基于 GPU 加速的边界面法正则积分的研究

张见明[1+]，余列祥[1],刘路平[2]

(1. 汽车车身先进设计与制造国家重点实验室 湖南大学 长沙 410082; 2. 中国水电顾问集团中南勘测设计研究院 长沙 410014)

摘要：相比传统的微处理器 CPU，图像处理单元 GPU 具备低成本、低功率，高性能等特点。由于 GPU 强大的并行计算能力和快速提升的可编程能力，除了其在图形处理上应用，GPU 越来越广泛的应用于数值计算上。正则积分是边界面法中计算较为密集的部分，具有高度并行性。本文首次在 CUDA 编程环境中运用 GPU 对边界面法正则积分进行加速，并在 NVIDIA GTX680 GPU 和英特尔(R)酷睿(TM)酷睿 i7-3770K CPU 的计算平台上与传统的正则单元积分进行对比。数值算例表明，在保证相同精度的前提下，加速比可达到 8.3。

关键字：GPU，并行计算，边界面法，CUDA，正则积分

中图分类号：TH164 文献标识代码：A

Research on Regular Integration in boundary face method Based on GPU-acceleration

Jianming Zhang[1+] Liexiang Yu[1] Luping Liu[2]

(1. State Key Laboratory of Advanced Design and Manufacturing for Vehicle Body, Hunan University, Changsha, China, 410082; 2. Hydrochina Zhongnan Engineering Corporation, Changsha, 410014)

Abstract: Graphics processing unit (GPU) is a low-cost, low-power (watts per flop) and very high performance alternative to conventional microprocessors. In addition to GPU's applications in graphics processing, it was used in numerical computing more and more widely due to its powerful ability of parallel computing and rapidly improved programmability. The regular integration possessing high level of parallelism is a computationally intensive part of the BFM. This paper makes a first try of applying GPU to accelerate computation for boundary face method (BFM) in CUDA (Compute Unified Device Architecture) programming environment. Comparative computations compared to traditional regular element integration are made on both NVIDIA GTX680 GPU and Intel(R) Core(TM) i7-3770K CPU. Numerical examples show that, a speedup of 8.3x has been achieved at the same level of accuracy.

Keywords: GPU, parallel computing, BFM, CUDA, regular integration

近年来，GPU已经具备了实现大规模快速计算的编程能力，不仅用来进行图形处理，还用于通用计算[1]。CPU的架构提供了2到4个多核，而GPU架构提供了一百多个核，

收稿日期：2012-09-07
基金项目：国家自然科学基金资助项目(10972074)；国家自然科学基金资助项目(11172098)

作者简介：余列祥（1988-），男，浙江温州人，硕士；张见明（1965-），男，湖北孝昌县人，教授，博士
[+]通讯联系人：张见明, E-mail: zhangjm@hnu.edu.cn

甚至更多，使得利用很多线程来进行并行处理[2]。GPU的每秒浮点操作数和带宽也远远超过同一时期的CPU，且价格相比CPU更为低廉。

NVIDIA 公司于 2006 年 11 月提出了 CUDA[3-5](Compute Unified Device Architecture，计算统一设备架构)，它不需要借助于图形学API，并采用了比较容易掌握的类C语言进行开发，开发人员能够从熟悉的C语言比较平稳地从CPU过渡到GPU，而不必重新学习语法，因此大大降低了CUDA程序的开发难度。CUDA是一个新的基础架构，是一个软硬件协同的完整的解决方案。这种架构使得GPU可以处理复杂的科学计算问题，特别是极大数据量的并行计算问题[6]。如今，CUDA被广泛应用于石油勘探[7]，天文计算[8]，计算流体动力学[9,10]，生物学[11]等领域，在很多应用中获得了几倍，几十倍，乃至上百倍的加速比。Yong Cai[12] 等人运用CUDA进行了显示有限元板材成形仿真系统的并行开发，获得27倍的加速比；M.Januszewski[13] 等人提出了随机微分方程的数值解的并行方案，加速比达到了675；Juan C. Pichel[14] 等人将稀疏矩阵向量乘法并行化，获得了2.6倍的加速比。

