基于 GPU 的星图配准算法并行程序设计

陈 茜[1,2],邱跃洪[1],易红伟[1]

(1. 中国科学院西安光学精密机械研究所,陕西 西安 710119; 2 中国科学院大学,北京 100049)

摘 要: 星图配准是星图处理应用中的一个重要步骤, 因此星图配准的速度直接影响了星图处理的整体速度。近几年来,图形处理器(GPU)在通用计算领域得到快速的发展。结合 GPU 在通用计算领域的优势与星图配准面临的处理速度的问题,研究了基于 GPU 加速处理星图配准的算法。在已有配准算法的基础上, 根据算法特点提出了相应的 GPU 并行设计模型, 利用 CUDA 编程语言进行仿真实验。实验结果表明:相较于传统基于 CPU 的配准算法,基于 GPU 的并行设计模型同样达到了配准要求,且配准速度的加速比达到 29.043 倍。

关键词: GPU; CUDA; 星图配准; 通用计算; 加速

中图分类号: TP391 **文献标志码:** A **文章编号:** 1007-2276(2014)11-3756-06

Parallel programming design of star image registration based on GPU

Chen Xi[1,2], Qiu Yuehong[1], Yi Hongwei[1]

(1.Xi'an Institute of Optics and Precision Mechanics, Chinese Academy of Sciences, Xi'an 710119, China; 2.University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: The speed of star image registration affects the whole speed of the processing of the star image as star image registration is one of the most important steps of star image processing. In recent years, the general purpose computing of graphic process unit(GPU)has a rapid development. In this paper, the computing power of GPU for the general purpose computing and the problem of the speeding up of processing of star image registration were combined to study the accelerated processing algorithm based on GPU. A parallel model of GPU for the registration algorithm was proposed and CUDA programming language was uesd to realize it. Experiment result shows that the parallel model also fulfills the purpose of the image registration and has a 29.043X speedup compared with the serial CPU program.

Key words: GPU; CUDA; star image registration; general purpose computing; acceleration

收稿日期: 2014-03-13; **修订日期:** 2014-04-15

基金项目: 国家自然科学基金(61178033/F050701;10978005/A030801)

作者简介: 陈茜(1988-),女,硕士生,主要从事图像处理及 GPU 在通用计算领域加速的研究工作。Email:chenxialice1207@163.com

导师简介: 邱跃洪(1968-),男,研究员,博士生导师,博士,主要从事高速实时信号处理、光电信息处理方面的工作。Email:yhqiu68@opt.ac.cn

0 引　言

相对于地球范围内的观测者而言，恒星可以看作是无穷远并具有一定光谱特性的点光源。恒星在 CCD 像平面上表现为较暗背景上的点状光斑[1]。星图是以恒星为观测目标所获得的星空图片。星图配准是星图处理过程中一个重要的环节。

图像配准是对两幅或多幅图像中的同一目标进行对正,确定图像间对应点的匹配关系,以消除或减小目标的位置差别及噪声引起的畸变[2]。典型的图像配准系统一般分为 4 个部分:特征检测、特征匹配、变换模型参数估算以及图像重采样与变换[3]。一般而言,图像配准的方法大致分为三类：基于像素的配准方法、基于特征的配准方法和基于模型的配准方法[4]。

传统的配准算法大多是基于 CPU 进行的串行程序设计。近几年来,图形处理器(GPU)的快速发展提出了一种新型的编程模型：并行程序设计。GPU 通过同时执行多个线程来处理一些计算密度高、逻辑分支简单的算法，大大降低了算法时间复杂度。GPU 在通用计算领域的快速发展也为图像配准的实时性要求提供了新的思路[5]。

GPU 从诞生之日起,处理能力就以超越摩尔定律的速度增长。GPU 面向通用计算主要有以下几个优势:(1)具有大量的并行计算核心，能够提供更多的计算资源;(2)具有高存储带宽，加快数据传输速度;(3)线程切换开销小，提高整体执行性能;(4)采用只读式缓存,能够快速读取处理数据[6]。

