Journal of Computer Applications
计算机应用,2015,35(6):1564 - 1566,1579
ISSN 1001-9081
CODEN JYIIDU
2015-06-10
http://www.joca.cn

文章编号:1001-9081(2015)06-1564-03
doi:10.11772/j.issn.1001-9081.2015.06.1564

基于图形处理器的球面 Voronoi 图生成算法优化

王 磊[1*],王鹏飞[1],赵学胜[1],卢立托[2]
(1.中国矿业大学(北京) 地球科学与测绘工程学院,北京 100083;
2.中国石油集团工程设计有限责任公司 北京分公司,北京 100085)
(*通信作者电子邮箱 wl890627@163.com)

摘 要:基于四元三角格网(QTM)之间距离计算与比较的球面 Voronoi 图生成算法相对于扩张算法具有较高的精度,但由于需要计算并比较每个格网到所有种子点的距离,致使算法效率较低。针对这一问题,利用图形处理器(GPU)并行计算对算法进行实现,然后从 GPU 共享内存、常量内存、寄存器等三种内存的访问方面进行优化,最后用 C++语言和统一计算设备架构(CUDA)开发了实验系统,对优化前后算法的效率进行对比。实验结果表明,不同内存的合理使用能在很大程度上提高算法的效率,且数据规模越大,所获得的加速比越高。

关键词:球面 Voronoi 图;统一计算设备架构;共享内存;常量内存;寄存器
中图分类号: TP391.41; P208 **文献标志码**:A

Optimization of spherical Voronoi diagram generating algorithm based on graphic processing unit

WANG Lei[1*], WANG Pengfei[1], ZHAO Xuesheng[1], LU Lituo[2]
(1. *College of Geoscience and Surveying Engineering, China University of Mining and Technology (Beijing), Beijing* 100083, *China*;
2. *Beijing Company, China Petroleum Engineering Company Limited, Beijing* 100085, *China*)

Abstract: Spherical Voronoi diagram generating algorithm based on distance computation and comparison of Quaternary Triangular Mesh (QTM) has a higher precision relative to dilation algorithm. However, massive distance computation and comparison lead to low efficiency. To improve efficiency, Graphic Processing Unit (GPU) parallel computation was used to implement the algorithm. Then, the algorithm was optimized with respect to the access to GPU shared memory, constant memory and register. At last, an experimental system was developed by using C++ and Compute Unified Device Architecture (CUDA) to compare the efficiency before and after the optimization. The experimental results show that efficiency can be improved to a great extent by using different GPU memories reasonably. In addition, a higher speed-up ratio can be acquired when the data scale is larger.

Key words: spherical Voronoi diagram; Compute Unified Device Architecture (CUDA); shared memory; constant memory; register

0 引言

随着数字地球的发展,球面 Voronoi 图(以下简称 V 图)在全球空间索引[1-2]、球面动态模拟[3]等方面得到了广泛的应用。基于四元三角格网(Quaternary Triangular Mesh, QTM)[4]单元之间距离计算与比较的球面 V 图生成算法[5]相对于扩张算法具有较高的精度,但由于需要计算并比较每个格网到所有种子点之间的距离,致使算法效率较低,无法满足实时应用的需求。

近年来,图形处理器(Graphic Processing Unit, GPU)技术快速发展,其浮点运算能力和内存带宽已远远超过中央处理器(Central Processing Unit, CPU)[6]。NVIDIA 公司开发的统一计算设备架构(Compute Unified Device Architecture, CUDA)为 GPU 增加了一个易用的编程接口[7],使得 GPU 并行计算在群体行为仿真[8]、三维数据并行可视化[9]、地球表面测绘[10]等领域得到广泛应用。

本文利用 CUDA 对基于距离计算与比较的球面 V 图生成算法进行实现,然后从 GPU 共享内存、常量内存、寄存器等三种内存的访问方面,对算法进行优化,最后利用 C++和 CUDA 开发了实验系统,对优化前后算法的效率进行对比。

1 基于 QTM 的球面 Voronoi 图并行生成算法

给定一个区域 Ω、一系列的实体 $s_i(i=1,2,\cdots,n)$ 和一个距离函数 d,则实体 s_i 的 Voronoi 区域 V_i 可以通过式(1)定义:

$$V_i = \{\omega \in \Omega \mid d(\omega,s_i) < d(\omega,s_j), \forall s_j \in \Omega, i \neq j\} \tag{1}$$