在传统的边界元法[15-18]中，边界单元不仅要用来进行边界积分和物理变量插值，而且用来近似几何体。当网格较为稀疏时，会引起较大的几何误差，从而影响计算精度。张见明[19] 等人提出了边界面法，边界积分和场变量插值都是在实体边界的参数空间内进行。在边界积分过程中，积分点的几何数据，如物理坐标、雅可比，外法向量都是直接由曲面算得，而不是通过单元插值近似，从而避免了几何误差[20]。

随着计算规模的增大，计算精度和效率之间的权衡成为了边界面应用的瓶颈，而单元积分的计算时间占了整个边界面法分析计算时间的相对较大的比重。为提高边界面法的计算效率，我们提出了一个基于 GPU 计算的并行积分方案来加速边界面法中的单元积分。在初始阶段，我们已经实现了单元积分中正则积分的并行化。

本文详细介绍了边界面法正则积分的 CUDA 并行加速方案，并给出了典型的数值算例。本文的后续内容组织如下：第 1 节概述了 CUDA；第 2 节我们介绍了边界积分方程，边界积分方程离散，和正则积分；第 3 节提出了正则积分的并行加速方案；第 4 节给出了两个数值算例来评估我们开发的并行代码的加速性能；第 5 节对本文作出了总结和工作展望。

1 CUDA 概述

CUDA 是 NVIDIDA 基于 C 语言编程的 GPGPU 模型，可被大多数学过 C 语言的人所掌握，无需图形芯片结构的知识便能开发出能在图形芯片上执行的 CUDA 代码。GPU 由若干个流多处理器 SM 组成，8 个标量流处理器 SP 组成一个 SM。作为 CPU 的协调处理器，CUDA 为 GPU 提供了大规模的多线程架构。相比 CPU 仅有的 1-4 个内核，GPU 架构的内核却达到数百个，使得其更适合用于单指令多数据(SIMD)计算[21]。如图 3 所示，CUDA 通用并行计算的过程包括四个步骤：(1)初始化主机端上的数据；(2)从主机端复制数据到设备端(GPU)；(3)并行计算；(4)从设备端传递数据回数据。

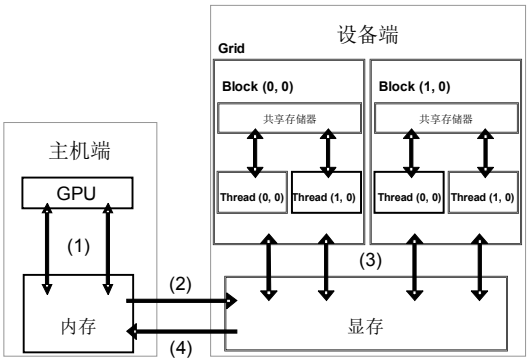

图 3 CUDA 通用并行计算过程

Fig.3 General parallel computing process of CUDA

在 CUDA环境下编程时，可把GPU视为一个能够进行多线程并行处理的运算设备。如果程序中的某个函数对于不同的数据要重复执行多次，就可以把这部分函数独立出来，分配给GPU 的线程来执行。将此函数编译为在GPU上执行的指令集，编译后的目标程序称之为核（Kernel）。核并不是完整的程序，而是整个程序中若干基本的关键数据的并行计算步骤[6]。

核(Kernel)以网格(Grid)的形式执行，如图4所示，每个网格由若干个线程块（Block）组成，每一个线程块又由最多512个线程(Thread)组成。在Fermi[22]架构中，支持的线程数可达1536个。处于同一block中的线程，不仅能并行执行，且能通过共享存储器(Shared memory)和栅栏(Barrier)进行通信。在同一个Block中的线程通过共享存储器交换数据，并通过栅栏同步保证线程间能够正确地共享数据。共享存储器和栅栏嵌套在粗粒度的数据并行和任务并行里，提供了细粒度的数据并行和线程并行[3]。在实际操作中，Block会被分割成更小的线程束(warp[3,5])。在采用Tesla架构的GPU中，一个线程束由连续的32个线程组成。

