分类号 TP391 学号 GS11062185

U D C 密级 公 开

工程硕士学位论文

CPU/GPU 异构系统下高光谱遥感影像线性降维并行算法研究与实现

硕士生姓名 方民权

工 程 领 域 软件工程

研 究 方 向 高性能图像处理

指 导 教 师 周海芳 副研究员

国防科学技术大学研究生院

二〇一三年九月

Parallel Algorithm Research and Realization of Linear Dimensionality Reduction for Hyperspectral Image on CPU/GPU

Candidate：Fang Minquan

Advisor：Asso.Prof. Zhou Haifang

A thesis

Submitted in partial fulfillment of the requirements

for the degree of Master of Engineering

in Software Engineering

Graduate School of National University of Defense Technology

Changsha，Hunan，P.R.China

September，2013

独 创 性 声 明

本人声明所呈交的学位论文是我本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表和撰写过的研究成果，也不包含为获得国防科学技术大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文题目：CPU/GPU异构系统下高光谱遥感影像快性降维并行算法研究与实现

学位论文作者签名：方民权 日期：2013 年 9 月 22 日

学位论文版权使用授权书

本人完全了解国防科学技术大学有关保留、使用学位论文的规定。本人授权国防科学技术大学可以保留并向国家有关部门或机构送交论文的复印件和电子文档，允许论文被查阅和借阅；可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密学位论文在解密后适用本授权书。）

学位论文题目：CPU/GPU异构系统下高光谱遥感影像快性降维并行算法研究与实现

学位论文作者签名：方民权 日期：2013年 9 月 22 日

作者指导教师签名：周湘芳 日期：2013年 9 月 22 日

目 录

表 目 录

图 目 录

摘 要

如何对高光谱信息进行高效处理是遥感领域近年来的研究热点，高光谱图像降维是做好后续地物分析识别的一个关键步骤。由于降维计算是一个典型的计算密集和访存密集型过程，计算复杂度较高，采用传统串行处理模式已无法满足军事、农林等高端应用实时处理的需求。CPU+GPU 异构系统可以满足应用对高度密集计算的需求，是目前乃至未来很长一段时间内高性能计算机体系结构主流发展方向之一。如何充分利用 CPU/GPU 异构系统的计算能力，有效提高高光谱遥感图像降维的处理速度是本文研究的关键。

CPU/GPU 异构系统具有 3 个并行层次，分别是结点间进程级、结点内线程级和 GPU 线程级。本文针对高光谱遥感影像线性降维中的 PCA 和 FastICA 两种典型降维算法，分析研究算法中多个加速热点的并行处理方案和优化策略，基于 MPI、openMP 和 CUDA 等 3 种主流的并行编程模型设计和实现上述 3 个层次的多级协同并行算法。本文的主要工作和创新有以下几点：

（1）深入研究了 CPU/GPU 异构系统 3 种主流并行模式。异构系统中，除了 GPU 的密集计算资源外，一般还包括多 CPU、多 GPU、多结点等特征，要完全开发系统的性能，必须熟练掌握 MPI、openMP 和 CUDA 等主流并行技术。本文重点针对这 3 种并行技术的编程模型和优化策略进行了深入研究，并研究了一些混合并行的优化策略。

（2）基于 CPU/GPU 异构系统研究并提出了一系列高光谱影像 PCA 并行降维算法。在分析 PCA 算法加速热点的基础上，提出了适用于不同并行层次的任务划分方案，给出了协方差矩阵计算、PCA 变换、高光谱数据 I/O 等热点步骤的并行及优化策略；分别基于 MPI、openMP、CUDA、MPI+CUDA、openMP+CUDA 和 MPI+openMP+CUDA 等单一或混合编程模式提出并实现了 6 种 PCA 多级并行降维算法。实验结果表明，6 种并行算法均获得了显著的性能提升，其中 MPI+openMP+CUDA 三级协同并行 PCA 降维算法获得了最高 145 倍的计算加速比（不含 I/O）和最高 128 倍的总加速比（含 I/O）。

（3）基于 CPU/GPU 异构系统研究并提出了一系列高光谱影像 FastICA 并行降维算法。在分析 FastICA 算法加速热点基础上，提出了白化处理、ICA 迭代、IC 变换等热点步骤的并行及优化策略，特别是针对复杂的 ICA 迭代过程，提出了一种“具体-抽象-具体”的并行设计模式，并将该模式应用于 ICA 迭代过程的并行程序设计；结合协方差矩阵、高光谱数据 I/O 等并行优化策略，提出了 6 种 FastICA 多级并行降维算法。实验结果显示，6 种并行算法均获得了良好的加速效果，其中 MPI+openMP+CUDA 三级并行 FastICA 降维算法获得了最高 169 倍的计算加速比和 159 倍的总加速比。

主题词：高光谱降维；PCA；FastICA；协方差；CUDA；多级并行

ABSTRACT

How to efficiently process hyperspectral information has become the research focus of remote sensing area. Dimensionality reduction of hyperspectral image is the key step of the following analysis and identification of landscape. Dimensionality reduction is a typical computing intensive and memory access intensive process with high complexity. Realizing data reduction in serial mode is impossible to satisfy the real-time need of many applications, such as military and agriculture. Heterogeneous system with general CPU and special GPU can satisfy different needs for computing resources, and is one of the most promising developments of future high performance computer architecture. How to effectively utilize the computing power of the heterogeneous systems is the key of this paper. At present and in the future, CPU + GPU heterogeneous system, which can satisfy the demand of application of highly intensive computing, is one of the mainstream development direction of high performance computer architecture. How to make full use of the computing ability of the CPU/GPU heterogeneous system to improve the processing speed of linear dimension reduction of hyperspectral remote image is the key of this study.

There are three parallel levels in the CPU/GPU heterogeneous system, such as processes between nodes , threads in a node and threads in the GPU. The parallel programming models of MPI, openMP, CUDA are chosed in this paper. PCA and FastICA, which are typical linear dimension reduction algorithms, were selected. we researched the strategies of parallel and optimization of different accelerations, and came up with some parallel algorithms of linear dimension reduction. In this paper, we have done the following works and innovations:

(1)Further study of 3 current parallel modes of CPU/GPU heterogeneous system was done. Multiple CPUs, GPUs and nodes are the characteristics of heterogeneous systems. the technologies of MPI, openMP and CUDA must be mastered. In this paper, we studied the proprramming models and the optimization strategies of signal and hybrid parallel.

(2)A series of parallel PCA algorithms were come up on the CPU/GPU heterogeneous system. On the basis of analyzing accelerate hots of PCA algorithm, we put forward the task assignment for different parallel levels, and came up with the parallel and optimization strategies of the hot steps of covariance matrix calculation, PCA transform and hyperspectral data I/O. There are 6 kinds of multi-parallel PCA algorithms proposed respectively based on MPI, openMP, CUDA, MPI+CUDA, openMP+CUDA and MPI+openMP+CUDA. The experimental results show that the 6 kinds of parallel algorithms all obtained the remarkable performance improvement, and the triple parallel PCA algorithm of MPI+openMP+CUDA get 145 times speed-up(without I/O) and 128 times of sum speed-up(include I/O).

(3)A series of parallel FastICA algorithms were proposed based on the CPU/GPU heterogeneous system. With analyzing accelerate hots of FastICA algorithm, we come up with the parallel and optimization strategies of the hot steps of white processing, the ICA iterations and IC transformation. An innovation of parallel design mode of “concrete –

abstract - concrete” was put forward to complex calculation of ICA iterations in particular. We have come up with 6 kinds of multi-parallel FastICA algorithms. Experiments showed that the acceleration of the 6 kinds of parallel algorithm is ideal. 169 times speedup(without I/O) and 159 times speedup(include I/O) were taken by the triple parallel FastICA algorithm of MPI+openMP+CUDA.

Key Words：Dimensionality Reduction for Hyperspectral Image, PCA, FastICA, Covariance, CUDA, Multilevel Parallel

第一章　绪论

本章首先介绍了课题的研究背景，接着介绍了课题的来源和研究重点，并对高光谱降维方法和高光谱并行处理现状进行了研究，然后说明了本文实验平台、数据来源及预处理、统一符号设定等，最后是本文结构。

1.1 课题研究背景

高光谱遥感（Hyperspectral Remote Sensing）是 20 世纪 80 年代出现的一种全新的遥感技术，是融合电磁学、光学、信号处理等多学科交叉领域，其应用成果已遍布林业[1]、农业[2,3]、地理学[1,4-6]、环境科学[1,4]、海洋学[4]等领域，尤其在军事[7]上应用更为广泛。高光谱遥感技术将确定地物性质的光谱与确定地物空间与几何特性的图像有机地结合在一起，在当前乃至今后一段时间都将处于遥感技术研究的前沿。

高光谱遥感图像具有以下特征：1）波段多，数据量大；高光谱图像是一个图像立方体（图 1.1），它由几十乃至几百幅波段图像组成，每幅图像都是高光谱成像仪在一定波段范围内对同一场景成的像[8,9]，其高光谱图像空间的量级为 GBs（NASA 每天向地球发送大概 850GB 的高光谱数据[10]）。2）光谱分辨率高、空间分辨率高；使得 AVIRIS（Jet Propulsion Laboratory's Airborne Visible-Infrared Imaging Spectrometer）能获得覆盖 2-12km 宽，数千米长区域[8]，精细的光谱分辨率反映了地物光谱的细微特征。3）波段间相关性高，冗余大；4）图谱合一；5）信噪比低，增加了处理难度。由于高光谱图像的这些特征，导致了高光谱图像处理复杂且耗时。同时，各应用领域对高光谱处理的实时性提出了自身的需求，比如在军事领域，在目标侦察中，需要尽快识别敌人的武器和伪装，急需对高光谱遥感信息实时处理[7]。伴随着计算机摩尔定律的快速发展，高光谱信息处理无论在算法上，还是在应用实现上都获得了巨大的发展，但依然面临着巨大的挑战[11]，其中海量影像数据的存储与计算[4]是高光谱信息处理中的重要难题之一。因此，如何对高光谱信息进行高效处理和利用已成为遥感领域近年来研究的热点。

由于“维数灾难（curse of dimensionality）”的存在，高维数据很难直接进行处理，一般都是先通过降维转换成低维数据进行处理。从空间几何的观点来看，高光谱遥感影像在空间中的分布属于高维问题。直接在高维空间上寻找高光谱数据间的统计关系，会带来样本类别训练困难、多元密度估计方法不精确[12]、高相关性和高冗余度[13]、“维数灾难”[14]、“空空间现象”[15]等严重的计算问题。因此，对高维光谱影像进行有效的维数压缩，提取其本维特征，是高光谱遥感影像处理中必不可少的重要步骤，成为该应用方向的重点研究问题。

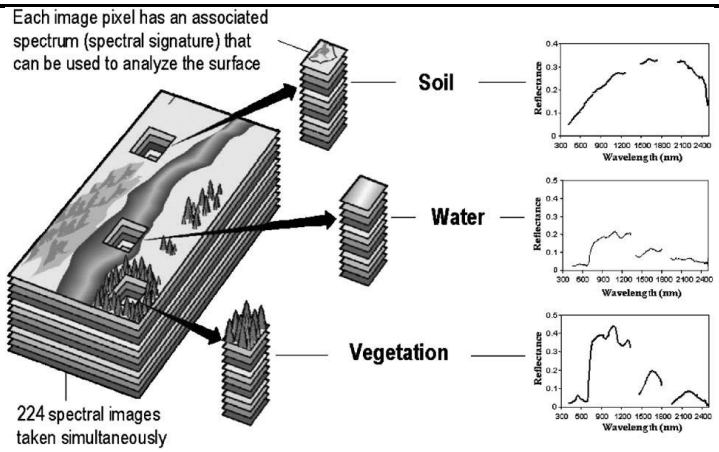

图 1.1 AVIRIS 高光谱图像组成

然而，大规模的高光谱影像降维带来的巨大计算量成为其应用的重要限制。降维算法的核心包含大量的矩阵（向量）运算、准则函数和最优化过程，涉及多次迭代和大规模循环，是一个典型的计算密集型和访存密集型过程，计算复杂度很高，十分耗时。例如，设高光谱数据集 X 的维数为 B，N 为 X 中的样本个数，则按传统的方法构造主成分分析算法的变换矩阵的复杂度为 $o(NB^2)$。如果用传统的串行方法处理该降维过程，将对军事、农林、灾害检测等高时效性的应用造成很大的约束与限制。

于是，近年来研究人员开始引入并行处理技术来解决遥感图像处理的计算性能问题[5-7]。并行计算是提高处理速度的最有效途径之一，不但可以使原有算法在不降低精度的同时达到实时性的要求，而且能够提高算法的自动化程度、扩大业务规模。

近年来，高性能计算系统越来越普及，尤其是集群系统和异构系统的出现。自 2007 年 nVidia 公司正式推出统一计算设备架构（Compute Unified Device Architecture, CUDA）[16]，以图形处理器（Graphic Processing Unit, GPU）作为协处理器辅助 CPU 进行通用计算，并获得了巨大的性能提升。CPU+GPU 的异构并行构架，其中 CPU 负责复杂逻辑和事务处理等串行计算，而 GPU 完成大规模并行计算。这样的系统构建方式充分利用了 CPU 和 GPU 的特点，并便于开发不同层次的并行性。随着 GPU 上编程环境的日益成熟，CPU+GPU 构建的异构系统具备了可编程性好、性价比高、计算功耗比低等诸多优点，恰可以有效应对传统同构高性能计算机在这些方面的缺陷。这种众核结构的协处理器成为了高性能发展的黑马，而搭载这种 CPU/GPU 异构系统的天河 1A 在 2010 年夺得高性能计算机 TOP500 排行第一，其后，具有相同架构的泰坦在 2012 年夺冠[17]。

新的高性能计算能力能为大规模高光谱遥感图像处理服务，并在复杂的图像处理分析中获得了合理的响应时间[18-21]。由于图像像素的 2D 局部组织[22]和底层计算的规范化，并行架构在图像分析应用领域越来越流行。然而，并行处理技术在多维图像构成的高光谱遥感图像处理领域特别是高光谱图像降维方向上的应用在国内还是一片空白，这是一个巨大的挑战和机会。

基于上述背景，本文旨在面向图像降维这一高光谱遥感信息处理中的重要步骤，针

对“线性降维”这一类最经典、应用最广的降维算法，基于 CPU/GPU 异构系统，对高光谱遥感图像线性降维多级混合并行算法展开研究及实现工作。

1.2 课题研究内容

1.2.1 课题来源

本课题来源于“国家自然科学基金”（项目编号：0901062412001）——CPU/GPU 异构系统下高光谱遥感影像降维多级协同并行计算方法及优化策略。项目属于计算机技术与遥感信息处理技术相结合的交叉学科项目，项目以线性降维方法（如 PCA 和 ICA）为牵引，重点研究基于流形学习的非线性高光谱影像降维算法（如 Isomap 和 LLE），建立性能分析模型，研究面向多结点 CPU/GPU 异构体系结构的多级协同并行算法，归纳此类应用面向该体系结构特征的一般性优化方法，同时反向指导降维算法在可并行度和可扩展性方面的创新实践，突破原有算法的“加速比墙”，出原创成果。作为该项目的一个组成部分，本文重点针对高光谱影像线性降维方法 PCA 和 FastICA，研究其在 CPU/GPU 异构系统下多级协同并行算法，并在相应的实验平台上实现本文所提出的并行算法。

1.2.2 课题研究重点

要充分发挥异构平台的性能优势，需要从研究分析体系结构的性能模型和应用程序的执行模型两个方面入手。其中基于 CPU/GPU 异构系统的特性包括：CPU 与 GPU 的结点配置、通信带宽、访存带宽和延迟、最佳的线程数量等，而应用程序的执行特性包括并行性、数据特征、访存局部性、占用的寄存器资源等。这些因素会造成程序优化时的搜索空间巨大，需要进行性能优化研究。

本文研究重点包括以下三个方面：

1）首先在研究高光谱遥感影像数据空间特性的基础上，深入研究 PCA、ICA 等线性降维算法的关键计算环节，结合 CPU/GPU 异构系统体系结构特点，分析可能影响性能的因素及性能瓶颈，确定此类降维方法的加速热点，为后续的并行算法设计和优化提供宏观指导和理论依据。

2）目前前沿的 CPU/GPU 异构系统一般采用结点内异构（CPU+GPU）、结点间同构的硬件架构，可同时支持结点间进程级（MPI）、结点内 CPU 线程级（openMP）和结点内 GPU 线程级（CUDA）三级协同并行。针对这三级并行的编程模型和性能优化策略进行深入研究，提出一系列混合并行优化策略。

3）选择高光谱影像线性降维方法中两种典型算法（PCA 和 FastICA）展开深入研究，针对算法中的加速热点提出并行和优化策略；结合线性降维算法，基于 MPI、openMP、

CUDA、MPI+CUDA、openMP+CUDA、MPI+openMP+CUDA 等单一和混合并行模式设计并实现相应的并行算法。特别是基于 MPI+OpenMP+CUDA 三级混合模式，要充分利用不同结点的 CPU 和 GPU 资源，使并行算法的加速性能达到最佳。

1.3 相关研究工作

1.3.1 高光谱遥感图像降维的方法

高维数据的降维技术大致可以分为非线性降维方法和线性降维方法两类。而非线性降维又包括基于核的方法和基于流形学习的方法。基于核的方法主要是利用核函数隐式地将数据从原始空间映射到更高维的特征空间，然后在高维特征空间利用现有的线性降维方法进行降维[23]。从某种意义上讲，基于核的非线性降维方法可以看作是线性降维方法的扩展；而基于流形学习的非线性降维方法则是基于全新的流形学习假设，认为高维数据在高维空间会形成低维的非线性流形实现降维[77]。非线性算法可以揭示线性方法无法发现的内在结构，但结果往往很难解释，计算复杂度高，运算量大[24,25]。而线性方法计算快速，结果是有限个参数的线性模型，可以更容易地解释数据；但高光谱影像由于光谱分辨率很高，导致很多异物同谱的地物在高维空间中会分布在不同的但却可能很接近的区域，即高维空间中的这些数据云团并不是线性分布的，线性方法会损失原始数据的信息量，从而无法完全发挥高光谱的效力[26,27]。

主成分分析（PCA）、独立成分分析（ICA）、多维尺度变换（MDS）、线性奇异分析(LDA)等是高光谱影像处理中应用最广泛、最经典的线性降维技术；而新的基于流形学习的降维方法（如等距映射法 Isomap、局部线性嵌入 LLE 等），在发现高维非线性数据集的内在维数和进行维数约简等方面具有很大的优势，应用潜力大。无论是改进传统基于统计意义的全局线性降维方法，还是提出新的非线性降维方法，近年来国内外都有大量的研究成果可供参考[12,25,28-39]。例如，Green[15]发展了主成分分析方法，提出了最小噪声分量变换，通过该变换可使变换后各成分按照信噪比而不是方差从大到小的顺序来排列；Yang[28]提出了的局部多维尺度分析(Local MDS, LMDS)；Tenenbaum[29]和 Roweis[30]从算法实现的角度提出了两种经典的流形学习算法 Isomap 和 LLE；董广军[36]和 Melba[37]通过实验验证和分析了各种非线性流形算法应用于高光谱影像降维分类的性能和特点；Luo 等人[39]提出了一种新的高光谱影像非线性降维分析算法，验证了流形学习是一种有效的高光谱遥感数据特征提取方法。

尽管目前在高光谱降维领域的主要研究集中在非线性降维上，但非线性算法是在线性降维算法基础上发展而来，并且线性降维算法应用最广，效率显著，能基本达到降维目的。作为整个项目的牵引，本文主要针对高光谱线性降维算法中最经典的 PCA 和 ICA 算法进行研究。

1.3.2 高光谱影像并行处理研究现状和趋势

国内外研究大规模高光谱影像并行处理技术主要面向传统的并行处理系统（如MPP，Cluster 机群等）。Plaza[40]总结了并行计算在高光谱遥感图像处理领域的研究现状和发展趋势。David[41]研究了基于异构 MPI 在网络机群系统上对高光谱影像进行处理的技术。Javier 等人[42]提出了基于神经网络的高光谱影像并行分类算法。刘春等人[43]运用 Matlab 建模分析了高光谱降维的可并行性。上述研究还尚未触及类似“天河 IA”采用的多 CPU/GPU 异构并行体系结构，但研究方法和思路可为本文提供很好的参考借鉴。

此外，随着近几年来 GPU 的高速发展，在国际上已开始利用 GPU 对高光谱影像处理加速的研究。Sergio Sa´nchez 等[44,45]于 2011 年对高光谱遥感图像解混的 GPU 实时实现进行研究；Mahmoud ElMaghrbay[46]对高光谱解混中 Endmember Extraction 的 N-FINDER 算法进行了 GPU 移植；He Yang, Qian Du 等[47,48]在多 GPU 实现高光谱影像快速波段选择；Qian Du 等[49]在 GPU 上实现高光谱随机工程降维。但在该领域，国内研究较少，Haicheng Qu 等在[50]对高光谱特征提取进行 GPU 加速研究。本文的研究恰好可以弥补国内高光谱影像 GPU 实时处理领域的空白。

上述研究现状说明，将 CPU/GPU 异构混合结构的并行计算系统引入高光谱遥感影像处理应用领域正是时机，研究空间巨大。

1.4 本文的主要工作和创新

本文重点研究了 PCA、ICA 等高光谱遥感影像线性并行降维算法及其优化实现技术，针对这两种线性降维算法，提出了其在 CPU/GPU 异构系统下的多种并行算法及优化方案，并在本文实验平台下进行实现并测试。本文的主要工作和创新体现在：

1）针对 PCA 降维算法，对 PCA 降维算法中的协方差矩阵计算、PCA 变换、高光谱图像数据输入和结果矩阵输出等加速热点深入研究，提出相应的并行方案和优化策略。提出并实现基于 MPI、openMP、CUDA、MPI+CUDA、openMP+CUDA 和 MPI+openMP+CUDA 等 6 种并行 PCA 降维算法。

2）针对 FastICA 降维算法，对 FastICA 算法中的白化处理、ICA 迭代和 IC 变换等加速热点进行了深度研究，特别是具有复杂计算的 ICA 迭代过程，提出一种新型的针对复杂计算的“具体-抽象-具体”的并行设计模式，并将该并行设计模式应用于 ICA 迭代过程，开发出多种 ICA 迭代并行策略。提出并实现基于 MPI、openMP、CUDA、MPI+CUDA、openMP+CUDA 和 MPI+openMP+CUDA 等 6 种并行 FastICA 降维算法。

1.5 本文算法实验平台

本文的研究工作基于目前主流的 CPU/GPU 异构系统展开，采用 AMAX 微型 GPU 超算作为本文的实验平台，其详细配置为：两个 8 核的 Intel Xeon CPU E5-2650，带 ECC 功能的 64GB 内存和 2 张最新 Kepler 架构的 nVidia Tesla K20 GPU。本文程序在 CentOS 6.2 的 Linux 系统下，采用 GCC4.4.6 编译器、CUDA5.0 工具包和 openMPI 库对本文的不同算法实现进行编译执行测试。

本文实验平台共有两个 8 核 CPU，故执行 MPI 程序时启动 16 个结点（进程）；openMP 程序执行前设置 export OMP_NUM_THREADS=16。

1.6 本文统一符号设定

本文统一符号设定如下：高光谱图像数据 X（W*H*B），B 表示高光谱图像的波段数量，W 表示单个波段图像的宽，H 表示单个波段图像的高。用 S 表示单个波段象元数目（即 S=W*H）。通过降维，获得 Y（W*H*m）的 PC 矩阵（IC 矩阵），其中 m<B。

CPU 中结点（进程）数量 N，线程数量 n，结点号 rank，线程号 threadid；GPU 内线程格尺寸 gridDim，线程块尺寸 blockDim，线程块号 blockIdx，线程号 threadIdx。

1.7 本文所采用的高光谱数据及其预处理

下表 1.1 罗列了本文实验所采用的 6 组高光谱遥感影像数据。其中前 4 组高光谱遥感数据均来自美国 AVIRIS 提供的免费高光谱数据；后两组高光谱数据是通过对第四组高光谱数据放大得到，目的是进一步研究本文所设计并行算法的处理瓶颈及极限。

表 1.1 高光谱遥感图像数据信息

序号	宽	高	波段数	原始大小（MB）	预处理后大小（MB）
1	614	512	224	134.31	67.16
2	614	1087	224	285.15	142.58
3	753	1924	224	618.98	309.49
4	781	6955	224	2320.74	1160.37
5	1562	6955	224	4641.48	2320.74
6	1562	13910	224	9282.96	4641.48

高光谱遥感图像数据预处理过程包括两个主要部分：

1）光谱信息转换成像素信息；

2）适用于降维算法的合理的数据存储结构。

由于高光谱遥感图像数据本身记录的是光谱信息，无法进行直接的降维处理，在此需要一个预处理的过程。预处理的主要目的是将高光谱数据中杂乱无章的光谱信息（16 位 int 类型）转换为像素信息（用 0~255 表示，unsigned char 类型）。其具体转换过程

如下：寻找每个波段中光谱信息 I_i 的最大值 max 和最小值 min，通过以下公式将其转换为像素信息 P_i：$P_i = \frac{I_i - \min}{\max - \min} * 255$。除了该方法外，遥感软件 ANVI 提供了将单个波段高光谱遥感图像数据转换为 BMP 图像的功能，再通过读取 BMP 图像即可以获得所需要的数据。

高光谱数据维度变换研究：高光谱遥感图像数据较为独特，其结构类似于一个立方体，参见图 1.1。这种结构的数据表示需要三维数组，但三维数组数据的访问特别是在流处理器 GPU 中会显得较其累赘。根据降维的目的，主要分析的是高光谱数据的光谱间相关性，故其中高光谱图像 X（W*H*B）的宽 W 和高 H 这两个维度可以合并，即每个波段的所有象元数据为一个维度。此时高光谱遥感数据已经从三维转换成二维，但由于需要利用 GPU 进行计算，考虑到数据需要从主机端发送到设备端，即使是二维的高光谱数据，其通信次数受限于波段数 B（AVIRIS 高光谱图像的波段数为 224），会产生很多额外的通信开销，无法最大化带宽利用率。通信合并是最大化带宽利用率的最佳选择，基于此，需要将二维的高光谱数据转成一维数据存储在主存储器中，方法是单个波段的所有象元数据存储在一起，然后存储下一个波段的数据。

1.8 论文结构

本文以高光谱遥感影像降维为主线，围绕 PCA、ICA 等线性降维算法展开研究，并设计在 CPU/GPU 异构系统下并行算法，并在本文实验平台下实现该算法，进行性能测评。

本文共分六章。第一掌为绪论，首先介绍课题背景，说明课题来源和主要研究内容，包括课题研究的重难点；研究了高光谱降维方法和高光谱影响并行处理现状；概述了本文的主要工作和创新；最后介绍了本文算法验证的实验平台、统一符号标准、高光谱图像数据及其预处理和论文的结构安排。第二章主要研究了 MPI、openMP 和 CUDA 等三种主流并行编程模型，并针对单一并行以及混合并行研究了相应的优化策略。论文第三章对高光谱遥感影像线性降维中的 PCA 算法展开研究，首先研究其串行方法，进行加速热点分析；针对不同加速热点，研究单一并行及混合并行下的并行及优化策略，提出了 6 种并行 PCA 算法；在本文实验平台上实现并测试这 6 种并行 PCA 降维算法，对其进行性能度量。论文第四章先研究 FastICA 线性降维算法理论，分析其加速热点；针对加速热点研究不同并行及优化策略，特别是针对具有复杂计算的 ICA 迭代过程，提出一种适用于复杂计算的“具体-抽象-具体”的并行设计模式；提出并实现了基于主流并行及混合并行的 6 种并行 FastICA 降维算法。论文第五章对本文的工作进行总结，并展望下一步工作。

第二章　CPU/GPU 异构多级并行及优化技术研究

针对本文的实验平台，首先要考虑以下问题：CPU/GPU 是如何协同工作的？GPU 的体系结构如何？采用什么开发平台能够实现高光谱遥感影像并行降维算法？怎样做到最优化？

2.1 CPU/GPU 异构协同工作方式研究

目前前沿的 CPU/GPU 异构系统一般采用结点内异构（CPU+GPU）、结点间同构的硬件架构，可同时支持结点间进程级（MPI）、结点内 CPU 线程级（openMP）和结点内 GPU 线程级（CUDA）三级协同并行。

异构系统中，CPU 与 GPU 经北桥（Bridge）通过 AGP 或 PCI-E 总线连接，各自有独立的外部存储器，分别是内存（Host Memory）和显存（Device Memory）。其中，CPU 负责逻辑性较强的事务控制，GPU 负责高密集度的浮点计算，图 2.1 是典型的 CPU+GPU 异构系统。在使用 GPU 计算前，CPU 必须先通过北桥将数据传到 GPU 显存中；在 GPU 计算完成后，GPU 再将结果返回主存。CPU 与 GPU 间的通信是比不可少的，由于接口 PCI-E 的通信带宽限制，数据通信开销是必需要考虑的问题。

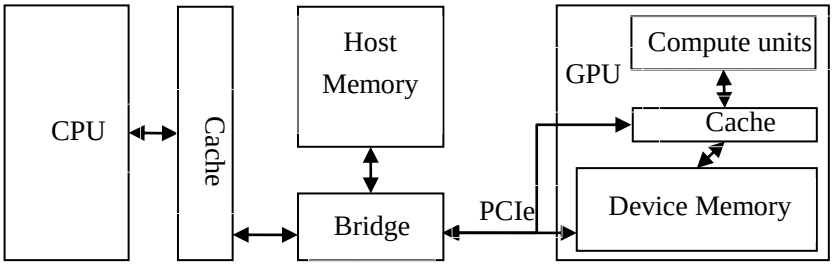

图 2.1　典型的 CPU/GPU 异构系统

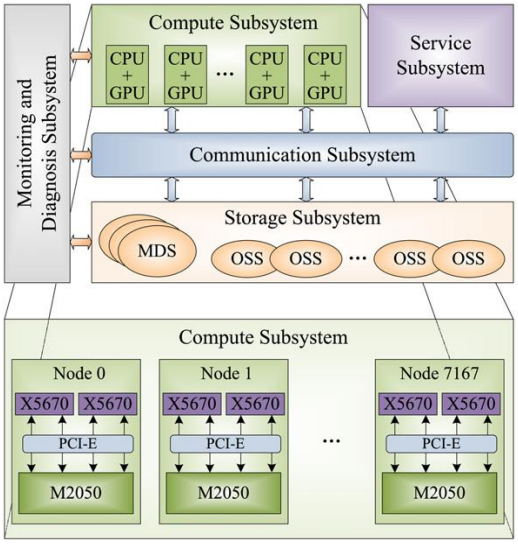

图 2.2　天河 1A 系统结构

图 2.2 描述了基于 CPU/GPU 异构搭建的天河 1A 超级计算机的系统结构。整个天河 1A 系统由计算子系统、服务子系统、通信子系统、存储子系统和监视与诊断子系统组成，其中计算子系统由 7168 个 CPU+GPU 异构结点构成。

