer(myLayer3)
90         myPlot.saveAsPNG(Path+Opt+"JavaTime.png")
91         myPlot.close()
92         return myModel
93
94
95     # the function TimeUsed Used for input the CountData
96     # and calculate each Operator's time cost
97     def TimeUsed(Opt,Path,begin,end,step):
```

```
98     theTimeUsed=0
99
100    myData=Complex1dTimeCost.TcInput(Opt,Path)
101    if(len(myData[0])==0):
102        return 0
103    myData=Long2d(myData)
104    nums=myData.getDimension(1)
105    rg=Range(0,nums)
106    myData1=myData.get(0,rg)
107    myData2=myData.get(1,rg)
108    Dim=myData.get(0,nums-1)
109    ss=Double1d(range(Dim+1))
110    myModel=fitTime(Opt,begin,end,step)
111    X=myModel(ss)
112
113    myData2=Double1d(myData2)
114    for i in range(nums):
115        j=myData1[i]
116        theTimeUsed=theTimeUsed+X[j]*myData2[i]
117    return theTimeUsed
118
119
120
121 Path="C:\\\\Users\\\\yfjin\\\\hcss\\\\workresult\\\\Complex1d\\\\"
122 begin=1
123 end=20001
124 step=500
125
126
127 SmallPipelineTimeUsed={"Add1":0,"Add2":0,"Sub1":0,"Sub2":0,"Mul1":0,"Mul2":0\
128             , "Div1":0,"Div2":0,"Pow":0,"Neg":0,"Abs":0,"Mod":0}
129 Operation=["Add1","Add2","Sub1","Sub2","Mul1","Mul2","Div1",\
130             "Div2","Pow","Neg","Abs","Mod"]
131 OptArray=[Complex1dNumCount.Add1,Complex1dNumCount.Add2,Complex1dNumCount.Sub1, \
132             Complex1dNumCount.Sub2,Complex1dNumCount.Mul1,Complex1dNumCount.Mul2, \
133             Complex1dNumCount.Div1,Complex1dNumCount.Div2,Complex1dNumCount.Pow, \
134             Complex1dNumCount.Neg,Complex1dNumCount.Mod,Complex1dNumCount.Abs]
135
136 SPTimeUsed=[]
```

```

137 AllTime=0
138 for Opt in Operation:
139     OptTimeUsed=TimeUsed(Opt,Path,begin,end,step)
140     AllTime=AllTime+OptTimeUsed
141     SPTimeUsed.append(OptTimeUsed)
142     SmallPipelineTimeUsed[Opt]=OptTimeUsed
143
144 print SmallPipelineTimeUsed
145 fp = open(Path+"result.txt",'w')
146 print >> fp, SmallPipelineTimeUsed
147 fp.close()
148
149 myPlot = PlotXY()
150 myLayer1 = LayerXY(Double1d(range(len(Operation))), \
151                     Double1d(SPTimeUsed),color=java.awt.Color.blue)
152 myLayer1.name="real time used of small map pipeline"
153 myPlot.addLayer(myLayer1)
154 myPlot.legend.visible =0
155 myPlot.titleText = "real time used of small map pipeline"
156 myPlot.xaxis.titleText = "Variety of the Calculate Operation"
157 myPlot.yaxis.titleText = "Time Used"
158 myPlot.saveAsPNG(Path+"AllOptTime.png")
159 myPlot.saveAsEPS(Path+"AllOptTime.eps")
160 #myPlot.close()
161

```

B.4 Double1d 计数和时间花费分析

B.4.1 Double1d 计数分析

Double1d后续处理脚本，用于计数处理，代码文件名为Double1dNumCount.py。

```

1 # the Double1dNumCount Program analysis
2
3 OptArray=[Double1dNumCount.Add1,Double1dNumCount.Add2,Double1dNumCount.Sub1, \
4           Double1dNumCount.Sub2,Double1dNumCount.Mul1,Double1dNumCount.Mul2, \

```