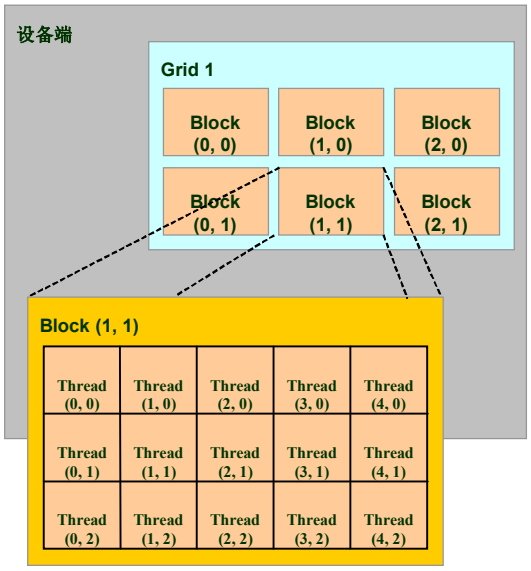

图 4 线程块网络

Fig.4 Grid of thread blocks

2 边界积分方程，边界方程离散，和正则积分

2.1 边界积分方程

对任意三维域 Ω，满足拉普拉斯控制方程的位势边界值问题可以表示为：

$$\begin{aligned} &u_{,ii}=0, \quad \forall x\in\Omega \\ &u=\bar{u}, \quad \forall x\in\Gamma_u \\ &u_{,i}n_i\equiv q=\bar{q}, \quad \forall x\in\Gamma_q \end{aligned} \tag{1}$$

式中 Ω 的整个边界为 $\Gamma=\Gamma_u+\Gamma_q$，$\bar{u}$ 和 $\bar{q}$ 分别是位势已知边界 Γ_u 和法向流已知边界 Γ_q 的边界值，n 是边界外法向矢量，n_i 是法矢量分量 i =1,2,3。

该问题可以等价地转化为以下的势问题正则化边界积分方程：

$$0=\int_\Gamma((u(s)-u(y))q^s(s,y)d\Gamma-\int_\Gamma q(s)u^s(s,y)d\Gamma \tag{2}$$

其中 Γ 为边界，u 和 q 代表位势和法向流量，$q=\partial u/\partial n$，y 和 s 为边界上的源点和场点，$q^s(s,y)$ 和 $u^s(s,y)$ 是基本解对于三维势问题：

$$u^s(s,y)=\frac{1}{4\pi}\frac{1}{r(s,y)} \tag{3}$$

$$q^s(s,y)=\frac{\partial u^s(s,y)}{\partial n}=\frac{1}{4\pi r^2(s,y)}\frac{\partial r(s,y)}{\partial n(s)} \tag{4}$$

2.2 边界积分方程离散和正则积分

首先把域边界离散成 M 个边界单元和 N 个相应的积分点。位势和法向流量已给出，对于任何一个节点，u 或者 q 其中一个是已知的，在边界上的 u 和 q 可表示成如下式所示：

$$\begin{aligned} u(s)=u(u,v)=\sum_{k=1}^{N}N_ku_k \\ q(s)=u(u,v)=\sum_{k=1}^{N}N_kq_k \end{aligned} \tag{5}$$

其中，u 和 v 表示边界参数曲面的二维参数坐标，三维物理空间坐标(x, y, z)满足 $x=x(u, v)$, $y=y(u, v)$, $z=z(u, v)$, $N_k(S)$ 是形函数。将方程(5)代入方程(2)后，可得：

$$\begin{aligned} 0=&-\sum_{j=1}^{M}\int_{\Gamma_j}q^s(s,y)\sum_{k=1}^{N}(N_k(s)-N_k(y))u_kd\Gamma \\ &+\sum_{j=1}^{M}\int_{\Gamma_j}u^s(s,y)\sum_{k=1}^{N}N_k(s)q_kd\Gamma \end{aligned} \tag{6}$$