CPU+GPU 异构集群体系结构对大规模并行计算研究提出了新的挑战，迫切需要研究人员深入研究与该体系结构相适应的并行算法。针对 CPU/GPU 异构系统的高性能计算平台，研究相应的并行通用计算技术，设计并实现高光谱遥感影像降维问题的多级协同并行算法，具有重大意义和价值。

2.2 GPU 体系结构

2.2.1 GPU 概述

GPU 初期用于图像处理，后来扩展用于通用计算。GPU 的处理能力和存储器带宽上相对 CPU 有明显优势，在成本和功耗上不需要付出太大代价，就能解决大规模数据处理问题[16]。CPU 的设计目标是使执行单元能够以很低的延迟获得数据和指令，因此采用了复杂的控制逻辑和分支预测，以及大量的缓存来提高执行效率；而 GPU 必须在有限的面积上实现很强的计算能力和很高的存储器带宽，因此需要大量执行单元来运行更多相对简单的线程，在当前线程等待数据时就切换到另一个处于就绪状态等待计算的线程，简而言之，CPU 对延迟将更敏感，而 GPU 则侧重于提高整体的数据吞吐量[78]。CPU 和 GPU 的设计目标的不同决定了两者在架构和性能上的巨大差距，图 2.3 对 CPU 与 GPU 中晶体管的数量及用途进行了比较，图 2.4 描述了近年 CPU 与 GPU 在浮点运算上的吞吐量差距[52]。

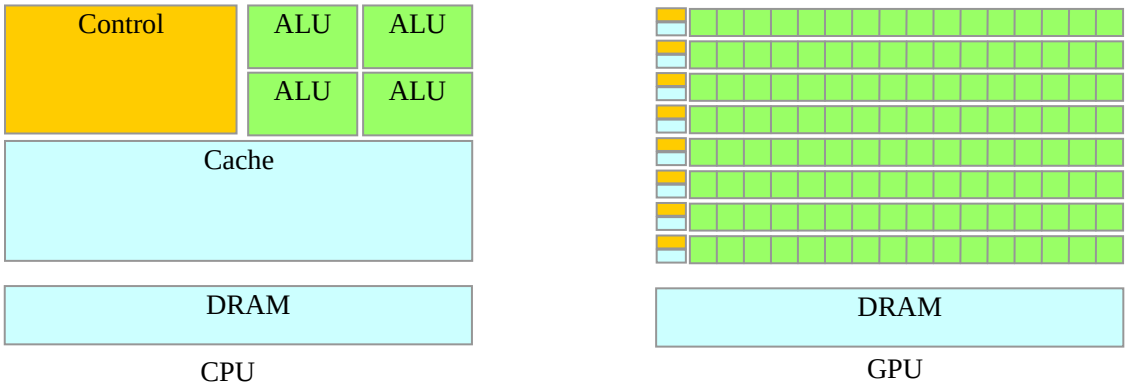

图 2.3 CPU 与 GPU 晶体管对比

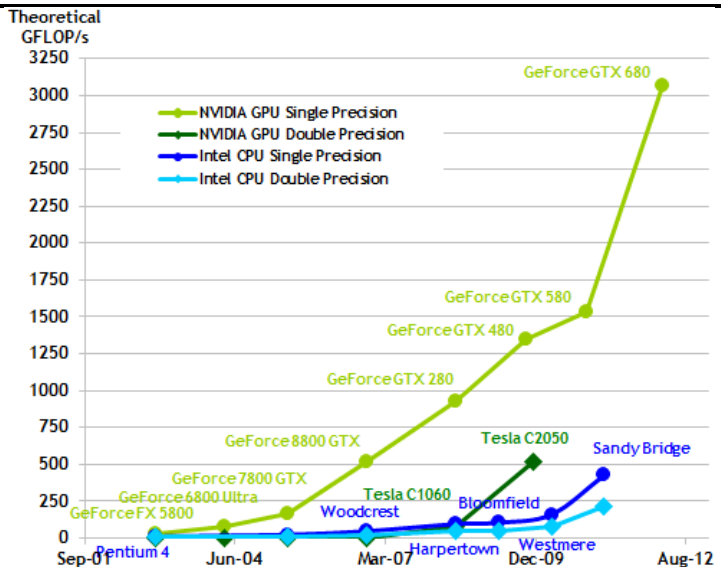

图 2.4 CPU 和 GPU 浮点处理性能对比

2.2.2 GPU 编程模型发展

伴随着 GPU 的不断发展，GPU 编程模型也在不断变化。早期 GPU 只能执行固定的几类操作，没有可编程 GPU 的概念，用 DirectX 和 OpenGL 等图形 API 进行程序映射，之后出现相对高级的着色语音（Shader Language），如基于 DirectX 的 HLSL 和 OpenGL 后端的 GLSL 及同时支持 DirectX 和 OpenGL 的 Cg。GPGPU（General Purpose GPU）通用计算图形处理器概念的提出，GPU 正式进入可编程时代。各个厂商推出了自己的通用计算 GPU 及相应的编程模型。目前主流的有 nVIDIA 的 CUDA 平台、Microsoft 在 Windows7 系统中集成的支持 GPU 通用计算的 DirectCopmute、AMD 的 Stream SDK 平台以及 Apple 提出 Khronos Group 发布的 OpenCL 并行编程平台。目前为了让普通程序员的使用 GPU 的高速运算，nVIDIA 正积极推出 OpenACC 编程平台，避免了一些底层的 GPU 代码编写。

其中 CUDA 是目前使用最广泛的 GPU 编程模型。CUDA 由 nVIDIA 公司在 2007 年正式推出，在短短几年时间内取得了巨大的发展，经历了三种不同的计算架构（Tesla、Fermi、Kepler）。CUDA 可以使用高级语言（扩展 C）编写程序，而不是 DirectX 和 OpenGL 的 API 接口，大大降低了编程难度，提高了编程性能。CUDA 为开发者利用 GPU 做大规模并行计算提供了条件，自 CUDA 推出后，已被广泛应用于流体力学模拟、数据挖掘、音视频编解码、石油探测、生物计算、气候与天气建模、图像处理和超级计算等领域，并获得了几倍、几十倍乃至上百倍的加速比。具体的 CUDA 编程平台将在 2.3.3 中详细介绍。

2.2.3 GPU 执行模型

CUDA 程序的串行部分在 CPU 上执行，并行部分通过 kernel 函数在 GPU 上启动指定 block 和 thread 数量的 Grid，GPU 执行时采用 SIMT（Single Instruction Multiple Thread, 单指令多线程）模式，即 GPU 端并行执行的程序称为 kernel 的内核程序，而在 CPU 端执行的程序被称为 host 宿主程序，宿主程序控制 kernel 内核程序的启动、数据交互以及少量串行计算，参见图 2.5。

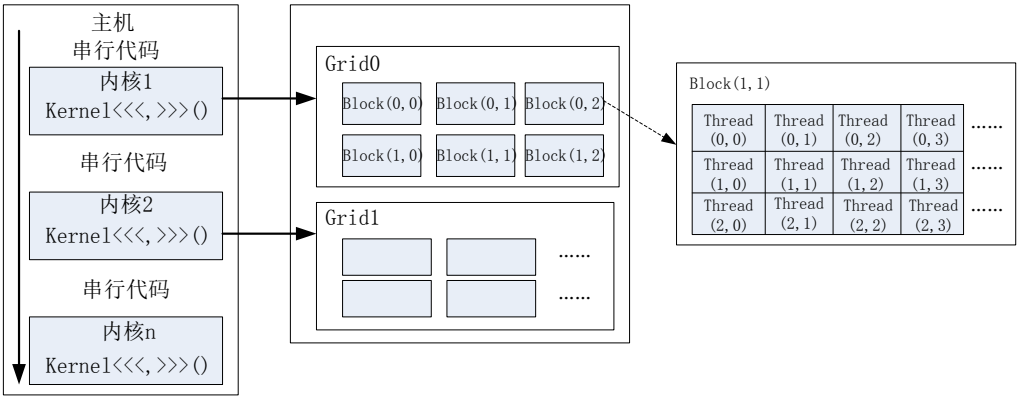

图 2.5 GPU 执行模型

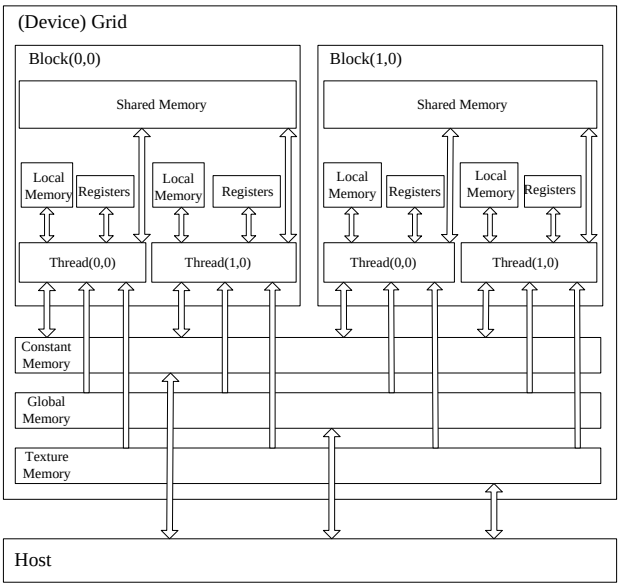

图 2.6 GPU 存储模型

2.2.4 GPU 存储模型

GPU 中拥有大量的存储器，每个 thread 拥有寄存器（register）和局部存储器（local memory），block 拥有共享存储器（shared memory），还包括可供所有线程访问的全局存储器（global memory）以及两种只读存储单元：常量存储器（constant memory，在最

新 Kepler 架构中已改为片内缓存）和纹理存储器（texture memory）。在 GPU 中并没有局部存储器，而是当寄存器不够时，从全局存储器中划分一部分作为局部存储使用。其基本结构如图 2.6 所示。

GPU 上各存储单元访问延迟各不相同，其中寄存器最快，共享存储器次之，全局存储器最慢。表 2.1 罗列了 K20 中各存储单元的位置、大小、访问延迟、权限、生命周期等相关属性。

表 2.1 K20 存储层次相关属性

存储空间	位置	大小	延迟	访问权限	生命周期范围
寄存器	片上	255/线程	≈20cycle	读写	线程内
共享存储器	片上	48KB/SM	≈40cycle	读写	函数内
纹理存储器	片上	最大 4G	>200cycle	只读	全局
局部存储器	片外	最大 4G	400-800cycles	读写	函数内
全局存储器	片外	4GB	400-800cycles	读写	全局

2.3 并行编程模型

2.3.1 MPI

消息传递指用户通过显式地发送和接收消息来实现处理机间的数据交换、协调步伐、控制执行。在这种并行编程里，每个进程拥有自己独立的地址空间，无法直接相互访问，必须通过显式的消息传递来实现数据的交互。由于消息传递程序设计要求用户很好地对问题进行分解，合理组织进程间数据交换，并行计算粒度大，适合大规模可扩展并行计算。这种并行方式广泛用于 MPP 和 Cluster 等面向分布式存储的计算机系统，也可以应用于共享内存 SMP 系统。

消息传递接口（Massage Passing Interface，MPI）是消息传递函数库的标准规范，由 MPI 论坛开发。MPI 作为一种消息传递编程模型，有三大目标：①通信性能高、②程序可移植性好、③功能强大。以 MPI 编写的消息传递程序，与语言和平台无关，具有实用、可移植、高效、灵活等特点。

MPI 是一个库，而非一门编程语言，故要用其编程必须与特定的语言（Fortran、C）绑定使用。MPI 不提供具体实现，目前主要的 MPI 实现有 MPICH、openMPI 等。通过 MPI 编写的程序，需要采用对应的编译器进行编译，比如 C 语言需要 mpicc 编译，C++ 则需要 mpicxx 编译，执行时通过 mpirun 或者 mpiexec 启动多个进程。

MPI 程序的框架如图 2.7 所示，首先需要包含头文件（mpi.h）；定义程序中需要的 MPI 相关变量（机器名 procsname，进程号 rank，进程数 numprocs）；接着用 MPI_Init 函数开始 MPI 程序；程序体包含计算部分和 MPI 通信部分；最后用 MPI_Finalize 结束程序。

图 2.7 MPI 程序框架结构

尽管 MPI 有很多函数，但只有 6 基本函数，只要掌握了这六个基本函数，就能进行 MPI 程序设计了。这 6 基本函数是：

① MPI_INIT,这是 MPI 程序第一个调用的函数，完成 MPI 程序的初始化工作。

② MPI_COMM_SIZE，函数返回给定通信域内包括的进程数量，不同的进程可以通过该函数的调用获知给定通信域内正在执行的进程数量。

③ MPI_COMM_RANK，返回调用函数进程的 ID 标识，有了这个标识号，不同进程可以将自身与其他进程区分开来，实现不同的功能或进程间协作。

④ MPI_SEND，通过该函数，进程可以将自己的数据发送给目标进程，并通过消息标志将消息与其他消息区分开来。

⑤ MPI_RECV,进程可通过该函数接收由相应的 MPI_SEND 函数发送过来的数据。

⑥ MPI_FINALIZE，MPI_FINALIZE 是进程最后调用的函数，可以结束 MPI 程序的运行，是 MPI 程序最后一句可执行程序，否则程序运行结果将不可预知。

在 MPI 程序中，有个返回状态 status，它是 MPI 定义的数据类型，使用前需要用户为其分配空间。在 C 中，status 由三个基本域（MPI_SOUECE、MPI_TAG、MPI_ERROR）组成。通过 status.MPI_ SOUECE 返回状态中包含的发送数据进程标识，通过 status.MPI_TAG 返回发送数据使用的 tag 标识，用 status.MPI_ERROR 显示接收操作返回的错误代码。还可以通过对 status 执行 MPI_GET_COUNT 调用可以得到接收到的消息的长度信息。

MPI 通信域包括进程组合通信上下文。进程组是所有参加通信的进程集合；通信上下文提供了一个相对独立的通信区域，不同的消息在不同的上下文进行传递，不同上下文的消息互不干涉，通过通信上下文可以将不同的通信区别出来。

MPI 并行程序有两种基本的设计模式，分别为对等模式和主从模式。MPI 程序一般是 SPMD 程序，也可以用 MPI 来编写 MPMD 程序，但是所有的 MPMD 程序都可以用 SPMD 程序来表达。对于 MPI 的 SPMD 程序，实现对等模式的问题是比较容易理解和接受的，因为各个部分地位相同，功能和代码基本一致，只不过是处理的数据或对象不同，也容易用同样的程序来实现；对于主从模式的问题，不管是从理论的角度还是从实践角度，都说明了主从模式的问题是完全可以用 SPMD 程序来高效解决的，即 SPMD

程序有很强的表达能力。而设计时一般也是以 SPMD 形式的程序为主。

除了带发送和接收的点点通信外，MPI 还提供了很多聚合通信函数（比如 MPI_BARRIER、MPI_BCAST、MPI_REDUCE、MPI_ALLREDUCE 和 MPI_SCAN 等），大大地降低了编程复杂性。聚合通信能够实现与点点通信相同的功能，且只能阻塞进行。

2.3.2 openMP

openMP 是目前基于共享存储器多处理机系统上最主流的并行设计模式，该模式称为共享内存编程模型。openMP 得到了所有硬件厂商（比如 SGI、IBM、Intel 等）和第三方软件（如 PGI、KAP 等）的广泛支持，是基于指导命令的并行编程语言，仅描述用户指导的并行化，不会对数据相关、死锁等进行检测。openMP 的主要优点是编程简单、可移植性强、可读性好等，其主要缺点是无法在分布式存储系统中使用，扩展性较差。

openMP 主要由编译指导命令、库函数和环境变量组成，其中编译指导命令通过并行结构、同步结构和数据环境实现程序的设计和编写。而环境变量和库函数共同构成了 openMP 的运行环境，图 2.8 为 openMP 的结构图。

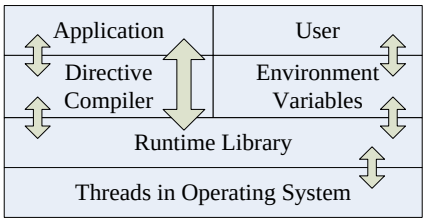

图 2.8 openMP 结构体系

openMP 的执行模式是 Fork-Join 模式，如图 2.9 所示，其中 fork 创建新线程或唤醒已有线程，join 即多线程会合。Fork-Join 执行模型在刚开始执行时，只有一个主线程，主线程在需要并行计算时，派生出从线程，主线程和从线程同时工作。并行区域结束时，从线程退出，程序回到主线程。成对的 fork 和 join 之间的区域称为并行域（Parallel Region），既表示代码又表示执行时间。

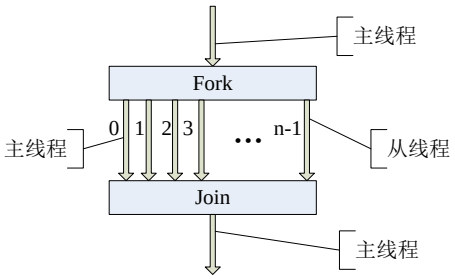

图 2.9 Fork-Join 执行模型

openMP 程序的编译执行及其方便。由于 openMP 受到几乎所有的编译器的支持，一般只需要通过在原编译命令上加“-omp”、“-openmp”或“-fopenmp”选项即可进行编译。通过设置环境变量 OMP_NUM_THREADS 即可控制 openMP 程序启动的线程

数量，后即可直接执行 openMP 程序。

openMP 专门设置了一些主流的并行指导命令供用户使用，比如 for、sections 和 single 等。for 指令可以控制循环的自动分配，可通过参数配置调度从句达到循环静态、动态分配的目的；sections 可以实现最简单的控制并行，每个 section 子句只能由一个线程执行一次，多余的线程在此区域等待，即 sections 包含了一次同步效果；single 表示单线程执行区域。openMP 还提供了线程同步指导命令 barrier，实现原子性操作的 atomic，以及主线程执行的 master 指导指令等，辅助完成复杂的编程任务。除此以外，openMP 还提供了一些子句，对以上指导指令所控制的区域进行具体的配置，比如 private 子句指定了线程的私有变量，reduction 子句实现了线程自动归约操作，nowait 子句使线程可以忽略指导命令内隐含的同步操作等等。其具体指导命令以#pragma omp 开始，形式为：#pragma omp 指令 [子句[，子句] …][51]。

openMP 库提供了丰富的 API 函数，用户可以通过这些库函数对并行区域进行独特的控制。比如用户可以通过 omp_set_num_threads 函数设置后续并行区域的线程个数；通过 omp_get_thread_num 函数可以得到线程 ID，得到线程 ID 后，用户即可对不同的线程进行独特的控制；通过 omp_get_num_threads 函数可以获取并行区域的线程数；通过 omp_get_max_threads 函数可以获取最大线程数等等[51]。

openMP 定义了在一定程度上控制 openMP 程序的环境变量，比如前面提到的 OMP_NUM_THREADS 可以控制启动并行区域的线程数；OMP_SCHEDULE 可以控制 for 循环并行化后调度的类型；OMP_DYNAMIC 用来确定是否允许动态设置并行区线程数；OMP_NESTED 指出是否可以并行嵌套。

在 openMP 编程中，一种比较好用的技巧是通过#pragma omp parallel 启动并行区域，后通过 omp_get_thread_num 函数获得当前线程 ID，配合 omp_get_num_threads 函数获得的线程数，即可实现类似于 MPI 的编程方式（SPMD），可通过同步指导命令替代 MPI 中的消息通信。

2.3.3 CUDA

在 CUDA 编程中，GPU 端称为 Device，CPU 端称为 Host。CUDA 程序采用 nvcc 编译器进行编译，编译时将源代码分为 Host 端代码和 Device 端 PTX（Parallel Thread eXecution）代码，再经过动态编译器（Just in time compiler, JIT）将 PTX 代码编译到目标 GPU 上。

程序员通过定义 kernel 内核函数调用 CUDA 线程，kernel 函数用__global__声明符定义，通过<<<blocks, threads>>>执行配置语法指定内核函数启动的相应线程数量（该 kernel 的 Grid 中启动 blocks 个 block，每个 block 启动 threads 个 thread），共启动 blocks*threads 个线程。每个启动的 block/thread 具有独一无二的 ID，可通过

blockIdx/threadIdx 变量在 kernel 函数中访问，相应的 Grid 或 Block 的尺寸可以在 kernel 函数中用 gridDim 或 blcokDim 进行访问。其中，根据 GPU 计算能力的不同，一个 block 中能启动的 thread 数量也不同，目前最新的 Tesla K20 GPU 一个 block 最多可以启动 1024 个 thread。

threadIdx 具有 3 个分量，分别为 threadIdx.x、threadIdx.y、threadIdx.z，利用这些分量可以唯一地标识线程。比如对于一个(Dx,Dy,Dz)的块，索引为(x,y,z)的线程 ID 为(x+yDx+zDxDy)。同样的道理，可以对 block 进行索引。因此启动参数 blocks 和 threads 可以是整形或 dim3 类型。

在线程块内的线程间，可以通过__syncthreads()函数实现栅栏同步的作用，在其调用点，块内线程必须等待直到所有线程到达该点才继续向前执行；而线程块必须独立执行，且可以以任意顺序串行或并行执行。这种 block 的独立性能够保证 block 可以在任意数目的核心（SP）上进行调度。

在 kernel 执行期间，CUDA 线程可以访问多级存储器上的数据，如图 2.10。每个线程可以访问私有的寄存器和本地存储器，线程块内线程可以访问 shared memory，所有线程都能访问 global memory。另外还有常量和纹理存储器两种只读存储器，能被所有线程访问。

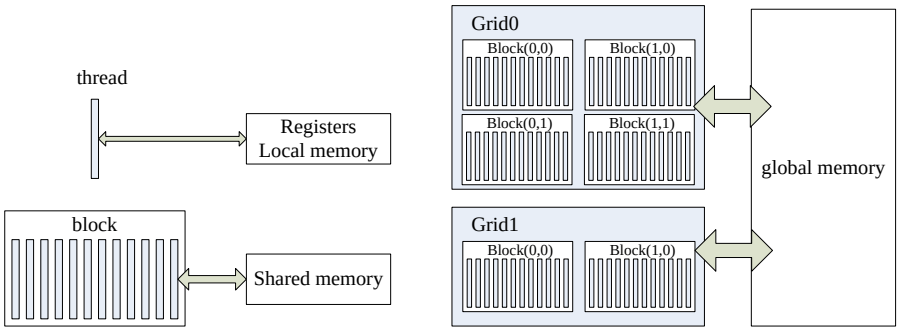

图 2.10 GPU 存储访问示意图

CUDA 程序在 CPU/GPU 异构系统下执行，其中内核 kernel 函数在 GPU 上执行，而程序串行部分在 CPU 上执行。CUDA 程序假设主机和设备都有独立的 DRAM 存储器空间，分别称为主机存储器空间和设备存储器空间。程序通过调用 CUDA runtime API 或 CUDA driver API，对设备上的存储器空间（global memory、shared memory、texture memory）进行管理。包括设备存储器的分配和释放，以及主机和设备的数据通信。

CUDA runtime API 或 CUDA driver API 都提供了对设备管理、线程管理、流管理、事件管理、存储器管理、纹理索引管理、执行控制、与 OpenGL 互操作、与 Direct3D 互操作和异常处理的应用程序接口，其功能是一样的。但所处的层次不同，runtime API 在 driver API 上层，对 driver API 进行了封装，比较适合初学者使用；而 driver API 是基于句柄、命令式的 API，大多数对象通过句柄引用，可以加载 PTX 汇编代码或 CUBIN 二进制代码形式的内核函数模块，能直接控制硬件执行，实现更复杂的功能，获得更好

的性能[16]。

不同的 GPU 拥有不同的计算能力，计算能力用主修订号和次修订号进行定义，比如 Tesla 架构的 GPU，主修订号为 1；Fermi 架构的设备，主修订号为 2；最新的 Kepler 架构 GPU，主修订号为 3；不同的计算能力支持的功能各不相同，具体可参考[52]。

```
__global__ void gpu_function();

void main(){
…
float *d_a,*d_b;
cudaMalloc((void**)&d_a,size);
cudaMalloc((void**)&d_b,size);
cudaMemcpy(d_a, a, size, cudaMemcpyHostToDevice);
gpu_function<<<blocks,threads>>>(a,b...);
cudaMemcpy(b, d_b, size, cudaMemcpyDeviceToHost);
cudaFree(a);
cudaFree(b);
...
}
```

图 2.11 CUDA 程序基本步骤

图 2.11 给出了简单的 CUDA C 程序基本步骤，在获得 GPU 设备后，首先用 cudaMalloc()函数在 GPU 上分配存储空间；接着用 cudaMemory()将数据从 host 端传输给 device 端；再启动 kernel 函数；然后将结果用 cudaMemory()传回 CPU；最后用 cudaFree()释放存储。通过以上几个基本步骤，一个简单的 CUDA 程序就完成了。

2.4 CPU/GPU 异构并行优化技术研究

2.4.1 串行优化技术研究

一个高效的并行程序=单机+并行 （2.1）

串行程序性能优化在并行程序优化中占据着不可缺少的地位。串行程序性能优化主要有三种途径：微处理器层次式的存储结构、串行程序优化和提高速度的途径。微处理器通过提高主频、流水线、cache 和并行技术（超标量、超流水、超长指令字等）实现计算速度的提升；存储器利用程序的时间和空间局部性，通过层次式存储结构实现快速的存储访问；以上优化由硬件工程师和系统工程师完成，本文涉及的优化主要通过串行程序优化来提升程序性能。

串行程序优化包括以下策略：1）尽量采用快速算法（如 KRYLOV）；2）研究编译器的优化开关（-O1、-O2、-O3、-fast）；3）充分利用已优化的库（BLAS）；4）基于 cache 和指令级并行对源程序进行优化（循环展开）。

编译器优化开关的选择是最主要且最方便的优化方法。一般编译器（比如 ICC、GCC）都提供了四级编译优化，其功能如表 2.2 所示。

表 2.2 编译优化开关及其作用

优化开关	作用
-O0	禁止优化
-01	常规优化技术，例如指令调度、寄存器分配、无用代码删除、常数传播等等
-O2/-O	循环展开、软流水、内部函数的内联
-03	数据预取、循环转换、标量替换

cache 主要有三种命中失效：首次访问，强制失效；容量不足，换出后又被取入；组相联或直接映射到同一组的块数过多，导致冲突失效。基于上述 cache 失效原因，有下列增加 cache 利用率的手段：循环交换，改变嵌套循环中访问内存的次序，在 fortran 采用列优先原则，C 中采用行优先原则；数组合并，利用块长，改善空间局部性；循环合并，增加数据可重用性，改善时间局部性；分块，集中访问可读入 cache 的块状矩阵，避免全行或全列读写，增强时间局部性 [79]。

一些程序中实现存储优化的原则：尽量使用局部变量，避免全局变量；使用 do 循环替代 goto 循环，可使用 load/store 指令的后增方式；避免使用等价语句；避免数组越界；所有函数中使用相同方式定义公用区。

在分支优化方面，将可能发生的分支放在前面；避免使用嵌套 if，尽量使用 switch；避免数据相关。在循环优化上，尽量保持循环简单，避免循环中出现复杂的控制流和函数调用。

在上述优化策略的基础上，提出串行程序优化的一般性步骤：

step 1 运行并得到正确结果，调用性能分析工具进行切步性能分析；

step 2 抽取程序运行关键部分，找到耗时最长的子程序；

step 3 使用优化开关优化程序，找到最佳优化开关后再进行性能分析；

step 4 改变源代码进行进一步优化，减少内存访问、使用已优化的库，发挥 cache 性能和流水线。

2.4.2 消息传递优化研究

基于 MPI 的并行算法，最敏感的部分就是通信开销。因此，MPI 消息传递并行程序的主要优化都是围绕着通信优化展开。

1）最小化通信量和通信开销

减少通信量和通信开销的主要方法有两种，一是通过合理的算法变形，改变计算过程中的通信次数和通信量，达到减少通信开销的目的，比如第三章中将要介绍的协方差计算公式变形。第二种方法是将单独的消息合并成一个大消息，进行集中通信，尽量减少消息传递的启动次数和时间。另外可以尽量采用 MPI 函数库中提供的广播等功能。

2）使用非阻塞通信，计算通信重叠

MPI 的主要通信方式有阻塞和非阻塞两种（图 2.12），阻塞指函数调用的返回则意味着本调用涉及的资源可重新使用；非阻塞通信中，如果一个函数调用可能在本次操作完成之前或允许用户重新使用调用中涉及的资源(缓冲区)之前即返回。其中阻塞的消息发送和接收使用 MPI_Send 和 MPI_Recv 两个函数，而非阻塞的消息发送和接收则使用 MPI_Isend 和 MPI_Irecv，并通过配合 MPI_Wait 的使用来实现。其中非阻塞方式的通信有下列优点：避免缓冲待发送的消息，避免缓冲不足引发的死锁，计算通信重叠，避免分配缓冲区和拷贝数据到缓冲区引起的开销。图 2.13 展示了标准非阻塞通信的优势。