若将式(1)中的 ω 定义为一个 QTM 格网,Ω 定义为球面上所有 QTM 格网的集合,距离函数 d 定义为球面大弧距离,则式(1)表示的即为基于 QTM 的球面 V 图。

收稿日期:2015-01-07;**修回日期**:2015-04-02。
基金项目:国家自然科学基金资助项目(41171306);高等学校博士学科点专项科研基金资助项目(20130023110001)。
作者简介:王磊(1989 -),男,安徽宿州人,博士研究生,主要研究方向:球面 Voronoi 图、GPU 并行计算; 王鹏飞(1991 -),男,山东德州人,硕士研究生,主要研究方向:三维建模、海量三维模型可视化; 赵学胜(1967 -),男,山东菏泽人,教授,博士,主要研究方向:三维 GIS、数字地球空间建模; 卢立托(1986 -),男,河北石家庄人,助理工程师,硕士,主要研究方向:并行计算、全球定位系统。

基于 QTM 格网之间距离计算与比较的球面 V 图生成算法[5],通过计算并比较每个格网到所有种子点的距离,来确定格网单元所属的 Voronoi 区域,算法具有计算密集性、指令一致性及相互独立性的特点,本文采用 GPU 单指令多数据流(Single Instruction Multiple Date,SIMD)并行计算模型来执行这些运算,算法实现的核函数(Kernel)伪码如下(Kernel_1)。

```
算法　Kernel _1。
输入　gridVerts:格网中心点的坐标数据(全局内存);
      siteVerts:种子点坐标数据(全局内存);
      siteCount:种子点数量。
输出　bestSites:所有格网的最近种子点数据(全局内存)。
1)  Dim tempDist ← Max
2)  Dim tempBest ← Null
3)  Dim tid ← blockDim. x * blockIdx. x + threadIdx. x
4)  Dim center ← gridVerts[tid]
5)  Loop i From 0 To siteCount Do
6)    Dim curDist ← GetDist(center, siteVerts[i])
7)    If (curDist < tempDist) Then
8)      tempDist ← curDist
9)      tempBest ← i
10)   End If
11) End Loop
12) bestSites[tid] ← tempBest
```

2　算法的优化

在 CUDA 架构中有多种内存可供使用,如全局内存、局部内存、共享内存、常量内存、寄存器等,每种内存的空间大小、作用范围及访问速度都各不相同。因此,在实现算法时使用不同的内存、不同的访问方式,将会对程序的性能产生巨大的影响。本文将从 GPU 内存访问方面(包括共享内存、常量内存及寄存器等)对上述算法进行优化。

2.1　共享内存优化

GPU 全局内存位于显存,具有较大的存储空间,可被 Kernel 调用的所有线程访问,但全局内存的访问具有较高的访存延迟(一次访问需要 400 ~ 600 个时钟周期)。而共享内存是 GPU 片内的高速存储器,它的缓冲区驻留在物理 GPU 上,而不是 GPU 之外的系统内存中。因此,共享内存的访存延迟要远远低于全局内存(只有全局内存的 1% 左右)[11-12]。共享内存可供一个线程块内所有的线程访问,是实现线程间通信的延迟最小的方法[11]。然而,共享内存存储空间较小,可供每个线程块使用的共享内存大小仅为 16 KB(计算能力 1.1 的设备)。

Kernel_1 所述算法中,格网中心点的坐标数据及种子点坐标数据,都存储在全局内存中,在计算每个 QTM 格网到所有种子点的距离时,需要频繁地从全局内存获取种子点数据。设种子点数为 N, 球面上 QTM 格网总数为 G,则生成球面 V 图所需要的计算和比较的总次数为 $2NG$,所需要的全局内存访问的次数为 $(G + NG)$,二者之比为 $2NG:(G + NG) \approx 2:1$。也就是说,每进行两次计算(或比较)就需要进行一次全局内存的访问,这将严重影响算法的效率。

