基于 GPU 的高光谱 PCA 降维多级并行算法

方民权，周海芳，申小龙

（国防科学技术大学 计算机学院，湖南 长沙 410073）

摘　要：高光谱影像降维是一个典型的计算密集型和存储密集型应用。面向目前主流的 CPU/GPU 异构并行系统，研究高光谱降维算法中经典的主成分分析（PCA）方法，提出一种基于单 GPU 的 G-PCA 并行算法和一种基于多 GPU 的 MPI+openMP+CUDA 多级并行 Gs-PCA 算法，并在 GPU 微型超算平台上实现了这两种算法。实验结果显示，G-PCA 算法获得了显著的加速效果，但会面临存储受限问题；Gs-PCA 算法能突破存储限制，最高可获得 128 倍的加速比。

关键词：图像处理单元；多级并行；高光谱降维；主成分分析；统一计算设备架构

中图分类号：TP391　　**文献标志码：**A

Multilevel Parallel Algorithm of PCA Dimensionality Reduction for Hyperspectral Image on GPU

FANG Min-quan, ZHOU Hai-fang, SHEN Xiao-long

(School of Computer Science, National University of Defence Technology, Changsha, 410073, China. Corresponding author: FANG Min-quan, E-mail: 877086820@qq.com)

Abstract： Dimension reduction for hyperspectral image is computationally expensive and data intensive. Based on the CPU/GPU heterogeneous system, the classical Principal Component Analysis (PCA) method is researched. A G-PCA algorithm for single GPU and a Gs-PCA algorithm for multiple GPUs are implemented and validated on a mini-supercomputer system. Experiments show that the G-PCA algorithm can remarkably enhance the performance but the simulation is constricted by the limited memory; in comparison, the Gs-PCA algorithm overcomes the problem, and can reach a maximum speed-up of 128X in test.

Key words： GPU; Multilevel Parallel; Hyperspectral Image Dimensionality Reduction; PCA; CUDA

高光谱影像是利用成像光谱仪以连续很窄的波段从感兴趣物体上获得的一组图像，一般由数百个波段图像组成[1]。高光谱遥感技术在军事、农业、环境科学、地质、海洋学等领域有广泛应用，这些应用大都需要及时处理[2]。

高光谱数据波段多、数据量大、相关性强、冗余多等特点，直接处理非常复杂，会导致样本类别训练困难、多元密度估计方法不精确[3]、高相关性和高冗余度[4]、维数灾难、空空间现象[5]等严重的计算问题。因此，降维是高光谱处理过程中必不可少的步骤。降维是通过一定的映射，将高维数据转换成低维数据，同时保持信息量基本不变。由于高光谱数据的特点，降维过程复杂且耗时。

自 2007 年 nVidia 正式推出统一计算设备架构（Compute Unified Device Architecture, CUDA），图像处理单元（Graphic Processing Unit， GPU）在通用计算领域发展迅猛。CPU+GPU 的异构系统可编程性好、性价比高、功耗比低，已成为高性能计算领域的黑马。搭载这种异构系统的天河 1A 和泰坦分别在 2010 年和 2012 年夺得 TOP500 榜首[6]。CPU/GPU 异构系统的快速发展为高光谱图像降维的实时处理提供了可能。

高光谱影像并行处理在传统并行系统已有成熟的应用，David[7]基于异构 MPI 在网络集群系统研究了高光谱影像处理的技术。Javier 等[8]提出基于神经网络的高光谱影像并行分类算法。在 CPU/GPU 异构系统上，Sergio Sanchez [9] 和 Mahmoud ElMaghrbay[10] 对高光谱解混进行了 GPU 移植，He Yang 等[11]在多 GPU 实现高光谱影像快速波段选择。这些研究较少涉及基于 GPU 研究高光谱影像降维，本文研究正是时机。

主成分分析（Principal Component Analysis，

收稿日期：2014-03-20

基金项目：国家自然科学基金项目(61272146)。

作者简介：方民权（1989 -），男，浙江东阳人，国防科技大学博士研究生，CCF 会员；周海芳（1975-），女，上海普陀人，国防科技大学教授。

PCA）是经典的高光谱遥感影像线性降维方法。本文以 PCA 算法为对象，基于 CPU/GPU 异构系统展开研究，提出一种单 GPU 的 G-PCA 并行算法和一种多 GPU 的 Gs-PCA 多级并行算法，并在本文实验平台上实现这两种算法。