公式(6)可以组装成矩阵形式：

$$\begin{bmatrix} H_{11} & H_{12} & \cdots & H_{1N} \\ H_{21} & H_{22} & \cdots & H_{1N} \\ \vdots & \vdots & \ddots & \vdots \\ H_{N1} & H_{N2} & \cdots & H_{NN} \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{Bmatrix} = \begin{bmatrix} G_{11} & G_{12} & \cdots & G_{1N} \\ G_{21} & G_{22} & \cdots & G_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ G_{N1} & G_{N2} & \cdots & G_{NN} \end{bmatrix} \begin{Bmatrix} q_1 \\ q_2 \\ \vdots \\ q_N \end{Bmatrix} \tag{7}$$

或

$$Hu = Gq \tag{8}$$

其中，

$$H_{ik} = \sum_{j=1}^{M} \int_{\Gamma_j} q^s(s, y_i)(N_k(s) - N_k(y_i))d\Gamma \tag{9}$$

$$G_{ik} = \sum_{j=1}^{M} \int_{\Gamma_j} u^s(s, y_i) N_k(s) d\Gamma \tag{10}$$

方程(6)右端第一项在任何情况下都是正则的。因此在任一个单元内可用正则高斯积分方法计算。对于方程(6)右端第二项，当源点 y 趋近于场点 s 时会出现弱奇异性。假使 y 距离 s 足够远，则认为方程右端第二项是正则的。以矩形单元为例，如图 5 所示，L 为三角形最长边，d 表示 y 和 s 的距离。如果 $d > Lk$ (k 为系数，本文其值为 4.0)，该单元便是正则单元。

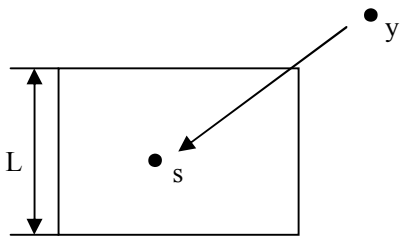

图 5 矩形单元

Fig.5 Rectangle cell

3 CUDA 并行方案

本文通过运用 CUDA 计算规则单元的正则积分得到矩阵 H 和矩阵 G 每一项的值。在传统的正则积分程序中，矩阵 H 和矩阵 G 的计算过程是串行的。图 7 描述了该串行算法：在第一层循环中有 N 个单元，每个单元包含一定数量的场点(三角形线性单元有三个场点)。在第二层循环里，CPU 串行计算矩阵 H 和矩阵 G，一次计算得到 H 和 G 中的一项。在第二层循环结束，返回第一层循环再对其他单元进行循环计算。