文中通过对传统星图配准算法的分析，结合 GPU 编程模型的特点对原有基于 CPU 的配准算法进行相关的并行设计优化并利用 CUDA 编程语言进行仿真实验。实验结果表明:基于 GPU 的星图配准并行算法较原有串行算法相比在时间复杂度上得到了大幅降低。

1 CUDA 编程模型及存储结构

2007 年 6 月,NVIDIA 推出了统一计算设备架构(CUDA)[7]。CUDA 是一种将 GPU 作为数据并行计算设备的软硬件体系。CUDA 通过 C 语言扩展可以编写 CUDA 程序应用于 GPU 通用计算。与以往的 GPU 相比，支持 CUDA 架构的GPU 更适于通用计算。一是由于其采用了统一处理架构,可以更加有效地利用过去分布在顶点渲染器和像素渲染器的计算资源;二是引入了片内共享存储器,支持随机写入和线程间通信。

1.1 CUDA 编程模型

NVIDIA 公司开发的 CUDA 语言是 C 语言的扩展,这使得熟悉 C 语言的编程人员能够快速掌握应用 CUDA 开发通用计算程序。CUDA 编程模型采用中文单指令多线程(SIMT)模式执行程序,将 CPU 作为主机称为 Host,GPU 作为协处理器称为 Device。图 1 是 GPU 编程模型示意图。CPU 负责进行逻辑性强的事务处理和串行计算,GPU 则专注于执行高度线程化的并行处理任务。用户将运行在 GPU 上的 CUDA 并行计算函数称为内核函数 (kernel)。一个 kernel 函数并不是一个完整的程序，而是整个 CUDA 程序中可以被并行执行的步骤。一个完整的 CUDA 程序是由一系列设备端 kernel 函数并行步骤和主机端的串行处理步骤共同组成的。kernel 以线程网格(grid)的形式组织,每个 grid 由若干线程块(block)组成,每个线程块又由若干线程(thread)组成。

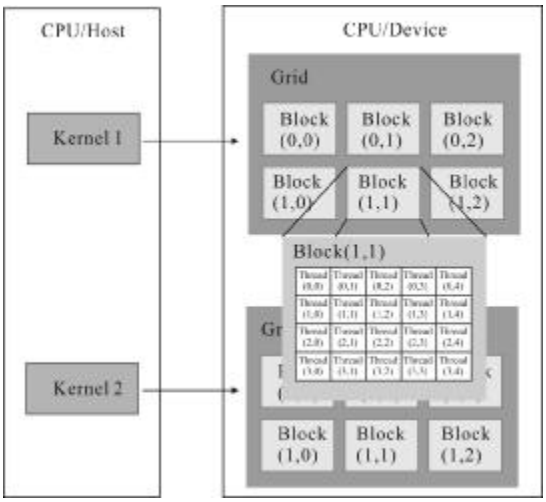

图 1 CUDA 编程模型

Fig.1 CUDA programming model

为了克服传统 GPU不能实现线程间通信的缺点,CUDA 中同一 block 中的 thread 不仅能够并行执行，还能够通过共享存储器和栅栏同步实现 block 内的线程间通信。这样同一 grid 内不同 block 之间的并行称为粗粒度并行，同一 block 内的线程间并

行称为细粒度并行。在编程时通过对不同算法进行分析,对将要执行的任务进行合理划分,利用细粒度和粗粒度两级并行来实现资源的高效利用达到算法的加速效果。

1.2 CUDA 存储结构

CUDA 中一共定义了 8 种存储器模型，分别是寄存器(register),局部存储器(local memory),共享存储器(shared memory),全局存储器(global memory),可分页内存 (pageable memory)，页锁定内存(pinned memory),纹理存储器(texture memory),常数存储器(constant memory)。每一个线程拥有自己的存储器和局部存储器;每一个线程块拥有一块共享存储器;grid 中的所有线程都可以访问全局存储器。常数存储器和纹理存储器是可以被所有线程访问的只读存储器。图2 为 CUDA 存储结构示意图。