```

5      Double1dNumCount.Div1,Double1dNumCount.Div2,Double1dNumCount.Pow,\n
6      Double1dNumCount.Neg,Double1dNumCount.Mod,Double1dNumCount.Abs,\n
7      Double1dNumCount.DotProduct1,Double1dNumCount.DotProduct2,\n
8      Double1dNumCount.OutProduct]\n
9  Path="C:\\\\Users\\\\yfjin\\\\hcsl\\\\workresult\\\\Double1d\\\\"\n
10 Double1dNumCount.Double1dNcOutputALL(Path)\n
11\n
12 for Opt in OptArray:\n
13     s=Double1dNumCount.Double1dGetDtl(Opt)\n
14     if(len(s[0])==0):\n
15         continue\n
16     ss=Double1dNumCount.Double1dGetDim(s)\n
17     Array=Long1d(ss)\n
18     Dim=len(Array)\n
19\n
20     myPlot=PlotXY()\n
21     myLayer1 = LayerXY(Double1d(range(Dim)),Double1d(Array),color=java.awt.Color.blue)\n
22     myPlot.addLayer(myLayer1)\n
23     myPlot.titleText = "the Number Count of the Double1d " + Opt.toString() + " Operation"\n
24     myPlot.xaxis.type = Axis.LINEAR\n
25     myPlot.yaxis.type = Axis.LOG\n
26     myPlot.xaxis.titleText = "Array Scale"\n
27     myPlot.yaxis.titleText = "Number"\n
28     myPlot.saveAsPNG(Path+Opt.toString()+"file.png")\n
29     myPlot.saveAsEPS(Path+Opt.toString()+"file.eps")\n
30     myPlot.close()\n
31

```

B.4.2 Double1d 时间花费分析

Double1d后续处理脚本，用于模拟并拟合目标脚本的Double1d类时间花费，代码文件名为Double1dFitTime.py。

```

1 # fit the time curve\n
2 # the function clip Used for clear the strange point\n
3 def clip(ksigma,n,Xrg,Yrg,Wrg):\n
4     myModel = PolynomialModel(1)\n
5     myFitter = Fitter(Xrg, myModel)

```

```
6     fitresults = myFitter.fit(Yrg,Wrg)
7     Yrg1=myModel(Xrg)
8     Variance=0
9     N=0
10    for i in range(len(Xrg)):
11        if(Wrg[i]!=0):
12            Variance=Variance+POW(ABS(Yrg[i]-Yrg1[i]),2)
13            N=N+1
14    Variance=Variance/N
15    Sigma=SQRT(Variance)
16
17    if ABS(Yrg[n]-Yrg1[n]) > ksigma*Sigma:
18        return 0
19    else:
20        return 1
21
22
23 def cliper(ksigma,X,Y,W):
24     lenX=len(X)
25     for i in range(5):
26         rg=Range(i,i+10)
27         Xrg=X.get(rg)
28         Yrg=Y.get(rg)
29         Wrg=W.get(rg)
30         Wrg.set(0,0)
31         W[i]=clip(ksigma,0,Xrg,Yrg,Wrg)
32         for i in range(5,lenX-5):
33             rg=Range(i-5,i+5)
34             Xrg=X.get(rg)
35             Yrg=Y.get(rg)
36             Wrg=W.get(rg)
37             Wrg.set(5,0)
38             W[i]=clip(ksigma,5,Xrg,Yrg,Wrg)
39             for i in range(lenX-5,lenX):
40                 rg=Range(i-10,i)
41                 Xrg=X.get(rg)
42                 Yrg=Y.get(rg)
43                 Wrg=W.get(rg)
44                 Wrg.set(9,0)
```