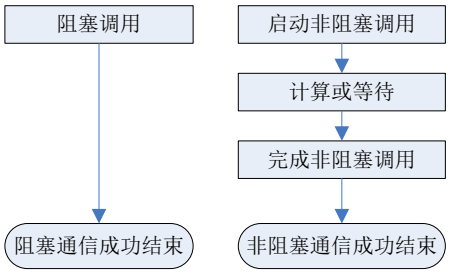

图 2.12 阻塞与非阻塞通信对比

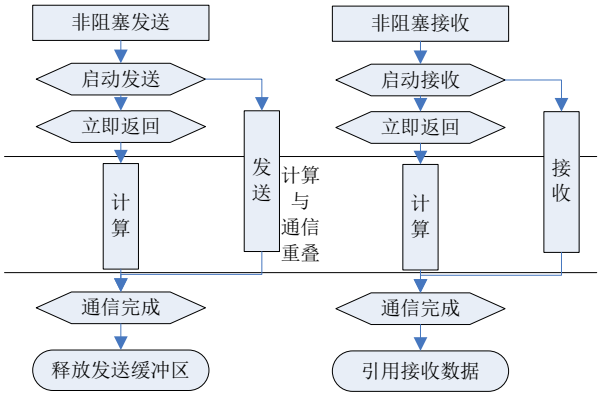

图 2.13 标准非阻塞消息通信

除了以上通信优化外，在进行 MPI 程序优化时，还需要考虑数据划分、任务划分以及动态负载均衡等问题。数据的划分决定了负载平衡和通信开销，数据划分要使通信开销最小化，同时要提高程序局部性。合理的任务划分通过数据控制相关性转换与消除、并行映射等达到减少消息通信的目的。动态均衡负载主要用于负载动态变化的情况，需要同时在负载均衡的系统开销和负载不均衡的计算间进行折中。

2.4.3 多线程优化研究

针对 openMP 程序，影响其性能的主要因素包括程序并行部分比率、openMP 本身的开销、负载均衡、程序局部性、程序同步带来的开销等。其中 openMP 本身的开销包括：并行性开发不足；并行性表达低效；对系统体系结构欠优化；启动并行区引发的开销；冗余的指导语句及同步操作引发的开销；其他开销。针对上述影响 openMP 程序性能的因素，可以在以下几个方面对程序进行优化：

1）合理的并行化设计。在整体程序并行算法设计时，需要综合考虑该高光谱遥感影像降维算法的特点及实验平台系统的特点，对并行区域进行合理的设计安排。

2）并行区的扩展与合并。通过并行区的扩展和合并可以减少并行区数量，达到减少并行串并行执行的切换开销。并行区的扩展和合并主要包括相邻并行区的合并和并行区向外层扩展。相邻并行区合并是将两个或多个相邻的并行区合并成一个；并行区的外层扩展主要是将并行区扩展到控制结构外，比如 if、for、while 等。

3）删除冗余指导语句。删除冗余的指导语句可以减少对 openMP 运行时库的调用，一般有两类冗余指导语句需要删除，一类是固有冗余指导语句，即某些指导语句在所有情况下都是冗余的；第二类是面向特定环境的冗余删除，这种冗余主要对通常情形构成冗余。

4）barrier 同步优化。分析并行区不同线程间的数据依赖关系，分析相应的优化策略，尽量减少并行区的同步次数。

5）确保负载均衡。在基于 openMP 对计算任务进行划分时，应尽量确保每个线程负载均衡。

6）其他优化。比如尽量避免多重循环、避免在循环内部启动并行区、合理的全局和私有变量设计等。

2.4.4 GPU 性能优化研究

针对 nVidia 公司的 GPU 体系结构，主要有以下优化策略：

GPU 存储优化：在 GPU 体系结构中，存储单元包括全局存储、共享存储、寄存器、常量和纹理存储等，新一代的 GPU 还具备了两级缓存，如何有效利用各级存储器的特点，降低数据流访存开销是优化的关键。当访存对齐时（GPU 的一个 half-warp 的读写操作如果能够满足合并访问条件），那么多次访存操作会被合并成一次完成，从而成倍提高访问效率，优化存储带宽利用率。存储优化时的一条总指导方针是利用访问延迟少的存储器替代访问延迟长的存储器（访问延迟参考表 2.1）比如用寄存器代替共享存储器；减少全局存储器访问，转变为共享存储访问等等。

Kernel 和循环优化：kernel 合并是常用的减少 CPU 和 GPU 之间通信的优化技术。在 PCA、ICA 等线性降维算法中都存在对同一数据集的多次运算变换，有大量小规模中间结果。拟设计适合该应用的 kernel 合并的方法，尽量消除中间计算结果的传输。此外，针对算法中有依赖关系的 DOAROOSS 循环，采用循环变换、循环倾斜等技术增加细粒度线程的并行性；再结合循环分块（Tiling）技术，根据性能模型精确估计分块大小，以提高访存局部性。

线程设计和最大化 occupancy：GPU 上有众多计算核心，要想发挥 GPU 的性能，首先要确保 GPU 满载。然而由于 GPU 上寄存器等资源十分有限，对于给定的计算任务，

并不是 GPU 派生出的线程越多越好。将根据各种降维算法的特点，派生数量适当的线程，减少线程之间的资源冲突，实现线程间并行度和 GPU 片上资源利用率的最佳折衷。设置线程数量的一个基本原则是最大化 occupancy 占用率，理论上 occupancy 占用率越大，计算性能越好。

多流并发：新一代 GPU 最多支持 32 个 kernel 函数并发执行，并且它们之间的切换代价很小，例如 NVIDIA 的 Kepler 架构流程序之间上下文切换只需 20-25 微秒。这样用户就可以使用 GPU 提供的类似操作系统中分时服务的特性，将多个流程序同时执行以提升性能。当一个流程序在访存时，另一个流可以用来计算。单个流程序的性能可能降低，但是整个流程序的性能有机会获得提升。试图尝试这样的机制，研究将其用于高光谱影像降维这类访存密集型应用的可行性。

主机设备通信优化：主机端和设备端的通信必不可少，进行主机端和设备端的通信优化是 GPU 优化的重要部分。主机设备通信优化手段有两种：通信与计算重叠，主机端使用页锁定内存。通信与计算重叠主要利用多流并发技术，在一个流负责通信的同时，另一个流仍可以进行计算任务。用 cudaHostAlloc()分配的页锁定内存（Pinned memory）能够确保内存不被调出，相比较可分页内存（Pagable Memory），具有明显的传输速率。比如在 Tesla K20 GPU 中，经 CUDA-Z 测试，页锁定存储的带宽为 6016.69 MB/s，可分页存储的通信带宽仅为 2200.97 MB/s。但页锁定内存会额外消耗主机端资源，使用时需要综合考虑优劣。

2.4.5 一些混合并行优化技术研究

多核集群的通信优化研究。针对多核结点构成的集群，结点内部通信和结点间通信存在巨大差距，采用分层通信策略，即将通信分为结点内通信和结点间通信。数据通信时首先结点内首进程对结点内部需要通信的数据进行收集和整合，后与其他结点的首进程进行通信，完成结点间通信后，结点内首进程再通过结点内通信传递数据。考虑到结点内 cache 的特点，可以通过利用高速 cache 通信来提高结点内通信效率，通过调整通信数据的大小（将通信数据划分成符合 cache 的较小的数据块），使其适合 cache 大小，确保数据能够驻留在 cache 中，确保通信在 cache 内完成，增加 cache 的利用率。

GPU 集群的通信优化研究。针对 GPU 集群，nvidia 推出了一项 CUDA-Aware MPI 技术，可以较好地优化不同结点间 GPU 上数据通信。配合 nvidia 推出的 GPUDirect 技术，CUDA-Aware MPI 可以实现不同结点 GPU 间的直接通信，避免了 CPU 的参与。图 2.14 展示了 CUDA-Aware MPI 技术的使用方法。

Without CUDA-Aware MPI

```
//MPI rank 0
cudaMemcpy(s_buf_h,s_buf_d,size,cudaMemcpyDeviceToHost);
MPI_Send(s_buf_h,size,MPI_CHAR,1,100,MPI_COMM_WORLD);

//MPI rank 1
MPI_Recv(r_buf_h,size,MPI_CHAR,0,100,MPI_COMM_WORLD, &status);
cudaMemcpy(r_buf_d,r_buf_h,size,cudaMemcpyHostToDevice);
```

CUDA-Aware MPI

```
//MPI rank 0
MPI_Send(s_buf_d,size,MPI_CHAR,1,100,MPI_COMM_WORLD);

//MPI rank n-1
MPI_Recv(r_buf_d,size,MPI_CHAR,0,100,MPI_COMM_WORLD, &status);
```

图 2.14 CUDA-Aware MPI 技术

结点内多 GPU 的通信优化研究：结点内的多个 GPU 间通信一般有两种方法，一是 GPU-CPU-GPU 的通信方式，另一种是 GPU-GPU 的通信。很明显，第二种方法更加方便快捷，但在使用第二种方法时，需要将 GPU 内需要通信的数据指针声明为共享变量，才能在其中一个线程能正确启动 cudaMemcpy 函数进行传输。图 2.15 显示了结点内 GPU 间通信的具体实现方法。

GPU-CPU-GPU

```
//thread 0
cudaMemcpy(buf_h,buf_d0,size,cudaMemcpyDeviceToHost);

#omp barrier;
//thread 1
cudaMemcpy(buf_d1,buf_h,size,cudaMemcpyHostToDevice);
```

GPU-GPU

```
//buf_d1为共享变量，在thread 0内可见
//thread 0
cudaMemcpy(buf_d1,buf_d0,size,cudaMemcpyHostToDevice);
```

图 2.15 结点内 GPU 间通信

2.5 本章小结

本章基于 CPU/GPU 异构集群系统，首先介绍了 CPU/GPU 异构系统的协同工作方式；第 2 节介绍了 GPU 体系结构，包括 GPU 概述、GPU 编程模型的发展、GPU 的执行模型和存储模型；第 3 节介绍了本文所涉及的 3 个并行层次（MPI、openMP、CUDA）的不同编程模型；第 4 节重点介绍的各并行层次的优化方法，分别从串行、MPI、openMP 和 CUDA4 个方面阐述单独优化策略，研究了混合并行下的优化策略。本章的研究工作为后续的高光谱遥感影像降维并行算法开发和优化奠定了技术基础。

第三章　PCA 降维多级并行算法研究与优化

在高光谱遥感图像降维领域，主成分分析（Principal Component Analysis，PCA）是一种最经典的线性降维方法，可以根据高光谱图像数据协方差矩阵的特征值和特征向量，计算主成分的贡献率，贡献率直接反映了信息量的大小，通过保留贡献率大的分量，丢弃贡献率小的分量，从而达到降维的目的。PCA 降维概念简单，算法高效，但由于算法实现计算量大、计算复杂，随着高光谱图像数据量不断增大，该算法已无法满足实时性需求。因此，采用高效的并行技术，研究实时并行 PCA 降维算法具有较好的理论与实用价值。

本章针对已有的高光谱 PCA 降维算法进行并行热点分析，针对不同热点提出多种并行方案和优化策略，分别提出基于 MPI、openMP、CUDA、MPI+CUDA、openMP+CUDA 及 MPI+openMP+CUDA 的 6 种并行 PCA 降维算法。实验结果表明这些并行算法能获得良好的加速比，多级混合并行各并行机制能发挥各自的优势，实现更好的加速效果。

3.1 主成分分析流程及并行热点分析

3.1.1 PCA 变换

主成分分析又称 K-L 变换，是在统计特征基础上的多维正交线性变换，把原始特征空间的特征轴旋转到平行于混合集群结构轴的方向，得到新的特征轴。

在文献[80]中定义了主成分分析及其原理：设有向量集 $X=\{X_i,\ i=1,2,\cdots,N\}\in R^n$，E（X）为 X 的数学期望，U 是 X 对协方差矩阵的特征向量按特征根由大到小顺序排列而成的变换矩阵，称 $Y_i=UX_i$ 为主成分分析，其中 $Y=\{Y_i,\ i=1,2,\cdots,N\}\in R^n$，主成分分析性质如下：

（1）　主成分分析是一正交变换；

（2）　主成分分析所得到的向量 Y_n 中各个元素互不相关；

（3）　从离散主成分分析后所得到的向量 Y_n 中删除后面的（n-d）个元素而只保留前 d（d<n）个元素时所产生的误差满足平方误差最小的准则。

主成分分析的原理[80]如图 3.1 所示，原始数据为二维数组，两个分量 x_1，x_2 之间存在相关性，具有如图所示的分布，通过投影，各数据可以表示为 y_1 轴上的一维点数据，从二维空间中的数据变成一维空间中的数据会产生信息损失，为了使信息损失最小，必须按照使一维数据的信息量最大的原则来确定 y_1 轴的取向，新轴 y_1 称为第一主成分，为了进一步汇集剩余信息，求出与第一轴 y_1 正交、且尽可能多地汇集剩余信息的第二轴 y_2，新轴 y_2 称为第二主成分。

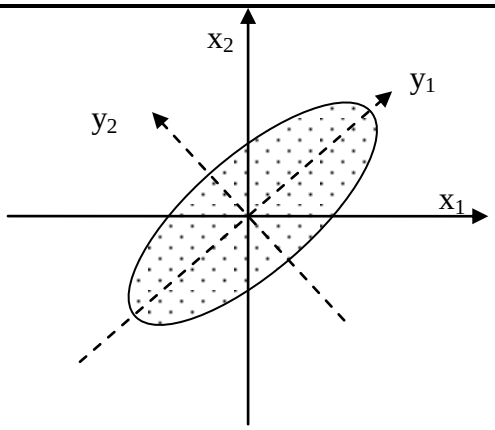

图 3.1 主成分分析的原理

3.1.2 基于高光谱遥感图像降维的 PCA 算法流程

在高光谱遥感图像处理中，波段间存在很强的相关性。主成分分析（PCA）是高光谱图像中用来降低图像冗余度的一种最常使用的技术。它通过一定变换把高维图像变换到低维空间，在低维空间波段之间的冗余度降到最低，因此原始数据的信息被压缩到了较少的几个波段，这样就实现了降维。

基于 PCA 的高光谱图像降维方法处理流程与具体方法为（图 3.2）：

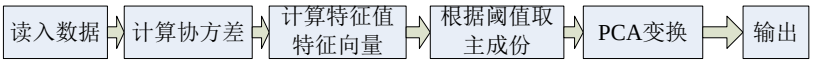

图 3.2 PCA 串行算法流程图

对于高光谱图像数据 X（W*H*B），B 表示高光谱图像的波段数量，W 表示单个波段图像的宽，H 表示单个波段图像的高。用 S 表示单个波段象元数目（即 S=W*H）。。

1) 计算高光谱图像的各波段间相互的协方差：

$$\sum_{ij}=\mathrm{cov}\left(X_i,X_j\right)=E\left[\left(X_i-\mu_i\right)\left(X_j-\mu_j\right)\right],\qquad \mu_i=E\left(X_i\right)=\frac{1}{N}\sum_{i=1}^{N}x_i \tag{3.1}$$

最后构建协方差矩阵

$$\sum=E\left[\left(X-E[X]\right)\left(X-E[X]\right)^T\right]$$

$$=\begin{bmatrix} E\left[\left(X_1-\mu_1\right)\left(X_1-\mu_1\right)\right] & E\left[\left(X_1-\mu_1\right)\left(X_2-\mu_2\right)\right] & \cdots & E\left[\left(X_1-\mu_1\right)\left(X_B-\mu_B\right)\right] \\ E\left[\left(X_2-\mu_2\right)\left(X_1-\mu_1\right)\right] & E\left[\left(X_2-\mu_2\right)\left(X_2-\mu_2\right)\right] & \cdots & E\left[\left(X_2-\mu_2\right)\left(X_B-\mu_B\right)\right] \\ \vdots & \vdots & \ddots & \vdots \\ E\left[\left(X_B-\mu_B\right)\left(X_1-\mu_1\right)\right] & E\left[\left(X_B-\mu_B\right)\left(X_2-\mu_2\right)\right] & \cdots & E\left[\left(X_B-\mu_B\right)\left(X_B-\mu_B\right)\right] \end{bmatrix} \tag{3.2}$$

其中Σ ij 的无偏估计为：$\sum_{ij}=\frac{1}{S-1}E\left[\left(X_i-\mu_i\right)\left(X_j-\mu_j\right)\right]$ （3.3）

2) 用雅克比（jacobi）算法计算协方差矩阵的特征值及相应的特征向量。

3) 给定一个阈值 T(比如 99%)，计算主成分贡献率并排序，找到满足贡献率之和 $v_m \geqslant T$ 的主成分的个数 m

4) 对原高光谱图像进行 PCA 变换，即 $Y=(Y_1,Y_2,\ldots,Y_m)^T=AX$，其中 $A=T^T$

3.1.3 并行热点分析

分析串行 PCA 算法的并行热点，最好的方法是执行一遍 PCA 串行算法的实现程序。表 3.1 显示了采用高光谱遥感图像数据 2 时，执行一遍 PCA 降维串行算法程序（未开优化开关）的各段具体时间。

表 3.1 最初 PCA 串行程序各段执行时间（ms）

输入	协方差	特征值	贡献率	PCA 变换	矩阵输出
811.32	196087.2	776.64	0.88	45306.34	171.71

分析表中串行 PCA 算法的各段执行时间，可以发现 PCA 算法的时间消耗最多的是协方差矩阵的计算和 PCA 变换过程，这两部分的消耗时间占总时间的 99%以上，是并行设计的重中之重。

其中特征值特征向量的计算和贡献率的计算都依赖于高光谱遥感图像的波段数 B，而目前的高光谱成像仪所得到的高光谱图像波段数在几十到数百之间，不会随输入数据的增大而需要更大的执行时间，且这两部分耗时较少，可不必考虑。特别是贡献率的计算几乎不需要 1ms 时间，在后续的实验结果测试中将其放入特征值计算中统一考虑，不再单独列出。

当输入高光谱数据线性增长时，其输入/输出时间也会呈线性增长。当协方差和 PCA 变换这两个部分获得较大的性能加速后，几何增长的 I/O 时间将会成为并行程序的瓶颈，故并行 I/O 也是本文的研究热点。

3.2 PCA 算法并行热点研究

根据 MPI、openMP 及 CUDA 三种并行机制，针对前文分析的并行热点进行研究，深入挖掘各步骤的最佳并行方案，并提出混合并行的多级协同并行策略，最终形成并行 PCA 降维算法。并行方案的任务划分中，MPI 主要侧重于划分高光谱图像数据 X，原因是高光谱图像数据巨大，可能会导致内存（显存）容量不足；而 openMP 的多线程程序是基于共享存储的，如何划分不会影响需要的内存总量，且为了多级混合并行，因此任务划分时侧重于寻找相异与 MPI 任务划分的其他划分方案。

3.2.1 协方差矩阵并行计算研究

基于 MPI 的协方差矩阵并行计算：要想利用 MPI 实现不同结点并行计算协方差矩阵，需要解决两个主要问题，首先是如何针对协方差计算进行任务划分，即各结点需要

完成的计算任务分别是什么；然后需要考虑如何协调得到最终结果，即通过通信交互各自的计算结果，最终计算得到高光谱遥感图像数据的协方差矩阵。

一种传统的协方差计算公式变形如下

$$COV(X,Y)=\frac{\sum_{i=1}^{s}(X_i-\overline{X})(Y_i-\overline{Y})}{s-1}$$

$$=[\sum_{i=1}^{s/N}(X_i-\overline{X})(Y_i-\overline{Y})+\sum_{i=s/N+1}^{2s/N}(X_i-\overline{X})(Y_i-\overline{Y})+\cdots+\sum_{i=(N-1)s/N+1}^{s}(X_i-\overline{X})(Y_i-\overline{Y})]/(s-1) \quad (3.4)$$

根据该公式变形，各结点的主要计算任务是计算$\sum(X_i-\overline{X})(Y_i-\overline{Y})$。而计算该过程，需要首先计算$\overline{X}$，即全局向量均值。要计算全局向量均值，首先计算得到局部数据的向量均值，然后通过通信在 root 结点计算得到全局数据的向量均值，再通过广播通信可以使各结点获得全局向量均值，该过程共需 2 次消息通信。在各结点完成$\sum(X_i-\overline{X})(Y_i-\overline{Y})$的计算任务后，需要将计算结果发送给 root 结点进行汇总计算得到最终的协方差矩阵，此时又需要 1 次消息通信。整个过程可参见图 3.3 所示的多结点协方差计算过程

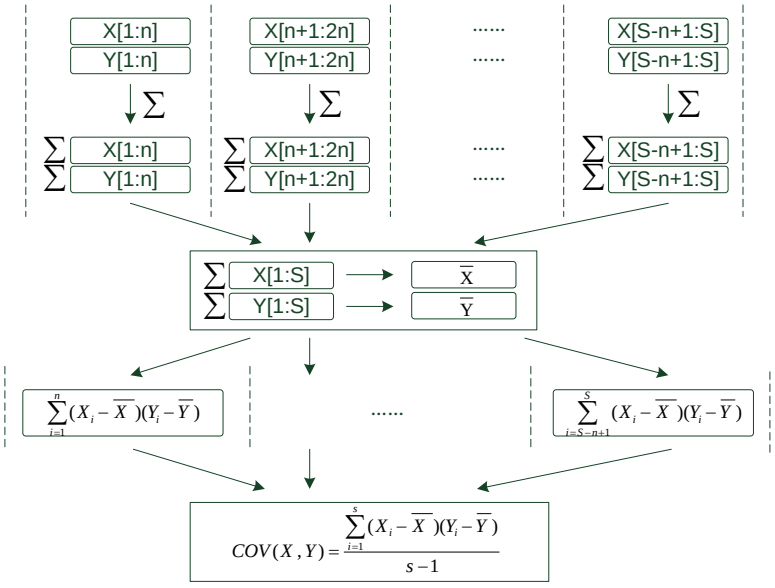

图 3.3 传统方法实现多结点协方差求解

为了获得更好的并行程序执行效率，对协方差计算公式进行如下变换：

$$COV(X,Y)=\frac{\sum_{i=1}^{s}(X_i-\overline{X})(Y_i-\overline{Y})}{s-1}=\frac{1}{s-1}\left[\sum_{i=1}^{s}\left(X_iY_i+\overline{X}\overline{Y}-X_i\overline{Y}-\overline{X}Y_i\right)\right]$$

$$=\frac{1}{s-1}\left[\sum_{i=1}^{s}X_iY_i+s\overline{X}\overline{Y}-\overline{Y}\sum_{i=1}^{s}X_i-\overline{X}\sum_{i=1}^{s}Y_i\right]=\frac{1}{s-1}\left[\sum_{i=1}^{s}X_iY_i+s\overline{X}\overline{Y}-\overline{Y}s\overline{X}-\overline{X}s\overline{Y}\right]$$

$$=\frac{1}{s-1}\left[\sum_{i=1}^{s}X_iY_i-s\overline{X}\,\overline{Y}\right]=\frac{1}{s-1}\left[\sum_{i=1}^{s}X_iY_i-\frac{1}{s}\sum_{i=1}^{s}X_i\sum_{i=1}^{s}Y_i\right]$$

$$=\frac{1}{s-1}\left[\left(\sum_{i=1}^{s/N}X_iY_i+\sum_{i=s/N+1}^{2s/N}X_iY_i+\cdots+\sum_{i=(N-1)s/N+1}^{s}X_iY_i\right)-\frac{1}{s}\left(\sum_{i=1}^{s/N}X_i+\sum_{i=s/N+1}^{2s/N}X_i+\cdots+\sum_{i=(N-1)s/N+1}^{s}X_i\right)\left(\sum_{i=1}^{s/N}Y_i+\sum_{i=s/N+1}^{2s/N}Y_i+\cdots+\sum_{i=(N-1)s/N+1}^{s}Y_i\right)\right]$$

（3.5）

通过该变换，计算协方差时，各结点的计算任务变成了计算ΣX 和ΣXY，得到计算结果后，通过消息传递机制将结果发送给 root 结点，root 结点最终计算得到协方差矩阵。其在多结点下的计算流程如图 3.4 所示，整个过程仅需要 1 次通信。通过该改进，通信量从原来的 3 次降为了 1 次，能够较好地提高 MPI 程序的并行效率。

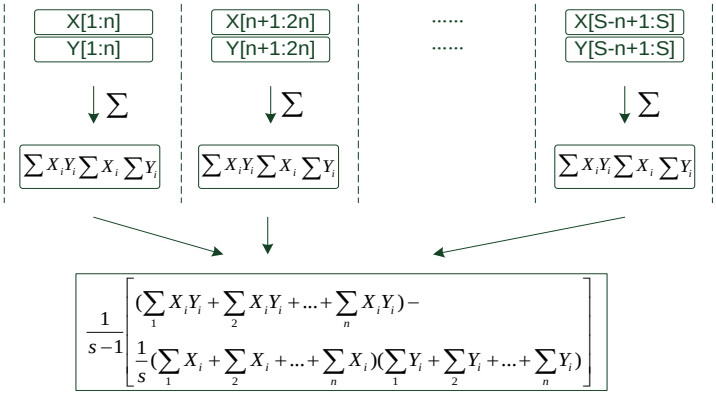

图 3.4 改进的多结点协方差求解方法

基于 openMP 的协方差矩阵并行计算：openMP 是结点内多线程并行方法，要想实现多线程同时计算高光谱遥感图像的协方差矩阵，需要明确两点，即协方差矩阵的计算任务如何均衡分配给各线程和怎样同步以确保协方差矩阵计算的正确性。

通过对协方差公式的变形可以看出，协方差计算的计算量主要集中在ΣX 和ΣXY，一种比较简单的并行思想是对该部分关键计算做并行，即采用 openMP 中的归约操作完成协方差的主要计算工作。由于 S（H*W）较大，归约计算的部分占了整个协方差计算的绝大多数，故可以获得较为理想的性能提升。但是，每次的归约操作都会进行一次并行区域的 Fork-Join，整个协方差矩阵共需进行 B*(B+1)/2 次协方差计算，其总的并行启动开销是较为明显的。

为了能够避免并行区域启动开销对整体程序性能的影响，提出一种协方差矩阵的补偿均匀划分策略。由于整个协方差矩阵是一个对称阵，即只需要计算其下三角阵（图 3.5 左）。而协方差矩阵内部的各个协方差计算是独立的，那么一个比较简单的划分方法是将这些协方差的计算均匀地分配给所有的线程。为了能够较为方便地将协方差矩阵的计算任务均衡地分配给每个线程，在此提出一种补偿划分方案，如图 3.5 右，在刨去对角线上的协方差后，①就能完美的放入②区域上，对应的映射关系为：①中的 COV（i,j）对应②中的 COV（B-1-i，B-1-j）。经补偿划分，协方差矩阵的计算任务可以形象地视为矩形，进行简单的切分即可实现不同线程的均衡负载。通过该策略实现的并行方案中，

只需要一次的 Fork-Join，即可完成所有的计算任务，即可较为有效地避免了前文中提到因并行区域启动开销而引发的性能损失。

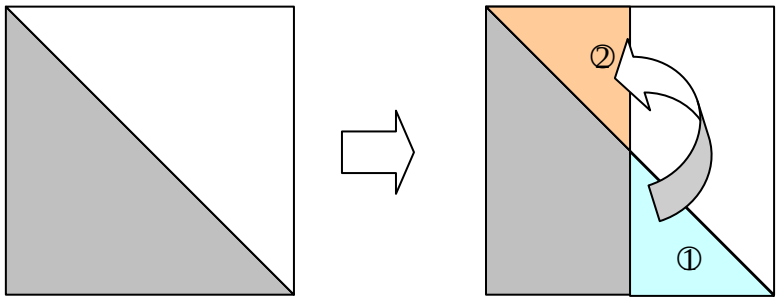

图 3.5 协方差矩阵的补偿划分

由于协方差矩阵上各个协方差值的计算是相互独立的，因此在协方差的计算过程中各个处理器间并不需要同步，而在协方差矩阵计算结束时，由于各处理器的处理速率的差异，尽管任务分配均匀，但全部处理器无法同时结束协方差矩阵的计算任务。而下一步的特征值和特征向量的计算依赖于完整的协方差矩阵，故在计算协方差矩阵结束前，需要一个同步操作来确保程序执行的正确性。当然，由于并行区域的 Fork-Join 中 Join 退出并行区时已经隐含了一次同步操作，故此处的同步可以省略。

基于 CUDA 的协方差矩阵并行计算：在利用 GPU 并行计算时，如何用 CUDA 将计算任务映射到 GPU 上是关键。本文针对协方差矩阵计算的特点，提出了 3 种基于 CUDA 的协方差矩阵计算映射方案，其中方案Ⅱ又提出 3 种不同的实现策略：

方案Ⅰ GPU 上的每个线程完成协方差矩阵中的一个协方差的计算任务。如图 3.6 左，启动 B 个 block（线程块），每个 block 中启动 B 个 thread（线程），将每个协方差矩阵中的协方差计算任务映射到其对应的 thread 中进行计算。由于协方差矩阵的对称性，可以采用前文叙述的补偿划分策略，启动 B/2 个 block 即可完成协方差矩阵的计算任务。

方案Ⅱ GPU 上的每个线程块完成协方差矩阵中一个协方差的计算任务。如图 3.6 右，启动 B*B/2 个 block，每个 block 中的 thread 可根据 GPU 选取最佳的 thread 数量。将协方差矩阵中单个协方差的计算任务映射到对应的 block 中，由一个 block 完成一个协方差的计算。在具体实现上可有多种策略：①采用补偿划分策略，先启动<<<B,blockDim>>>的 kernel 函数计算对角线上的协方差，然后启动<<<(B/2,B,1),blockDim>>>的 kernel 函数即可完成计算任务，②还是采用补偿划分，通过循环启动的方法，即通过 B/2 次循环启动<<<B,blockDim>>>的 kernel 函数完成协方差矩阵的计算工作。③直接循环启动 B 次，每次启动<<<i,blockDim>>>（i 为当前的循环次数）的 kernel 函数。

方案Ⅲ 每个发射到 GPU 上的 kernel 函数完成一个协方差的计算任务。由于每个协方差计算时循环较大，其循环次数为每个波段的图像象元个数（S=W*H），每次启动<<<gridDim,blockDim>>>的 kernel 函数，用来计算一个协方差值。其中 block 数量和

thread 数量可根据 GPU 型号进行自由选择，确保 occupancy 占用率达到最大。通过循环启动所有的协方差计算任务，即协方差矩阵的下三角（(1+B)*B/2 个）。