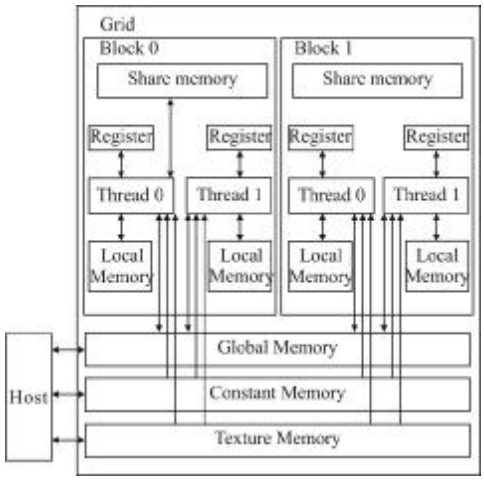

图 2 CUDA 存储结构

Fig.2 CUDA memory structure

寄存器是 GPU 片上高速缓存器,执行单元可以以极低的延迟访问寄存器。如果寄存器被消耗完,数据将被存储在局部存储器中。由于局部存储器中的数据被保存在显存中而不是片上寄存器或缓存中,因此对局部存储器的访问速度很慢。共享存储器也是 GPU 上的片内高速存储器。访问共享存储器的速度几乎和寄存器一样快，是实现线程间通信的延迟最小方法。全局存储器能提供很高的带宽,但同时也具有较高的访存延迟。要有效利用全局存储器带宽必须遵守合并访问要求来避免分区冲突。可分页内存即为通过操作系统 API 分配的存储器空间；而页锁定内存始终不会被分配到低速的虚拟内存中,能够保证存在于物理内存中并且能够通过 DMA 加速与设备端的通信。常数存储器具有缓存机制,可以节约带宽加快访问速度。纹理存储器适合于实现图像处理和查找表，对大量数据的随机访问或非对齐访问也有良好的加速效果。

2 基于GPU 的星图配准算法并行设计

根据星图的图像特点，传统的星图配准方法一般分为两个步骤:星点检测和星点匹配。当星图发生旋转与平移变换时，星图配准的关键就是测量出原始星图与待配准星图之间的旋转变换参数和平移参数。文中采用一种通过交叉投影来检测星点的具体位置的算法[8]。在确定星点的具体位置后计算出星点的质心，通过最小二乘法计算出待配准星图与原始星图之间的旋转参数矩阵和平移参数矩阵。最后利用双线性插值法对待配准星图进行插值从而获得最终配准后的星图。

2.1 星点检测

在星点检测之前首先对星图图像进行预处理。选取一个合适阈值对图像进行二值化处理。大于阈值的像素点灰度值为 1，小于阈值的像素点灰度值为 0。交叉投影星点检测算法的基本原理为:首先对星图图像进行竖直投影运算,并检测投影灰度值大于 0 的坐标区间，则这些坐标区间必然是存在星像点的区间。在每个区间范围内,对图像进行水平投影检测，进而确定该坐标区间内各星像点的纵坐标范围。显然,在进行水平投影检测时,每次检测的操作对象为由竖直检测结果确定的一个带状图像。经过竖直和水平两次投影检测，就得到了各星点外接四边形的 4 个顶点坐标,即确定了星像点的分布范围。图 3 给出了交叉投影算法的原理图。

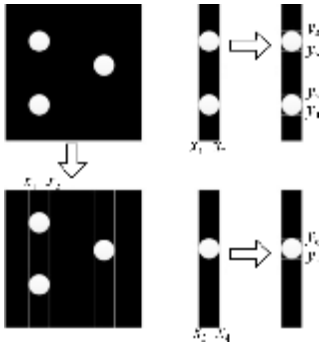

图 3 交叉投影算法示意图

Fig.3 Scheme of orthogonal projection algorithm

2.2 星点匹配

通过前文交叉投影的方法确定出星点外接四边形的坐标后，利用质心法即可确定出星点的具体位置。将原始星图与待配准星图的星点质心逐一进行空间欧氏距离的测量，将距离值小于测量前选择的合适阈值的星点质心点对作为匹配星点对。再利用最小二乘法根据测量出的 **N** 对匹配星点对计算出旋转参数 **R** 和平移参数 **T**。对于第 i 对匹配点对有：

$$\begin{bmatrix} \mathbf{Y}_i \\ \mathbf{X}_i \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \mathbf{x}_i \\ \mathbf{y}_i \end{bmatrix} + \begin{bmatrix} \mathbf{t}_x \\ \mathbf{t}_y \end{bmatrix} \tag{1}$$