本文结构如下，第 1 部分介绍 PCA 串行算法并分析加速热点；第 2 部分详细阐述各加速热点的并行策略，提出 G-PCA 和 Gs-PCA 两种并行算法；第 3 节描写本文算法采用的优化策略；第 4 节分析实验结果与算法性能；最后是总结和展望。

1 PCA 降维算法与加速热点分析

1.1 高光谱影像 PCA 降维算法

主成分分析又称为 K-L 变换，是在统计特征基础上进行的一种多维正交线性变换，其几何意义是将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去，得到新的特征轴。在高光谱遥感影像降维中，通过 PCA 变换，把高维图像变换到低维空间，原始数据的信息被压缩到较少的波段中，实现了高光谱影像降维。

假设 $\boldsymbol{X}=(\boldsymbol{X}_1,\boldsymbol{X}_2,\cdots,\boldsymbol{X}_S)=(\boldsymbol{X}_1,\boldsymbol{X}_2,\cdots,\boldsymbol{X}_B)^T$ 表示高光谱遥感图像，其中 S 表示高光谱遥感图像的象元数目，B 表示波段数目。下面简单对 PCA 降维算法进行描述：1）首先计算 $\boldsymbol{X}$ 中各波段间的相互协方差，构建协方差矩阵；2）求协方差矩阵的特征值和特征向量。并根据特征值计算贡献率 v，并根据给定的阈值 R（本文取 *99%*），按特征值从大到小选取主成份个数 m，使得累计贡献率 $\sum v \geq R$；3）对选中的主成分进行 PCA 变换。令 $\boldsymbol{A}=\boldsymbol{T}^T$，PCA 变换结果矩阵 $\boldsymbol{Y}=(\boldsymbol{Y}_1,\boldsymbol{Y}_2,\ldots,\boldsymbol{Y}_m)^T=\boldsymbol{AX}$。通过 PCA 降维，高光谱遥感影像数据从 B 个波段降为 m 个波段，大大减少了后续高光谱影像处理的计算量。

1.2 加速热点分析

实现高光谱影像 PCA 降维算法，用高光谱图像数据（波段数为 *224*，宽为 *614*，高为 *1087*）测试串行 PCA 降维各步骤的时间消耗，计算其占总时间的百分比（表 1）。观察该表，PCA 降维的时间主要消耗在协方差矩阵计算和 PCA 变换，因此这两个步骤是本文最重要的加速热点。其余 I/O 为次要加速热点，由于本文数据波段数恒为 224，协方差矩阵特征值求解时间消耗恒定且较小，可不考虑并行。

2 PCA 算法并行化研究

2.1 协方差矩阵计算并行化

2.1.1 协方差矩阵计算的 3 种 GPU 移植方案

协方差矩阵是对称阵，只需计算其下三角元素，本文设计了 3 种 GPU 移植方案，5 种实现策略：

表 1 各步骤耗时百分比

Tab.1 Time percents of different parts

Step	Time/ms	Percentage
input	774.82	0.45%
cov	129518.63	75.09%
eigenvalue	768.23	0.45%
PCA trans	41313.88	23.95%
output	116.14	0.07%

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。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，可以采用补偿划分策略（图 1，将①补偿到②），启动 *B/2* 个 block 即可完成协方差矩阵的计算任务，映射方案参考图 2 左。

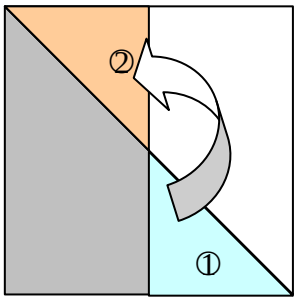

图 1 协方差矩阵的补偿划分

Fig.1 Compensation division of COV matrix

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 2 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 3 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（*(1+B)*B/2* 个）。