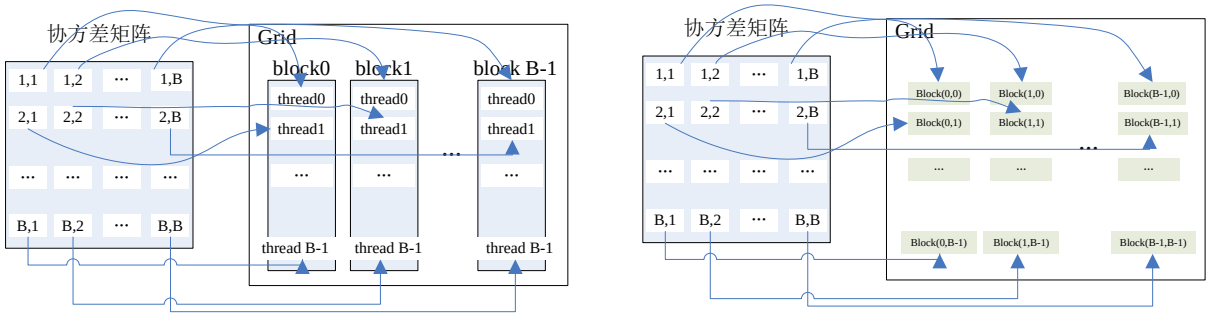

图 3.6 协方差矩阵计算任务的 GPU 映射方案

一种基于 GPU 的计算通信重叠优化策略：其中方案Ⅱ的③实现策略中，存在着通信计算重叠的可能。在图 3.6 右中，循环启动的第 i 行，使用的是高光谱数据中前 i 个波段中的数据。在高光谱数据第 1 个波段数据传输完成后，可以启动协方差矩阵第一行的计算，在计算第一行的同时传输第 2 波段数据，等 2 波段数据传输完成后即可同时启动第二行协方差计算和第 3 波段数据传输，以此类推，即能实现计算与通信重叠。

实现 CUDA 的三种方案，5 种实现策略，通过实验比较选择最优的方案，将其作为后续的多级并行开发中的基本协方差矩阵 CUDA 并行策略，从而获得最大化的程序性能提升。

基于 MPI+CUDA 的两级并行协方差矩阵计算：在采用 MPI+CUDA 两级协同并行加速时，首先需要明确两种并行的主要职责，然后才能依此完成针对高光谱数据协方差矩阵计算的两级并行设计。由于 GPU 突出的浮点计算性能，提出采用 MPI 控制不同结点启动不同的 GPU 实现多 GPU 并行计算的策略，MPI 主要负责不同结点间的任务调度、数据划分、通信及对各自 GPU 的控制，而 CUDA 负责将具体的协方差矩阵计算任务映射到 GPU 中实现并行计算。

由改进的 MPI 并行策略中的协方差多项式变换可知，每个结点需要计算ΣX 和ΣXY，即需要将ΣX 和ΣXY 映射到 GPU 上。其中ΣXY 的计算类似于单独的协方差计算，可采用基于 CUDA 的最优协方差矩阵计算并行策略，而ΣX 的计算可放在协方差任务矩阵的对角线上进行。

基于 openMP+CUDA 的两级并行协方差矩阵计算：类似于 MPI+CUDA 的两级协同并行，采用 openMP+CUDA 两级协同并行时，GPU 作为关键的浮点计算部件，而通过 openMP 启动的线程起控制、调度及同步作用。线程如何控制 GPU、计算什么任务以及如何实现不同 GPU 间的数据交互成为了本节研究的主要问题。

openMP+CUDA 两级协同并行加速时，通过 openMP 启动多个线程（为排除多余线程占用额外的资源，影响程序性能，启动的线程数量最好与 GPU 数量相同），不同线

程启动各自对应的 GPU 进行计算。由于本文能验证该算法的平台有两个 GPU，故将协方差矩阵的计算任务按协方差个数平均分成两份，用 openMP 启动的两个线程分别控制不同的 GPU 完成协方差的计算任务。

比较 openMP 并行协方差矩阵计算中提出的两种并行策略，很明显，归约的方案无法与 CUDA 结合进行两级混合并行。故在任务的划分上，采用协方差矩阵的补偿均匀划分策略。CUDA 的协方差并行计算可采用前文阐述的策略中最优的方案。其中针对 CUDA 并行策略中的方案Ⅱ的③实现策略，在双 GPU 的基于共享存储的统一编址空间中，有图 3.7 所示的优化方案：GPU0 执行图中①的计算任务，同时 GPU1 处理②中协方差计算；从右图中，①的上方三角区域及②中右侧三角区域需要的数据分别是高光谱图像数据的互补的一半，其中通信及其计算任务的安排可参考前文中提出通信计算重叠策略；在 GPU0 及 GPU1 各获得高光谱数据的一半时，即可在统一编址的情况下实现不同 GPU 的直接通信（直接通信的传输速率较主机-设备的页锁定通信还快了近 1 倍），且该通信也可与计算重叠；由于通信开销小于计算开销，故可以在不进行同步（保证右图矩阵部分计算时各 GPU 都有完整的高光谱图像数据）的情况下，进行后续的计算任务。

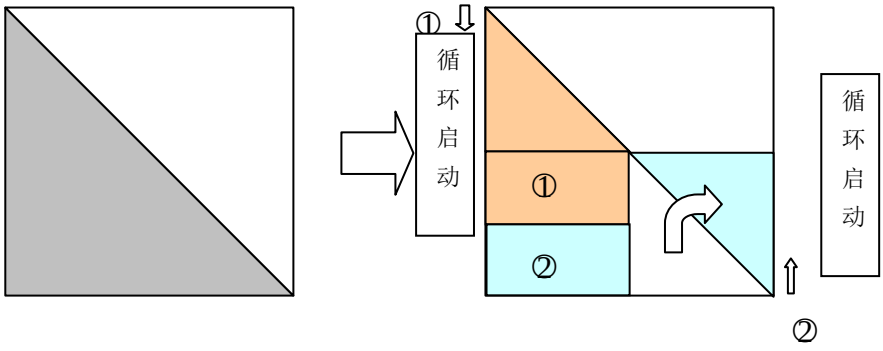

图 3.7 一种结点内双 GPU 的协方差计算优化策略

在各 GPU 计算完自身的任务后，其获得的是一个具有部分协方差值的协方差矩阵，进行交互得到完整的协方差矩阵也是问题的关键之一。为了能够较简单地交互，将非本 GPU 计算的协方差矩阵中协方差的值赋为 0，在 GPU0 中再额外申请一个协方差矩阵的空间，循环将其他 GPU 上的协方差矩阵通过设备端-设备端的通信方式传到额外开辟的协方差矩阵空间上，再在通过线程控制 GPU0 对两个协方差矩阵进行加法操作，最终 GPU0 上的协方差矩阵即为最终的协方差矩阵。

3.2.2 适用于 PCA 变换的并行计算研究

基于 MPI 的并行 PCA 变换：针对 PCA 变换过程的 MPI 并行研究的关键问题是研究如何划分各结点的任务（即各结点应该完成 PCA 变换中的什么运算任务），以及计

算过程如何交互（即完成 PCA 变换需要的通信及同步等）。本文通过下面的方法解决这些问题，最终实现 MPI 并行 PCA 变换。

首先，有 PCA 变换的公式演变：$Y = AX = \{AX_1, AX_2 \cdots AX_N\}$，其中 $X = \{X_1, X_2 \cdots X_N\}$。通过该变化，PCA 的变换过程可以分解为 N 个 AX_i 的运算，即每个结点仅需计算 AX_i 即可。这样就简单地完成了 PCA 变换过程的任务均衡划分。图 3.9 中左图具体阐述了高光谱图像数据矩阵 X 的均衡分配方案。

然后考虑通信，首先 A 矩阵（特征向量矩阵）是在 root 结点计算得到的，需要一次广播操作将其发送给所有的计算结点，各结点收到 A 矩阵后才能执行后续的计算过程。在计算得到 AX_i 局部结果后，有两种处理方案：①各结点将各自的局部结果矩阵发送给 root 结点，root 结点在收集到所有的局部结果后将其拼凑成完整的 PCA 结果矩阵，然后输出（图 3.8 左）。②结合在本文 3.2.4 节中将提到的 MPI 并行输出方案，各结点将各自独立的局部结果矩阵根据对应的映射输出到结果文件中（图 3.8 右）。其中①方案需要一次通信，且由于结果矩阵较大，其通信开销可能会比较明显；方案②避免了①中的通信，但需要写并行或写缓冲的支持。

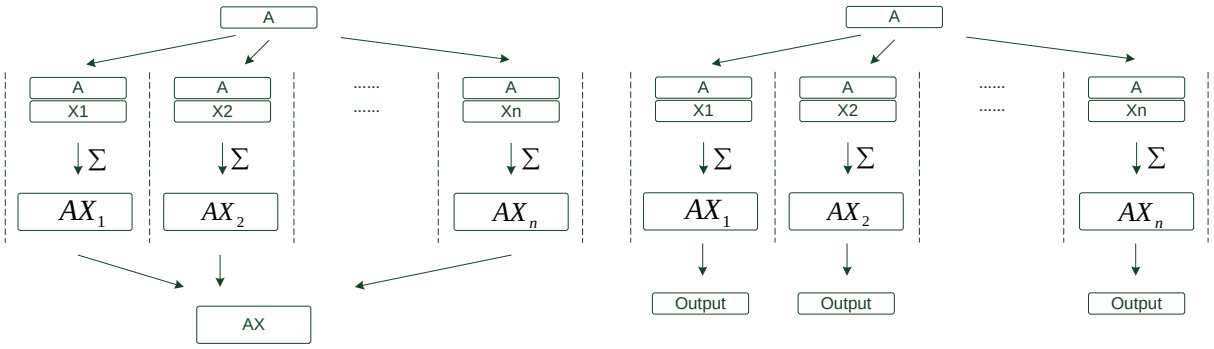

图 3.8 多结点 PCA 变换过程（左图适用于串行输出，右图为并行输出）

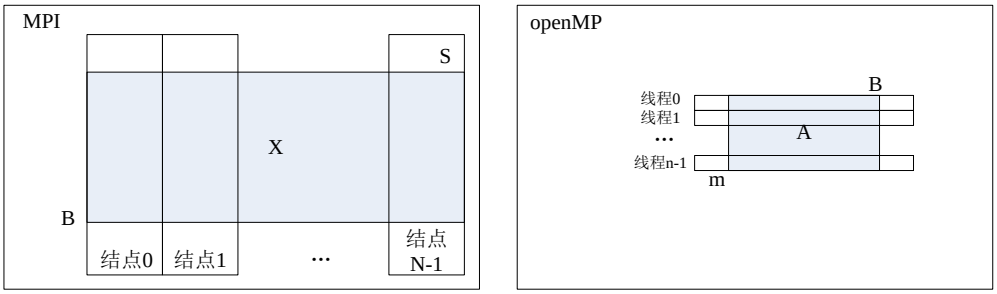

图 3.9 PCA 变换中 MPI 和 openMP 的划分策略

基于 openMP 的并行 PCA 变换：PCA 变换过程要实现结点内部线程级并行，要解决两个问题，即各独立线程需要完成 PCA 变换中的什么计算任务，如何同步来保证 PCA 变换的正确执行。下面针对这两个主要问题进行逐一解决。

PCA 变换公式可以进行另一种演变：$Y = AX = \{X^{\mathrm{T}}A_1^{\mathrm{T}}, X^{\mathrm{T}}A_2^{\mathrm{T}} \cdots X^{\mathrm{T}}A_n^{\mathrm{T}}\}^{\mathrm{T}}$，即对 A 分

割（$A=\{A_1^{\mathrm{T}},A_2^{\mathrm{T}}\cdots A_n^{\mathrm{T}}\}^{\mathrm{T}}$），其均匀分割方法见图 3.9 右侧。保证了 A 矩阵的均匀分割，即保证了 PCA 变换计算的均衡负载。由于各独立线程的任务相互独立（分别计算 PCA 变换矩阵中的部分值），故不需要同步来保证其计算的正确性，仅在独立线程计算结束时，需要一个同步来保证 PCA 变换矩阵已经完全计算，以便开始下一步操作（结果输出）。

基于 CUDA 的并行 PCA 变换：CUDA 并行是细粒度的，在 PCA 变换过程中寻找细粒度的并行计算任务，然后将这些细粒度可并行的计算任务通过 kernel 函数映射到 GPU 上执行，这是 PCA 的 CUDA 并行变换的关键。

在 PCA 变换中，A 是 B*m 的矩阵，X 的尺寸为 S*B，其结果矩阵 Y 大小为 S*m。对于结果矩阵中的每个元素的计算是独立的，且其计算为长度为 B 的两个向量内积。知道，高光谱遥感图像的波段数 B 一般为几十到上百不等，即该向量内积的计算量是有效的，且对于计算机来说是细粒度的，那么结果矩阵上的每个元素的计算可视为最小的粒度，可将其映射到 GPU 的线程上实现计算任务。对于整个结果矩阵，共需要 S*m 个这样的计算，那么理论上只需启动 S*m 个线程即可实现；由于 S=W*H，对于高光谱图像而言，其宽度 W 和高度 H 都比较大，约数百到上千，故其 S 较大，而 S*m 的数量级更大，在 GPU 上，一次启动的线程数量是有限制的，直接启动如此巨大数量的线程明显是不合理的。此时，可以采用循环计算的方案，即启动一定数量的线程块及线程，使其循环计算 S*m 个内积运算，即对于线程块 blockId 中的线程 threadIdx，其需要计算第 blockIdx*blockDim+i*（gridDim*blockDim）个任务（其中 gridDim 表示线程格中的线程块数，blockDim 表示线程块中线程数量，i 为循环变量），直到其值超过 S*m。通过这种方案，gridDim 和 blockDim 不受约束，可通过计算 occupancy 自由设定，使其占用率达到最大。

对比矩阵 A 和矩阵 X，矩阵 A 的大小远小于矩阵 X，知道共享存储器的访问延迟比全局存储器的访问延迟小了一个量级，那么能否将 A 矩阵放入共享存储器中，利用共享存储器加速程序性能？A 的尺寸为 B*m，假设 B 为 224，m 为 30，那么其需要的存储容量为 224*30*4/1024=26.25KB。尽管这个值小于 K20 的共享存储容量 48KB，但①由于 m 值未定，可能会出现出现大于共享存储容量的情况；②如此大的共享存储器占用会影响整个 kernel 的 occupancy 占用率，进而影响性能；③整个矩阵 A 参与的计算过程会产生大量的中间变量，在寄存器资源耗尽时又会使用全局存储器，影响 kernel 性能。因此将 A 完整放入共享存储器的想法并不可取。提出一种循环启动的方法，即将 A 分割成 m 个 B 长度的向量 Ai，每次启动一个 Ai*X 的计算任务，其需要的 shared memory 为 224*4=896B，计算产生的中间变量也较少，可用寄存器存储，可用保证最大化的 occupancy 占用率。一个 Ai*X 的计算结果作为 Y 矩阵的一行，在存储器上是顺序存储的，在完成一行的计算后，该行传回主机端和下一行的计算是可用并发进行的，因此可

开发计算通信重叠。

基于 MPI+CUDA 的两级并行 PCA 变换：结合前文中的 MPI 并行 PCA 变换中的任务划分策略，以及 CUDA 并行中的 GPU 映射策略，先用 MPI 将任务按结点均衡划分（即将原始高光谱遥感图像矩阵 X 对行 S 按结点数量 N 分割成 S/N 的小矩阵 Xi，各结点只需计算 A*Xi 即可），结点内部通过 CUDA 映射策略将自身的 PCA 变换的计算映射到 GPU 上，如图 3.10，就能充分利用不同结点的 GPU 资源进行两级并行 PCA 变换了。

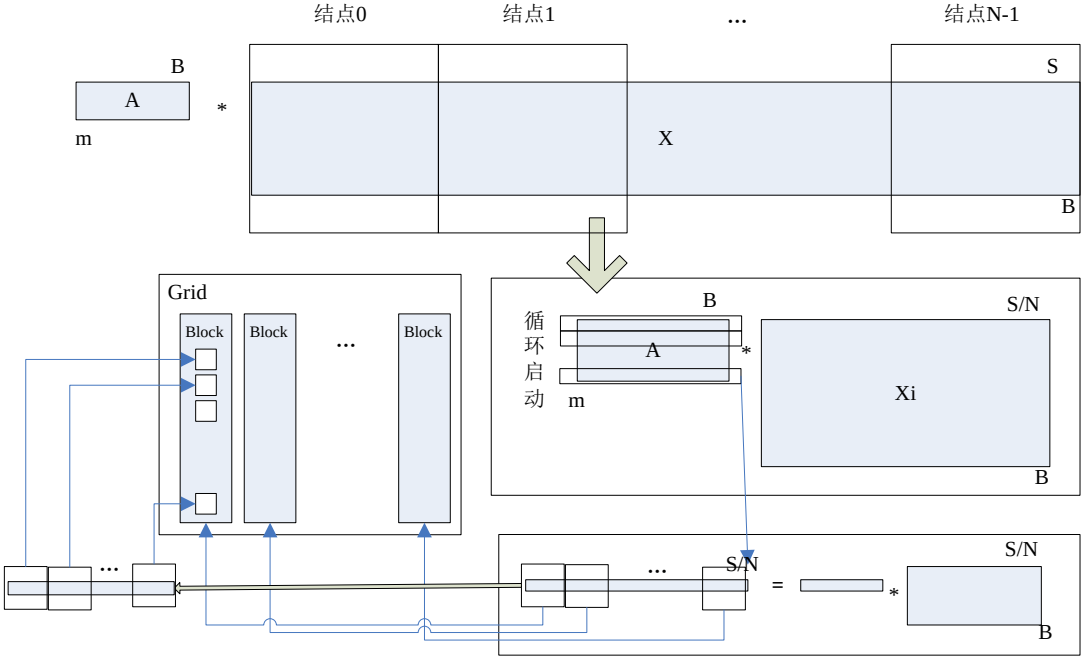

图 3.10 MPI+CUDA 两级并行 PCA

基于 openMP+CUDA 的两级并行 PCA 变换：类似于 MPI+CUDA，这里利用 openMP 实现任务的均衡划分，利用 GPU 众核处理器完成计算任务。先将任务按矩阵 A 的列 m 按线程数量 n 进行划分，其中线程数量根据结点内部 GPU 数量而定；然后各线程将自身分配到的 m/n 个 Ai*X 计算任务按前文叙述的 CUDA 映射策略映射到各自的 GPU 上计算，图 3.11 为 openMP+CUDA 两级并行 PCA 变换过程的分配示意图。

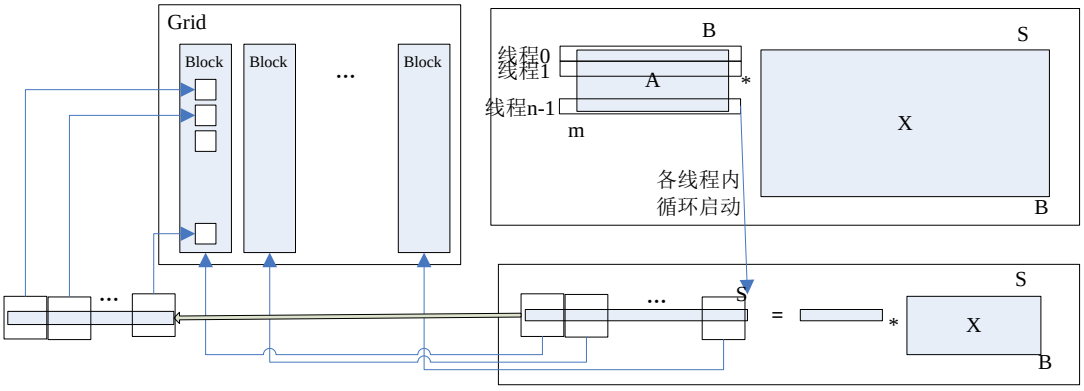

图 3.11 openMP+CUDA 两级并行 PCA 变换

3.2.3 面向高光谱数据的并行输入设计

输入数据主要考虑 MPI 和 openMP 两种并行，基于 GPU 的 CUDA 可以不做考虑。在本文中，输入数据主要是输入高光谱图像数据 X（W*H*B），整个高光谱图像数据类似于一个立方体，针对这一特点，提出下面的基于 MPI 和 openMP 的输入并行策略：

首先考虑 MPI 的数据划分：MPI 层采用数据并行的策略，通过对高光谱图像数据各波段图像进行均匀划分获得各结点的处理数据。根据高光谱图像的特点以及协方差矩阵和 PCA 变换的 MPI 并行策略，有两种划分策略，分别是水平划分和竖直划分，即分别根据 H 和 W 进行划分。本文采用水平划分的策略，对于 N 个结点，每个结点分配到的高为 h，其中 root 结点为 h_0，其中

$$\begin{cases} if\ (rank=0), 则 h_0=\dfrac{H}{N}+H\%N \\ \quad if\ (rank!=0), 则 h=\dfrac{H}{N} \end{cases} \tag{3.6}$$

根据以上数据划分方案，提出了基于 MPI 层的并行输入策略。 对于各结点而言，只需要输入自身需要的数据即可，很明显该过程是可以并行的。对于结点 rank，循环读取 B 次，每次只需读取 p 到 q-1 象元，

$$\begin{cases} \text{if（rank}=0\text{），则}p=0, q=h_0*W \\ \text{if（rank}!=0\text{），则}p=h_0*W+(\text{rank}-1)*h*W,\ q=h_0*W+\text{rank}*h*W \end{cases} \tag{3.7}$$

基于 openMP 的并行输入策略：根据高光谱遥感图像数据特点，在 openMP 层以波段划分（即根据 B 进行划分），对于 n 个线程，每个线程读取 B_0 到 B_1-1 个波段的图像数据，其中 threadid 为线程号，

$$\begin{cases} if\ (threadid+1!=n), 则 B_0=threadid*(B+n-1)/n, B_1=(threadid+1)*(B+n-1)/n \\ \quad if\ (threadid+1=n), 则 B_0=threadid*(B+n-1)/n, B_1=B \end{cases} \tag{3.8}$$

结合 MPI 和 openMP 的两级并行输入。分析上述的 MPI 层次和 openMP 层次的数据输入策略，可以发现 MPI 和 openMP 分别基于高光谱遥感图像数据的高 H 和波段 B 进行划分，因此这两个并行层次的数据输入操作可以结合起来。对于结点 rank 内 threadid 个线程而言，其数据输入任务为读取从 B_0 到 B_1 个波段的 p 到 q 个象元的数据，如图 3.12 所示。

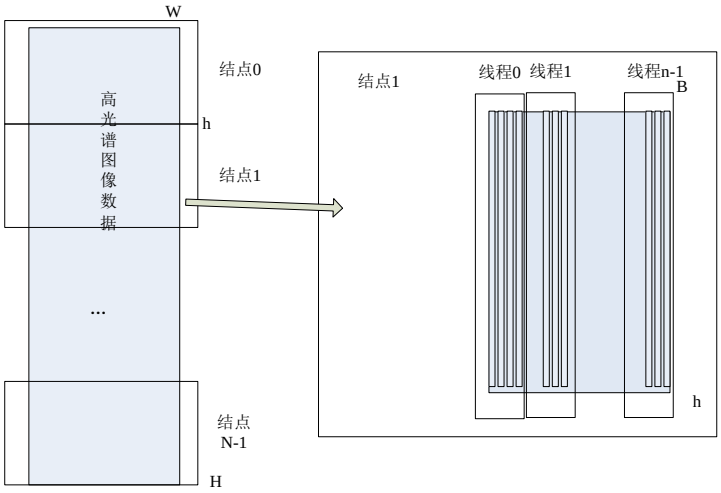

图 3.12 基于 MPI 和 openMP 的两级并行输入

3.2.4 适用于 PCA 变换矩阵的并行输出设计

输出类似于输入，主要需要考虑 MPI 和 openMP 的并行输出策略。对于 PCA 算法而言，输出的重点是 Y 矩阵（S*m）。

基于 MPI 的并行输出：根据 PCA 变换中的 MPI 并行策略，最终可计算得到 Y 局部矩阵，各结点将各自的 Y 局部矩阵输出到文件的相应位置即可，如图 3.13 左。对于结点 rank，循环写 m 次，每次只需写 p 到 q-1 象元，p、q 的取值参考公式 3.7。

基于 openMP 的并行输出：在 openMP 层对 m 划分，如图 3.13 右，对于 n 个线程，每个线程输出 B_0 到 B_1-1 个的 PCA 图像数据，B_0 和 B_1 的取值参考公式 3.8。

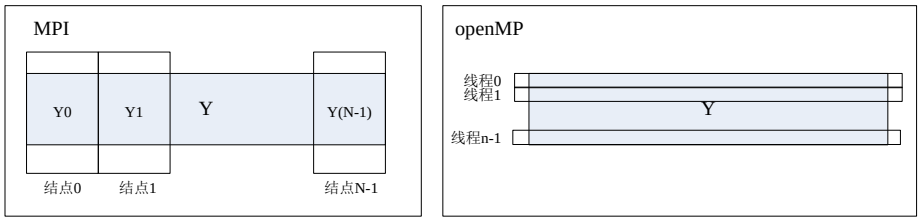

图 3.13 基于 MPI 和 openMP 的单级并行输出

基于 MPI+openMP 的二级并行输出：上述的 MPI 并行输出策略和 openMP 并行输出策略并不冲突，相反，还能进行互补。将上述两种并行策略结合，如图 3.14 所示，对于 rank 结点的 threadid 线程，其任务是输出 B_0 到 B_1-1 个的 IC 图像的 p 到 q-1 象元数据。

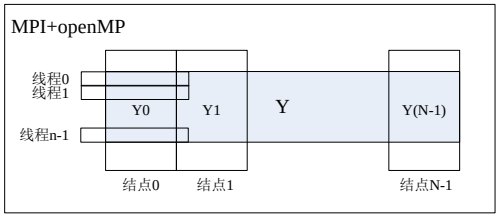

图 3.14 基于 MPI+openMP 的二级并行输出

3.3 6 种并行 PCA 降维算法

本节主要工作是将上述单独步骤的并行策略应用到 PCA 降维算法中，最终形成基于各种并行机制的完整的并行 PCA 高光谱遥感图像降维算法。

3.3.1 基于 MPI 的 PCA 降维算法

首先考虑单一的 MPI 并行，对于串行算法中的并行热点，采用前文叙述的并行策略中的基于 MPI 的协方差矩阵计算并行策略、基于 MPI 的并行 PCA 变换策略、基于 MPI 的 I/O 策略等，根据图 3.15 中的基于 MPI 的并行 PCA 降维算法流程图进行组织，最终形成基于 MPI 的并行 PCA 降维算法。图中高光谱图像的数据划分及各结点的数据读取详见 3.2.3；协方差的计算主要计算局部的ΣX、ΣXY，然后传给 root 结点综合计算协方差矩阵；PCA 变换前，root 结点需要将计算得到的主成分数组和特征向量广播给所有结点，然后各结点才能进行 PCA 变换；变换后的结果通过 3.2.5 节的策略输出。根据该算法实现的 MPI 并行 PCA 降维程序适用于共享存储系统及集群等通用的大规模并行计算机系统。

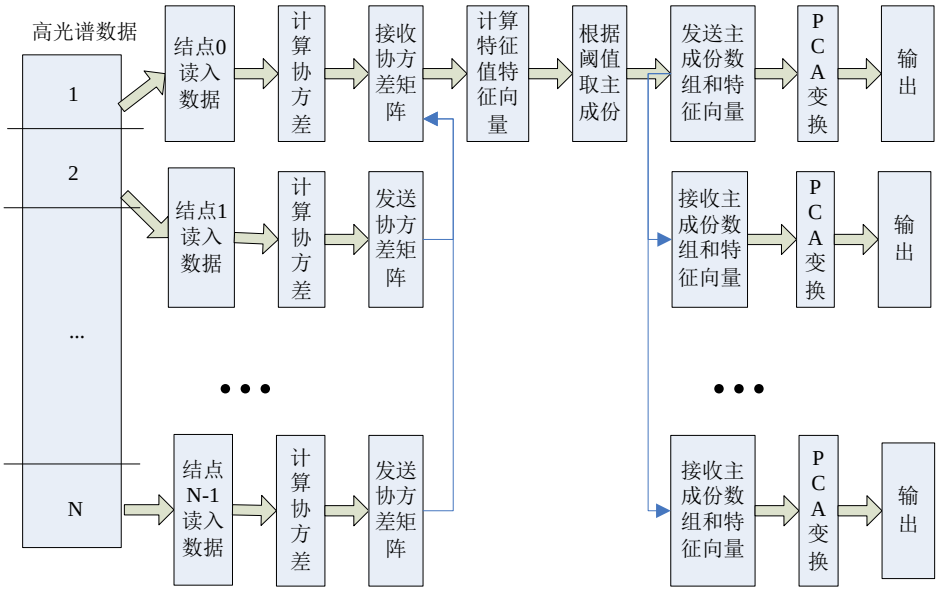

图 3.15 基于 MPI 的 PCA 降维算法流程图

3.3.2 基于 openMP 的 PCA 降维算法

针对单结点多处理器的共享存储系统，利用图 3.16 中的流程图，组织前文中的基于 openMP 的高光谱图像输入策略、基于 openMP 的协方差矩阵计算并行策略、基于 openMP 的并行 PCA 变换策略及基于 openMP 的结果输出策略，完成基于 openMP 的并行 PCA 降维算法。其中各个步骤间都需要同步来保证该步骤已经完成和下一个步骤的正确性。

