基于 GPU 的并行最小生成树算法的设计与实现*

郭绍忠，王 伟，王 磊
（解放军信息工程大学 信息工程学院，郑州 450002）

摘 要： 针对目前并行 Prim 最小生成树算法效率不高的问题，在分析现有并行 Prim 算法的基础上，提出了适于 GPU 架构的压缩邻接表图表示形式，开发了基于 GPU 的 min-reduction 数据并行原语，在 NVIDIA GPU 上设计并实现了基于 Prim 算法思想的并行最小生成树算法。该算法通过使用原语缩短关键步骤的查找时间，从而获得较高效率。实验表明，相对于传统 CPU 实现算法和不使用原语的算法，该算法具有较明显的性能优势。

关键词： 图形处理器；图论；最小生成树；Prim 算法；数据并行原语
中图分类号： TP311.52 **文献标志码：** A **文章编号：** 1001-3695(2011)05-1682-03
doi:10.3969/j.issn.1001-3695.2011.05.025

Design and implementation of GPU-based parallel minimum spanning tree algorithm

GUO Shao-zhong, WANG Wei, WANG Lei
(*Institute of Information Engineering, PLA Information Engineering University, Zhengzhou* 450002, *China*)

Abstract: Considering current parallel Prim's MST algorithms' limited speedup, based on analyzing existing parallel MST algorithms, this paper used compress adjacency list graph representation suited to GPU architecture, developed GPU-based min-reduction data parallel primitive, and designed parallel Prim's MST algorithm on NVIDIA GPU. The algorithm shortened the finding time via using primitives in the key step so achieved higher efficiency. Experimental results show that it obtains evident performance improvement over the traditional CPU implementation and non-primitives GPU implementation.

Key words: GPU; graphic theory; minimum spanning tree (MST); Prim's algorithm; data parallel primitive

最小生成树(MST)是图论中的一个基本概念，定义为：给定一个具有权值映射关系 $w: E \to R$ 的无向图 $G=(V,E)$，求一个具有 $|E'|=|V|-1$ 关系的连通子图 $T=(V,E' \subset E)$，使得 $\sum_{e \in E'} w(e)$ 最小化。MST 问题在网络规划、旅程问题及 VLSI 布局等许多领域发挥重要作用。Prim 算法是最常用的 MST 算法之一，目前对 Prim 算法的研究和使用以串行为主，并行 Prim 算法相关研究较少且加速效果不理想[1~3]，在图规模较大时问题尤其突出。

利用 GPU 进行应用程序加速是近年来的一个重要趋势。GPU 具有计算能力强、价格低廉等优点，CUDA、OpenCL 等新型开发环境的出现大大降低了 GPU 编程的难度，大量利用 GPU 进行应用程序加速的例子纷纷出现。由于自身架构的特殊性，GPU 加速效果良好的程序大多为规则问题，诸如图论、计算几何等非规则问题在 GPU 上的加速效果并不令人满意[4]。MST 属于非规则问题，据笔者所知，利用 GPU 进行 Prim 算法并行化的研究较为少见。使用数据并行原语将不规则问题映射到 GPU 是解决此类问题的一个有效解决方案。本文在 CUDA 环境下使用新开发的数据并行原语对 Prim 算法进行 GPU 并行化设计与实现，取得了较好的加速效果。

1 相关工作

1.1 并行 Prim 算法

由于具有内在的顺序性，Prim 算法的外层 while 循环难以并行化，但是其内层的两个循环步骤可以进行并行求解。两个内层循环为：求当前候选边集中的最小权值边以及加入新点后比较并更新候选边集。到目前为止，对 Prim 算法进行并行化的研究并不多见。文献[1]针对具有共享内存的 SMP 结构提出了一种并行 Prim 算法，取得了至多 2.64 倍的加速效果；文献[2]在 MPI 环境下提出了一种一次迭代增加多个顶点的并行 Prim 算法，取得了较好的加速效果；文献[3]提出了一种借鉴 Prim 算法思想的并行 MST 算法，也取得了一定的加速比。相对于另一种 MST 算法——Borůvka 算法的并行版本，这些算法的共同问题是加速比不高，在图规模较大时尤其突出。

1.2 基于 GPU 的数据并行原语

数据并行原语是指在开发并行程序时常用的基本并行操作，如 scan、sort、reduction 等。在不规则问题的 GPU 并行化过程中，数据并行原语的使用可以提高运行效率。文献[5]提出

收稿日期：2010-10-08；修回日期：2010-11-15 基金项目：国家“863”计划重点资助项目(2009AA012201)；上海市科委重大科技攻关资助项目(08dz501600)

作者简介：郭绍忠(1964-)，女，安徽合肥人，副教授，硕导，主要研究方向为分布式计算(xy_gsz@163.com)；王伟(1983-)，男，山东泰安人，硕士研究生，主要研究方向为分布式计算、GPU 通用计算；王磊(1977-)，男，陕西临潼人，讲师，硕士，主要研究方向为分布式计算、GPU 通用计算.