```
45     W[i]=clip(ksigma,9,Xrg,Yrg,Wrg)
46     return W
47
48 # the function fittime Used for fitting the time curve
49 def fitTime(Opt,begin,end,step):
50     Y=[]
51     for i in range(begin,end,step):
52         Y.append(Double1dTimeCost.TimeCal(Opt,i))
53     Y=Double1d(Y)
54     X=Double1d(range(begin,end,step))
55     myModel = PolynomialModel(1)
56     myFitter = Fitter(X, myModel)
57     # result 1
58     fitresults1 = myFitter.fit(Y)
59
60     myPlot = PlotXY()
61     myLayer1 = LayerXY(Double1d(X),Double1d(Y),color=java.awt.Color.blue)
62     myLayer1.name="real time cost curve of " +Opt
63     myLayer2 = LayerXY(Double1d(X),myModel(X),color=java.awt.Color.red)
64     myLayer2.name="fit curve of time cost of " +Opt
65     myPlot.addLayer(myLayer1)
66     myPlot.addLayer(myLayer2)
67     myPlot.legend.visible =1
68     myPlot.titleText = "Time Used of the Double1d " + Opt +"Operation"
69     myPlot.xaxis.titleText = "number of elements in one dimension"
70     myPlot.yaxis.titleText = "Time"
71
72     W=[]
73     lenX=len(X)
74     for i in range(lenX):
75         W.append(1)
76         W=Double1d(W)
77         W=cliper(5,X,Y,W)
78         W=Double1d(W)
79         yWeight=cliper(3,X,Y,W)
80
81         yWeights=Double1d(yWeight)
82         #print yWeights
83         fitresults2 = myFitter.fit(Y, yWeights)
```

```
84
85     myLayer3 = LayerXY(Double1d(X),myModel(X),color=java.awt.Color.green)
86     myLayer3.name="fit curve without singular point "+Opt
87     myPlot.addLayer(myLayer3)
88     myPlot.saveAsPNG(Path+Opt+"JavaTime.png")
89     myPlot.close()
90     return myModel
91
92
93 # the function TimeUsed Used for input the CountData
94 # and calculate each Operator's time cost
95 def TimeUsed(Opt,Path,begin,end,step):
96     theTimeUsed=0
97
98     myData=Double1dTimeCost.TcInput(Opt,Path)
99     if(len(myData[0])==0):
100         return 0
101     myData=Long2d(myData)
102     nums=myData.getDimension(1)
103     rg=Range(0,nums)
104     myData1=myData.get(0,rg)
105     myData2=myData.get(1,rg)
106     Dim=myData.get(0,nums-1)
107     ss=Double1d(range(Dim+1))
108     myModel=fittime(Opt,begin,end,step)
109     X=myModel(ss)
110
111     myData2=Double1d(myData2)
112     for i in range(nums):
113         j=myData1[i]
114         theTimeUsed=theTimeUsed+X[j]*myData2[i]
115     return theTimeUsed
116
117
118
119 Path="C:\\\\Users\\\\yfjin\\\\hcsl\\\\workresult\\\\Double1d\\\\"
120 begin=1
121 end=10001
122 step=100
```

```
123
124
125 SmallPipelineTimeUsed={"Add1":0,"Add2":0,"Sub1":0,"Sub2":0,"Mul1":0,"Mul2":0,\n
126     "Div1":0,"Div2":0,"Pow":0,"Neg":0,"Abs":0,"Mod":0,\n
127     "DotProduct1":0,"DotProduct2":0,"OutProduct":0}
128 Operation=["Add1","Add2","Sub1","Sub2","Mul1","Mul2","Div1","Div2",\n
129     "Pow","Neg","Abs","Mod","DotProduct1"]
130 OptArray=[Double1dNumCount.Add1,Double1dNumCount.Add2,Double1dNumCount.Sub1,\n
131     Double1dNumCount.Sub2,Double1dNumCount.Mul1,Double1dNumCount.Mul2,\n
132     Double1dNumCount.Div1,Double1dNumCount.Div2,Double1dNumCount.Pow,\n
133     Double1dNumCount.Neg,Double1dNumCount.Mod,Double1dNumCount.Abs,\n
134     Double1dNumCount.DotProduct1,Double1dNumCount.DotProduct2,\n
135     Double1dNumCount.OutProduct]
136
137 SPTimeUsed=[]
138 AllTime=0
139 for Opt in Operation:
140     OptTimeUsed=TimeUsed(Opt,Path,begin,end,step)
141     AllTime=AllTime+OptTimeUsed
142     SPTimeUsed.append(OptTimeUsed)
143     SmallPipelineTimeUsed[Opt]=OptTimeUsed
144
145 print AllTime
146 print SmallPipelineTimeUsed
147 fp = open(Path+"result.txt",'w')
148 print >> fp, SmallPipelineTimeUsed
149 print >> fp, "AllTime = ", AllTime
150 fp.close()
151
152 myPlot = PlotXY()
153 myLayer1 = LayerXY(Double1d(range(len(Operation))),\
154     Double1d(SPTimeUsed),color=java.awt.Color.blue)
155 myLayer1.name="real time used of Spectrometer pipeline"
156 myPlot.addLayer(myLayer1)
157 myPlot.legend.visible =0
158 myPlot.titleText = "real time used of the pipeline"
159 myPlot.xaxis.titleText = "Variety of the Calculate Operation"
160 myPlot.yaxis.titleText = "Time Used"
161 myPlot.saveAsPNG(Path+"AllOptTime.png")
```