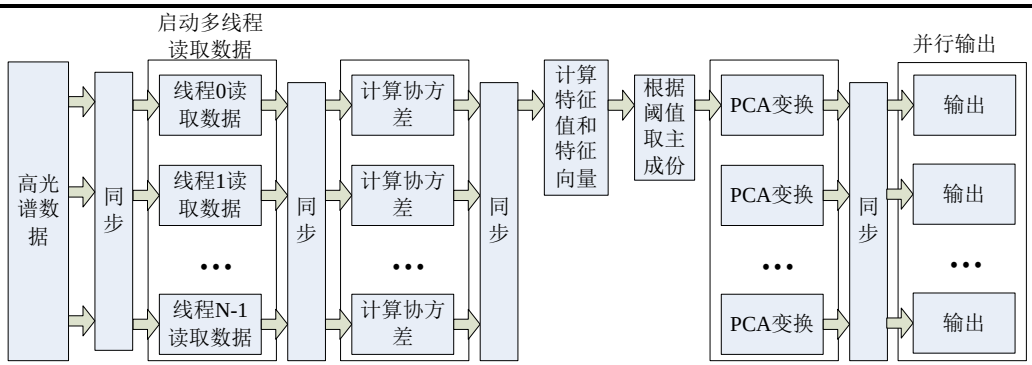

图 3.16 基于 openMP 的 PCA 降维算法流程图

3.3.3 基于 CUDA 的 PCA 降维算法

针对搭载了 GPU 众核处理器的 CPU/GPU 异构系统，提出一种基于 CUDA 的 PCA 降维算法。该算法采用了基于 CUDA 的协方差矩阵计算的 GPU 映射策略及基于 CUDA 的 PCA 变换映射方案，根据图 3.17 的流程图组织。算法中协方差的 GPU 映射共有 3 种方案 5 种实现策略，即可以实现 5 种不同的 CUDA 并行 PCA 降维程序，再比较产生性能最佳的方案，然后将该方案运用到后续的多级并行算法的开发中。

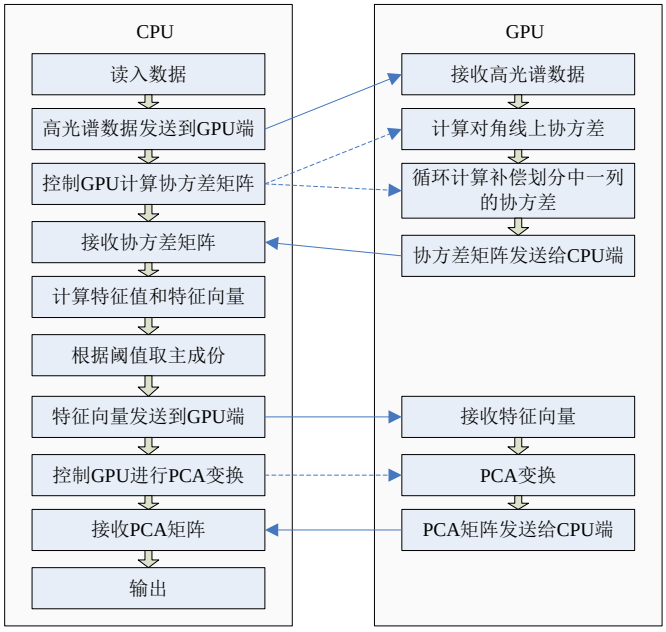

图 3.17 基于 CUDA 的 PCA 降维算法流程图

3.3.4 基于 MPI+CUDA 的 PCA 降维算法

基于 MPI+CUDA 的并行 PCA 降维算法：该算法的提出是为了充分利用多 GPU 集群（比如天河 1A 超级计算机）中的 GPU 计算能力，处理数据量巨大、计算复杂的高光谱遥感图像降维运算。算法根据图 3.18 中的流程图（图中 mul 表示∑XY），组织了基于 MPI 的并行高光谱图像数据输入、基于 MPI+CUDA 的两级并行协方差矩阵计算策略、

基于 MPI+CUDA 的两级协同并行 PCA 变换策略和 MPI 并行结果输出等并行策略。该算法的实现程序除了能够在 GPU 集群上正确计算高光谱 PCA 降维外，还能通过修改各进程的启动 GPU 的方法（比如进程 0 启动 GPU0，进程 1 启动 GPU1），使其能在多 GPU 结点上正确运行（本文实验平台下的算法验证就是采取了该方法）。

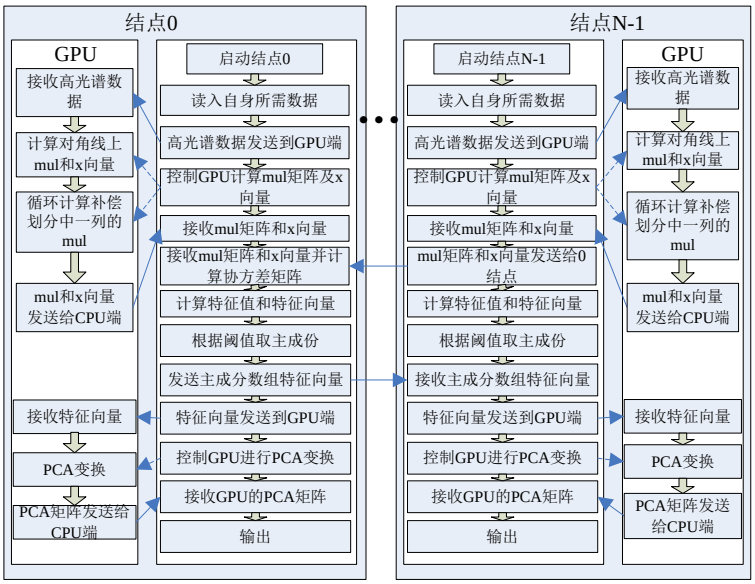

图 3.18 基于 MPI+CUDA 的 PCA 降维算法流程图

3.3.5 基于 openMP+CUDA 的 PCA 降维算法

利用图 3.19 中的基于 openMP+CUDA 的并行 PCA 降维算法流程图，结合前文描述的基于 openMP 的并行数据输入和并行结果输出策略、基于 openMP+CUDA 的两级并行协方差矩阵计算策略和两级并行 PCA 变换策略，形成基于 openMP+CUDA 的并行 PCA 降维算法。图中双横线间的部分表示并行区域，该区域的退出包含了同步操作，因此该算法并不需要额外的同步操作。该算法能在统一编址的多 GPU 结点内发挥巨大的并行优势，算法中利用统一的地址空间，实现了不同 GPU 间的直接数据通信，缩减了通信开销。在结点内部启动多个 GPU 运算 kernel 时，将会产生启动开销，因此在这里将数据输入和 GPU 启动空 kernel 函数并行，以减小不必要的开销。

3.3.6 基于 MPI+openMP+CUDA 的 PCA 降维算法

在 GPU 集群中，除了 GPU 巨大的计算能力外，还存在着多核 CPU 处理器资源，可以通过 MPI+openMP+CUDA 三级协同并行来充分开发这些资源。本文提出一种基于 MPI+openMP+CUDA 的 PCA 降维算法，其流程图见图 3.20（图中 mul 表示ΣXY），结合了基于 MPI+openMP 的两级并行高光谱数据输入、MPI+CUDA 的两级并行协方差矩阵计算、基于 MPI+CUDA 的两级并行 PCA 变换和基于 MPI+openMP 的两级并行结

果输出策略。该算法能最大化结点内单 GPU 的 GPU 集群的并行性能。

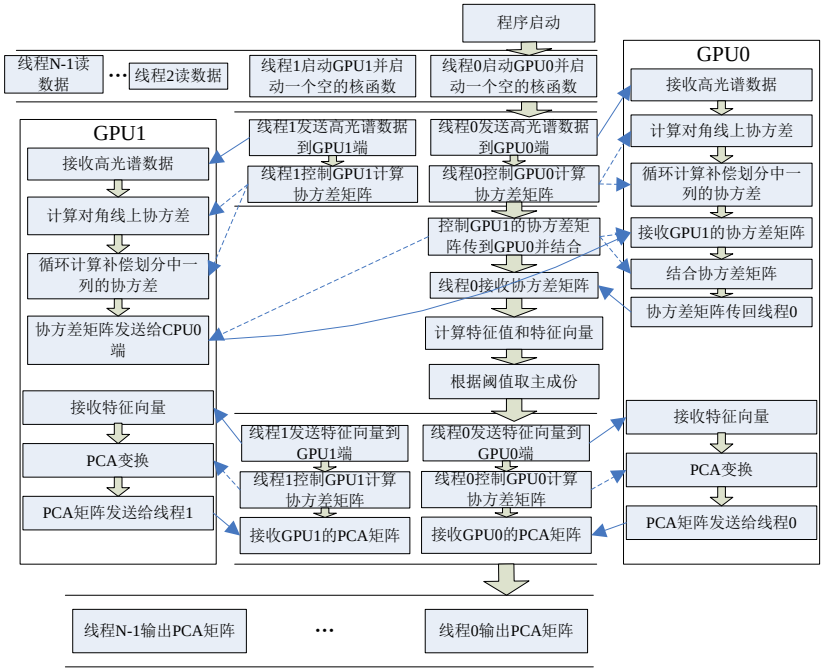

图 3.19 基于 openMP+CUDA 的 PCA 降维算法流程图

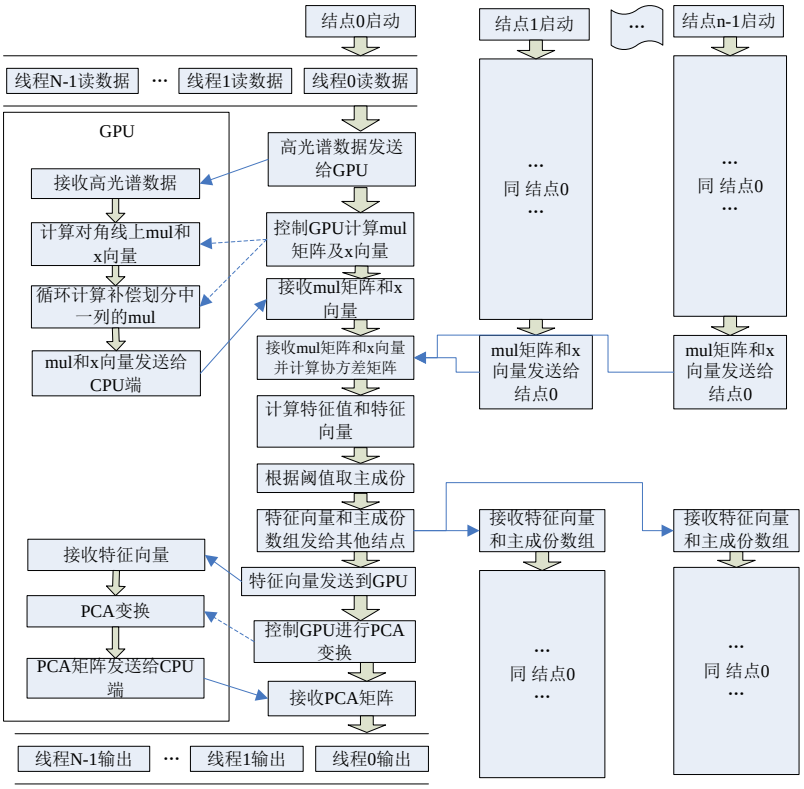

图 3.20 基于 MPI+openMP+CUDA 的 PCA 降维算法流程图

3.4 实验结果与性能分析

本章所采用的具体实验平台和实验数据详见本文 1.5 和 1.7 小节。

3.4.1 最佳优化开关的选取

并行程序的性能提升只有在对比最优的串行程序时才是有意义的，本节的目的就是获得最优的串行 PCA 降维程序，并以此为基准评价本章所开发的并行 PCA 算法的性能。

首先针对原始 PCA 降维程序进行串行优化，循环合并是其中的主要策略之一。在本章的协方差计算中，计算ΣX、ΣY、ΣXY 时，可以采用循环合并策略。在原始程序中，利用 3 次循环分别计算ΣX、ΣY 和ΣXY；经循环合并后，可在一次循环即可实现上述 3 种计算，相比较未优化的部分，可以节省 2 次循环开销。除了循环合并外，还有循环交换等优化策略，在本章串行程序中，PCA 变换采用了该策略。

进行串行程序的优化测试及优化开关选取。表 3.2 罗列了原始串行程序、优化后未开优化开关时及分别开启优化开关-O1、-O2、-O3 时，执行高光谱遥感图像数据 2 的执行时间。表 3.3 列出了其加速效果。

表 3.2 串行 PCA 程序优化执行时间表（ms）

数据 2	输入	协方差	特征值	PCA 变换	矩阵输出	总时间
原始	811.32	196087.2	776.64	45306.34	171.71	243154.1
O0	774.82	129518.6	768.23	41313.88	116.14	172492.1
O1	409.75	34144.43	311.17	32414.88	131.04	67411.76
O2	402.29	26619.43	283.7	31212.01	102.23	58620.11
O3	397.08	26784.33	282.56	32520.08	81.23	60065.79

表 3.3 串行 PCA 优化加速比

数据 2	输入	协方差	特征值	PCA 变换	矩阵输出	总时间
O0	1	1.5	1	1.1	1.5	1.4
O1	2	5.7	2.5	1.4	1.3	3.6
O2	2	7.4	2.7	1.5	1.7	4.1
O3	2	7.3	2.7	1.4	2.1	4

对比表 3.4 及表 3.5，经串行优化后，程序相对原始程序获得了一定的加速效果，其中在开启优化开关-O2 时，加速效果最明显，总运行时间对比原始程序获得了 4.1 倍加速比。因此选择打开优化开关-O2 的串行 PCA 降维程序作为 PCA 最优串行程序，并以此为标准与并行程序进行对比。

3.4.2 最优 CUDA 并行的选择

本章 3.2.1 中，提出了三种不同的协方差矩阵 GPU 映射策略以及 5 种不同的实现方案。在实现以上 5 种方案的基础上，对实验结果进行比较。表 3.4 为计算高光谱数据 2、

3、4 的协方差矩阵执行时间，其中Ⅱ1 表示方案Ⅱ中第①种实现策略，Ⅱ2 表示方案Ⅱ中第②种实现策略，Ⅱ3 表示方案Ⅱ中第③种实现策略。

表 3.4 5 种 CUDA 并行协方差矩阵计算时间（ms）

数据	Ⅰ	Ⅱ1	Ⅱ2	Ⅱ3	Ⅲ
2	6547.24	627.81	509.27	475.4	1089.46
3	23078.55	1355.88	1217.94	1033.33	1795.95
4	164218	5096.1	5463.58	4049.66	5332.14

表 3.4 中明显地显示方案Ⅰ的性能最差，方案Ⅱ中第③种实现策略是最好的，将方案Ⅱ中的③策略作为多级混合并行的参考策略。方案Ⅱ中的③策略能达到最佳性能，不仅因为方案Ⅱ的任务映射策略是最适用于 GPU 计算架构的，更因为③策略中我们提出了一种计算和通信重叠的优化策略，该策略几乎能隐藏全部的通信开销，而在数据量巨大时，通信开销是显著的。

这里解释下方案Ⅲ在数据小时加速性能差，而当数据量扩大后，其加速性能增长并接近方案Ⅱ中的映射策略。主要原因是方案Ⅲ中启动的 kernel 函数个数与波段数 B 相关，在高光谱图像较小（S=W*H 较小）时，计算所占比重较小，而大量的 kernel 启动开销较为明显，故其执行时间较慢；而当数据量大时，计算比重增加，相对的启动开销比率下降，导致其执行时间接近方案Ⅱ。

3.4.3 单级并行执行时间与性能分析

分析单级并行性能时主要采取数据量相对较小的数据 1、2、3、4，对于单级并行而言，这四组数据的测试结果已经足够说明其加速性能了。

协方差矩阵的三种单级并行执行时间（单位 ms）见表 3.5，图 3.21 描述了三种并行的加速比。图中显示，MPI 和 openMP 两种并行方案都获得了近 10 倍的加速效果，而 CUDA 并行获得了 50 多倍加速比，说明设计的 MPI、openMP、CUDA 这三种单级并行方案是合理的。对比而言，CUDA 并行获得了比 MPI 和 openMP 并行都明显的加速效果，说明了 GPU 在通用浮点计算上相对于 CPU 具有巨大的优势。

表 3.5 协方差矩阵单级并行执行时间（ms）

数据	最佳串行	MPI	OMP	CUDA
1	12688.89	1335.09	925.74	232.8
2	26619.43	2353.94	2484.85	475.4
3	58123.34	4821.48	5627.64	1033.33
4	219365.4	23177.49	21576.94	4049.66

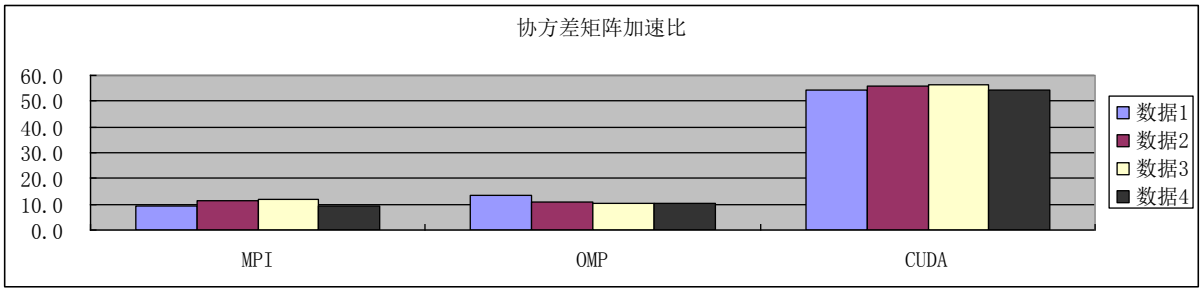

图 3.21 协方差矩阵的单级并行加速比

表 3.6 为 PCA 变换的最佳串行、MPI 并行、openMP 并行和 CUDA 并行的执行时间，其加速比见图 3.22。从实验数据中，可以明显看出 MPI 并行 PCA 变换较 openMP 并行快，且其加速比超过了结点数 16，获得了超线性加速比。主要原因是在 PCA 变换中，对高光谱图像 X 的切分成小数据存储，使得 cache 对其命中率提高，最终获得超线性加速比。同时 CUDA 并行能获得 300 多倍的性能提升，相对比其他两种并行有着巨大的优势。

表 3.6 PCA 变换单级并行执行时间（ms）

数据	最佳串行	MPI	OMP	CUDA
1	9469.55	815.2	696.18	37.39
2	31212.01	1099.52	3153.97	86.23
3	58557.2	2581.55	5820.16	185.11
4	128288.1	6588.23	10323.02	361.37

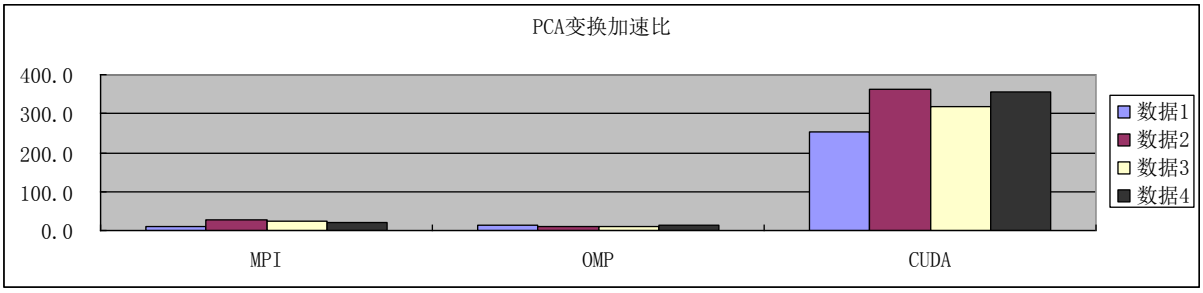

图 3.22 PCA 变换的单级并行加速比

测试并获得高光谱遥感图像数据 1、2、3、4 的单级并行的输入时间（表 3.7），与最佳串行时间比较，计算得到其加速效果（图 3.23）。结果显示 MPI 和 openMP 的并行输入都获得了较为明显的加速比，验证了本文提出的并行输入策略是有效的。由图 3.23 看出，尽管 CUDA 程序的输入是串行的，但其相对原始程序也获得了性能提升，主要原因是经本文预处理得到的高光谱遥感图像数据是以 unsigned char 类型（255 以内）存储的，在 GPU 上，经实验验证 unsigned char 类型直接参与运算不会产生过多开销，且 unsigned char 类型（1B）比 float 类型（4B）小，在通信和数据访问上更具优势；而在 CPU 上执行时，必须先将 unsigned char 类型转换成 float 类型，然后参与运算，否则计算时会产生巨大的延迟。因此 CUDA 程序的数据输入少了数据转换过程，比最佳串行程序输入更快。

表 3.7 单级并行输入执行时间（ms）

数据	最佳串行	MPI	OMP	CUDA（串）
1	208.59	40.31	51.3	31.99
2	402.29	95.79	76.31	67.66
3	763.38	275.63	105.79	144.38
4	2616.03	890.66	268.63	546.14

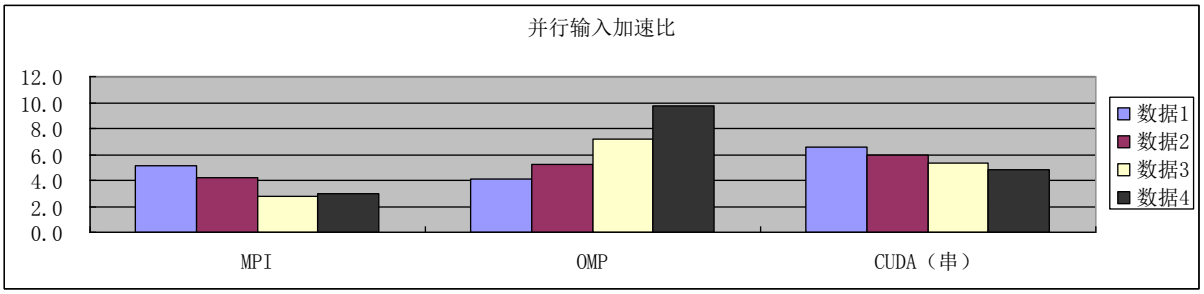

图 3.23 并行输入加速比

最后分析单级并行输出，时间测试见表 3.8，图 3.24 统计了各种并行的加速比。本文设计的两种并行输出 MPI 和 openMP 都获得了较为明显的性能提升，且该性能提升随着输出结果的扩大而增加，直到一个较为稳定的加速比。CUDA 程序的输出时间与最优串行程序的输出时间基本一致。

表 3.8 单级并行输出执行时间（ms）

数据	最佳串行	MPI	OMP	CUDA（串）
1	17.4	18.17	19.8	23.9
2	102.23	78.64	76.6	84.03
3	536.89	245.2	230.85	477.41
4	1030.59	321.4	264.49	1062.36

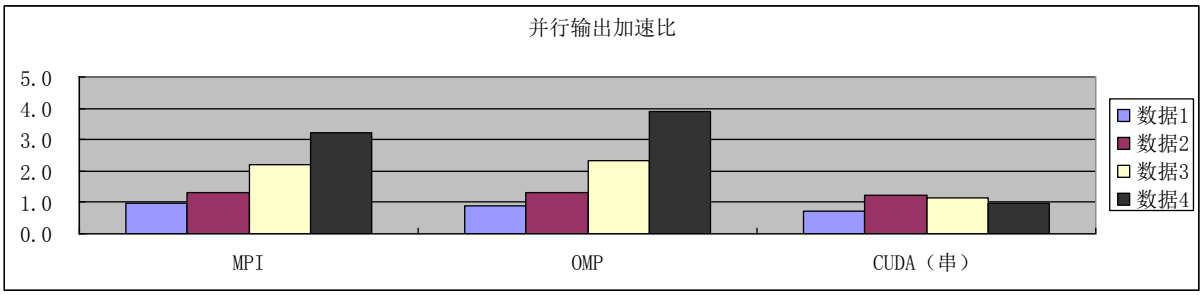

图 3.24 并行输出加速比

3.4.4 多级并行的优势

分析多级并行性能时，主要分析 4、5、6 三组高光谱图像数据，在数据量较大时，多级并行相对单级并行的优势将更明显，而且可以分析并行算法中针对数据量过大情况下的瓶颈问题。

实现并测试多级并行程序，统计多级并行协方差矩阵计算执行时间以及串行和单

CUDA 并行的执行时间，表 3.9 分别罗列了对高光谱数据 4、5、6 三组数据的执行时间（多级并行的优势在大数据中发挥得更好），计算各种并行的加速比，绘制成图 3.25。在本文实验平台（2 个 GPU）下，本章设计的 MPI+CUDA 和 openMP+CUDA 两个两级并行协方差计算策略均取得了理想的加速效果，其加速比达到了单一 CUDA 并行的两倍左右。

表 3.9 多级协方差矩阵并行计算执行时间（ms）

数据	最佳串行	CUDA	MPI+CUDA	OMP+CUDA
4	219365.4	4049.66	1958.38	2244.68
5	441330.1	8942.62	4133.3	4761.67
6	877791.3		9011.42	

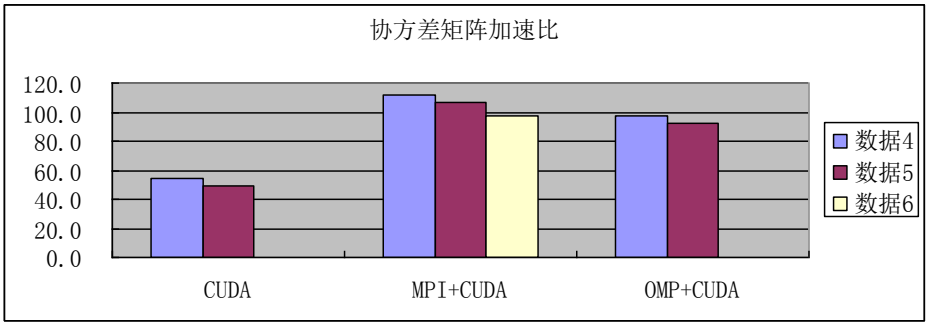

图 3.25 多级协方差矩阵并行计算加速比

从 3.9 表中，可以发现，当数据量大到一定的程度后，CUDA 程序和 openMP+CUDA 程序都无法运行，而 MPI+CUDA 程序却能正确执行。主要是由于高光谱遥感图像数据过于庞大，超过了 K20 GPU 的全局存储器容量，导致程序无法分配存储空间而无法运行。从设计上也能看出 CUDA 并行和 openMP+CUDA 并行都需要完整的高光谱图像数据才能计算协方差矩阵，而显存的不足必将导致其无法正常执行；而 MPI+CUDA 的两级并行设计中每个 GPU 计算时仅需用到局部数据，即使面临更加巨大的高光谱遥感图像数据降维问题，这种并行策略也能通过 GPU 数量的增加得到解决。

接着分析多级并行 PCA 变换的性能，表 3.10 为多级并行 PCA 变换的执行时间，其加速比见图 3.26。比较加速比，两级协同并行算法比 CUDA 并行算法快了近一倍，说明本章提出的基于 MPI+CUDA 的并行 PCA 变换和基于 openMP+CUDA 的并行 PCA 变换是合理有效的。

表 3.10 多级并行 PCA 变换执行时间（ms）

数据	最佳串行	CUDA	MPI+CUDA	OMP+CUDA
4	128288.1	361.37	160.86	206.69
5	253528.8	600.39	327.58	327.12
6	439860.4		487.97	

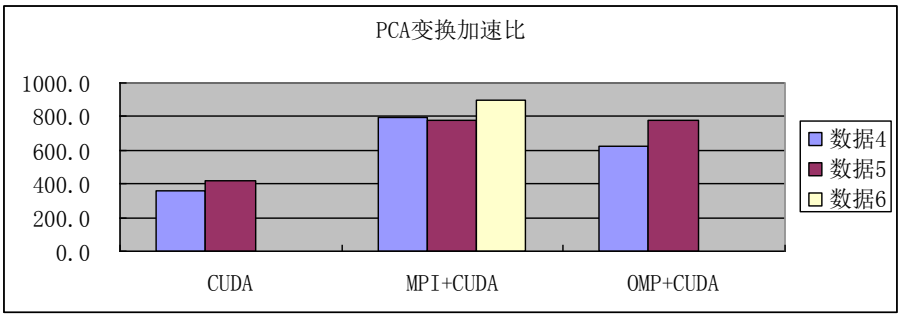

图 3.26 多级并行 PCA 变换加速比

表 3.11 为多级并行 I/O 执行时间，其中 MPI（CUDA）表示该时间在 MPI+CUDA 程序中测得，该程序在本文实验平台下共启动 2 个进程；输入部分以 CUDA 程序中的输入（不包含数据转换）作为比较。结果比较明显，启动了两个进程的 MPI 并行 I/O 加速了近两倍，而 MPI+openMP 的两级并行 I/O 的延迟比 MPI 的短很多，说明了两级并行 I/O 合理地利用了系统中剩余的 CPU 资源。

表 3.11 多级并行 I/O 执行时间（ms）

数据	输入			输出		
	CUDA	MPI（CUDA）	MPI+OMP	串行	MPI（CUDA）	MPI+OMP
4	546.14	246.23	79	1030.59	517.43	192.69
5	1034.6	501.65	144.62	2321.85	1123.63	959.49
6		996.6	259.32	3985.4	2012.63	602.6

3.4.5 总时间与总加速比

实现并执行 3.3 节提出的 6 种并行算法，测试其执行时间（不考虑 I/O 时间），记录在表 3.12 中，其中 MCU 表示 MPI+CUDA 程序，OCU 表示 openMP+CUDA 程序，MOC 表示 MPI+openMP+CUDA 程序，并计算加速比绘制成表 3.13，其中 MCU 和 MOC 程序的计算部分（不含 I/O）是相同的，因此这里仅列出了 MOC 的数据。观察这两个表格，可以发现并行算法的加速效果是较为明显的，特别是 MPI+openMP+CUDA 的三级并行程序，其计算加速比（不含 I/O）最高达到了 145 倍。

表 3.12 PCA 算法执行时间（不包括 I/O）（s）

数据	最佳串行	MPI	OMP	CUDA	OCU	MOC
1	22.41	2.44	1.91	0.62	0.51	0.50
2	58.12	3.78	5.96	0.94	0.71	0.70
3	117.00	7.83	11.79	1.54	1.02	1.00
4	347.99	30.15	32.26	4.74	2.82	2.46
5	695.19	44.10	61.21	9.87	5.40	4.79
6	1317.98	87.90	117.70			9.81