了 GPU 上的基于 CUDA 的 reduction 原语实现方法；文献 [6] 指出，在 GPU 程序中运用 sort、reduction 等数据并行原语并且将问题的不规则步骤映射到这些原语或其组合之上，可以提高程序的运行效率。文献 [7] 将原语应用于求解诸如图聚类、子图提取等图论问题，获得了一定的加速比。

2　设计与实现

2.1　压缩邻接表的设计

传统的图数据结构包括邻接矩阵与邻接表两种。邻接矩阵适用于稠密图表示，但存储空间 $O(|V|^2)$ 要求太高，而 GPU 设备端的显存空间相对有限，所以不适用于本文的设计；邻接表适用于稀疏图表示，存储空间要求仅为 $O(|V|+|E|)$。在 CUDA 架构中，设备端内存空间都被视为数组，对数组可以进行高效的读写操作。基于以上分析，本文对传统邻接表进行改进，提出一种适用于 GPU 的纯数组形式的压缩邻接表。如图 1 所示，压缩邻接表包含三个数组：顶点数组 V、边数组 E、权值数组 W。V 长度为 $|V|$，每一个元素指向其连接顶点在 E 和 W 中的开始位置；V 和 E 保存相关联的顶点号及相应边的权值，由于 MST 处理的是无向图，E 与 W 长度都为 $2|E|$。

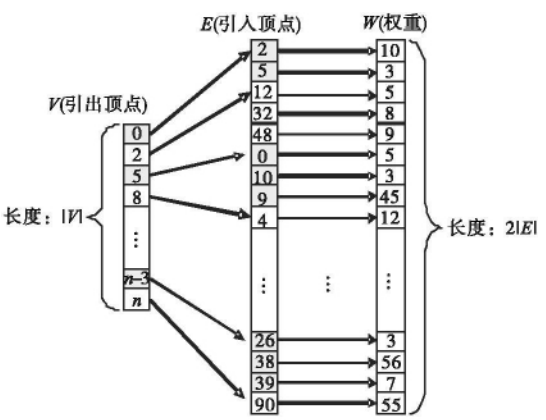

图1　适用于GPU的压缩邻接表

2.2　GPU min-reduction 原语的设计

Reduction 数据并行原语是指对一组数据元素进行某种特定操作（如求和、乘积、最小值等），最终得出一个元素的并行处理过程。本文对文献 [5] 提出的 GPU sum-reduction 原语进行了扩展，得出改进的 GPU min-reduction。

2.2.1　改进与扩展

改进主要体现在三方面：a）待比较数组的元素个数不限于 2 的乘方，可为任意数；b）只需要至多两步 kernel 调用即可得出最终比较结果；c）在比较过程中添加了 index 数组，结果包括最小权值和对应 index。改进的 GPU min-reduction 原语包含全局内存和共享内存比较两种。重要变量及数组的名称及作用如表 1 所示。

2.2.2　全局内存比较

图 2 所示为 GPU min-reduction 的全局内存比较，用于 kernel_1。由于线程对全局内存的存取改为以数组长度为边界，从而不限制其长度为 2 的乘方。以少量性能损失换取更高的灵活性，从而能够更好地适用于 Prim 算法等比较元素长度不固定的特点；由于规定了 max_blocks，可以预估调用 kernel_1 后的临时结果个数，从而使得至多在调用 kernel_2 后即可完成最终比较，便于主机端调用。

表 1　重要变量及数组

名称	作用
V、E、W	存放顶点集、边集、权值集
R_1、R_2、R_3	共同表示 MST 边表集，分别为引出点、引入点、权值
T_1、T_2	临时数组，存放 min-reduction 结果最小权值与索引号
R_3'	R_3 中当前用于比较的权值集部分
C	全局变量，存放最近加入 MST 的边的 R_2 顶点
kernel_1、kernel_2	min-reduction 的两个 CUDA kernel 调用