式中：$\mathbf{Y}_i$ 和 $\mathbf{X}_i$ 分别为待配准星图中第 i 个对应点的横纵坐标，$\mathbf{x}_i$ 和 $\mathbf{y}_i$ 分别为原始星图中第 i 个对应点的横纵坐标。上式可以分解为以下两式：

$$\mathbf{X}_i = \mathbf{t}_x + \cos\theta \cdot x_i + \sin\theta \cdot \mathbf{y}_i \tag{2}$$

$$\mathbf{Y}_i = \mathbf{t}_y + (-\sin\theta) \cdot x_i + \cos\theta \cdot \mathbf{y}_i \tag{3}$$

对于公式(2)，令 $\mathbf{a}=\mathbf{t}_x$，$\mathbf{b}=\cos\theta$，$\mathbf{c}=\sin\theta$，则：

$$\mathbf{X}_i = \mathbf{a} + \mathbf{b} \cdot x_i + \mathbf{c} \cdot \mathbf{y}_i \tag{4}$$

参数 **a**、**b**、**c** 的值是使误差函数取得最小时的值，误差函数的表达式为：

$$\mathbf{E}_x = \sum_{i=1}^{N} (\mathbf{a} + \mathbf{b} \cdot \mathbf{x}_i + \mathbf{c} \cdot \mathbf{y}_i - \mathbf{X}_i)^2 \tag{5}$$

令误差函数 $\mathbf{E}_x$ 分别对 **a**、**b**、**c** 求偏导数并令它们为零，可以得到：

$$\begin{bmatrix} \sum_{i=1}^{N} 1 & \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 & \sum_{i=1}^{N} x_i \cdot y_i \\ \sum_{i=1}^{N} y_i & \sum_{i=1}^{N} x_i \cdot y_i & \sum_{i=1}^{N} y_i^2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} X_i \\ \sum_{i=1}^{N} x_i \cdot X_i \\ \sum_{i=1}^{N} y_i \cdot X_i \end{bmatrix} \tag{6}$$

解方程(6)即可得到 **a**、**b**、**c** 的值，进而求得 $\mathbf{t}_x$、$\cos\theta$、$\sin\theta$ 的值。对于公式(3)，用同样的方法可以得到 $\mathbf{t}_y$、$\cos\theta$、$\sin\theta$ 的值。

利用上述最小二乘法计算出来的旋转参数 **R** 和平移参数 **T**，可以得到原始星图发生旋转平移变换后对应的星点坐标。再利用得到的这些对应星点的灰度值根据双线性插值的方法对待配准星图进行插值，最终得到配准后的星图。

2.3 星图配准算法的并行设计

在对 CPU 串行算法进行并行优化程序设计时，最关键的步骤在于分析出原始串行算法中计算密度大且具有高度并行性的部分。再根据具体算法进行任务的划分，从而充分利用 GPU 的并行计算优势达到算法加速的目的。

CPU 端首先将原始星图和待配准星图的图像像素数据读入 CPU 主存，然后将其加载于 GPU 的 global memory。通过启动 kernel 函数进行并行运算，最后再将 GPU 计算好的结果写回 CPU 端。通过前文对基于 CPU 的星图配准串行算法的描述，可以分析发现该算法的计算量主要存在于两个部分：星点的检测和原始星图发生旋转平移变换后对应星点坐标的计算。因此，利用 GPU 的并行计算优势对这两部分算法进行并行优化设计，从而达到配准加速的目的。