表 3.13 各并行 PCA 算法的加速比（不包括 I/O）

数据	MPI	OMP	CUDA	OCU	MOC
1	9.2	11.7	36.0	44.3	45.0
2	15.4	9.8	61.7	82.2	83.3
3	14.9	9.9	76.0	114.7	117.2
4	11.5	10.8	73.4	123.4	141.6
5	15.8	11.4	70.5	128.8	145.2
6	15.0	11.2			134.3

本文除了对传统的计算部做了并行研究外，还对高光谱数据 I/O 等部分进行了并行化设计。测试其总执行时间（包括 I/O），记录在表 3.14 中，并计算各并行程序相对最佳串行程序的加速比，如表 3.15 所示。在考虑了 I/O 时间后，各种并行算法的加速效果依然明显，其中结合了三种并行机制的 MPI+openMP+CUDA 三级并行 PCA 算法获得了最佳的加速效果，总加速比达到了 128 倍。

表 3.14 PCA 算法总执行时间（包括 I/O）（s）

数据	最佳串行	MPI	OMP	CUDA	MCU	OCU	MOC
1	22.63	2.5	1.98	0.68	0.55	2.03	0.52
2	58.62	3.97	6.11	1.09	0.92	2.27	0.76
3	118.3	8.62	12.13	2.16	1.43	2.68	1.11
4	351.64	32.11	32.79	6.35	3.36	5.02	2.73
5	702.12	48.3	62.48	13.13	6.67	7.46	5.89
6	1331.37	97.46	119.64		13.21		10.68

表 3.15 各并行 PCA 算法的总加速比（包括 I/O）

数据	MPI	OMP	CUDA	MCU	OCU	MOC
1	9	11.4	33.3	41.1	11.2	43.7
2	14.8	9.6	53.5	63.9	25.9	76.8
3	13.7	9.8	54.7	82.6	44.2	107.1
4	11	10.7	55.4	104.7	70.1	128.8
5	14.5	11.2	53.5	105.3	94.1	119.2
6	13.7	11.1		100.8		124.7

将表 3.15 中的总加速比绘制成图 3.27，从图中的整体趋势中可以看出，当数据量扩大时，并行算法的加速比呈线性增长，当数据量达到一定规模后，其加速比达到最大，继续扩大数据，加速比将有所下降并处于一个较为稳定的区间。分析其原因：数据量较小时，并行计算所占比重较少，加上并行引发的额外开销较明显，导致了其加速比较小；数据扩大后，计算比率增加，加速比随之增加；当达到某个值（比如 MPI+openMP+CUDA 三级并行算法处理数据 4 时），加速比达到了最大，此时整个平台的 GPU 计算能力达到了饱和状态；超过该值后，加速比会有所下降并处于一个稳定的区间，主要因为是 GPU 计算能力达到饱和后，数据增大时，非 GPU 计算部分（比如并行 I/O）没法达到如 GPU 相当的加速效果（不考虑 I/O 时能有 140 多倍的加速比，而考虑 I/O 后加速比仅

能达到 120 倍左右），根据 Amdahl 定律，将影响整体的程序性能。

要想进一步提升程序性能，主要有以下几种方法：1）增加 GPU 数量，GPU 数量增加可以进一步提高并行度，增加并行计算部分加速比；2）使用更强的 GPU，Tesla K20 是目前最强的 GPU，更强的 GPU 还需时间开发；3）提升 CPU 性能，增加 CPU 性能可以压缩串行计算部分时间（比如协方差的计算），达到提升加速比的目的；4）增加 CPU 核数和减少并行开销，增加核数可以提高并行度，并减少并行区域的启动、调度、结束等开销，可以增加 I/O 部分的并行性，进一步提升程序执行时间。

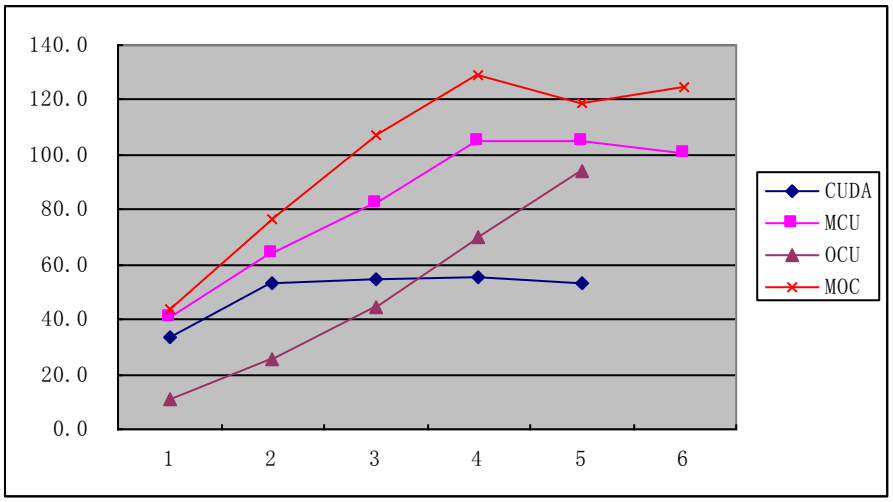

图 3.27 总加速比趋势图

3.5 本章总结

本章主要研究了经典高光谱遥感图像线性降维算法中的主成分分析 PCA 算法。本章首先介绍了主成分分析理论及其在高光谱遥感图像降维领域的应用，并对其进行了加速热点分析。然后重点根据并行热点介绍了对于不同并行及混合并行的设计思路和策略，提出了相应的优化措施；将这些热点的并行策略进行组合，形成完整的 PCA 并行降维算法。最后在实验中验证了本章提出的并行热点设计，各种并行 PCA 降维算法均取得了良好的加速效果，其中 MPI+openMP+CUDA 三级并行 PCA 降维算法取得了最高 145 倍的执行加速比（不含 I/O）和最高 128 倍的总加速比（含 I/O）。

第四章　FastICA 降维多级并行算法研究与优化

除了第三章所叙述的 PCA 方法，独立成分分析（ICA）也是一种经典的线性降维方法。ICA 利用统计独立目标来分离独立成分信号，利于保留小类别、小目标信息，且与高光谱图像中的特征是有联系的[53]，因此经 ICA 降维的高光谱图像，相对 PCA 降维，更有利于高光谱图像处理。

本文对基于负熵最大的 FastICA 高光谱降维算法进行研究，提出基于 MPI、openMP 及 CUDA 三个并行层次的多种并行 FastICA 算法，并在不同层次上对算法进行优化研究。实验表明本文提出的并行算法都能获得优良的加速效果，各并行层次能进行互补，获得更大的加速效果。

4.1 独立成分分析流程及算法并行性分析

4.1.1 ICA 基本模型

独立成分分析是从线性混合信号中恢复基本源信号的方法，其模型如图 4.1 所示。该模型描述了未知源信号生成观测数据的过程，其中源信号是隐藏的，不能直接观测，而混合矩阵未知，能观测到的数据是随机变量 x。

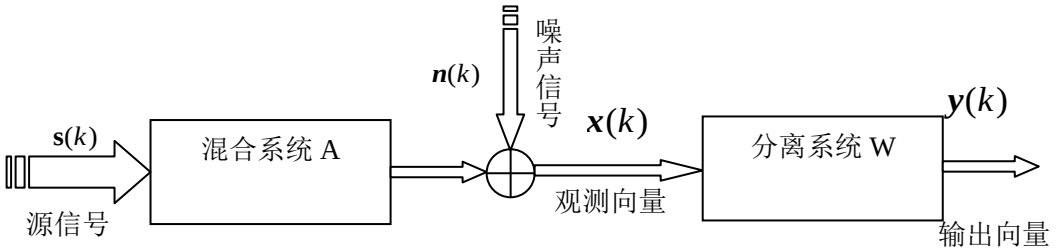

图 4.1 ICA 模型

文献[81]对独立成分分析做了如下解释：

假定第 i 个观测信号 x 由 n 个相互独立的未知信号 s 线性混合而成

$$\boldsymbol{x}_i = a_{i1}\boldsymbol{s}_1 + a_{i2}\boldsymbol{s}_2 + \cdots + a_{in}\boldsymbol{s}_n \quad , \quad i = 1,2,\cdots,m \tag{4.1}$$

假定每个观测变量和未知源变量都是随机变量，矢量 $\boldsymbol{x}_i$ 表示观测变量 $[x_1, x_2, \cdots, x_3]$，矢量 $\boldsymbol{s}_j$ 表示源变量 $[s_1, s_2, \cdots, s_3]$，A 表示混合矩阵 a_{ij}，上式可以用矢量矩阵形式表示：$\boldsymbol{X} = \boldsymbol{A}\boldsymbol{s}$，也可写成：$x_i = \sum_{j=1}^{n} a_{ij} s_j$

为保证 ICA 模型可解，需要假设：①统计独立性、②非高斯分布、③独立分量数据等于观测混合信号数量（即 A 矩阵是方的）、④各观测器引入噪声 n 最小，可不予考虑。

然后通过计算矩阵 A 的逆 W，就可解出独立分量 $s = \boldsymbol{W}x$ 。

4.1.2 基于负熵最大的 FastICA 算法

FastICA 算法，固定点(Fixed-Point)算法，该算法有基于负熵最大、基于似然最大、基于峭度等 3 种形式，本文主要研究基于负熵最大的 FastICA 算法。该算法以负熵最大作为搜寻方向，实现独立源的顺序提取，充分体现传统线性变换中的投影追踪（Projection Pursuit）思想，该算法还采用定点迭代的优化算法，使收敛更加快速、稳健[82]。

负熵判决准则[83]：由信息论理论可知：在所有等方差的随机变量中，高斯变量的熵最大，因而可以利用熵来度量非高斯性，常用熵的修正形式，即负熵。根据中心极限定理，若一随机变量 X 由许多相互独立的随机变量 $S_i(i = 1,2,3,...N)$ 之和组成，只要 S_i 具有有限的均值和方差，则不论其为何种分布，随机变量 X 较 S_i 更接近高斯分布。换言之，S_i 较 X 的非高斯性更强。因此，在分离过程中，可通过对分离结果的非高斯性度量来表示分离结果间的相互独立性，当非高斯性度量达到最大时，则表明已完成对各独立分量的分离。

负熵的定义[83]：$N_g(Y) = H(Y_{Gauss}) - H(Y)$，$Y_{Gauss}$ 是一与 Y 具有相同方差的高斯随机变量，$H(Y) = -\int p_Y(\xi)\lg p_Y(\xi)d\xi$ 是随机变量的微分熵；根据信息理论，在具有相同方差的随机变量中，高斯分布的随机变量具有最大的微分熵；当 Y 具有高斯分布时，$N_g(Y) = 0$，Y 的非高斯性越强，其微分熵越小，$N_g(Y)$ 值越大，所以 $N_g(Y)$ 可以作为随机变量 Y 非高斯性的测度；计算微分熵需要知道 Y 的概率密度分布函数，这显然不切实际，于是采用如下近似公式：

$$N_g(Y) = \{E[g(Y)] - E[g(Y_{Gauss})]\}^2 \tag{4.2}$$

其中，$E[\cdot]$ 为均值运算；$g(\cdot)$ 为非线性函数，可取 $g_1(y) = \tanh(a_1 y)$，或 $g_2(y) = y\exp(-y^2/2)$ 或 $g_3(y) = y^3$ 等，其中，$1 \le a_1 \le 2$，通常取 $a_1 = 1$。

FastICA 学习规则[83]是找一个方向以便 $W^T X(Y = W^T X)$ 具有最大的非高斯性，非高斯性用式给出的负熵 $N_g(W^T X)$ 的近似值来度量，$W^T X$ 的方差约束为 1，对于白化数据而言，这等于约束 W 的范数为 1。

实践中，FastICA 算法中用的期望必须用它们的估计值代替，当然最好的估计是相应的样本平均；理想情况下，所有的有效数据都应该参与计算，但这会降低计算速度。所以通常用一部分样本的平均来估计，样本数目的多少对最后估计的精确度有很大影响。迭代中的样本点应该分别选取，假如收敛不理想的话，可以增加样本的数量[84]。

4.1.3 基于高光谱遥感图像降维的 FastICA 算法流程

对于高光谱数据 X（W*H*B），X 可看成是 B 行 S 列的二维矩阵，矩阵的一行表示一个波段的高光谱数据。通过 ICA 高光谱降维，可以获得 m 个 IC 图像，其中 m<B，从而实现降维效果。本文 m 的选取通过 X 的协方差矩阵特征值的比较阈值进行选择。具体算法如下：

step1. 计算 X 的协方差矩阵；

step2. 计算Σ x 的特征值及相应的特征向量，并根据阈值 T 选取最终 IC 图像数量 m；

step3. 计算白化矩阵 $M = D^{-\frac{1}{2}}V^T$，其中 D 为 S 的特征值矩阵，V 为对应的特征向量矩阵；

step4. 高光谱图像数据白化处理 $Z = MX$，这里的 X 为减去均值之后的 X（即零均值处理）；

step5. 迭代开始：

①初始化 W_i；

②计算 $W_i = E\{Zg(W_i^T Z)\} - E\{g'(W_i^T Z)\}W$ （4.3）

其中 g 为非线性函数，可取 $g_1(y) = \tanh(a_1 y)$，或 $g_2(y) = y\exp(-y^2/2)$ 或 $g_3(y) = y^3$ 等。

③正交化 $W_i = W_i - \sum_{j=1}^{i-1}(W_i^T W_j)W_j$ （4.4）

④归一化 $W_i = W_i / \|W_i\|$ （4.5）

⑤判断是否收敛，不收敛则返回到步骤②；若收敛，如果所有的 w_i 已经计算结束（即 i=m），则退出，否则令 $i = i+1$，返回到步骤①计算新的 w_{i+1}

step6. 计算独立成分 Y=WZ，其中 $W_1^T, W_2^T, \cdots W_m^T$ 为 W 所对应的行向量

4.1.4 并行热点分析

串行实现上述算法，执行串行程序并测试时间，获得表 4.1 所示的时间数据（开启优化开关-O2）。观察表 4.1，在 ICA 算法中，除了特征值的计算外，其他步骤所消耗的时间均随高光谱数据量的扩大（数据 1 到 6 数据量递增）而线性增长。特征值计算部分仅占总时间的较少一部分，特别是数据量扩大时，其所占比例几乎可以忽略不计，根据 Amdahl 定律，即使通过并行加速特征值的计算部分，也几乎无法获得的总的性能提升，故特征值的计算不纳入加速热点。

表 4.1 串行 ICA 执行时间（s）

数据	输入	协方差	特征值	白化	ICA 迭代	IC 变换	输出	总时间
1	0.22	12.67	0.25	9.91	10.31	0.16	0.02	33.55
2	0.41	26.90	0.28	33.62	49.26	1.61	0.08	112.16
3	0.76	59.38	0.32	60.97	253.54	3.78	0.59	379.34
4	2.61	218.08	0.33	135.83	368.17	2.56	1.07	728.65
5	5.13	431.57	0.33	253.47	741.25	4.24	2.18	1438.16
6	10.30	865.82	0.33	449.21	520.27	5.52	4.06	1855.51

计算各步骤执行时间所占整体程序的总执行时间的比率，参见表 4.2。明显的，协方差、白化和 ICA 迭代这三个步骤所占的比率是最高的，这三部分的占总执行时间的比重达到了惊人的 97%~99%，毫无疑问，这三个步骤是本文研究的主要并行热点。其余的高光谱数据输入、IC 变换及结果输出这三部分占总时间的比重较为稳定，在数据量扩大时，该三个部分的时间消耗会线性增长；但其所占的比重较少，故可以作为次要并行热点进行研究。

表 4.2 ICA 各步骤占总时间百分比

数据	输入	协方差	特征值	白化	ICA 迭代	IC 变换	输出
1	0.66%	37.77%	0.75%	29.54%	30.74%	0.48%	0.05%
2	0.37%	23.98%	0.25%	29.97%	43.92%	1.44%	0.07%
3	0.20%	15.65%	0.08%	16.07%	66.84%	1.00%	0.15%
4	0.36%	29.93%	0.05%	18.64%	50.53%	0.35%	0.15%
5	0.36%	30.01%	0.02%	17.62%	51.54%	0.29%	0.15%
6	0.56%	46.66%	0.02%	24.21%	28.04%	0.30%	0.22%

4.2 FastICA 算法并行热点研究

本节根据 MPI、openMP 及 CUDA 三种不同的并行机制，针对 FastICA 算法的主要并行热点和次要并行热点分别进行研究，深入挖掘各步骤的最佳并行方案，并提出混合并行机制的多级协同并行策略，最终形成并行 FastICA 降维算法。

并行热点中协方差矩阵计算、高光谱图像数据输入、结果输出的并行方案和优化策略已在第三章详细研究，本章的并行 FastICA 程序将直接使用第三章中提出并验证了的一些有效的并行方案。

4.2.1 并行白化处理研究

白化处理 $Z = MX$ 作为高光谱图像降维 FastICA 算法中的重要部分，且其执行时间占总执行时间比例较高，是并行研究的重点之一。白化处理包含了高光谱图像数据的特点，在对其并行研究时需要考虑该点。从宏观上，白化处理可以看成一个简单的矩阵乘法模型（图 4.2），可根据其宏观模型开发并行，然后在实现时考虑其具体操作，比如

X 需要减去均值等。

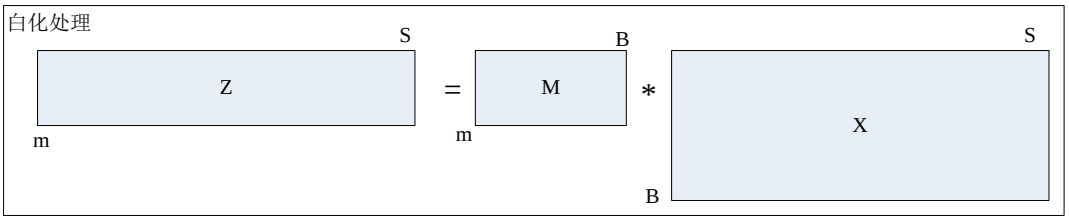

图 4.2 白化处理矩阵乘模型

在将白化处理转换成矩阵乘模型后，就可以发现该模型与第三章所研究的 PCA 变换过程及其类似，因此可以借鉴第三章中的并行 PCA 变换研究成果。

基于 MPI 的并行白化处理：在 MPI 层，将 X 矩阵按图 3.9 左的方案划分成 N 个 Xi 局部矩阵，然后各结点（进程）计算局部数据 $Z_i=M*X_i$，最后通过通信将局部结果收集到 root 结点，获得完整的 Z 矩阵。当然，这个局部结果矩阵是巨大的，会带来额外的开销。事实上，局部矩阵的通信并不需要，因为通过后面对 ICA 迭代和 IC 变换的并行 MPI 改进，结点后续计算只需要使用其局部结果 Z_i 即可。

基于 openMP 的并行白化处理：在 openMP 层，对矩阵 M 划分，将 m 按线程数量 n 进行均匀划分成 n 个 M_i 局部矩阵，分配给各独立线程，每个线程只需处理 M_i*X。最后通过同步操作来保证所有线程均已完成局部白化处理。

基于 CUDA 的并行白化映射策略：仿照 PCA 变换中的映射策略，为发挥共享存储器的性能，获得最大化的 occupancy 占用率，在 CUDA 处理白化时，循环启动 m 个<<<gridDim,blockDim>>>核函数计算白化过程，每次完成 M 的一行和 X 的乘法操作。每个 kernel 函数启动 gridDim 个 block，每个 block 中启动 blockDim 个 thread，对于每个线程，需要循环处理 X 矩阵中的第（blockIdx*blockDim+threadIdx）+gridDim*blockDim*i 个列中的向量内积运算。

基于 MPI+CUDA 的两级并行白化处理：针对于白化处理过程，MPI 负责对矩阵 X 的分配和相关的通信，各结点通过基于 CUDA 的白化处理映射策略，将局部矩阵的运算发送到 GPU 上实现并行计算。除了计算任务的分配，由 MPI 启动的各进程还要负责该结点内主机端和设备端的通信。

基于 openMP+CUDA 的两级并行白化处理：openMP+CUDA 两级并行白化处理时，程序的线程数量根据 GPU 数量而定，然后按上述 openMP 并行中的划分策略对 M 矩阵划分，每个线程分配到 m/n 个 M 行向量乘 X 矩阵的任务，然后采用上述 CUDA 并行策略，通过 kernel 函数，将每个行向量乘 X 矩阵的任务映射到 GPU 上进行计算。最后需要同步保证后续任务的正确执行。

4.2.2 ICA 迭代并行研究

分析 ICA 迭代过程，可以发现该过程中每次的迭代都需要大量复杂计算，无法进行简单的任务划分。本文提出一种“具体→抽象→具体”的并行设计模式，首先分析具体的算法流程，抽象出简单的宏观模型，针对该模型进行并行设计，在实现时，将具体的计算步骤映射到并行策略中。

针对 ICA 迭代，由于数据依赖，不同的迭代间无法实现并行处理，那么只能在单次迭代中开发并行。分析迭代中各个步骤的计算量，初始化、正交化、归一化和判断操作的计算量有效，主要的计算量集中在 $W_i = E\{Zg(W_i^T Z)\} - E\{g'(W_i^T Z)\}W$ 中，再分解该式，计算量最集中的过程包括 $W_i^T Z$ 和 $Zg(W_i^T Z)$。对计算密集的步骤进行分析，可以抽象出图 4.3 中的 ICA 迭代宏观模型。

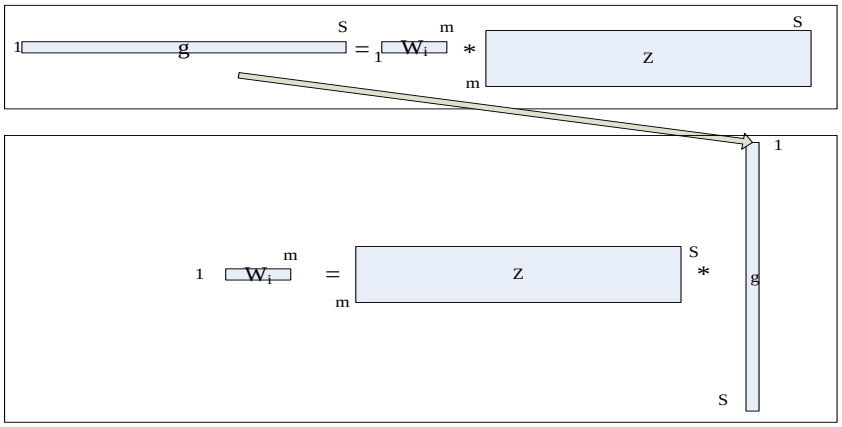

图 4.3 ICA 迭代宏观模型

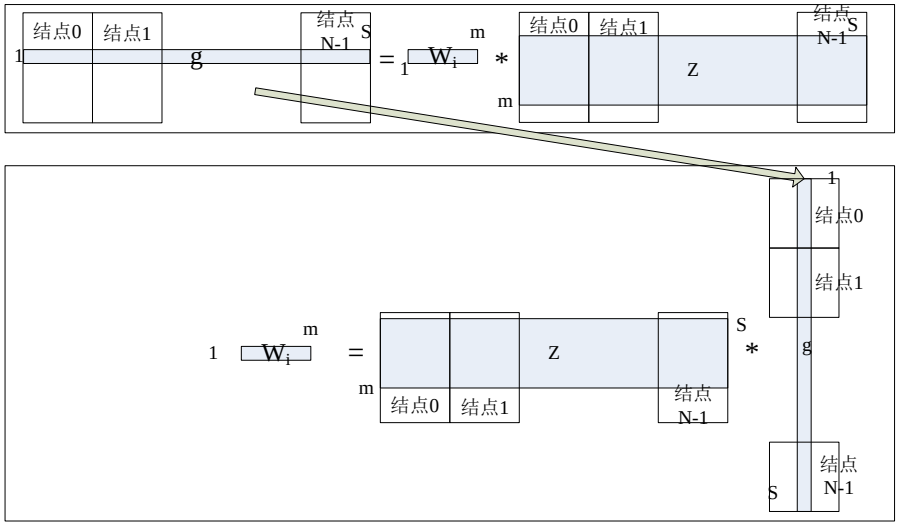

图 4.4 基于 MPI 的 ICA 迭代并行模型

基于 MPI 的 ICA 迭代并行设计：针对 ICA 迭代宏观模型，存在一种较为简单的划分方法，见图 4.4。在该方案中，各结点能较好地达到均衡负载，且仅需要两次通信，

即各个结点计算得到各自的 W_{ij}，j=0,1...N-1，然后将其传递给 root 结点，再由 root 结点对其进行综合计算。root 结点计算得到结果后再将 Wi 广播给所有结点，由于归一化和正交化计算量较少，每个结点可自行计算，并以此判断是否收敛。

一种比较简单的 ICA 迭代 openMP 并行设计：矩阵乘实质上就是求向量内积，在 openMP 中提供了较好的求向量内积的方法，即循环归约（for reduction）。对上述两个过程进行分析，首先在 $W_i^T Z$ 中，实质上是求长度为 m 的两个向量内积，共需求 S 次，由于 S>>m，如果用归约，需要启动 S 个并行区域，每个并行仅处理 m 的运算，这种方法明显并不可取，此时考虑增加 openMP 中程序粒度，类似 MPI 中的划分方法将其映射到线程上，即可获得较好的效果。再考虑 $Zg\left(W_i^T Z\right)$，此时相当于循环 m 次求 S 长的向量内积，此时启动 m 个并行区域，每个并行区域的计算量为 S，S>>m，比较合理；除此之外，还可以对 Z 矩阵中的 m 按线程进行均衡划分，即每个线程处理 m/n 个（n 为线程数量）S 长的向量内积运算。

基于 CUDA 的 ICA 迭代 GPU 映射方案设计： $W_i^T Z$ 的计算过程类似于白化处理过程，在用 CUDA 开发并行时，可以借鉴白化处理的方案，即启动<<<gridDim,blockDim>>>的核函数计算。 $Zg\left(W_i^T Z\right)$ 过程的特点是单次向量内积计算量大，计算次数少，故可以启动<<<gridDim,blockDim>>>的核函数，用来计算单次向量内积计算中的乘法操作，对于每个线程，循环处理 X 矩阵中的第（（blockIdx*blockDim+threadIdx）+gridDim*blockDim*i）个乘法并相加，可以得到 gridDim*blockDim 个结果，再进行 block 内部归约，即可得到 gridDim 个中间结果；由于 GPU 内 block 间无法同步，所以需要重新启动一个<<<1，gridDim >>>的核函数，让其对 gridDim 个中间结果进行归约，最终得到大向量的内积。如此循环 m 次，即完成 ICA 迭代过程中的一次迭代计算。

基于 MPI+CUDA 的两级并行 ICA 迭代计算。分析上述的 MPI 和 CUDA 的 ICA 迭代并行策略，两种并行策略是互补的，可利用 MPI 进行划分、控制和通信，CUDA 负责计算。由于 GPU 的计算结果在设备端，而不同结点间的设备端是无法直接通信的，因此在交互时，需要先将数据传输到主机端，然后通过消息传递到目标结点的主机端，再利用 CUDA 函数将消息传入设备端。

基于 openMP+CUDA 的两级并行 ICA 迭代计算。除了上述较为简单的 openMP 并行方法外，基于 ICA 迭代抽象模型，提出一种可以配合 CUDA 使用的 openMP 并行划分方案，如图 4.5 所示。这是一种跟 MPI 完全不同的分割方案，且又能与 MPI 进行互补。通过该方案，不同线程间的任务更加明确，但也会引入一些额外的运算（比如图中 g0+g1），还有一些不同 GPU 间的通信开销（比如 g0、g1、Wi 等）。其中具体的计算通过 CUDA 程序映射到 GPU 上进行。

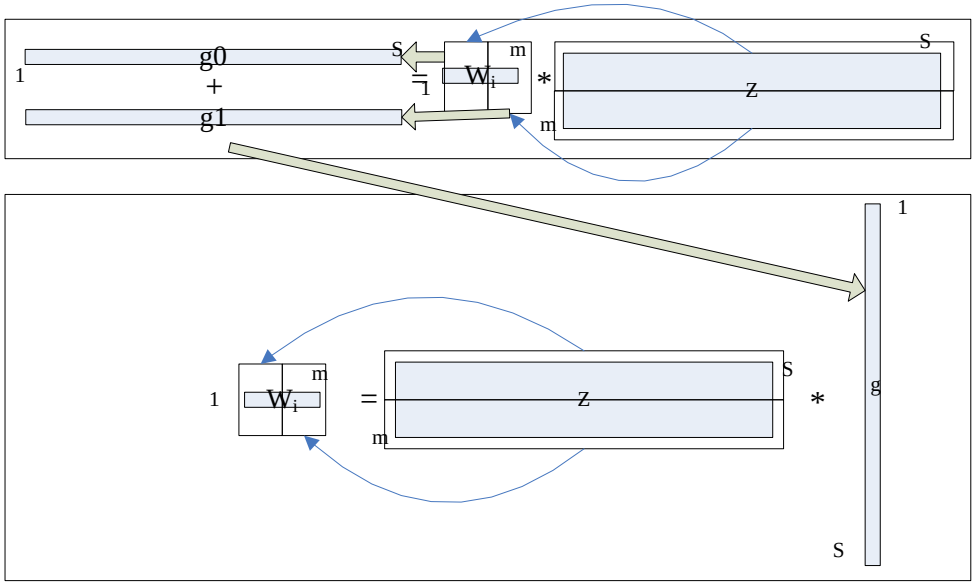

图 4.5 基于 openMP 的 ICA 迭代并行计算

在具体实现 openMP+CUDA 算法时，由于数据交互，因此需要大量的同步操作，其同步和通信的具体设计详见 4.3.5。且其中需要交互的数据 g0、g1 数据量庞大（S 个浮点数据），通信将产生巨大的开销。该开销引发的时间耗费甚至可能大于多 GPU 带来的计算收益。