```

162 myPlot.saveAsEPS(Path+"AllOptTime.eps")
163 myPlot.close()

```

B.5 Double2d 计数和时间花费分析

B.5.1 Double2d 计数分析

Double2d后续处理脚本，用于计数处理，代码文件名为Double2dNumCount.py。

```

1 Path="C:\\\\Users\\\\yfjin\\\\hcss\\\\workresult\\\\Double2d\\\\"
2
3 Double2dNumCount.Double2dGetALL()
4 OptArray=[Double2dNumCount.Add1,Double2dNumCount.Add2,Double2dNumCount.Sub1, \
5 Double2dNumCount.Sub2,Double2dNumCount.Mul1,Double2dNumCount.Mul2, \
6 Double2dNumCount.Div1,Double2dNumCount.Div2,Double2dNumCount.Abs, \
7 Double2dNumCount.Pow,Double2dNumCount.Mod,Double2dNumCount.Neg, \
8 Double2dNumCount.DotProduct1,Double2dNumCount.DotProduct2]
9 for Opt in OptArray:
10    Double2dNumCount.Double2dNcOutput(Opt,Opt._name,Path)

```

B.5.2 Double2d 时间花费分析

Double2d后续处理脚本，用于模拟目标脚本的Double2d类时间花费，代码文件名为Double2dTimeCost.py。

```

1 # MatrixMultiply TimeUsed Calculate
2 Path="C:\\\\Users\\\\yfjin\\\\hcss\\\\workresult\\\\Double2d\\\\"
3 OptArray=[Double2dNumCount.Add1,Double2dNumCount.Add2,Double2dNumCount.Sub1, \
4 Double2dNumCount.Sub2,Double2dNumCount.Mul1,Double2dNumCount.Mul2, \
5 Double2dNumCount.Div1,Double2dNumCount.Div2,Double2dNumCount.DotProduct1, \
6 Double2dNumCount.DotProduct2,Double2dNumCount.Pow,Double2dNumCount.Abs, \
7 Double2dNumCount.Neg,Double2dNumCount.Mod]
8 TimeUsed={"Add1":0,"Add2":0,"Sub1":0,"Sub2":0,"Mul1":0,"Mul2":0,"Div1":0,"Div2":0, \
9 "DotProduct1":0,"DotProduct2":0,"Pow":0,"Neg":0,"Abs":0,"Mod":0,}

```

```

10 timeList=[]
11 #MaTimeCost=MatrixTimeCost()
12 D2dTimeCost=Double2dTimeCost("Double2dTimeCost")
13 for Opt in OptArray :
14     D2dtime=D2dTimeCost.TimeCost(Opt,Opt._name,Path)
15     timeList.append(D2dtime)
16     D2dTimeUsed[Opt._name]=D2dtime
17
18 print "Double2d Time Cost = ", timeList
19
20 fp = open(Path+"result.txt",'w')
21 print >> fp, "Double2d Time Cost = ", timeList
22 fp.close()
23
24
25

```

B.6 FFT 计数和时间花费分析

B.6.1 FFT 计数分析

FFT后续处理脚本，用于计数处理，代码文件名为FFTNumCount.py。

```

1 # FFT number Count
2 Path="C:\\\\Users\\\\yfjin\\\\hcss\\\\workresult\\\\FFT\\\\"
3
4 FFTNumCount.FFTGetALL()
5 FFTNumCount.FFTNcOutputALL(Path)
6 OptArray=[FFTNumCount.NCfft,FFTNumCount.NCifft]
7
8 FFTGetD=Long1d(FFTNumCount.FFTGetDim(FFTNumCount.NCfft))
9 Dim=len(FFTGetD)
10 FFTPlot=PlotXY()
11 FFTLayer1 = LayerXY(Double1d(range(Dim)),Double1d(FFTGetD),color=java.awt.Color.blue)
12 FFTPlot.addLayer(FFTLayer1)
13 FFTPlot.xaxis.type = Axis.LINEAR
14 FFTPlot.yaxis.type = Axis.LOG

```