共享内存相对于全局内存具有更高的访存速度,因此可先将种子点数据从全局内存中取出,存储到共享内存中,供一个线程块内所有的线程(每个线程处理一个格网)使用,每次计算只需要从共享内存中读取数据即可。但由于可供每个线程块使用的共享内存仅为 16 KB,当种子点数量较多时,无法一次性放入共享内存。这就需要分成多次,每次将部分种子点数据放入共享内存,当该部分数据使用完成后,再取出一部分种子点数据进行计算,依次进行,直至所有的种子点都计算完毕。算法实现的伪码如下(Kernel_2)。

```
算法　Kernel _2。
输入　gridVerts:格网中心点的坐标数据(全局内存);
      siteVerts:种子点坐标数据(全局内存);
      siteCount:种子点总数量;
      Count:每次放入共享内存中的种子点数(取线程块尺寸)。
输出　bestSites:所有格网的最近种子点数据(全局内存)。
1)  Dim tempDist ← Max
2)  Dim tempBest ← Null
3)  Dim tid ← blockDim. x * blockIdx. x + threadIdx. x
4)  Dim center ← gridVerts[tid]
5)  Loop i From 0 To siteCount/Count Do
6)    Dim subSites ← GetSubSites(siteVerts, i * Count)
7)    Dim sharedSites[Count]
8)    sharedSites[threadIdx. x] ← subSites[threadIdx. x]
9)    syncthreads()
10)   Loop j From 0 To Count
11)     Dim curDist ← GetDist(center, sharedSites[j])
12)     If (curDist < tempDist) Then
13)       tempDist ← curDist
14)       tempBest ← j + i * Count
15)     End If
16)   End Loop
17)   syncthreads()
18) End Loop
19) bestSites[tid] ← tempBest
```

若每个线程块中的线程数量为 B,则每个线程块内所要进行的计算和比较的总次数为 $2NB$,需要进行的全局内存访问次数为 $(B + N)$(包括从全局内存中取出 B 个 QTM 格网数据和 N 个种子点数据),则计算(或比较)的次数与全局内存访存次数之比为 $2NB:(B + N) = 2N:(1 + N/B)$。若取 $N = 1\,024$、$B = 256$,则计算与访存次数之比为 $2\,048:5 \approx 400:1$。这相对于原算法大幅度减小了全局内存访问的次数,在很大程度上提高了算法的效率。

2.2　常量内存优化

GPU 中的常量内存是只读内存。如果半个线程束(Half-Warp, 16 个线程)中的每个线程都从常量内存的相同地址读取数据,GPU 只会产生一次读取请求并在随后将数据广播到每个线程;同时,由于常量内存的内容是不会发生变化的,硬件会主动把这个常量数据缓存在 GPU 上,在第一次从常量内存的某个地址上读取后,当其他半线程束请求同一个地址时,将命中缓存,同样减少了额外的内存流量。因此,与从全局内存中读取数据相比,从常量内存中读取相同的数据可以节约大量的内存带宽[11]。

由于算法中的种子点数据不会发生改变,且每个线程都从第 1 个种子点数据开始读取,因此,可将种子点数据直接从 CPU 内存复制到 GPU 常量内存中,以消除全局内存访问对算法效率的影响。与使用共享内存类似,在计算能力为 1.1 的设备中,GPU 常量内存的大小只有 64 KB,当种子点数据较大时,需要分成多次将数据从 CPU 内存复制到 GPU 常量内存,并多次调用核函数进行计算。算法实现的伪码与 Kernel_1 类似,只是将种子点数据存储在 GPU 常量内存中。

2.3 寄存器优化

在 2.2 节所述的算法中，每个 GPU 线程处理一个 QTM 格网，即在处理每个 QTM 格网时，都要从常量内存中获取所有的种子点数据。尽管 GPU 常量内存的访问速度较快，但频繁的访问也将带来效率上的影响，为减少对常量内存的访问次数，本节采取如下策略：在每个线程中，将多个 QTM 格网数据从全局内存中取出，存储在寄存器中，同时从常量内存中获取一次种子点数据，供这些 QTM 格网使用。算法实现的伪码如下(Kernel_3)。

```
算法  Kernel _3。
输入  gridVerts:格网中心点的坐标数据(全局内存);
      siteVerts:种子点坐标数据(常量内存);
      siteCount:种子点总数量;
      Num:每个线程处理格网的数量。
输出  bestSites:所有格网的最近种子点数据(全局内存)。
1)  Dim tempDist[Num] ← Max
2)  Dim tempBest[Num] ← Null
3)  Dim No ← (blockDim.x * blockIdx.x + threadIdx.x) * Num
4)  Dim center[Num]
5)  Dim site
6)  Loop i From 0 To Num Do
7)    center[i] ← gridVerts[No + i]
8)  End Loop
9)  Loop i From 0 To siteCount Do
10)   site ← siteVerts[i]
11)   Loop j From 0 To Num Do
12)     Dim curDist ← GetDist(center[j],site)
13)     If (curDist < tempDist[j]) Then
14)       tempDist[j] ← curDist
15)       tempBest[j] ← i
16)     End If
17)   End Loop
18) End Loop
19) Loop i From 0 To Num Do
20)   bestSites[No + i] ← tempBest[i]
21) End Loop
```