4.2.3 IC 变换并行研究

独立成分计算过程 Y=WZ 是典型的矩阵乘法，类似白化处理。对于 IC 变换的并行设计可以借鉴白化处理的并行设计，在 MPI 层对 Z 矩阵（S*m）按列划分，将 S 列均匀分配给每个结点，各结点只需处理各自的局部矩阵乘即可，并不需要额外的数据通信（需配合 MPI 并行输出，否则非 root 结点需将 Y 局部矩阵发送给 root 进行组合输出）；在 openMP 层，将 W 矩阵（m*m）按行划分，将 m 行平均分配给 n 个线程；基于 CUDA，通过循环启动 m 次<<<gridDim,blockDim>>>的 kernel 函数，kernel 函数内每个线程处理 S/(gridDim*blockDim)个向量内积运算。

基于 MPI+CUDA 的两级并行 IC 变换：根据上述 MPI 并行策略，确定每个结点的计算任务，即原始公式 Y=WZ 中 Z 矩阵的大小由 S 变为 S/N；然后将这些计算任务按照 CUDA 并行的计算策略映射，通过循环启动 m 次<<<gridDim,blockDim>>>的 kernel 函数实现 IC 变换过程。

基于 openMP+CUDA 的两级并行 IC 变换：设置 GPU 数量为启动的线程数量，启动各自的 GPU，然后按上述 openMP 并行中的划分策略对 W 矩阵按行划分，每个线程分配到 m/n 个行向量乘 Z 矩阵的任务；后通过 kernel 函数，将每个行向量乘 X 矩阵的任务映射到关联的 GPU 上进行计算。

4.3 6 种 FastICA 并行算法

根据前面所述的并行策略，可以组织成多种不同的并行 FastICA 降维算法：

4.3.1 基于 MPI 的 FastICA 算法

采用各种并行策略中的 MPI 策略，比如 MPI 并行输入（详见 3.2.3 小节）、MPI 并行协方差矩阵计算（详见 3.2.1 小节）、MPI 并行白化处理、MPI 并行 ICA 迭代、MPI 并行 IC 变换、MPI 并行输出（详见 3.2.4 小节）等，结合 FastICA 算法流程图（图 4.6），提出了一种基于 MPI 的 FastICA 并行降维算法。

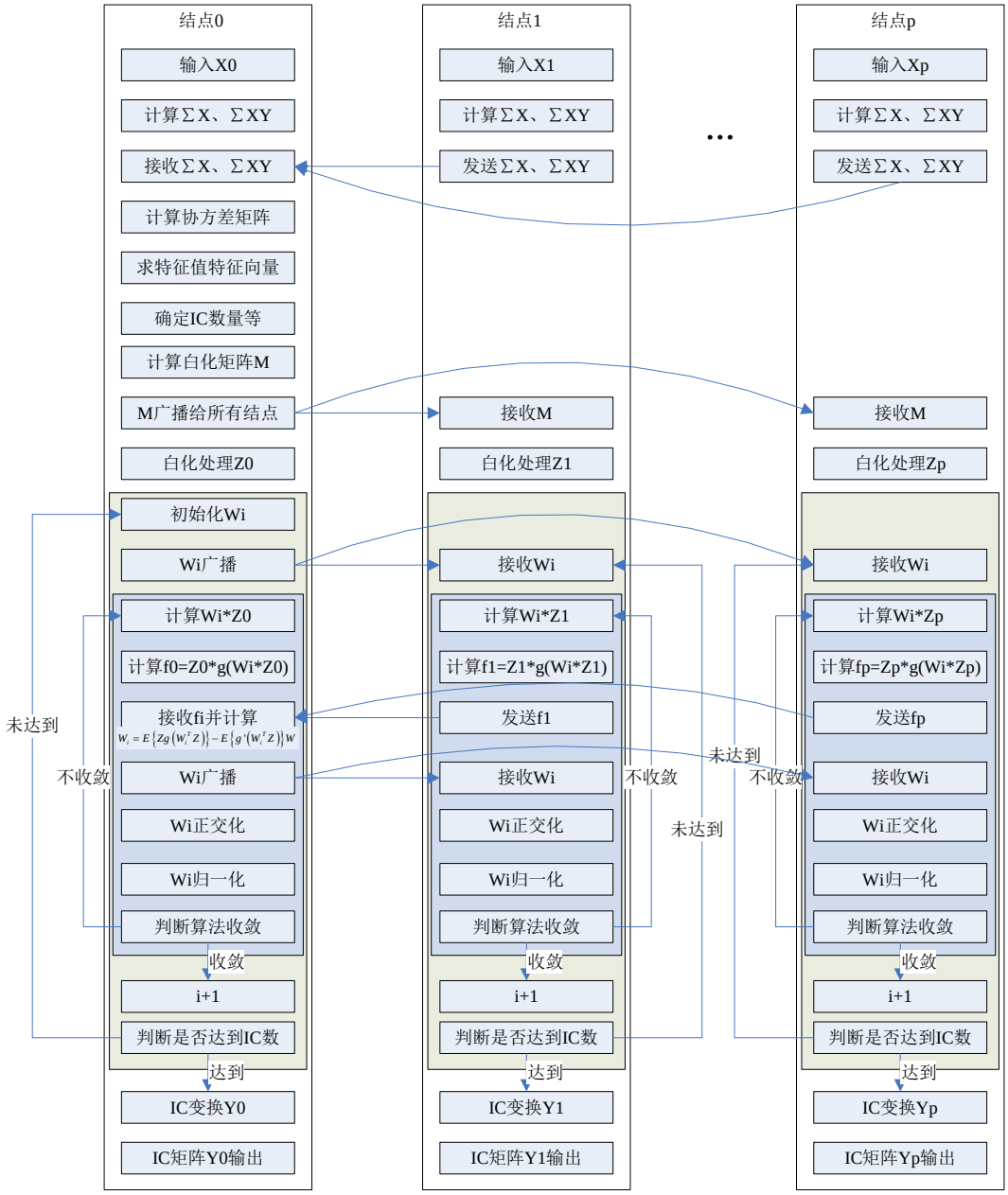

图 4.6 基于 MPI 的 FastICA 降维算法流程图

4.3.2 基于 openMP 的 FastICA 算法

按照 FastICA 算法流程，组织各并行热点的并行策略中的基于 openMP 并行策略，可以得到一种基于 openMP 的 FastICA 并行降维算法，图 4.7 为算法流程图。其中，基于 openMP 的高光谱并行输入见 3.2.3 小节，多线程协方差矩阵计算见 3.2.1 小节，结果矩阵并行输出见 3.2.4 小节，其他并行区域的并行策略见本章 4.2 节。各并行区域隐含同步操作，因此不必考虑其他同步即可保证程序正确执行。

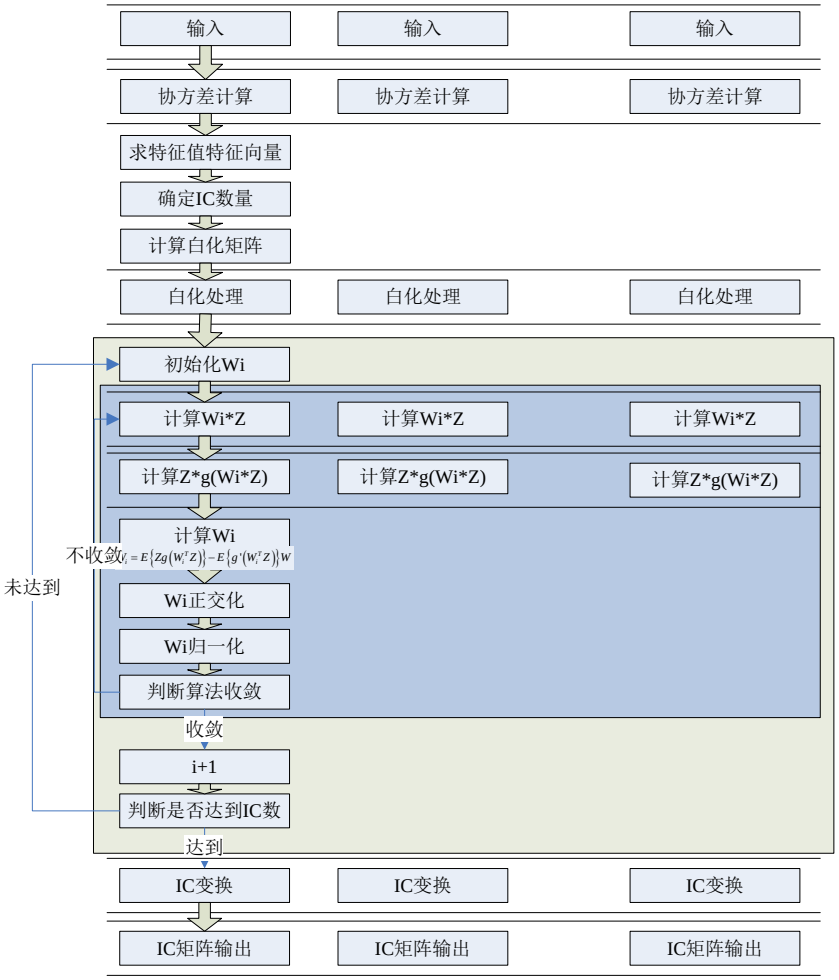

图 4.7 基于 openMP 的 FastICA 降维算法流程图

4.3.3 基于 CUDA 的 FastICA 算法

基于 CPU/GPU 异构系统，CPU 负责控制和少量计算，GPU 负责中大规模并行计算。根据算法流程图（图 4.8），结合本文提出的 CUDA 并行策略，包括协方差矩阵计算（详见 3.2.1）、白化处理、ICA 迭代计算、IC 变换等 CUDA 并行策略，提出了一种基于 CUDA 的 FastICA 算法。

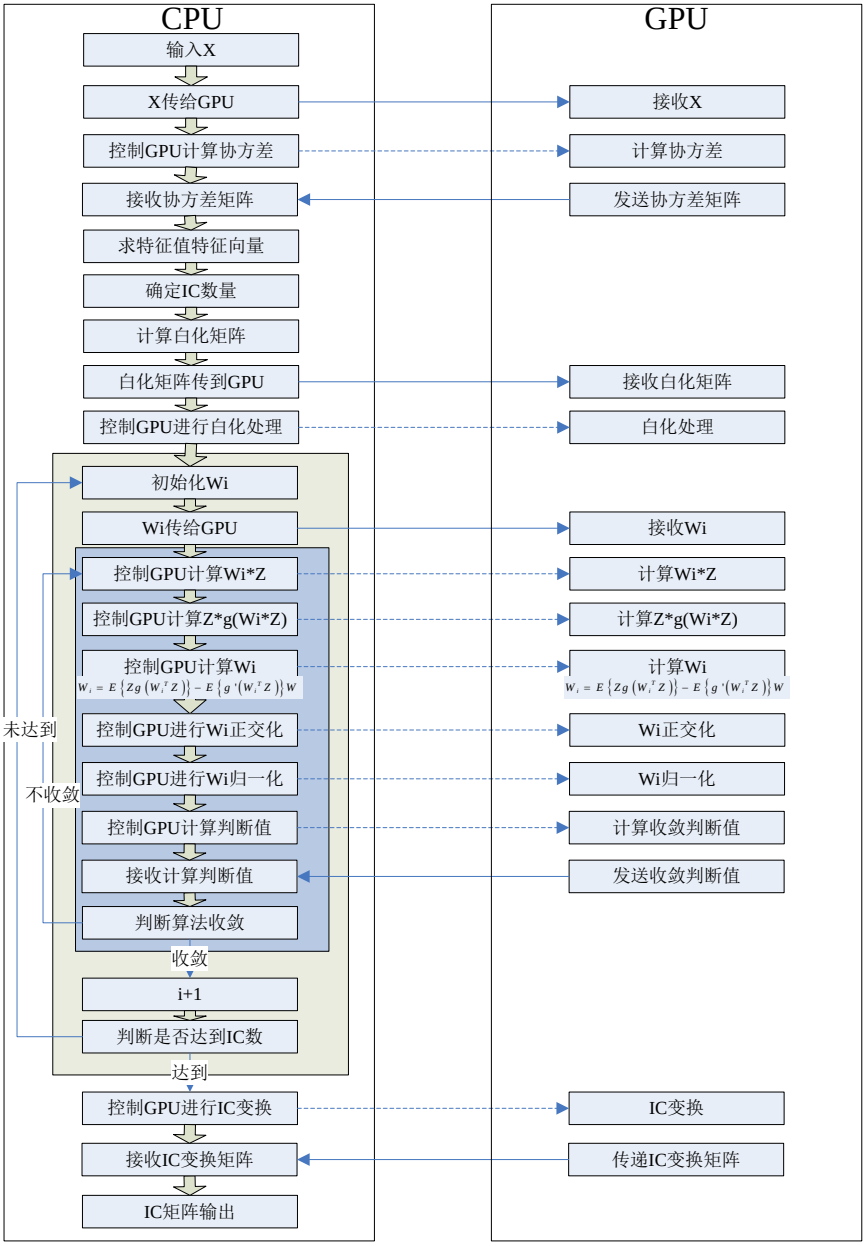

图 4.8 基于 CUDA 的 FastICA 降维算法流程图

4.3.4 基于 MPI+CUDA 的 FastICA 算法

针对 GPU 集群和多 GPU 计算机结点，结合 MPI 和 CUDA 两种并行机制，在基于 MPI 层的数据划分基础上，将各个结点的计算任务分配给 GPU。在各步骤采用 MPI 并行输入（3.2.3）、MPI+CUDA 协方差矩阵并行计算（3.2.1）、MPI+CUDA 并行白化处理、MPI+CUDA 两级并行迭代、MPI+CUDA 并行 IC 变换和 MPI 并行输出（3.2.4），进行合理控制和通信安排，提出了图 4.9 所示的 MPI+CUDA 并行 FastICA 降维算法。

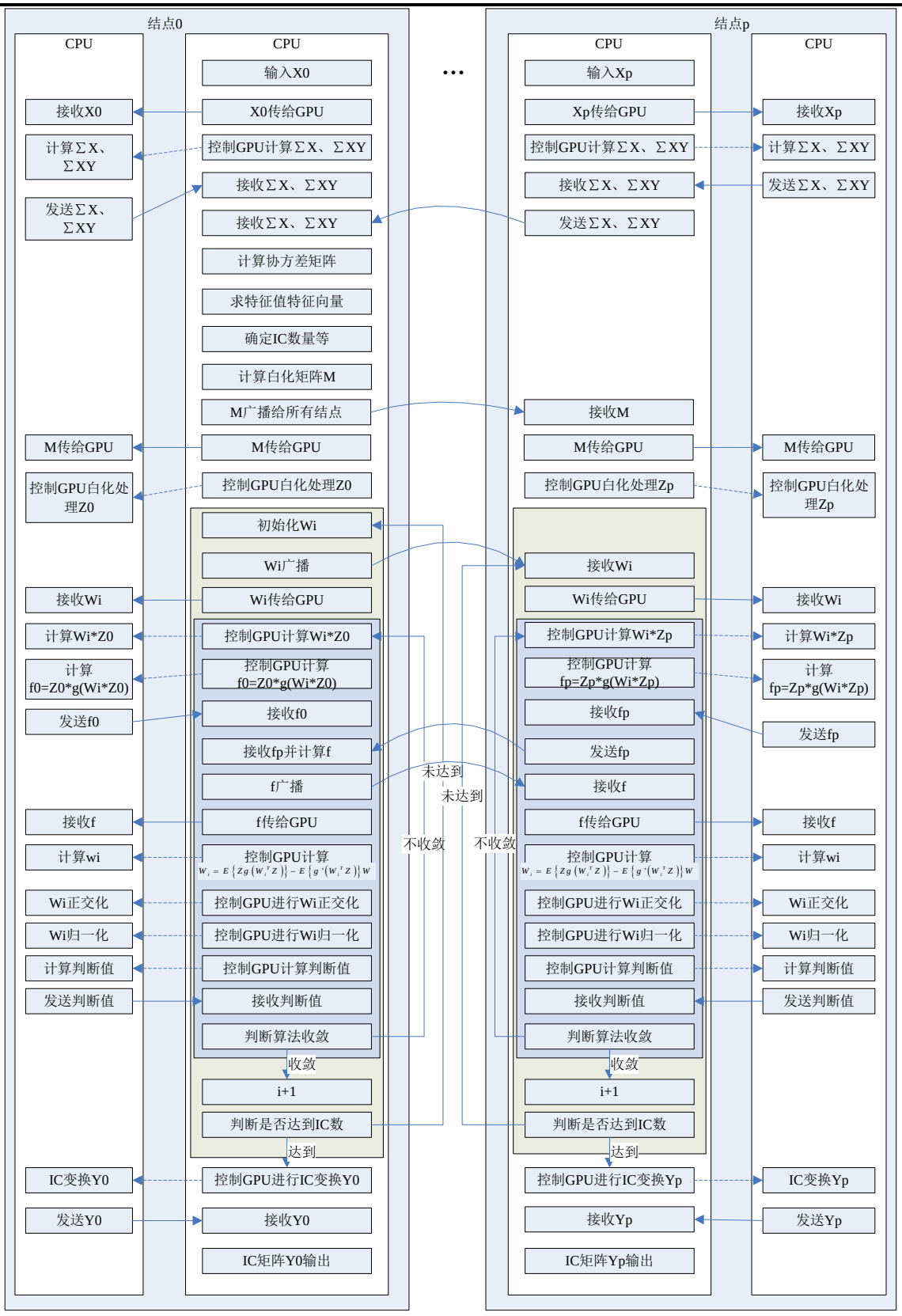

图 4.9 基于 MPI+CUDA 的 FastICA 降维算法流程图

4.3.5 基于 openMP+CUDA 的 FastICA 算法

对于单结点多 GPU 的微型超算，结合 FastICA 算法的特点，根据前文所述的并行策略（如 openMP 并行输入、openMP+CUDA 协方差矩阵、openMP+CUDA 白化处理、openMP+CUDA 并行 ICA 迭代、openMP+CUDA 并行 IC 变换、openMP 并行输出），提出一种基于 openMP+CUDA 的两级并行 FastICA 降维算法，图 4.10 为其流程图。

整个流程共有三个并行区域，openMP 并行输入和并行输出各有一个并行区域，而中间需要 GPU 计算的部分是一个并行区域。这样安排的主要原因是 I/O 部分可利用所有的 CPU 核，而中间的数据处理部分仅需要用到控制 GPU 的线程数；且在迭代的过程中需要同步操作，如果此时启动超过 GPU 数量的线程将导致无法同步而令程序崩溃。GPU 启动和并行输入不能同时进行，这里将并行输入提到 GPU 启动之前完成。

4.3.6 基于 MPI+openMP+CUDA 的 FastICA 算法

在 MPI+CUDA 两级并行 FastICA 降维算法的基础上，开发其中能采用 openMP 并行的部分（比如 MPI+openMP 并行输入和输出等，具体策略见 3.2.3 和 3.2.4 小节），提出基于 MPI+openMP+CUDA 的三级协同并行 FastICA 降维算法，图 4.11 为其流程图。该算法能够充分利用 MPI+CUDA 两级并行 FastICA 算法中为开发的 CPU 多核资源，进一步提高并行性。

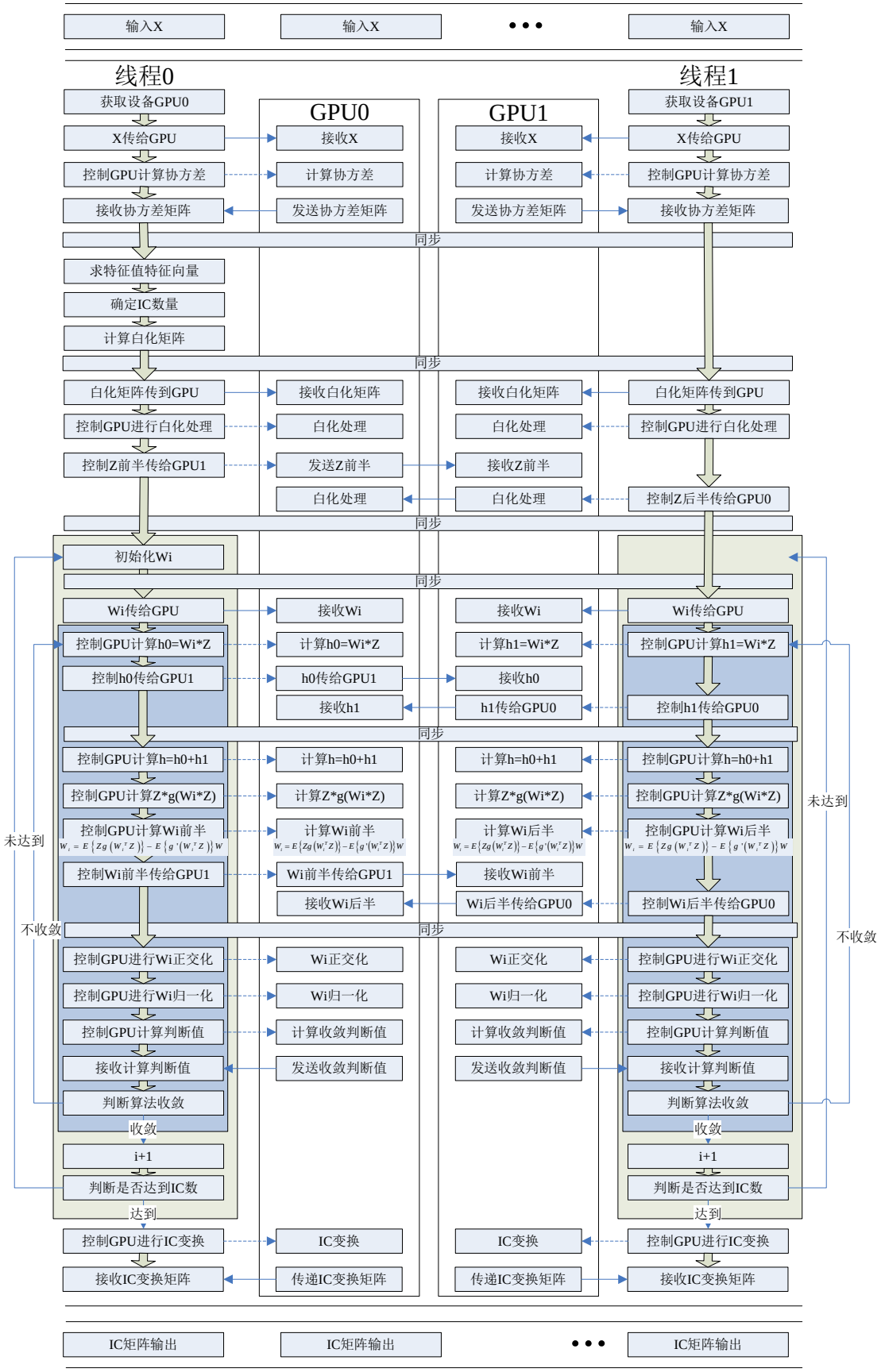

图 4.10 基于 openMP+CUDA 的 FastICA 降维算法流程图

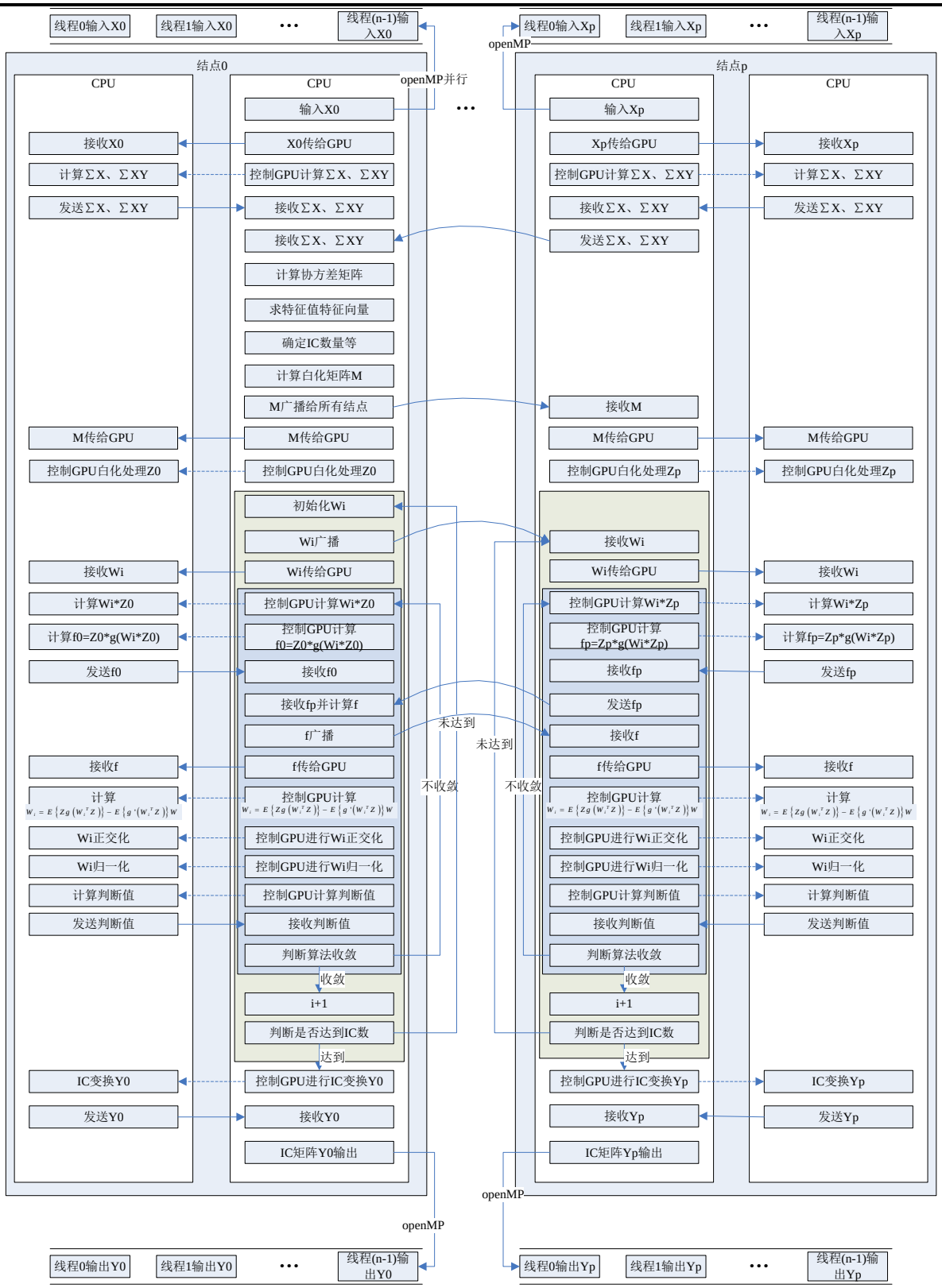

图 4.11 基于 MPI+openMP+CUDA 的 FastICA 降维算法流程图

4.4 实验结果与性能分析

本章所采用的具体实验平台和实验数据详见本文 1.5 和 1.7 小节。

4.4.1 迭代时间算法设定及串行程序最优化

在高光谱遥感图像 FastICA 算法中，一个典型的特点就是需要迭代，由于每次程序执行时所取得的随机数不同，同一算法的实现程序的不同的执行过程中所需要的迭代次数也不相同，而对于不同的高光谱图像数据，其需要的迭代次数又不同，这对于如何衡量程序性能提出了挑战。本文在此设定，针对每个不同的高光谱图像数据，选择一个固定的迭代次数（多次执行获得其平均迭代次数，本文所采用的高光谱数据迭代次数统计见表 4.3）。在测试程序性能时，计算本次执行的单次迭代时间，与设定的迭代次数相乘即为该次执行的总迭代时间。

表 4.3 各数据的迭代次数统计

数据	1	2	3	4	5	6
迭代次数	209	333	859	468	486	177

实现串行程序并通过打开各种优化开关编译，对其中的高光谱遥感数据 1 进行测试，其结果见表 4.4。观察并得到最优化的串行程序为打开优化开关-O2 的串行程序，将该最优化串行程序作为标准衡量本文开发的并行程序性能。

表 4.4 不同优化开关下的串行 ICA 执行时间（s）

	输入	**协方差**	**特征值**	**白化**	**迭代时间**	**IC 变换**	**输出**	**总时间**
O0	0.42	61.49	0.69	13.03	15.89	0.46	0.02	92.00
O1	0.21	16.13	0.28	9.75	10.60	0.18	0.02	37.16
O2	0.22	12.67	0.25	9.91	10.31	0.16	0.02	33.55
O3	0.21	12.73	0.28	10.16	10.28	0.14	0.02	33.82

4.4.2 单级并行结果与性能分析

实现并执行本章提出的三种单级并行算法，表 4.5 统计了高光谱数据 1、2、3 的白化处理、ICA 迭代和 IC 变换 3 个并行热点的执行时间。图 4.12、4.13、4.14 分别计算并统计了 3 个热点的并行加速比。其中白化处理和 IC 变换中，MPI 并行获得了超线性加速比，主要是由于大矩阵切分成小矩阵导致了更好的 Cache 命中率。在 IC 变换中，CUDA 的加速效果较 MPI 差，主要是数据量较小时，IC 变换计算量也相对较小，CUDA 计算需要通信开销，GPU 并行计算带来的收益比通信开销小。从整体看，CUDA 并行的优势是非常明显的。

表 4.5 各并行热点单级并行执行时间统计（ms）

	数据	O2	OMP	MPI	CUDA
白化	1	9909.50	748.97	617.09	32.02
	2	33617.81	3250.72	733.77	121.26
	3	60969.61	6043.69	1661.52	264.18
ICA 迭代	1	10311.71	811.48	1389.38	244.71
	2	49262.40	3868.36	6154.36	1026.53
	3	253543.45	33287.67	30630.89	4542.50
IC 变换	1	162.14	11.54	5.16	8.43
	2	1612.92	118.58	63.72	42.32
	3	3778.39	518.51	176.07	92.26