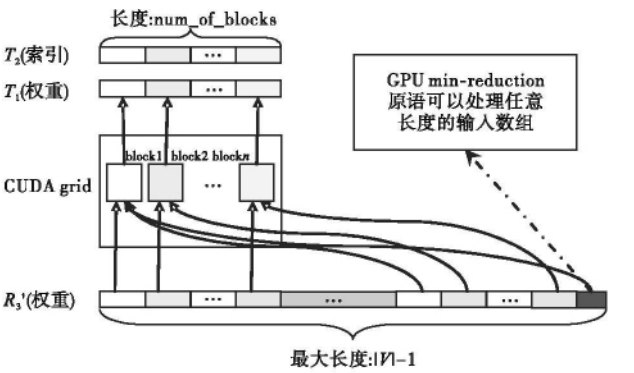

图2　全局内存比较

2.2.3　共享内存比较

图 3 所示为 GPU min-reduction 的共享内存比较，用于 kernel_1 的后半部分和 kernel_2。由于在进行候选边集调整时需要用到最小权值边索引号，在比较全过程中添加了 index 数组。在这里用到了两个技巧：a）相邻 thread 操作相邻元素，避免了共享内存的 bank conflict；b）由于一个 warp 内的运算总是能满足顺序一致性，在最后 32 个线程比较时取消 _syncthreads()，减少同步操作性能损耗。

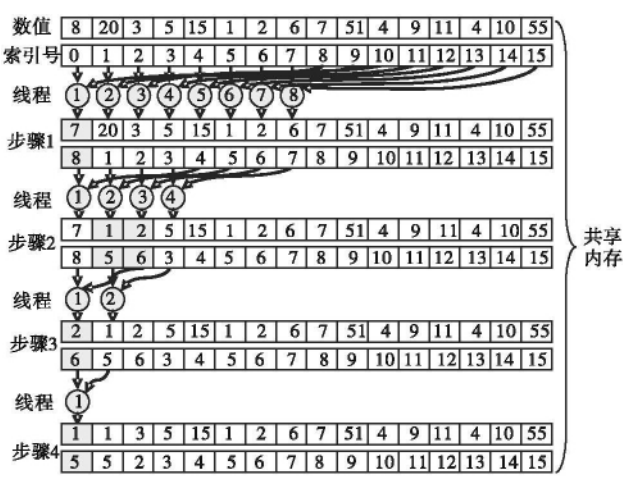

图3　共享内存比较

2.3　算法的设计与实现

Prim 算法是一种贪心算法。它开始选取图中任一顶点作为 MST 根节点，然后通过增加距离当前树最近（即最小权值边）的顶点来逐渐增长树，并将其连接的最小权值边也增加到树中。当所有顶点都增加到 MST 中时，算法结束。本文实现基于 CUDA 2.3 架构，算法输出为 MST 边表。

2.3.1　增长 MST

a）初始化 MST 边表。将传统 Prim 算法中的结果 MST 边

表分拆为三个数组(R_1 , R_2 , R_3) ,便于充分利用 min-reduction 原语的效率。三个数组长度都为 $|V|-1$,相同索引位置分别存放 MST 中一条边的三个属性: 输出顶点、输入顶点、权值。初始化时存放顶点 0 至其他各顶点的边,若无相连边,用 max_weight 填充。

b) 寻找最小权值边。使用 GPU min-reduction 数据并行原语在 R_3 中寻找最小值及其索引号,如图 4 所示。定义两个临时数组 T_1 、T_2 ,分别存放临时结果的最小权值与其索引号。具体过程分为 kernel_1 和 kernel_2 两步。其中, kernel_1 的结果存放于临时数组 T_1 、T_2 ; kernel_2 为可选项,只有 $|R_3'| > \text{max\_block} \times \text{max\_threads\_per\_block}$ 时才进行调用,调用时,与 kernel_1 不同,其操作对象为临时数组 T_1 、T_2 ,最终结果存放于 $T_1[0]$ 与 $T_2[0]$ 。在具体实现中,由于 block 个数有最大限制,使用了静态共享内存,每个 block 分配空间大小为 $|\text{max\_threads\_per\_block}| \times 2$ 。

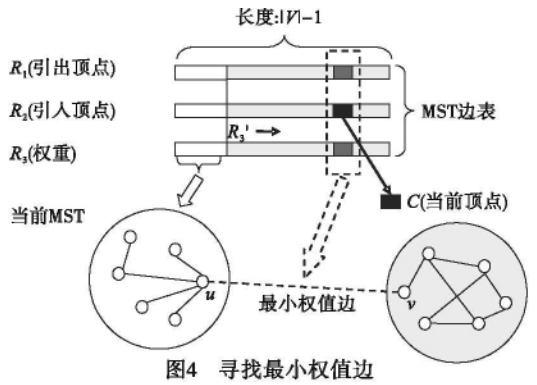

图4 寻找最小权值边

c) 加入新 MST 边。寻找到最小权值边后,本步骤读取 $T_1[0]$ 及 $T_2[0]$,将最小权值边及其连接的顶点一起加入 MST。方法为将其交换到候选 MST 边表最前端,同样涉及到 T_1 、T_2 、T_3 的同时调整。这一操作可能发生在两处: 如果 kernel_2 执行则在 kernel_2 之后,若 kenrel_2 不执行则在 kernel_1 之后。之后需要保存当前顶点号,即 $R_2[T_2[0]]$,为此,需要定义全局变量 C 进行保存。