3 实验与分析

实现本文算法所用的编程语言为C++和 NVIDIA CUDA 6.5；硬件环境为 Intel Core 2 Quad CPU Q8400 @2.66 GHz，2.00 GB内存，NVIDIA GeForce 9600 GT GPU，计算能力 1.1。在该环境下，将基于距离计算与比较的球面 V 图生成算法及上述各优化算法进行实现，并进行了对比分析。

基于第 9 层 QTM 格网(格网总数 2 097 152)，分别用不同数量的种子点对以上算法进行测试，原算法(CPU 串行)生成球面 V 图所用的时间如表 1 所示，各优化算法在不同数据规模下所获得的加速比如图 1 所示。图 2 为利用不同数量的种子点数生成的球面 V 图。由图 1 可以看出，使用 GPU 全局内存对算法进行实现时，频繁的全局内存访问致使算法执行的时间与 CPU 串行算法相当，仅可获得较小的加速比；而通过使用共享内存、常量内存、寄存器对算法进行加速后，在不同的数据规模(种子点数)下都能够获得几十甚至上百倍的加速比，使算法执行的时间大大降低。从图 1 中还可以看出，当种子点数较少时，所获得的加速比较低，随着种子点数的增多，加速比也随之增大。这是由于 GPU 更适合于密集型的计算，当数据量(计算量)较小时，算法在 GPU 上的执行时无法隐藏访存和数据传输的延迟[12]，而随着数据量的增大，这些延迟逐渐被隐藏，因此加速比会逐渐增大；而当数据量大到能使 GPU 满负荷工作时，所有的访存和数据传输的延迟都已被很好地隐藏，加速比也将趋于稳定，如图 1 所示。

表 1 原算法 CPU 串行生成不同种子点数球面 V 图所用时间

种子点数	生成时间/ms	种子点数	生成时间/ms
100	2 950	5 000	129 229
500	13 882	10 000	287 545
1 000	27 499	50 000	1 458 757

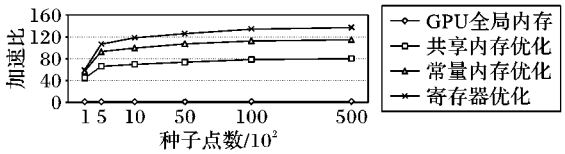
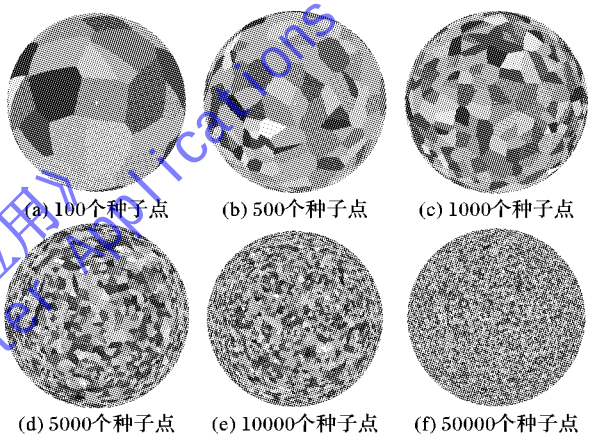


图 1 加速比随数据规模的变化曲线



(a) 100个种子点 (b) 500个种子点 (c) 1000个种子点

(d) 5000个种子点 (e) 10000个种子点 (f) 50000个种子点

图 2 不同数量种子点数生成的球面 Voronoi 图

4 结语

本文利用 CUDA 对基于距离计算与比较的球面 V 图生成算法进行实现，并从 GPU 共享内存、常量内存、寄存器等三种内存的访问方面对算法进行优化，通过实验得出如下结论：