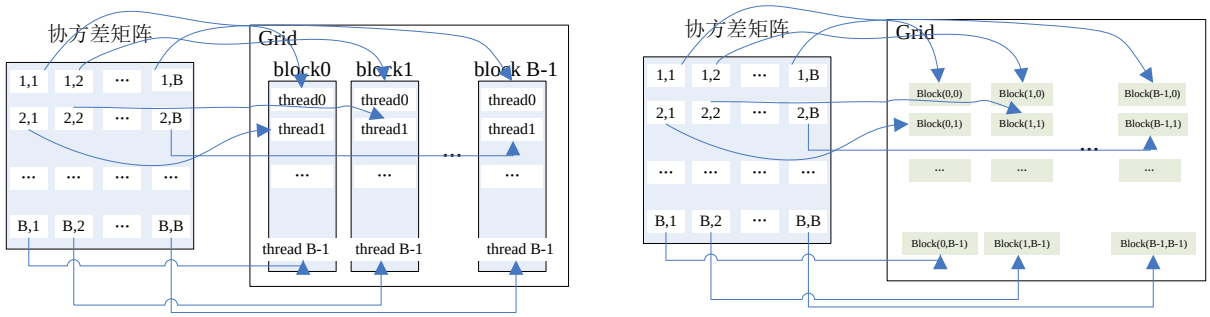

图 2 协方差矩阵计算任务的 GPU 映射方案

Fig.2 GPU scheme of COV matrix calculation

2.1.2 协方差矩阵计算的多 GPU 扩展

实现上述 5 种实现策略，通过实验比较选择最优方案，作为多 GPU 扩展的基础。为了适应单节点多 GPU 与多节点多 GPU 等不同的高性能计算体系结构，本文采用 MPI 进程控制 GPU 进行计算。

为了均衡分配协方差矩阵计算任务，首先对协方差计算公式进行如下变形：

$$\mathrm{COV}(\boldsymbol{X},\boldsymbol{Y})=\frac{1}{s-1}[\sum_{i=1}^{s}(\boldsymbol{X}_i-\overline{\boldsymbol{X}})(\boldsymbol{Y}_i-\overline{\boldsymbol{Y}})]$$

$$=\frac{1}{s-1}\left[\begin{array}{l}\left(\sum_{i=1}^{s/N}\boldsymbol{X}_i\boldsymbol{Y}_i+\sum_{i=s/N+1}^{2s/N}\boldsymbol{X}_i\boldsymbol{Y}_i+\cdots+\sum_{i=(N-1)s/N+1}^{s}\boldsymbol{X}_i\boldsymbol{Y}_i\right)\\-\frac{1}{s}\left(\sum_{i=1}^{s/N}\boldsymbol{X}_i+\sum_{i=s/N+1}^{2s/N}\boldsymbol{X}_i+\cdots+\sum_{i=(N-1)s/N+1}^{s}\boldsymbol{X}_i\right)\\*\left(\sum_{i=1}^{s/N}\boldsymbol{Y}_i+\sum_{i=s/N+1}^{2s/N}\boldsymbol{Y}_i+\cdots+\sum_{i=(N-1)s/N+1}^{s}\boldsymbol{Y}_i\right)\end{array}\right]$$

通过该变换，协方差的计算转变成了计算 $\Sigma\boldsymbol{X}$ 和 $\Sigma\boldsymbol{XY}$，划分成数据和计算均较独立的模块，更加适合任务划分。在进程级，不同的进程可以控制 GPU 进行独立的计算，得到结果后将其发送给 root 进程，由 root 进程最终计算得到协方差矩阵。整个计算过程仅需要 1 次进程间通信。

2.2 PCA 变换并行化

2.2.1 设备端并行 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，$\boldsymbol{A}$ 是 $B*m$ 的矩阵，$\boldsymbol{X}$ 的尺寸为 $S*B$，其结果矩阵 $\boldsymbol{Y}$ 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。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限的，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（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，可根据 occupancy 自由设定，使占用率最大化。

2.2.2 基于 MPI+CUDA 的两级并行 PCA 变换

PCA 变换的公式演变：$\boldsymbol{Y}=\boldsymbol{AX}=\{\boldsymbol{AX}_1,\boldsymbol{AX}_2\cdots\boldsymbol{AX}_N\}$，其中 $\boldsymbol{X}=\{\boldsymbol{X}_1,\boldsymbol{X}_2\cdots\boldsymbol{X}_N\}$。通过该变化，PCA 的变换过程可以分解为 N 个 $\boldsymbol{AX}_i$ 的运算，即每个进程仅需计算 $\boldsymbol{AX}_i$，这样就简单地完成了 PCA 变换过程的任务均衡划分。通过该方案划分好各进程的任务后，利用 MPI 控制进程调用 GPU 计算 $\boldsymbol{AX}_i$，实现多 GPU 资源的充分利用。