```

15 FFTPlot.titleText = "Number Count of the Complex1d FFT Operation"
16 FFTPlot.xaxis.titleText = "Array Scale"
17 FFTPlot.yaxis.titleText = "number Count"
18 FFTPlot.saveAsPNG(Path+"FFTfile.png")
19 FFTPlot.saveAsEPS(Path+"FFTfile.eps")
20 FFTPlot.close()

```

B.6.2 FFT 时间花费分析

FFT后续处理脚本，用于模拟目标脚本的FFT类时间花费，代码文件名为FFTTimeCost.py。

```

1 # FFT Time Cost
2 Path="C:\\Users\\yfjin\\hcss\\workresult\\FFT\\"
3 OpsArray=["NCfft","NCifft"]
4 AllTime=0
5 for Ops in OpsArray:
6     AllTime=AllTime+FFTTimeCost.TimeCost(Ops,Path)
7     print AllTime
8 print "FFT Time Cost = ", AllTime
9 fp = open(Path+"result.txt",'w')
10 print >> fp,"FFT Time Cost = ", AllTime
11 fp.close()

```

B.7 MatrixMultiply 计数和时间花费分析

B.7.1 MatrixMultiply 计数分析

MatrixMultiply后续处理脚本，用于计数处理，代码文件名为MatrixNumCount.py。

```

1 # MatrixMultiply Number Count
2
3 MatrixNumCount.MatrixGetALL()
4
5 Path="C:\\Users\\yfjin\\hcss\\workresult\\Matrix\\"

```

```
6
7 OptArray=[MatrixNumCount.Bool1,MatrixNumCount.Bool2,MatrixNumCount.Byte1,\ 
8 MatrixNumCount.Byte2,MatrixNumCount.Short1,MatrixNumCount.Short2,\ 
9 MatrixNumCount.Int1,MatrixNumCount.Int2,MatrixNumCount.Long1,\ 
10 MatrixNumCount.Long2,MatrixNumCount.Float1,MatrixNumCount.Float2,\ 
11 MatrixNumCount.Double1,MatrixNumCount.Double2]
12
13 for Opt in OptArray :
14     MatrixNumCount.MatrixNcOutput(Opt,Opt._name,Path)
15     MNumCount=MatrixNumCount.MatrixOptGetV(Opt)
16     if(MNumCount > 0):
17         MNumCount=MatrixNumCount.MatrixDimGet(Opt)
18         MNumCount=Long2d(MNumCount)
19         detail=MatrixNumCount.MatrixNcInput(Opt,Opt.get_name(),\
20             "C:\\\\Users\\\\yfjin\\\\hcsl\\\\workresult\\\\Matrix\\\\")
21         detail=Long2d(detail)
22
23 p=PlotXY()
24     p.plotSize=(4,4)
25     p.setLayout.vgap=0
26     z_grid=Double1d(detail.get(1))
27     p1_all= Double1d(detail.get(3))
28     p1=SubPlot()
29     p.addSubPlot(p1)
30     p1L_all=LayerXY(z_grid, p1_all, \
31         name="the Matrix Multiply number",color=java.awt.Color.blue)
32     p1L_all.style=Style(chartType=Style.HISTOGRAM_EDGE)
33     p1L_all.xaxis=Axis(type=Axis.LINEAR, range=[0,3000])
34     p1L_all.yaxis=Axis(range=[0,160000])
35     p1L_all.xaxis.tick.setFixedValues(Double1d([0,500,1000,1500,2000,2500,3000]))
36     p1L_all.xaxis.getTick().setLineWidth(1)
37     p1L_all.xaxis.getTick().setHeight(0.05)
38     #p1L_all.xaxis.tick.visible=0
39     p1L_all.xaxis.tick.label.visible=1
40     p1L_all.xaxis.title.visible=1
41     p1L_all.xaxis.tick.label.fontSize=8
42     p1L_all.xaxis.title.text="Matrix dimension"
43     p1L_all.xaxis.title.fontSize=8
44     p1L_all.yaxis.tick.label.fontSize=8
```