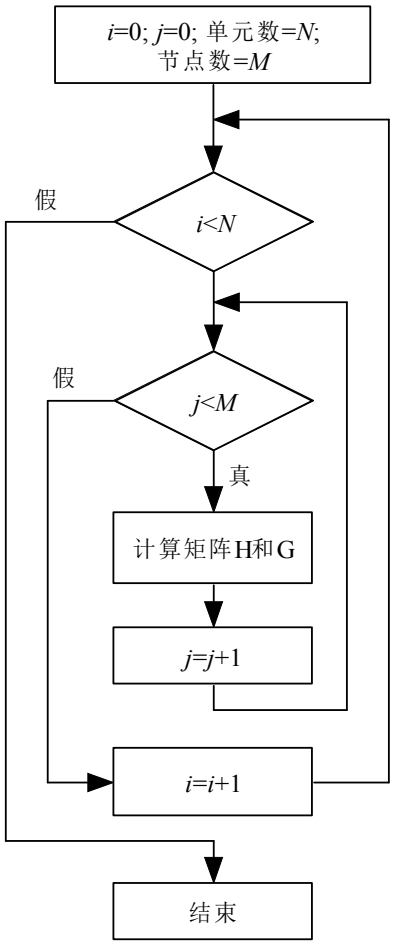

图 6 CPU 计算流程图

Fig.6 Flowchart of CPU computing

本文基于 CUDA 开发 GPU 并行程序，每一个线程并行计算矩阵 H 和矩阵 G 中的一项。考虑到 GPU 全局内存即显存的大小限制和图 6 第二层循环中正则积分所需的数据大小，因此在一次数据传输中尽可能传输更大的数据量。如图 7 所示，本文将 P 个单元的数据量打包一次性传入显存中，而不采用一次只传单个单元的数据量的方式。这样做的原因在于，在数据传输量相同的前提下，一次数据传输比分多次数据传输节省更多的时间。

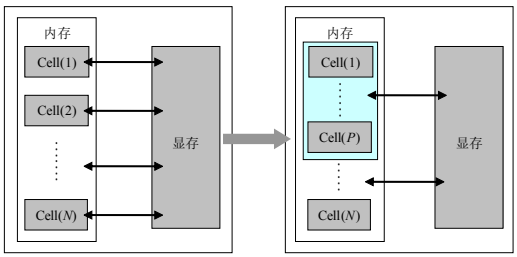

图 7　数据传输方式

Fig.7 Scheme of data transmission

GPU 并行程序的算法如图 8 所示。首先，通过传入数据量除以 GPU 显存得到传输数据的次数 *GPU_Compute_div*， *N* 表示单元的个数。在第二层循环中，将从内存传输 *P* 个单元的数据量到显存中。*tid_in_grid* 表示线程在整个 *Grid* 中的位置，每一个线程计算并行计算矩阵 *H* 和矩阵 *G* 中的一项。每一个执行内核的线程被赋予唯一的线程 ID，它能在 Kernel 中通过内建变量访问[7]。*blockIdx.x* 和 *threadIdx.x* 从 CUDA 运行时中获得,*blocksize* 定义为 Block 在 x 方向上的维度。本文使用一维 Grid 和一维 Block，每个 Grid 里拥有 *Q* 个线程并行计算矩阵 *H* 和矩阵 *G*。

```
1:  P ← N/ GPU_Compute_div
2:  for i=0 to GPU_Compute_div do
3:      for j=0 to P do
4:          tid_in_grid ← blockIdx.x * blocksize + threadIdx.x
5:          if tid_in_grid < Q then
6:              compute H and G
7:          end if
8:      end for
9:  end for
```

图 8　GPU 计算的算法

Fig.8 Algorithm of GPU computing

4 数值算例

本文选用两种三维几何模型[23]对 GPU 并行程序进行测试：一种模型是槽钢，另一种是散热片。为评估该 GPU 程序的计算精度，本文使用以下立方体解析场：

$$x^3+y^3+z^3-3yx^2-3xz^2-3zy^2 \qquad (11)$$

GPU 程序的计算机测试平台如表 1 所示：

表 1 计算平台

Tab.1 Computing platform

操作系统	Windows 7 64bit，16GB 内存
CPU	Intel(R) Core(TM) i7-3700K CPU 3.5GHZ
GPU	NVIDIA GTX680 GPU，NVIDIA driver version 296.10, 包含 8 个多流处理器 SM, 2GB 显存. 计算能力为 3.0
编译器	Visual studio 2008, CUDA toolkit 4.2.9 64bit

GPU 并行程序的执行效率与 CPU 程序作对比。加速比通过 CPU 程序的计算时间除以 GPU 并行程序的计算时间计算得到，本文只记录计算正则积分所耗费的时间。