将星点的检测和原始星图发生旋转平移变换后对应星点坐标的计算分别设计为一个 kernel 函数，GPU 通过调用完成计算。在星点检测算法中，首先对原始星图和待配准星图按照 grid 的 block 划分。实验的测试图像为 256×256 的星图。进行竖直投影时，其中每个 block 对应星图图像的一列像素，则图像被划分为 256 个 block。再将每个 block 划分为 256 个 thread。这样就使星图图像中的每个像素对应为一个 thread。同理，水平投影时一个 block 对应星图图像的一行像素。这样划分后，进行双向投影的 kernel 函数设计完成，以下列出水平投影的 kernel 函数代码：

```
__global__ voidSum_row(float*A,float*B)
{
    int bid = blockIdx.x;
    inttid = threadIdx.x;
        float sum = 0;
        for(tid=0;tid<N;tid++){
            sum = sum+A[bid*N+tid];
        }
        B[bid] = sum;
}
```

其中 threadIdx.x，blockIdx.x 分别为线程网格中定位线程位置的内建变量。threadIdx.x 表示一个线

程在 block 中的 ID,blockIdx.x 表示一个 block 在 grid 中的ID。在对计算原始星图发生旋转平移变换后对应星点坐标的算法进行并行设计时，由于原始串行算法中涉及到矩阵的相乘,因此可以利用 GPU 对矩阵乘法进行优化。由于星图图像是以二维形式组织,因此为了设计方便将 GPU 上的线程也按照二维形式组织。令星图图像像素与线程网格在横轴方向变量号为,纵轴方向变量标号为,则线程网格与星图图像像素之间的映射关系为：

$$i = threadIdx.x + blockIdx.x \times blockIDim.x$$

$$i = threadIdx.y\ v + blockIdx.y \times blockIDim.y$$

进行二维映射后，优化矩阵相乘的 kernel 函数代码为：

```
__global__ void MatMulKernel (float *d_c, float *d_a, float *d_b, float *d_T)
{
 int i = blockIdx.x*blockDim.x+threadIdx.x;
      int j =blockIdx.y*blockDim.y+threadIdx.y;
      for(int j=0;j<256;j++)
 {
           for(int i=0;i<256;i++)
  {
                d_c [j*256 +i] =d_a [j*256 +i]* (*(d_T + 0)) + d_b[j*256+i]*(*(d_T+1)) + (*(d_T+2));
           }
      }
}
```

3 实验结果与分析

实验的测试环境为：CPU 是 4 核的 Intel(R) Core(TM)i5-2400,内核主频率为 3.10 GHz,内存为 4 GB。GPU 为 NVIDIAGeforceGTX660Ti,内含 1 344 个流处理器,显存为 2 GB。操作系统为 Windows7,并在 Visual Studio2010 中配置 CUDA4.2 进行基于 GPU 的并行 CUDA 程序的实验。测试的对象是大小为 256×256 的星图。图 4(a)为原始星图,图 4(b)为待配准星图。由于 CUDA 算法是基于原始串行算法进行的优化设计,因此图表中只列出利用 CUDA 进行改进的串行算法部分。实验测试对比结果如表 1 所示。

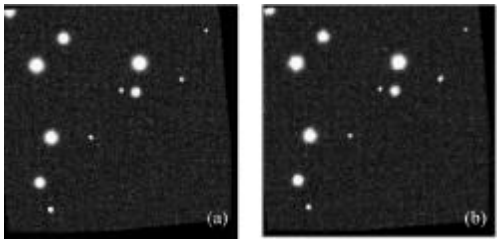

图 4 测试星图

Fig.4 Tested star image

表 1 算法主要性能

Tab.1 Major properties of algorithm

Properties	Executed by CUDA/ms	Serial executed /ms	Speedup ratio
Star detection	18.283	789.411	43.1776
Star matching	25.017	328.971	13.150
Total time	46.693	1 356.089	29.043

实验结果表明基于 GPU 的 CUDA 并行优化程序在配准精度方面达到了原有串行程序的配准要求,图 5(a)和图 5(b)分别为串行程序执行和 CUDA 执行后生成的配准星图。由表 1 可以看出，经过 CUDA 编程进行并行优化的程序运行时间得到了显著的降低，相较串行程序执行总时间加速比达到 29.043 倍。尤其是在星点检测部分的优化加速比达到 43.177 倍,这是因为星点检测采用的交叉投影算法具有高度并行，运算量集中的特点，充分利用 GPU 通用计算的高度并行性和高带宽的特点。