```

45     p1L_all.yaxis.title.text="Number"
46     p1L_all.yaxis.title.fontSize=8
47     p1.addLayer(p1L_all)
48     p1L_all.xaxis.auxAxes[0].tick.visible=0
49
50
51     p.legend.visible=1
52     p.legend.columns=1
53     p.legend.position=PlotLegend.CUSTOMIZED
54     p.legend.setLocation(1,4.0)
55     p.legend.halign=PlotLegend.LEFT
56     p.legend.valign=PlotLegend.BOTTOM
57     p.legend.borderVisible = False
58     #p.addAnnotation(Annotation(390,2500,"SVD",color=BLACK,fontSize=11))
59     p.titleText="the detail count of MatrixMultiply"
60
61 #     p.close()

```

B.7.2 MatrixMulyiply 时间花费分析

MatrixMultiply后续处理脚本，用于模拟目标脚本的FFT类时间花费，代码文件名为MatrixTimeCost.py。

```

1  # MatrixMultiply TimeUsed Calculate
2  Path="C:\\\\Users\\\\yfjin\\\\hcss\\\\workresult\\\\Matrix\\\\"
3
4  OptArray=[MatrixNumCount.Bool1,MatrixNumCount.Bool2,MatrixNumCount.Byte1, \
5    MatrixNumCount.Byte2,MatrixNumCount.Short1,MatrixNumCount.Short2, \
6    MatrixNumCount.Int1,MatrixNumCount.Int2,MatrixNumCount.Long1, \
7    MatrixNumCount.Long2,MatrixNumCount.Float1,MatrixNumCount.Float2, \
8    MatrixNumCount.Double1,MatrixNumCount.Double2]
9
10
11 MaTimeUsed={"Bool1":0,"Bool2":0,"Byte1":0,"Byte2":0,"Short1":0,"Short2":0, \
12   "Int1":0,"Int2":0,"Long1":0,"Long2":0,"Float1":0,"Float2":0,"Double1":0,"Double2":0}
13 timeList=[]
14 #MaTimeCost=MatrixTimeCost()
15 MaTimeCost=MatrixTimeCost("MatrixTimeCost")

```

```

16 for Opt in OptArray :
17
18     Matime=MaTimeCost.TimeCost(Opt,Opt._name,Path)
19     timeList.append(Matime)
20     MaTimeUsed[Opt._name]=Matime
21
22 print "Matrix Multiply Time Cost = ", timeList
23
24 fp = open(Path+"result.txt",'w')
25 print >> fp, "Matrix Multiply Time Cost = ", timeList
26 fp.close()

```

B.8 SingularValueDecomposition 计数和时间花费分析

B.8.1 SVD 计数分析

SVD后续处理脚本，用于计数处理，代码文件名为SVDNumCount.py。

```

1 # SVD Number Count
2 SVDNumCount.SVDGetALL()
3
4 Path="C:\\\\Users\\\\yfjin\\\\hcsl\\\\workresult\\\\SVD\\\\"
5 SVDNumCount.SVDNcOutputALL(Path)
6 OptArray=[SVDNumCount.SVDF1,SVDNumCount.SVDF2,SVDNumCount.SVDD1,SVDNumCount.SVDD2]
7
8 for Opt in OptArray :
9     svd=SVDNumCount.SVDGetCount(Opt)
10    if(svd>0):
11        svd=SVDNumCount.SVDNcInput(Opt,Path)
12        svd=Long2d(svd)
13        print "the number of SVD= ", svd
14
15        p=PlotXY()
16        p.plotSize=(4,4)
17        p.setLayout.vgap=0

```