4.1 槽钢模型

本文对所用的槽钢模型进行了简化，去除了圆角特征，并采用线性三角形单元进行网格划分。如图 9 所示，该槽钢被离散划分为 1026 个单元和 3078 个节点。

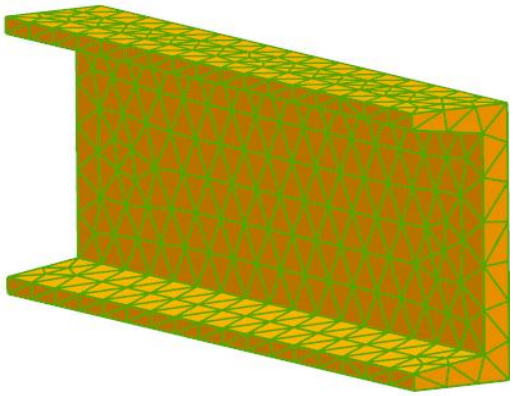

图 9 边界面法的槽钢网格划分

Fig.9 The BFM mesh for channel-section steel

表 2 记录了 CPU 程序和 GPU 程序的正则积分计算时间、精度，加速比。从图中可以明显看出数据传输占用的时间占据 *GPU* 计算时间很大一部分的比重，这是由于本文所使用的 GPU 的带宽不足导致的。图 10 和图 11 为 CPU 程序和 GPU 程序加速比和精度的对比，可以看出 GPU 程序的计算速度明显比 CPU 程序快得多。在计算精度得到保证的前提下，随着节点数的增加，加速比也随之增大，但当节点数增大到一定值时，继续增大节点数，加速比提高得却不明显。

表 2 槽钢计算结果

Tab.2 Result of cube computing

节点数	CPU 计算时间(s)	GPU 计算时间(s)（注：此表未列出 GPU 计算时数据在内存之间的传输时间）			误差(%)	加速比
		总计	数据传输时间	内核计算时间		
1254	0.125	0.047	0.031	0.000	0.9853	2.6595
1806	0.233	0.078	0.047	0.015	0.5467	4.7949
3078	0.903	0.156	0.094	0.015	0.2300	5.7885
6714	4.103	0.624	0.374	0.063	0.1389	6.5753
9744	9.348	1.357	0.687	0.205	0.1135	6.8887
13338	18.034	2.231	1.279	0.219	0.1058	8.0833
15072	22.898	2.746	1.778	0.320	0.0908	8.3387

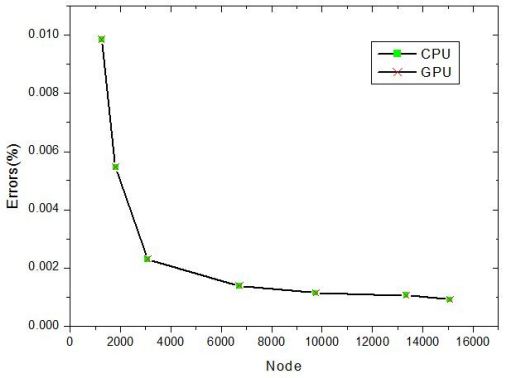

图 10 CPU 程序和 GPU 程序的精度

Fig.10 Errors of CPU-based code and GPU-based code

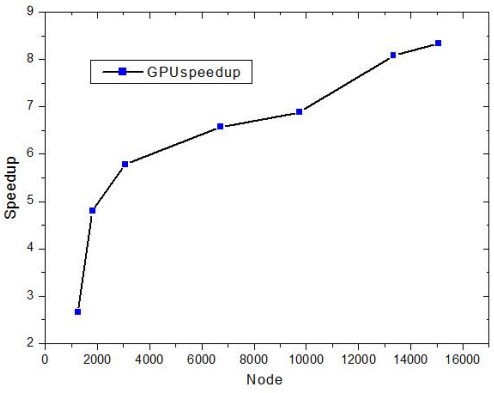

图 11 GPU 程序的加速比

Fig.11 Speedup of GPU-based code

4. 2 散热片模型

如图 12 所示，将散热片模型划分为 2046 个三角形线性单元和 6138 个节点。CPU 程序和 GPU 程序的计算精度及其加速比见图 13 和图 14，从图中可以看出，模型划分网格数越密，计算精度越高，加速比也随之增大。且 CPU 程序和 GPU 程序的计算精度保持高度一致，在节点数为 14478 时，加速比达到了 8.1。但同样，并行程序在计算规模增大到一定程度时加速效果提高得并不明显。