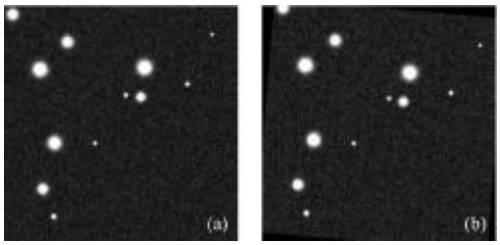

图 5 实验结果

Fig.5 Experiment result

4 结 论

文中在对传统串行星图配准算法进行分析后，将算法中计算密度高、并行度强的部分提取出来,利

用 GPU 对其进行了并行程序设计。相较于传统基于 CPU 的串行程序配准速度得到了显著的提升,同时也为实时星图配准提供了新的思路, 为星图图像处理拓展了新的研究方向。这种将串行算法进行 CUDA 并行算法优化的思路对于图像处理中的其他算法也具有一定的参考意义。未来研究的方向主要是面向 CUDA 存储级的优化,进一步充分利用 GPU 的并行运算资源,达到更好的加速效果。

参考文献:

[1] Zhai You, Zeng Luan, Xiong Wei. Star matching based on invariant feature descriptor [J]. Optics and Precision Engineering, 2012, 20(11): 2531-2539. (in Chinese)
翟优,曾峦,熊伟. 基于不变特征描述符实现星点匹配[J]. 光学 精密工程, 2012, 20(11): 2531-2539.

[2] Zeng Wenfeng, Li Shushan, Wang Jiang'an. Translation, rotation and scaling changes in image registration based affine transformation model [J]. Infrared and Laser Engineering, 2001, 30(1): 18-20. (in Chinese)
曾文锋,李树山,王江安. 基于仿射变换模型的图像配准中的平移、旋转和缩放 [J]. 红外与激光工程, 2001, 30(1): 18-20.

[3] Zhang Chunhua, Zhou Xiaozhong, Wang Xuewei, et al. Research of star-sky image registration method for dim target detection [J]. Journal of Image and Graphics, 2012, 20(11): 435-442. (in Chinese)
张春华,周晓中,王学伟,等. 一种面向弱小目标检测的序列星图配准算法 [J]. 中国图像图形学报, 2012, 20(11): 435-442.

[4] Yuan Jinsha, Zhao Zhenbing, Gao Qiang, et al. Review and prospect on infrared/visible image registration [J]. Laser & Infrared, 2009, 39(7): 694-697. (in Chinese)
苑津莎, 赵振兵, 高强,等. 红外与可见光图像配准研究现状与展望 [J]. 激光与红外, 2009, 39(7): 694-697.

[5] Wu Enhua, Liu Youquan. General purpose computation on GPU [J]. Journal of Computer-Aided Design & Computer Graphics, 2004, 16(5): 602-603. (in Chinese)
吴恩华, 柳有权. 基于图形处理器 (GPU) 的通用计算[J]. 计算机辅助设计与图形学学报, 2004, 16(5): 602-603.

[6] Zhou Haifang, Zhao Jin. Parallel programming design and storage optimization of remote sensing image registration based on GPU[J]. Journal of Computer Research and Development, 2012, 49: 281-286. (in Chinese)
周海芳,赵进. 基于 GPU 的遥感图像配准并行程序设计与存储优化 [J]. 计算机研究与发展, 2012, 49: 281-286.

[7] Zhang Shu, Chu Yanli, Zhao Kaiyong, et al GPU Performance Computing of CUDA [M]. Beijing: China Waterpower Press, 2009:11-13. (in Chinese)
张舒,褚艳利,赵开勇,等. GPU 高性能运算之 CUDA[M].北京:中国水利水电出版社, 2009: 11-13.

[8] Wang Zhaokui, Zhang Yulin. Algorithm for CCD star image rapid locating [J]. Chin J Space Sci, 2006, 26 (3): 209-214. (in Chinese)
王兆魁,张育林.一种 CCD 星图星点快速定位算法[J]. 空间科学学报, 2006, 26(3): 209-214.