本步骤的特殊之处在于前两步都可以多线程并行执行,这一步由某一线程执行,防止访存冲突。

2.3.2 降距操作(decrease-distance)

本步骤读取全局变量 C ,得到当前顶点号,通过查询图数据结构中的三个数组 E 、V 、W 计算其与每个顶点的边权值。假设当前比较顶点索引号为 n ,计算结果用 W_n 表示。若 $W_n < R_3[n]$,则调整边表: $R_3[n] = W_n$, $R_1[n] = C$ 。由于边表各组数据元素独立不相关,调整边表可由 GPU 各线程并行执行。需要指出的是,在查询图数据结构计算顶点 C 与顶点 n 之间的边权值时,为避免 warp 内的线程产生分支,最好让所有线程都查询 $C \rightarrow n$ 的边权值而不是 $n \rightarrow C$,这样可以在各线程加载数组 E 和 W 中的 C 连接的边相关值至共享内存和查询共享内存时获得尽可能多的一致性,这对于 CUDA 架构下的运行效率至关重要。

2.3.3 外层循环调用

外层循环调用即将上面两个步骤进行循环操作,直至算法结束。循环次数为固定值 $|V|-2$ 。为避免固定划分 grid 和 block 带来的负载不均衡问题,采取动态分配的方法,即每次调用前都进行 grid 和 block 的重新划分。在 min-reduction 步骤相关 kernel_1 中,为充分展开循环,使用 C++ 模板技术,在可操作元素数小于最大线程数时,视具体情况将 block 内线程数划分为以下各数之一: 256、128、64、32、16、8、4、2、1。

2.4 算法完整描述

算法 1 CUDA_PRIM_MST

a) 从输入文件中读取图数据,初始化数组 E 、V 、W ;

b) 在 CPU 内存中构建 MST 边表数组 R_1 、R_2 、R_3 ,并进行初始化,构建临时变量 C ,初始化为顶点 0;

c) 将 E 、V 、W 、R_1 、R_2 、R_3 、C 传输至 GPU 显存,构建临时数组 T_1 、T_2 ;

d) 根据进行 min-reduction 操作的元素个数,配置 CUDA grid 尺寸;

e) 调用 kernel_1 ,结果写入 T_1 、T_2 ,若 $|R_3'| \leqslant \text{max\_block} \times \text{max\_threads\_per\_block}$,则至步骤 g) ;

f) 如果未得出 min-reduction 最终结果,则调用 kernel_2 ,结果放入 $T_1[0]$ 、$T_2[0]$;

g) 读取 $T_1[0]$, $T_2[0]$,并将其代表的边和顶点加入 MST 边表,同时将新加入边另一顶点写入临时变量 C ;

h) 根据进行比较的元素个数,重新配置 CUDA grid 及 block 尺寸;

i) 读取全局变量 C ,得到当前顶点号,通过查询 E 、V 、W ,计算 C 与各顶点的边权值;

j) 对每个顶点,若新权值 < 旧权值,则调整 R_1 、R_3 对应值;

k) 循环调用步骤 d) ~ j) ,固定 $|V|-2$ 次,直至得出最终结果;

l) 将 GPU 显存内 R_1 、R_2 、R_3 传输回 CPU 内存,作为 MST 边表结果写入输出文件。

3 实验测试

3.1 对比算法

选取两个对比算法: a) boost graph library (BGL) 中的 CPU 串行版本 Prim 算法实现; b) 笔者开发的非原语的普通 CUDA 版本的 Prim 算法实现,与原语版本算法的区别是在寻找最小值时采用普通 GPU 并行化。

3.2 实验平台

Intel Pentium4 3 GHz CPU,2 GB 内存,NVIDIA GeForce GTX260 GPU,896 MB 显存,OS 为 Linux RedHat 5。

3.3 来源数据

选取 GTgraph 图数据生成套件中的 random 生成工具,它生成的数据特点为: 顶点的度数在一个小范围之间,很多顶点具有相似的度数。选取顶点数目为 128 ~ 16384 范围内的数据作为算法输入数据,每项数据的边数一般为顶点数的 10 倍,边权值范围限定为 1 ~ 1k。