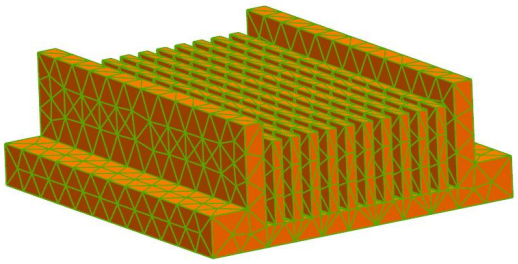

图 12 边界面法的散热片网格划分

Fig.12 The BFM mesh for heatsink

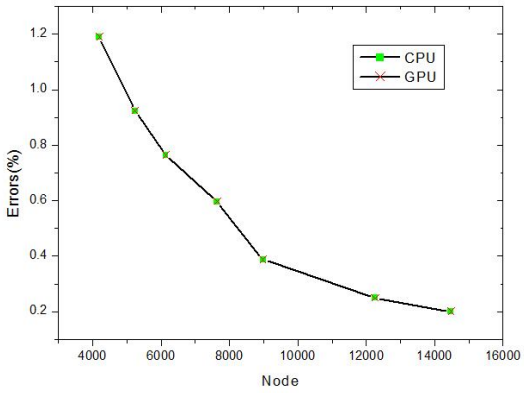

图 13 CPU 程序和 GPU 程序的精度

Fig.13 Errors of CPU-based code and GPU-based code

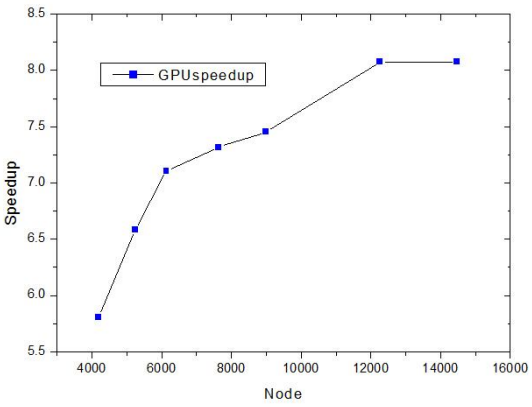

图 14 GPU 程序的加速比

Fig.14 Speedup of GPU-based code

5 结论

本文将 GPU 和边界面法相结合，开发

出了基于 CUDA 的边界面法正则积分的并行程序，并验证边界面法在 CUDA 并行硬件平台的可行性。通过两个数值算例，在获得高计算精度的前提下，加速比达到了 8.3 倍。加速比随着节点数的增大而增大，但不是线性增长，计算规模增大到一定程度时，加速比提高的不明显，这是因为用于传输的数据量较大，花费在显存与内存之间的数据传输时间将严重制约着程序的计算速度，因此 GPU 的带宽便成了程序的瓶颈。对于并行性好，数据传输量少的程序，GPU 的加速效果将会非常显著。在本文中，为与传统正则积分程序保持一致，GPU 程序同样采用双精度浮点运算，数值结果表明 GPU 并行程序同样获得了良好的计算精度。

本文仅是对边界面程序中正则积分进行了并行化，为更大限度得运用 GPU 对边界面程序进行并行化，在未来的工作中，我们将运用 CUDA 对边界面法中的近奇异积分、奇异积分，大型线性方程组的 LU 分解进行加速。具备低成本和高性能的 GPU，伴随着其通用计算流处理技术的发展，GPU 并行计算在工程应用上的潜力是巨大的。

参考文献：

[1]Takehiko Nawata, Reiji Suda. APTCC: Auto Parallelizing Translator From C To CUDA. Procedia Computer Science 2011;4:352–361.