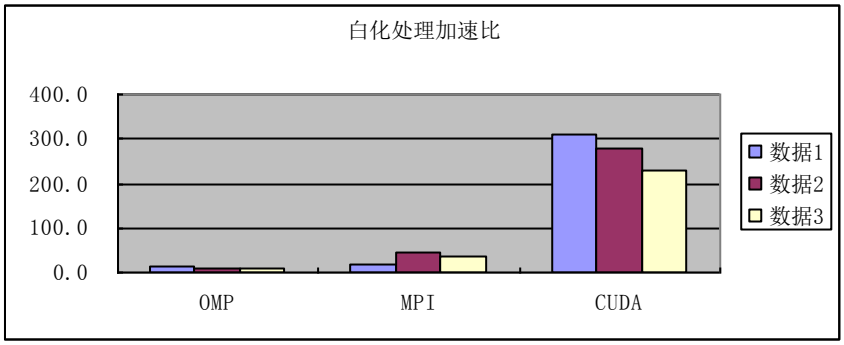

图 4.12 白化处理单级并行加速比

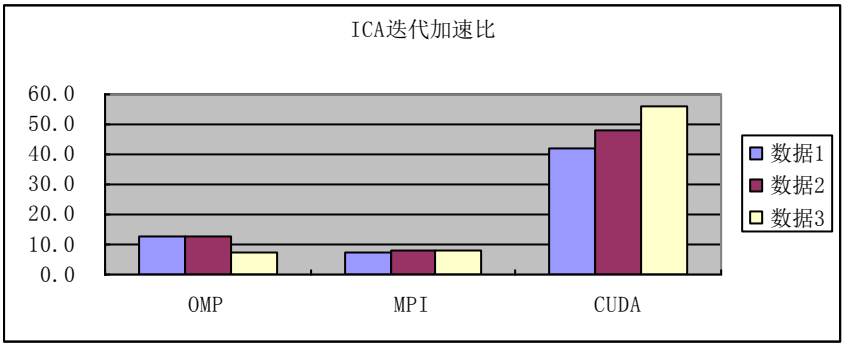

图 4.13 ICA 迭代单级并行加速比

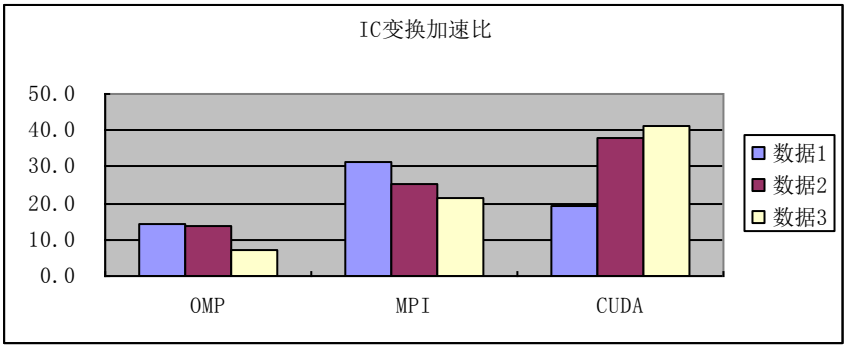

图 4.14 IC 变换单级并行加速比

4.4.3 多级并行的优劣

表 4.6 和 4.7 分别统计了执行高光谱数据 4、5、6 时 CUDA 并行和多级并行的三个主要并行热点的执行时间和加速比。其中 ICA 迭代的 openMP+CUDA 两级并行执行时间较单 CUDA 并行还长了近一倍，验证了 4.2.2 中关于同步与通信开销大于多 GPU 的计算收益的猜测，其主要原因已在前文分析。其余并行热点的多级并行均取得了比单级并行更加显著的性能提升。

表 4.6 各并行热点多级并行执行时间统计（ms）

	数据	**O2**	**CUDA**	**OCU**	**MCU**
白化	4	135831.59	446.72	218.01	227.85
	5	253473.42	839.60	393.82	428.52
	6	449209.22			705.01
ICA 迭代	4	368165.10	3502.38	6182.62	1920.96
	5	741251.24	6786.05	12180.06	3510.10
	6	520272.65			2176.57
IC 变换	4	2562.71	122.58	92.81	61.97
	5	4240.08	215.83	161.51	108.38
	6	5523.40			163.61

表 4.7 各并行热点多级并行加速比

	数据	**CUDA**	**OCU**	**MCU**
白化	4	304.1	623.1	596.1
	5	301.9	643.6	591.5
	6			637.2
ICA 迭代	4	105.1	59.5	191.7
	5	109.2	60.9	211.2
	6			239.0
IC 变换	4	20.9	27.6	41.4
	5	19.6	26.3	39.1
	6			33.8

4.4.4 总执行时间与总加速比

实现并执行 4.3 节设计的 6 种并行 FastICA 降维算法，测试其执行时间（不包含 I/O），记录在表 4.8 中，其中 MCU 表示 MPI+CUDA 程序，OCU 表示 openMP+CUDA 程序，MOC 表示 MPI+openMP+CUDA 程序，由于 MCU 和 MOC 程序的执行部分（不含 I/O）是相同的，因此这里仅列出了 MOC 的时间数据。计算并行程序加速比并绘成表 4.9。由表 4.8 可以看出本章设计的并行 FastICA 算法的加速效果显著，特别是基于 MPI+openMP+CUDA 的三级协同并行 FastICA 算法的计算加速比最高达到了 169 倍。

表 4.8 FastICA 执行时间（不含 I/O）统计（s）

数据	O2	OMP	MPI	CUDA	OCU	MOC
1	33.31	2.82	3.71	0.87	2.52	0.73
2	111.67	10.13	10.15	2.03	3.38	1.57
3	377.99	45.98	37.85	6.25	7.45	4.01
4	724.97	69.83	63.91	8.45	10.52	4.52
5	1430.86	151.19	96.81	17.11	24.55	8.44
6	1841.15	170.85	145.27			12.38

表 4.9 FastICA 总加速比（不含 I/O）

数据	OMP	MPI	CUDA	OCU	MOC
1	11.8	9.0	38.4	13.2	45.4
2	11.0	11.0	54.9	33.1	71.1
3	8.2	10.0	60.4	50.7	94.4
4	10.4	11.3	85.8	68.9	160.2
5	9.5	14.8	83.6	58.3	169.5
6	10.8	12.7			148.7

表 4.10 和表 4.11 分别统计了高光谱降维并行 FastICA 算法的总时间（含 I/O）和总加速比。除了 openMP+CUDA 实现的两级并行 FastICA 算法外，其他算法都获得了较为理想的加速比，其中结合了 MPI、openMP、CUDA 三种并行优势的三级并行 FastICA 降维算法获得了最高 159 倍的性能提升。整体加速比增长趋势与第三章的 PCA 并行算法类似，详见 3.4.5 小节，其中进一步提升算法并行性能的方法在并行 FastICA 降维算法中仍然适用。

分析 openMP+CUDA 未达到比单 CUDA 高的性能提升，其主要有两个原因：首先，单结点内多 GPU 的首个 kernel 函数启动时会产生启动开销，对本文实验平台下产生了 1.5s 左右的时间开销；然后是 ICA 迭代部分的设计中，包含了大量的同步和通信，产生了比多 GPU 并行计算节约时间更多的额外开销。一个比较简单的改进策略是在 ICA 迭代阶段采用单 GPU 并行方案替代 openMP+CUDA 并行方案，或者舍弃 openMP+CUDA 而采用 MPI+CUDA 并行策略。

表 4.10 FastICA 总执行时间统计（不含 I/O）（s）

数据	O2	OMP	MPI	CUDA	OCU	MCU	MOC
1	33.55	2.89	3.78	0.92	2.56	0.76	0.76
2	112.16	10.38	10.34	2.18	3.48	1.69	1.66
3	379.34	46.28	38.46	6.86	7.69	4.46	4.20
4	728.65	70.37	65.47	10.01	10.86	5.72	5.08
5	1438.16	152.20	100.01	20.32	25.26	10.17	9.03
6	1855.51	172.50	150.36			16.69	13.45

表 4.11 FastICA 总加速比（不含 I/O）

数据	OMP	MPI	CUDA	OCU	MCU	MOC
1	11.6	8.9	36.5	13.1	44.3	44.0
2	10.8	10.8	51.5	32.2	66.3	67.5
3	8.2	9.9	55.3	49.3	85.0	90.3
4	10.4	11.1	72.8	67.1	127.3	143.3
5	9.4	14.4	70.8	56.9	141.4	159.2
6	10.8	12.3			111.2	138.0

4.5 本章总结

本章深入研究了高光谱遥感图像 FastICA 降维算法，针对热点白化处理、ICA 迭代和 IC 变换等加速热点提出了并行方案及优化策略。给出了一种针对复杂计算的“具体-抽象-具体”的并行开发模式，并将其应用于 ICA 迭代过程的并行开发。提出并实现了 MPI、openMP、CUDA、MPI+CUDA、openMP+CUDA 和 MPI+openMP+CUDA 等 6 种并行 FastICA 算法，实验结果显示，本章设计的并行算法相对最优化串行获得了优良的性能提升，其中结合三种并行优势的并行 FastICA 降维算法计算加速比（不含 I/O）最高达到了 169 倍，总加速比（包括 I/O）最高达到了 159 倍。

第五章　结束语与工作展望

5.1 工作总结

高光谱遥感图像降维是高光谱图像处理最重要的步骤之一，对其研究具有重大的理论和应用价值。随着成像光谱仪的发展，高光谱图像的分辨率和维度越来越大，使得高光谱图像数据量海量增长，这也导致了高光谱图像降维的必要性和复杂性。于此同时，军事、农林等领域的现实应用精度需求不断提高，导致降维算法朝计算更复杂的方向发展，而这些应用又提出了实时性的需求。传统的串行方法已经无法满足高光谱图像降维的现实需求。

随着近年来 GPU 的快速发展，CPU/GPU 异构系统已经成为了未来高性能计算发展的必然趋势。如何有效地利用 CPU/GPU 异构系统实现高光谱遥感图像降维的并行处理是一个重大挑战。本文主要基于 CPU/GPU 异构系统对高光谱遥感图像线性降维进行并行研究。主要贡献如下：

（1）深入研究了 CPU/GPU 异构系统下 3 种主流并行编程模式。研究了 CPU/GPU 异构系统的工作原理，并对适用于异构系统的 MPI、openMP 和 CUDA 等 3 种并行模式进行了深入学习，研究了其编程模型与优化策略，为后续的高光谱遥感图像线性降维并行开发奠定了基础。

（2）研究并实现了 6 种并行 PCA 线性降维算法。针对高光谱遥感图像主成分分析降维方法，通过分析 PCA 串行算法的并行热点，对协方差矩阵计算、PCA 变换、高光谱图像数据输入和结果输出等 4 个加速热点进行了深入研究，提出了基于 MPI、openMP、CUDA 及混合并行的并行解决方案及优化策略；提出并实现了 6 种不同的并行 PCA 降维算法，实验结果表明本文提出的并行算法均获得了良好的性能提升，特别是混合并行能发挥出较单级并行更好的性能优势，其中 MPI+openMP+CUDA 三级协同并行 PCA 降维算法获得了最高 145 倍的计算加速比（不含 I/O）和 128 倍的总加速比（含 I/O）。

（3）针对 FastICA 降维算法进行并行研究。通过分析 ICA 原理、应用于高光谱图像降维的 FastICA 算法，和算法中的并行热点，提出了一系列并行及优化策略。提出一种用于开发复杂计算的并行设计模式：“具体-抽象-具体”，并将该模式应用到 ICA 迭代的并行开发过程中。提出并实现了 MPI、openMP、CUDA、MPI+CUDA、openMP+CUDA 和 MPI+openMP+CUDA 等 6 种并行 FastICA 降维算法，实验结果表明并行算法均获得了良好的加速效果，其中三级并行 FastICA 算法获得了最高 169 倍的计算加速比（不含 I/O）和 159 倍的总加速比（含 I/O）。

5.2 工作展望

由于时间限制，本课题的研究还不够完善，还有许多研究工作需要进一步的开展与完善，主要包括以下 3 个方面：

（1）除了本文研究的 PCA 和 FastICA 降维算法，还有很多常用的高光谱图像线性降维算法，比如线性奇异分析 LDA、最大噪声分数变换 MNF 等，这些线性降维算法都面临着计算量大、复杂度高等问题，需要进行并行研究。

（2）线性降维仅是高光谱降维方法中的一类，还有非线性降维方法，比如等距映射法 Isomap、局部线性嵌入 LLE 等。相对而言，非线性降维具有更好的降维效果，但其计算量更大、计算复杂度更高、时间消耗更长。非线性降维方法对高性能并行计算提出了更大的挑战。

（3）随着高性能计算体系结构的发展，特别是去年 Intel 公司推出了 MIC 协处理器，搭载 MIC 的天河 2 号成功夺得 TOP500 榜首。未来的高性能计算道路将越来越广，因此，除了 CPU+GPU 异构系统的并行研究外，还要展开基于其他高性能体系结构（比如 MIC）的并行研究工作。

致　谢

在毕业论文完成之际，我怀着一颗感恩的心，在此向所有关心、支持和给予我帮助的人表示诚挚的谢意！

首先，我要真诚地感谢我的导师周海芳老师。从读研开始，周老师就给予我无微不至的帮助与指导。周老师作为课题负责人，指导我进入 CPU/GPU 异构系统下多级并行加速处理高光谱遥感图像降维这一领域，使我整个硕士期间学习工作充实收获巨大。在我刚接触课题时，我没有找到正确的研究方法，周老师耐心地指导我，不仅在专业领域教我知识，更在学习方法上引导我，使我获益匪浅。周老师治学严谨认真，对学生要求严格，以使我们能够学到真知识，学到真本领。周老师在我硕士期间还为我们提供很多与相关领域专业人员交流的机会，支持和鼓励我们多写论文和参加学术会议，全面培养我们各方面的能力和素质。周老师作为教研室副主任，辛苦工作的同时尽心尽力的给予我指导，我想发自内心的对周老师说：老师，您辛苦了，谢谢您！

感谢杜云飞老师、朱小谦老师、张理论老师以及 Nvidia 的罗华平总工程师在我研究上遇到困难时帮我解答疑问；感谢天津超算和长沙超算为我提供高性能计算设备上的支持。如果没有你们，我的学习研究将困难重重，再次感谢，谢谢！

真诚地感谢师门的师兄弟：姚天问、申小龙等，他们在我研究生的学习和生活上，给予了最大的帮助和鼓励。感谢唐波、刘丹、程至圣等同学在我研究刚入门时提供了技术、设备上的帮助。感谢同实验室的钟珀辰、明月伟、项俊、邵则铭、李建立等陪伴我度过一年多的实验室生涯，谢谢各位师兄弟们！

真诚地感谢学员队的领导们，他们是：贺毅政委、唐又旺政委、朱涛队长。他们在思想上教育我、在生活上关怀我、在学业上鼓励我，为全体学生营造了良好的学习环境和生活环境。真诚地感谢来到科大认识并一直陪伴在身边的朋友们，特别是阮泓科、余松平等等。我的生活因为有你们而更加快乐和精彩，我非常珍惜我们一起度过的这段时光，这份深厚的友谊将会一直留在我心中。

真诚地感谢我的父母。父母含辛茹苦无怨无悔地供我读书，二十年如一日，我深受感动。我感恩有这样的好父母，谢谢爸爸妈妈！

最后，再一次真诚地感谢所有帮助我、支持我和理解我的人！

参考文献

[1] 于涛，李希灿等. 高光谱技术应用进展与展望[J]. 测绘科学. 2012.

[2] 王为. 高光谱遥感技术的发展及其在农业上的应用[J]. 江西农业学报. 2009.

[3] 房华乐，任润东等. 高光谱遥感在农业中的应用[J]. 测绘通报增刊. 2012.

[4] 杨国鹏, 余旭初等. 高光谱遥感技术的发展与应用现状[J]. 测绘通报. 2008.

[5] 张宗贵,王润生等. 成像光谱岩矿识别方法技术研究和影响因素分析[M]. 北京: 地质出版社,2006. 108~136.

[6] 甘甫平,王润生. 高光谱遥感技术在地质领域中的应用[J]. 国土资源遥感. 2007.

[7] Sergio Sa ́nchez, Rui Ramalho. Real-time implementation of remotely sensed hyperspectral image unmixing on GPUs[J]. Real-Time Image Proc. 2012.

[8] Green, R.O., Eastwood, M.L.. Imaging spectroscopy and the Airborne Visible/Infrared Imaging Spectrometer(AVIRIS)[J]. Remote Sens Environ,1998(65):227~248.

[9] Goetz, A.F.H., Vane, G., Solomon, J.E., Rock, B.N.: Imaging spectrometry for Earth remote sensing[J]. Science,1985(228): 1147~1153.

[10] Antonio Plaza ,David Valencia. Commodity cluster-based parallel processing of hyperspectral imagery[J]. Parallel Distrib Comput, 2006 (66) 345~358.

[11] J. Le Moigne, J.C. Tilton. Refining image segmentation by integration of edge and region data[J]. IEEE Trans Geosci Remote Sensing,1995,(33): 605~615.

[12] Green A, Berman M, Switzer P, Craig M D. A transformation for ordering multispectral data in terms of image quality with implications for noise removal[J]. IEEE Transaction on Geoscience and Remote Sensing, 2000, 26(1):65~74.

[13] Kaarna A, Zemcik P, and iainen H, et al.. Compression of multispectral remote sensing images using clustering and spectral reduction[J]. IEEE Trans. on Sci. Remote Sensing, 2000, 38(2): 1588~1592.

[14] Bellman R. Adaptive Control Processes: A Guided Tour. 1st ed[M]. Princeton, New Jersey: Princeton University Press, 1961.

[15] Scott D, Thompson J. Probability density estimation in higher dimensions[C]. In Proceedings of the 15th Symposium on the Interface Computer Science and Statistics. 1983: 173~179.

[16] 张舒, 褚艳利. GPU 高性能运算之 CUDA[M], 北京:中国水利水电出版社, 2009.

[17] TOP500 http://www.top500.org.

[18] T. Achalakul, S. Taylor. A distributed spectral-screening PCT algorithm[J]. Parallel Distrib Comput, 2003 (63): 373~384.

[19] M.K. Dhodhi, J.A. Saghri, I. Ahmad, R. Ul-Mustafa, D-ISODATA: a distributed

algorithm for unsupervised classification of remotely sensed data on network of workstations[J]. Parallel Distrib. Comput,1999 (59) :280~301.

[20] M. Gil, C. Gil, I. Garcia. The load unbalancing problem for region growing image segmentation algorithms[J]. Parallel Distrib. Comput,2003 (63):387~395.

[21] K. Sano, Y. Kobayashi, T. Nakamura. Differential coding scheme for efficient parallel image composition on a PC cluster system[J]. Parallel Comput,2004 (30): 285~299.

[22] K.A. Hawick, P.D. Coddington, H.A. James, Distributed frameworks and parallel algorithms for processing large-scale geographic data, Parallel Comput. 29 (2003) 1297~1333.

[23] Shawe-Taylor J, Cristianini N. Kernel methods for pattern analysis[M]. CambridgeUK: Cambridge Univ Press, 2004.

[24] John A. Lee, Michel Verleysen, Nonlinear Dimensionality Reduction[M], Springer, 2007.

[25] Yangchi Chen, Crawford, M.M., Ghosh, J., Improved Nonlinear Manifold Learning for Land Cover Classification via Intelligent Landmark Selection[J]. IEEE International Conference on Geoscience and Remote Sensing Symposium, IGARSS 2006:545~548.

[26] Levina E, Bickel P. Maximum likelihood estimation of intrinsic dimension[J]. Advances in Neural Information Processing Systems, 2005 (15) : 777~784.

[27] Costa J, Hero AO. Geodesic entropic graphs for dimensiona and entropy estimation in manifold learning[J]. IEEE Transactions on Signal Processing, 2004,52 (8) : 2210~2221.

[28] Yang L. Alignment of Overlapping Locally Scaled Patches for Multidimensional Scaling and Dimensionality Reduction[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2008, 30 (3): 438~450.

[29] Tenenbaum J, Silva V, Langford J. A global geometric framework for nonlinear dimensionality reduction [J]. Science, 2000, 290 (5500): 2319~2323.

[30] Roweis S, Saul L. Nonlinear dimensionality reduction by locally linear embedding[J]. Science, 2000, 290 (5500): 2323~2326.

[31] Rosman G., Bronstein M. M., Bronstein A. M. and Kimmel R., Nonlinear Dimensionality Reduction by Topologically Constrained Isometric Embedding[J]. International Journal of Computer Vision, 2010, 89(1): 56~68.

[32] 赵连伟, 罗四维, 赵艳敞等. 高维数据流形的低维嵌入及嵌入维数的研究[J], 软件学报,2005, 16(8):1423~1430.

[33] Lin T, Zha H. Riemannian Manifold Learning[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence,2008, 30 (5): 796~809.

[34] Charles M., Thomas L. and Robert A. Exploiting Manifold Geometry in Hyperspectral Imagery[J]. IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING, VOL. 43, NO. 3, MARCH 2005.

[35] Gashler, M. and Ventura, D. and Martinez, T.. Iterative Non-linear Dimensionality

Reduction with Manifold Sculpting[C]. Advances in Neural Information Processing Systems 20, MIT Press, Cambridge, MA, 2008: 513~520.

[36] 董广军. 基于流形学习的高光谱遥感影像分类技术[J]. 仪器仪表学报, 2008, 29(6):1990~1992.

[37] Melba M. Crawford, Li Ma and Wonkook Kim. Exploring Nonlinear Manifold Learning for Classification of Hyperspectral Data[J]. Optical Remote Sensing Advances in Signal Processing and Exploitation Techniques. LNCS: Augmented Vision and Reality, 2011, Volume 3, 207~234, DOI: 10.1007/978-3-642-14212-3_11.

[38] 杜培军, 王小美, 谭琨, 夏俊士. 利用流形学习进行高光谱遥感影像的降维与特征提取[J].武汉大学学报：信息科学版,2011.36(2): 148~152.

[39] Xin Luo; Ming-Fei Jiang.A new dimensionality analysis algorithm for hyperspectral imagery[J]. Computer Science and Service System (CSSS), 2011 International Conference on 1952~1956.

[40] Plaza, A.J. Recent developments and future directions in parallel processing of remotely sensed hyperspectral images[J]. Proceedings of 6th International Symposium on Image and Signal Processing and Analysis, ISPA 2009 : 626~631.

[41] David Valencia, Alexey Lastovetsky, et al. Parallel Processing of Remotely Sensed Hyperspectral Images On Heterogeneous Networks of Workstations Using HeteroMPI[J]. International Journal of High Performance Computing Applications. 2008, 22(4): 386~407.

[42] Javier Plaza, Antonio Plaza, Rosa Pérez and Pablo Martínez. Parallel Classification of Hyperspectral Images Using Neural Networks[J]. Computational Intelligence for Remote Sensing Studies in Computational Intelligence, 2008, 133: 193~216.

[43] 刘春, 陈燕, 辛亮. 基于 Matlab 的高光谱遥感数据降维并行计算分析[J]. 遥感技术, 2010.3 : 13~17.

[44] Sergio Sánchez,Rui Ramalho. Real-time implementation of remotely sensed hyperspectral image unmixing on GPUs[J]. Real-Time Image Proc,2012.

[45] Sergio Sáncheza, Antonio Plazaa. Real-Time Implementation of a Full Hyperspectral Unmixing Chain on Graphics Processing Units[J]. Satellite Data Compression, Communications, and Processing VII, edited by Bormin Huang, Antonio J. Plaza, Carole Thiebaut, Proc. 2011.

[46] Mahmoud ElMaghrbay, Reda Ammar. Fast GPU Algorithms for Endmember Extraction from Hyperspectral Images[J]. IEEE,2012.

[47] He Yang, Qian Du. FAST BAND SELECTION FOR HYPERSPECTRAL IMAGERY[J]. IEEE, 2011.

[48] He Yang, Qian Du. Unsupervised Hyperspectral Band Selection Using Graphics Processing Units[J]. IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING, 2011.

[49] Qian Du, James E. Fowler. RANDOM-PROJECTION-BASED

DIMENSIONALITY REDUCTION AND DECISION FUSION FOR HYPERSPECTRAL TARGET DETECTION[J]. IEEE, 2011.

[50] Haicheng Qu. High-Performance Computing in Remote Sensing II, edited by Bormin Huang, Antonio J. Plaza,Proc. 2012.

[51] 罗秋明, 明仲. OpenMP 编译原理及实现技术[M]. 北京:清华大学出版社, 2012.

[52] nVIDIA. CUDA C PROGRAMMING GUIDE. v5.0[EB/OL]. 2012.

[53] David, B. K., W. H. Wen-wei. 大规模并行处理器编程实战[M]. 北京:清华大学出版社, 2010.

[54] 余旭初, 冯伍法, 林丽霞. 高光谱-遥感测绘的新机遇[J]. 测绘科学技术学报, 2006, 23(2): 101~105.

[55] KRAM ER. Earth Observation H istory on T echno logy Introduction[EB/OL]. http://www.eoporta.org/documents/kramer/history.pdf 2008-03-12.

[56] P.Wang, K.Y. Liu, T. Cwik, R.O. Green, MODTRAN on supercomputers and parallel computers, Parallel Comput. 28 (2002) 53~64.

[57] Plaza, A. Preface to the special issue on architectures and techniques for real-time processing of remotely sensed images[J]. Real Time Image Process. 2009,4(3), 191~193 .

[58] Plaza, A., Du, Q., Chang, Y.-L., King, R.L.. Foreword to the special issue on high performance computing in Earth observation and remote sensing[J]. IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens, 2011,4(3), 503~507.

[59] Lee, C.A., Gasster, S.D., Plaza, A., Chang, C.-I., Huang, B.. Recent developments in high performance computing for remote sensing: a review. IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens, 2011,4(3), 508~527.

[60] 王恩东，张清等. MIC 高性能计算编程指南[M]. 北京: 中国水利水电出版社. 2012.

[61] 曾泳泓, 成礼智, 周敏. 数字信号处理的并行算法[M]. 长沙：国防科技大学出版社. 1999.

[62] 李晓梅, 莫则尧, 胡庆丰等. 可扩展并行算法的设计与分析[M]. 北京：国防工业出版社.2000.

[63] Dietz H. G, Cohen W. E., Grant B. K. Would You Run it Here or There? AHS: Automatic Heterogeneous Supercomputing[C]. 1993 International Conference on Parallel Processing, ICPP 1993, Vol2: 217~221.

[64] Braun T., Siegel H., Maciejewski A. Heterogeneous computing: Goals, methods, and open problems[J]. High Performance Computing HiPC, 2001: 307~318.

[65] Kistler M., Gunnels J., Brokenshire D., Benton B. Petascale computing with accelerators[C]. Proceedings of the 14th ACM SIGPLAN symposium on Principlesand practice of parallel programming, 2009, 44(4):241~250.

[66] http://www.coema.org.cn/study/optics/20080717/144631.html. 2008.7.

[67] 都至辉. 高性能计算之并行编程技术——MPI 并行程序设计[M]. 2001.

[68] Sanders, J., E. Kandrot. GPU 高性能编程 CUDA 实战[M]. 北京:机械工业出版社, 2010.

[69] Wilkinson, B., M. Allen. 并行程序设计[M]. 北京:机械工业出版社, 2002.

[70] 张武生, 薛魏. MPI 并行程序设计实例教程[M]. 北京:清华大学出版社, 2009.

[71] 朱述龙, 张占睦. 遥感图像获取与分析（第一版）[M]. 科学出版社. 2000.8.

[72] Robila S A and Varshney P K. A fast source separation algorithm for hyperspectral image processing[J]. IEEE International Conference on Geoscience and Remote Sensing, Canada, 2002,6 : 3516~3518.

[73] Jan Kybic, Michael Unser. Fast Parametric Elastic Image Registration[J]，IEEE Trans. image processing, 2003,12(11): 1427~1442.

[74] Jing Zhou, Haifang Zhou, Yuhua Tang. Study and Implementation of Parallel Registration Algorithm for Multimodality Remote Sensing Images Based on Mutual Information[J]. SPIE Multispectral Image Processing and Pattern Recognition (MIPPR2009). 2009.10.30~11.1: 7498 3D-1~12.

[75] Steele, J. and Cochran, R. Introduction to GPGPU programming. Proceeding of the 45th Annual Southeast Regional Conference[M]. North Carolina, 2007: 508~508.

[76] Hopcroft J. Future Directions in Computer Science[C]. In The 2nd International Frontiers of Algorithmics Workshop. Changsha, China, June 2008.

[77]詹宇斌. 流形学习理论与方法及其应用研究[D]. 国防科学技术大学,2011.

[78]许桢. 关于 CPU+GPU 异构计算的研究与分析[J]. 科技信息.2010.

[79]邱桑敏. 嵌入式智能监视系统的研究[D]. 上海交通大学,2003.

[80]胡冰. 遥感图像融合并行算法的研究及实现[D]. 国防科学技术大学,2006.

[81]陈艳. 基于独立分量分析的盲信号分离研究[J]. 中国西部科技,2009.

[82]胡楠. 独立分量分析在滑坡预报中的应用[D]. 成都理工大学,2008.

[83]徐秋平. 独立分量分析在 PET 图像去噪处理中的应用[D]. 哈尔滨理工大学,2009.

[84]顾军. 基于 FastICA 算法的敌我识别信号分选方法研究[J]. 舰船电子对抗,2009.

[85]赵进. 基于 GPU 的遥感图像并行处理算法及其优化技术研究[D]. 国防科大,2011.

作者在学期间取得的学术成果

[1] 方民权, 周海芳, 姚天问, 申小龙. 基于 GPU 异构系统的 PCA 图像融合算法实现[C]. 2013 全国计算机网络与通信学术会议, 2013:52~56.

[2] 方民权, 周海芳, 姚天问, 申小龙. 基于 GPU 的高光谱遥感图像并行降维算法研究及其存储优化[C]. 第 19 届全国信息存储技术学术会议, 2013. [已录用]

[3] 姚天问, 周海芳, 方民权, 申小龙. 基于 C6748 DSP 的小波变换算法的低功耗优化方法[C]. 第六届全国通信新理论与新技术学术大会,2013,89~94.

[4] 姚天问, 周海芳, 方民权, 申小龙. 基于 DSP 软加固下的功耗优化方法[C]. 第四届全国智能信息处理学术会议，2013. [已录用]

[5] Tianwen Yao, Haifang Zhou, Minquan Fang, Haibin Hu. Low Power Consumption Scheduling Based on Software Fault-tolerance[C]. The 9th International Conference on Natural Computation The 10th International Conference on Fuzzy Systems and Knowledge Discovery.[已录用]