2.3 并行 I/O 设计

从前文基于多 GPU 的协方差矩阵并行计算和并行 PCA 变换方案中，各进程用到的高光谱数据 $\boldsymbol{X}_i$ 是独立的，在高光谱数据读入时各进程仅读入自身所需要的数据即可。其中 $\boldsymbol{X}_i$ 对高光谱数据中的 S 进行均匀分割。同理，输出时，结果矩阵 $\boldsymbol{Y}$ 写到一个单独的文件中，由子结果 $\boldsymbol{Y}_i$ 组成，因此可以通过 MPI 控制进程将各自的 $\boldsymbol{Y}_i$ 写到文件中对应的位置。并行输出还可避免巨大的结果矩阵通信开销。

2.4 G-PCA 算法与 Gs-PCA 算法

针对单 GPU，根据上述协方差矩阵计算和 PCA 变换的 GPU 映射方案，用其替代串行算法中的对应部分，即可得到 G-PCA 算法，图 3 为其流程图。其中前文提出了 5 种协方差矩阵计算的 GPU 实现策略，因此流程图中 GPU 协方差矩阵计算部分也需要不同的实现。结合 MPI+openmp 的输入输出、MPI+CUDA 的两级并行协方差矩阵计算、基于 MPI+CUDA 的两级并行 PCA 变换等策略，本文提出一种基于多 GPU 的 Gs-PCA 算法（图 4），图中 ***mul*** 矩阵由 $\Sigma \boldsymbol{XY}$ 组成，$\Sigma \boldsymbol{X}$ 组成 $\boldsymbol{x}$ 向量。该算法可移植性好，可适用于多 GPU 结点和 GPU 集群。

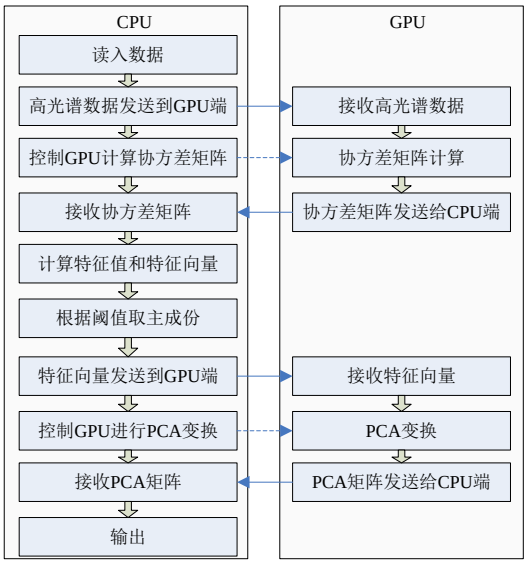

图 3 G-PCA 算法流程图

Fig.3 Flow chart of G-PCA

3 优化策略

3.1 GPU 细粒度并行优化

GPU 存储优化：GPU 提供了层次分明的存储单元系统，按离核距离由近及远分别是寄存器、共享存储器、全局存储器，可以人为指定这些存储单元的使用。离核距离近的存储单元延迟低，访存效率高，因此应尽量使用离核近的存储单元。本文通过下面两项技术进行优化（GPU 映射设计时已保证访存对齐）：1. 共享存储访问取代全局存储访问；2. 采用寄存器存储数量较少的中间变量。

基于 GPU 计算通信重叠优化策略：协方差矩阵计算方案 II 的③实现策略中，存在着通信计算重叠的可能。在图 1 右，第 i 行计算使用高光谱数据前 i 个波段中的数据。在传输完第 1 个高光谱波段数据后，同时启动协方差矩阵第一行计算和第 2 波段数据传输，即能实现计算与通信重叠。PCA 变换后的结果矩阵输出也可以实现计算通信重叠。通过 PCA 变换得到结果矩阵的一行后，在 GPU 计算下一行的同时，该行的数据可以传输到 CPU 端。