[2]李波,赵华成,张敏芳.CUDA 高性能计算并行编程[J]. 微型电脑应用:25(9):55-57.

[3]Nvidia Corporation, CUDA C Programming Guide. Version 4.0. 2011.

[4]Nvidia Corporation, http://www.Nvidia.corn/cuda.

[5]Nvidia Corporation, CUDA C Best Practices Guide Version 4.0,2011.

[6]焦良葆,陈瑞.GPU 核函数细化研究[J].计算机工程:2010, 36 (18):10-12.

[7]E. O. Aksnes, H. Hesland, and A. C. Elster. GPU Techniques for Porous Rock Visualization. Technical report, 2009.

[8]Harris, Chris; Haines, Karen; Staveley-Smith, Lister. GPU accelerated radio astronomy signal convolution. Experimental Astronomy 2008;22(1):129–142.

[9]Andrew Corrigan, Fernando F. Camelli, Rainald Löhner, John Wallin. Running unstructured grid-based CFD solvers on modern graphics hardware. International Journal for Numerical Methods in Fluids 2011;66(2):221–229.

[10]Crane K, Llamas I, Tariq S. Real-time simulation and rendering of 3d fluids. GPU Gems 2007;3:663–675.

[11]Juan C. Pichel, Francisco F. Rivera, Marcos Fernández, Aurelio Rodríguez. Optimization of sparse matrix–vector multiplication using reordering techniques on GPUs. Microprocessors and Microsystems 2012;36(2):65–77

[12]Yong Cai, Guangyao Li, Hu Wang, Gang Zheng, Sen Lin. Development of parallel explicit finite element sheet forming simulation system based on GPU architecture. Advances in Engineering Software 2012;45(1):370–379.

[13]M. Januszewski, M. Kostur. Accelerating numerical solution of stochastic differential equations with CUDA. Computer Physics Communications 2010;181(1):183–188.

[14]Juan C. Pichel, Francisco F. Rivera, Marcos Fernández, Aurelio Rodríguez. Optimization of sparse matrix–vector multiplication using reordering techniques on GPUs. Microprocessors and Microsystems 2012;36(2):65–77

[15]A.H.-D. Cheng, C.S. Chen, M.A. Golberg, Y.F. Rashed BEM for theomoelasticity and elasticity with body force — a revisit. Engineering Analysis with Boundary Elements 2001;25(4–5):377–387.

[16]Kok HwaYu, A. Halim Kadarman, Harijono Djojodihardjo. Development and implementation of some BEM variants — A critical review. Engineering Analysis with Boundary Elements 2010;34(10):884–899.

[17]Bonnet M. Boundary Integral Equation Methods for Solids and Fluids. Wiley: New York, 1995.

[18]龙述尧编著. 边界单元法概论[M]. 中国科学文化出版社, 2002.3.

[19]Zhang JM, Qin XY, Han X, Li GY. A boundary face method for potential problems in three dimensions. Int. J. Num. Meth. Eng 2008;80:320–337.

[20]Xianyun Qin, Jianming Zhang, Guangyao Li,

Xiaomin Sheng, Qiao Song, Donghui Mu. An element implementation of the boundary face method for 3D potential problems. Engineering Analysis with Boundary Elements 2010;34:934–943.

[21]Yang Liu, Wayne Huang, John Johnson and Sheila Vaidya. GPU Accelerated Smith-Waterman .Computer Science 2006; 3994:188–195.

[22]NVIDIA C. NVIDIA's next generation CUDA compute architecture; Fermi 2009.

[23]Xianyun Qin, Jianming Zhang, Guangyao Li, Xiaomin Sheng, Qiao Song, Donghui Mu. An element implementation of the boundary face method for 3D potential problems. Engineering Analysis with Boundary Elements 2010;34:934–943.