1)仅使用 GPU 全局内存时，算法的效率仅与 CPU 串行算法相当；

2)GPU 共享内存、常量内存、寄存器的合理使用，能够大大提高算法的效率；

3)在一定的数据规模下，随着种子点数的增多，GPU 加速比逐渐增大。

本文仅从 GPU 内存访问的角度对基于距离计算与比较的球面 V 图生成算法进行了优化，下一步的工作是综合考虑算法指令、任务划分、内存访问等因素，对算法进一步优化。

参考文献：

[1] LUKATELA H. Ellipsoidal area computations of large terrestrial objects [C/OL]// Proceedings of the First International Conference on Discrete Grids. [2014-12-01]. http://ncgia.ucsb.edu/global-grids-book/eac/.

[2] LUKATELA H. Hipparchus geopositioning model: an overview [C/OL] // Proceedings of the Eighth International Symposium on Computer-Assisted Cartography, 1987. [2014-12-01]. http://www.geodyssey.com/papers/hlauto8.html.

(下转第 1579 页)

的数据请求;通过标准安全模型证明了方案的选择安全性。

后续工作的重点将放在:1)数据请求 DataRequest 过程中不同用户的同步问题;2)用户组二叉树的存储开销优化问题。

表 2　空间占用比较

方案	密钥长度	密文长度
APPCP_ABE	$(\|S\|+O(\log T)^2)G_1+O(\log N)$	$(3+2l)G_1+G_2$
文献[1]方案	$(\|S\|+2)G_1$	$(2+l)G_1+G_2$
文献[6]方案	$(\|S\|+O(\log T)^2)G_1$	$(2+l)G_1+G_2$
文献[9]方案	$(\|S\|+l)G_1+O(\log N)$	$(3+2l\log N)G_1+G_2$

表 3　安全性对比

方案	属性撤销安全性	密钥泄漏安全性
APPCP_ABE	前后向安全	前后向安全
文献[1]方案	未考虑	未考虑
文献[6]方案	未考虑	前向安全
文献[9]方案	前向安全	未考虑

参考文献:

[1] SAHAI A, WATERS B. Fuzzy identity based encryption [C]// Proceedings of 2005 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques. Berlin: Springer-Verlag, 2005: 457 – 473.

[2] GOYAL V, PANDEY O, SAHAI A, *et al.* Attribute-based encryption for fine-grained access control of encrypted data [C]// Proceedings of the 13th ACM Conference on Computer and Communications Security. New York: ACM, 2006: 89 – 98.

[3] TANG Y, LEE P, LIU J, *et al.* Secure overlay cloud storage with access control and assured deletion [J]. IEEE Transactions on Dependable and Secure Computing, 2012, 9(6): 903 – 916.

[4] HONG C, ZHANG M, FENG D. AB-ACCS: a cryptographic access control scheme for cloud storage [J]. Journal of Computer Research and Development, 2010, 47(Z1): 259 – 265. (洪澄, 张敏, 冯登国. AB-ACCS: 一种云存储密文访问控制方法[J]. 计算机研究与发展, 2010, 47(Z1): 259 – 265.)

[5] HONG C, ZHANG M, FENG D. Achieving efficient dynamic cryptographic access control in cloud storage [J]. Journal on Communications, 2011, 32(7): 125 – 132. (洪澄, 张敏, 冯登国. 面向云存储的高效动态密文访问控制方法[J]. 通信学报, 2011, 32(7): 125 – 132.)

[6] WEI J, LIU W, HU X. Forward-secure ciphertext-policy attribute-based encryption scheme [J]. Journal on Communications, 2014, 35(7): 38 – 45. (魏江宏, 刘文芬, 胡学先. 前向安全的密文策略基于属性加密方案[J]. 通信学报, 2014, 35(7): 38 – 45.)

[7] WANG P, FENG D, ZHANG L. CP-ABE scheme supporting fully fine-grained attribute revocation [J]. Journal of Software, 2012, 23(10): 2805 – 2816. (王鹏翩, 冯登国, 张立武. 一种支持完全细粒度属性撤销的 CP-ABE 方案[J]. 软件学报, 2012, 23(10): 2805 – 2816.)

[8] YU S, WANG C, REN K, *et al.* Achieving secure, scalable and fine-grained data access control in cloud computing [C]// Proceedings of the 2010 INFOCOM. Piscataway: IEEE, 2010: 1 – 9.