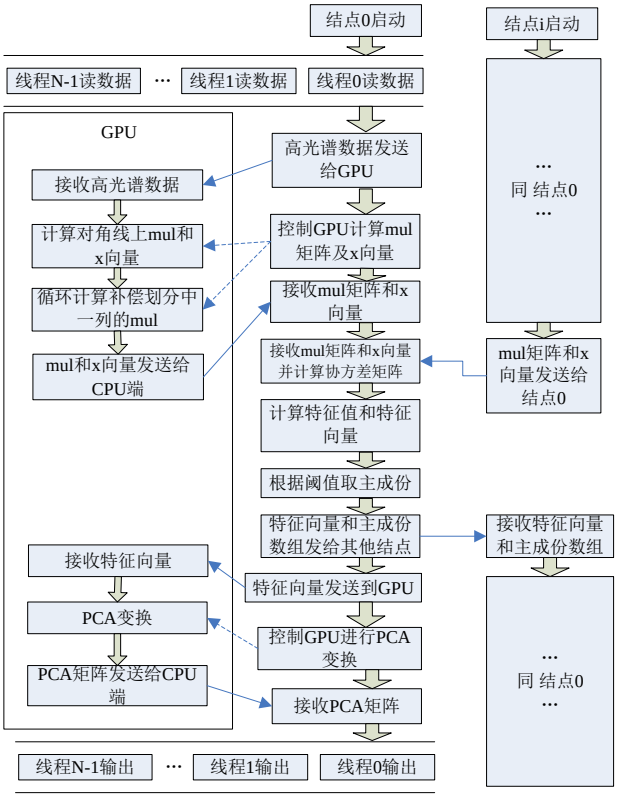

图 4 Gs-PCA 算法流程图

Fig.4 Flow chart of Gs-PCA

表 2 统计了相同映射策略情况下，GPU 存储优化和计算通信重叠优化策略的效果，其中优化手段 0 表示无优化，1 和 2 分别依次采取了 GPU 存储优化中 1、2 两种 GPU 存储优化技术，3 采取了计算通信重叠策略。表中数据显示，GPU 存储优化策略和计算通信重叠优化策略的效果显著。

表 2 各项优化技术效果(ms)

Tab.2 Effect of different optimizing methods(ms)

optimizing method	COV	PCA trans
0	16009.95	770.70
1	8019.15	659.46
2	6180.91	407.39
3	4077.66	335.59

4 实验结果与性能分析

4.1 实验平台

本实验采用 GPU 微型超算平台，搭配 2 个 8 核 Intel Xeon CPU E5-2650，64GB 的 ECC 内存和 2 张 Kepler 架构的 nVidia Tesla K20 GPU。本文程序在 Linux CentOS 6.2 操作系统下，采用 GCC 4.4.6 编译器、CUDA5.0 工具包和 openMPI 库对本文的不

同算法实现编译执行测试。

4.2 实验数据

本文采用的高光谱遥感图像数据来自 AVIRIS 的免费高光谱数据，共 *6* 组，表 3 罗列了这些数据的详细信息。该数据已经过前期预处理（通过 envi4.8 打开高光谱图像数据，将每个波段保存成 BMP 图像，后将 BMP 图像中的象元信息读取到一个流文件中，作为本文算法的输入数据）。原始数据是 *16* 位 int 类型保存的光谱信息，经预处理后转换成 unsigned char 类型的像素信息。

表 3 高光谱数据详细信息

Tab.3 Information of hyperspectral data

序号	宽	高	波段数
1	614	512	224
2	614	1087	224
3	753	1924	224
4	781	6955	224
5	1562	6955	224
6	1562	13910	224

4.3 串行程序最优化

首先进行串行程序的优化测试及优化开关选取。表 4 罗列了未开优化开关和分别开启优化开关-O1、-O2、-O3 时，处理高光谱遥感图像数据 2 的执行时间。选择优化开关-O2 时，串行程序执行时间最短，以此做为与并行算法对比的标准。

表 4 串行程序优化执行时间表

Tab.4 Run-time list of serial programs

优化开关	00	01	02	03
总时间/s	172.49	67.41	58.62	60.07

4.4 最优 G-PCA 的选择

本文共用GPU实现了*5*种不同的协方差矩阵计算，通过实验，对其结果进行比较。表 5 为计算高光谱数据 2、3、4 的协方差矩阵时间。

表 5 5 种 CUDA 并行协方差矩阵计算时间（ms）

Tab.5 Run time of 5 kinds for COV matrix calculation(ms)

数据	I	II ①	II ②	II ③	III
2	6547.2	627.8	509.2	475.4	1089.4
3	23078.5	1355.8	1217.9	1033.3	1795.9
4	164218	5096.1	5463.5	4049.6	5332.1

表中数据显示方案 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，将用方案 II 策略③实现的程序结果作为 G-PCA 算法的结果，与本文其他算法比较性能。方案 II 中的③策略能达到最佳性能，主要有两个因素：首先方案 II 的任务映射策略是最适用于 GPU 计算架构的；其次③策略中使用了一种计算和通信重叠的优化策略，几乎能隐藏所有的通信开销，特别是数据量巨大时，通信开销明显，该策略能获得更显著的效果。

4.5 并行效果及性能分析

分别执行用优化开关-O2 编译的串行 PCA 程序、G-PCA 和 Gs-PCA 程序处理 *6* 组高光谱数据，连续重复执行 *3* 次获得时间消耗的平均值，分别计算并行算法对串行算法的加速比（图 5）。