```
18     z_grid=Double1d(svd.get(0))
19     p1_all= Double1d(svd.get(2))
20     p1=SubPlot()
21     p.addSubPlot(p1)
22     p1L_all=LayerXY(z_grid, p1_all, name="the SVD number",color=java.awt.Color.blue)
23     p1L_all.style=Style(chartType=Style.HISTOGRAM_EDGE)
24     p1L_all.xaxis=Axis(type=Axis.LINEAR, range=[380,435])
25     p1L_all.yaxis=Axis(range=[0,7000])
26     p1L_all.xaxis.tick.setFixedValues(Double1d([390,400,410,420,430]))
27     p1L_all.xaxis.getTick().setLineWidth(1)
28     p1L_all.xaxis.getTick().setHeight(0.05)
29     #p1L_all.xaxis.tick.visible=0
30     p1L_all.xaxis.tick.label.visible=1
31     p1L_all.xaxis.title.visible=1
32     p1L_all.xaxis.title.fontSize=8
33     p1L_all.xaxis.title.text="SVD dimension"
34     p1L_all.xaxis.title.fontSize=8
35     p1L_all.yaxis.tick.label.fontSize=8
36     p1L_all.yaxis.title.text="Number"
37     p1L_all.yaxis.title.fontSize=8
38     p1.addLayer(p1L_all)
39     p1L_all.xaxis.auxAxes[0].tick.visible=0
40
41
42     p.legend.visible=1
43     p.legend.columns=1
44     p.legend.position=PlotLegend.CUSTOMIZED
45     p.legend.setLocation(1,4.0)
46     p.legend.halign=PlotLegend.LEFT
47     p.legend.valign=PlotLegend.BOTTOM
48     p.legend.borderVisible = False
49     #p.addAnnotation(Annotation(390,2500,"SVD",color=BLACK,fontSize=11))
50     p.titleText="the detail count of SingularValueDecomposition"
51     p.saveAsPNG(Path+"SVDfile.png")
52     p.saveAsEPS(Path+"SVDfile.eps")
53     #         p.close()
```

B.8.2 SVD 时间花费分析

SVD后续处理脚本，用于模拟目标脚本的FFT类时间花费，代码文件名为SVDTimeCost.py。

```
1 # SVD time cost
2 Path="C:\\\\Users\\\\yfjin\\\\hcss\\\\workresult\\\\SVD\\\\"
3 OpsArray=["SVDF1","SVDF2","SVDD1","SVDD2"]
4
5 svdd=SVDNumCount.SVDDnCInput(SVDNumCount.SVDD1,Path)
6 svdd=Long2d(svdd)
7 #print svdd
8
9 import time
10 timeBegin=time.clock()
11 AllTime=0
12 for Ops in OpsArray:
13     AllTime=AllTime+SVDTimeCost().TimeCost(Ops,Path)
14
15 print AllTime
16 timeEnd=time.clock()-timeBegin
17 print "SVD Time Cost = ", timeEnd
18 fp = open(Path+"result.txt",'w')
19 print >> fp, "SVD Time Cost = ", timeEnd
20 fp.close()
```

致谢

首先我非常感谢辛勤指导我工作的黄茂海研究员，黄老师不仅使我对工作的大体方向有了较好的把握，还抽出大量时间和我讨论工作细节，及时帮我引出误区，帮助我深入问题的本质，对本工作的完成起了巨大的作用。此外还要感谢黄老师为本工作提供了良好的硬件支持和工作环境。另外，我要感谢指导我HCSS软件编程的李波工程师和齐晓峰工程师，是他们细心的指导才是我很快的掌握了HCSS 软件编程的基本技能。最后还要感谢张墨学长、朱佳丽学长和其他为我提供支持和帮助的老师同学，感谢他们对本工作许多工作细节上的帮助和建议。

感谢中国科学院知识创新工程重要方向项目“中国参加赫歇尔空间天文台设备在轨观测运行和开展相关红外天文研究”(KJCX2-YW-T20)对本工作的支持。

北京大学学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

论文作者签名： 日期： 年 月 日

学位论文使用授权说明

(必须装订在提交学校图书馆的印刷本)

本人完全了解北京大学关于收集、保存、使用学位论文的规定，即：

- 按照学校要求提交学位论文的印刷本和电子版本；
- 学校有权保存学位论文的印刷本和电子版，并提供目录检索与阅览服务，在校园网上提供服务；
- 学校可以采用影印、缩印、数字化或其它复制手段保存论文；
- 因某种特殊原因需要延迟发布学位论文电子版，授权学校在 一年 / 两年 / 三年以后在校园网上全文发布。

(保密论文在解密后遵守此规定)

论文作者签名： 导师签名： 日期： 年 月 日