[9] HUR J, NOH D K. Attribute-based access control with efficient revocation in data outsourcing systems [J]. IEEE Transactions on Parallel and Distributed Systems, 2011, 22(7): 1214 – 1221.

[10] ATTRAPADUNG N, IMAI H. Conjunctive broadcast and attribute-based encryption [C]// Proceedings of the Third International Conference on Pairing-Based Cryptography — Pairing 2009, LNCS 5671. Berlin: Springer, 2009: 248 – 265

[11] WAN Z, LIU J, DENG R H. HASBE: a hierarchical attribute-based solution for flexible and scalable access control in cloud computing [J]. IEEE Transactions on Information Forensics and Security, 2012, 7(2): 743 – 754.

[12] SAHAI A, SEYALIOGLU H, WATERS B, *et al.* Dynamic credentials and ciphertext delegation for attribute-based encryption [C]// Proceedings of the 32nd Annual Cryptology Conference on Advances in Cryptology — CRYPTO 2012, LNCS 7417. Berlin: Springer, 2012: 199 – 217.

[13] LU R, LIN X, SHI Z, *et al.* PLAM: A privacy-preserving framework for local-area mobile social networks [C]// Proceedings of the INFOCOM 2014. Piscataway: IEEE, 2014: 763 – 771.

[14] NAOR D, NAOR M, LOTSPIECH J. Revocation and tracing schemes for stateless receivers [C]// Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology — CRYPTO 2001, LNCS 2139. Berlin: Springer, 2001: 41 – 62.

(上接第 1566 页)

[3] MOSTAFAVI M A, GOLD C. A global kinetic spatial data structure for a marine simulation [J]. International Journal of Geographical Information Science, 2004, 18(3): 211 – 227.

[4] DUTTON G. Modeling locational uncertainty via hierarchical tessellation [M]. Accuracy of Spatial Databases. London: Taylor & Francis, 1989: 125 – 140.

[5] WANG L, ZHAO X, CAO W, *et al.* A GPU-based algorithm for the generation of spherical Voronoi diagram in QTM mode [J]. International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 2013, XL-4/W2: 45 – 50.

[6] NVIDIA. CUDA C programming guide Version 6.5 [EB/OL]. [2014 – 12 – 01]. http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf.

[7] COOK S. CUDA programming: a developer's guide to parallel computing with GPUs [M]. SU T, LI D, LI S, *et al.* translated. Beijing: China Machine Press, 2014: 12 – 13. (COOK S. GPU 并行程序设计——GPU 编程指南[M]. 苏统华, 李东, 李松泽, 等译. 北京: 机械工业出版社, 2014: 12 – 13.)

[8] HE Y, YE C, LIU Z, *et al.* Parallel simulation and optimization of CUDA-based real-time huge crowd behavior[J]. Journal of Computer Applications, 2012, 32(9): 2466 – 2469. (贺毅辉, 叶晨, 刘志忠, 等. 基于 CUDA 的大规模群体行为实时仿真并行实现及优化[J]. 计算机应用, 2012, 32(9): 2466 – 2469.)

[9] XU S, ZHANG E. CUDA-based parallel visualization of 3D data [J]. CT Theory and Applications, 2011, 20(1): 47 – 54. (徐赛花, 张二华. 基于 CUDA 的三维数据并行可视化[J]. CT 理论与应用研究, 2011, 20(1): 47 – 54.)

[10] DESCHIZEAUX B, BLANC J-Y. Imaging earth's subsurface using CUDA [C/OL]// GPU Gems. 2007. [2014 – 12 – 01]. http://http.developer.nvidia.com/GPUGems3/gpugems3_ch38.html.

[11] SANDERS J, KANDROT E. CUDA by example: an introduction to general-purpose GPU programming [M]. NIE X, *et al.* translated. Beijing: China Machine Press, 2011: 54 – 55, 78. (SANDERS J, KANDROT E. GPU 高性能编程——CUDA 实战[M]. 聂雪军, 等译. 北京: 机械工业出版社, 2011: 54 – 55, 78.)

[12] ZHAN S, ZHU Y, ZHAO K, *et al.* GPU high performance computation -CUDA [M]. Beijing: China Water & Power Press, 2009: 46 – 47, 141 – 142. (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: 46 – 47, 141 – 142.)