图 5 数据验证了本文提出的 G-PCA 和 Gs-PCA 算法整体的良好性能，单 GPU 可获得最高 *55* 倍的加速比，而双 GPU 可最高加速 *128* 倍，超过了单 GPU 的两倍，其原因是在协方差矩阵计算和 PCA 变换过程并行后执行时间大大下降，I/O 部分的比重提升，Gs-PCA 算法对 I/O 进行了并行化，从而获得了超过 G-PCA 两倍的性能提升。

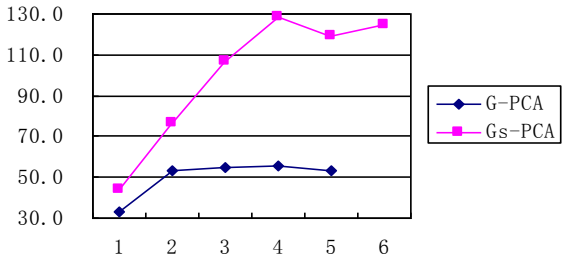

图 5 对 CPU 串行的总加速比

Fig.5 Total speed-up from CPU serial program

GPU 的片上存储空间是有限的，若程序的数据存储需求大于存储空间总量，程序将无法执行。图 10 中，由于 GPU 全局存储容量受限，G-PCA 算法无法处理第 *6* 组高光谱数据。而在 Gs-PCA 算法中，通过 MPI 层的数据划分，每个 GPU 的数据相互独立且仅为原来的一半左右，可以正常处理。

5 总结与展望

本文研究了高光谱遥感影像 PCA 降维方法，提出并实现了基于单 GPU 的 G-PCA 并行算法和基于多 GPU 的 Gs-PCA 算法。用多种高光谱遥感数据测试性能，实验结果显示本文设计的并行算法均能够取得良好的加速效果，G-PCA 算法能比最优串行程序加速 *55* 倍，而基于多 GPU 的 Gs-PCA 算法在 2 个 GPU 上运行能获得最高 *128* 倍的加速比。

本文研究验证了 CPU/GPU 异构系统上进行高光谱遥感影像降维能获得显著的性能提升。下一步工作是基于 CPU/GPU 异构系统开发其他高光谱影像降维方法，或基于最新的 MIC（Many Integrated Core）架构开发并行高光谱影像降维算法。

参考文献

[1] Green, RO, Eastwood, ML, Sarture, C.M., *et al*. Imaging spectroscopy and the Airborne Visible/Infrared Imaging Spectrometer(AVIRIS)[J]. *RSE*. 1998,65:227–248.

[2] Tom GT, Radhakrishna M. High-Speed Neural Network Filtering of Hyperspectral Data[C]. *SPIE, Bellingham, WA*, 2003:245-254.

[3] Green A, Berman M, Switzer P, *et al*. A transformation for ordering multispectral data in terms of image quality with implications for noise removal[J]. *IEEE TGRS*, 2000, 26(1):65-74.

[4] Kaarna A, Zemcik P, iainen H, *et al*. Compression of multispectral remote sensing images using clustering and spectral reduction[J]. *IEEE TSRS*, 2000, 38(2): 1588-1592.

[5] Scott D, Thompson J. Probability density estimation in higher dimensions[C]. *PSICSS*. 1983: 173–179.

[6] TOP500. TOP500 Supercomputer Sites[EB/OL].http://www.top500.org/

[7] David V, Alexey L, *et al*. Parallel Processing of Remotely Sensed Hyperspectral Images On Heterogeneous Networks of Workstations Using HeteroMPI[J]. *IJHPCA*. 2008, 22(4): 386-407.

[8] Javier P, Antonio P, *et al*. Parallel Classification of Hyperspectral Images Using Neural Networks[J]. *CIRSSCI*, 2008, 133: 193-216.

[9] Sergio S, Rui R. Real-time implementation of remotely sensed hyperspectral image unmixing on GPUs[J]. *JRTIP*,2012.

[10] Mahmoud E, Reda A. Fast GPU Algorithms for Endmember Extraction from Hyperspectral Images[C]. *ISCC, Cappadocia: IEEE*, 2012:631-636.

[11] He Yang, Qian Du. Unsupervised Hyperspectral Band Selection Using Graphics Processing Units[J]. *IEEE JSTAEORS*, 2011:660-668.