3.4 实验结果

实验结果数据如图 5 和 6 所示。分析可知,非原语版本的 CUDA 程序在整体上性能低于 CPU BGL 串行版本程序; 在顶点数量大于 4 096 时,原语版本的 CUDA 程序性能高于其他两个对比程序,并展现出稳定的优势。加速比如图 5 所示。分析可知,在实验数据范围内,非原语版本的 CUDA 程序加速比低于 1; 原语版本的 CUDA 程序加速比在数据量大时大于 1,最高加速比约 2 倍,实际应用中可以考虑采用。实验结果展现了采用数据并行原语方法在提升 GPU 程序性能方面的有效性。

4 结束语

本文在 GPU 上设计和实现了基于 Prim 算(下转第 1702 页)

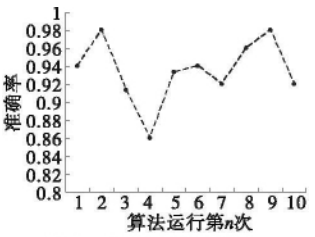

图3 聚类准确率统计

同时，为表现 ABC-Cluster 算法的聚类效果，本文选取 IRIS 数据中的两维特征和三维特征来进行描述，如图 4 和 5 所示，其中点▽表示算法迭代获取的聚类中心，X_1、X_2、X_3 表示该数据集的特征值。

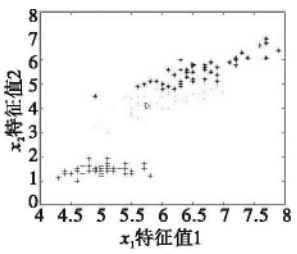

图4 二维特征聚类效果图

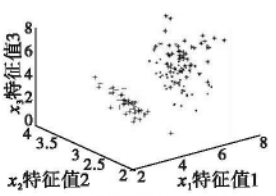

图5 三维特征聚类效果图

从图 4 和 5 可以看出，本文算法能获取较好的聚类中心，从而获得较为准确的分类。

仿真实验表明，本文算法充分发挥了蜂群算法良好的自组织性和收敛性，对聚类问题的聚类中心进行搜索，在搜索过程中引入紧密度指标和分离度指标来评价聚类有效性，可以获取最佳聚类数 K 值，解决了 K-means 算法等划分聚类算法受聚类数 K 影响的缺点，通过聚类准确率检验，虽然算法表现出不太稳定的缺点，但是在实验过程中算法的最差解仍然表现出算法的高有效性，这就从整体上验证了算法的可行性。

4 结束语

本文基于蜂群原理提出了一种新型的划分聚类算法，可以在不用给定聚类数 K 值的前提下，自动搜索最佳聚类数，实现自适应聚类过程。实验表明，算法不仅对最佳聚类数有良好的确定能力，而且与其他经典聚类算法比较也表现出良好的分类能力。此外，对于本文算法(ABC-CA)中适应度函数是最重要的一环，将有待于进一步研究适应度函数(指标函数)的合理性，从而设计更好的适应度函数，以提高算法性能。

参考文献：

[1] HAN Jia-wei, KAMBER M. Data mining concepts and techniques [M]. 2nd ed. [S. l.]: Morgan Kaufmann Publishers, 2006.

[2] DING C, LI Tao. Adaptive dimension reduction using discriminate analysis and K-means clustering [C]//Proc of the 24th International Conference on Machine Learning. New York: ACM Press, 2007: 521-528.

[3] SONG Le, SMOLA A, BORGWARDT K M. A dependence maximization view of clustering [C]//Proc of the 24th International Conference on Machine Learning. New York: ACM Press, 2007: 815-822.

[4] 谢娟英，张琰，谢维信，等. 一种新的密度加权粗糙 K-均值聚类算法 [J]. 山东大学学报: 自然科学版，2010，45(7): 1-6.

[5] 淦文燕，李德毅，王建民. 一种基于数据场的层次聚类方法 [J]. 电子学报，2006，34(2): 258-262.

[6] 张丽，刘希玉，李章全. 基于蚁群算法的聚类优化 [J]. 计算机工程，2010，36(9): 190-194.

[7] 刘靖明，韩丽川，侯立文. 一种新的聚类算法——粒子群聚类算法 [J]. 计算机工程与应用，2005，20(5): 183-185.

[8] BASTURK B, KARABOGA D. An artificial bee colony (ABC) algorithm for numeric function optimization [C]//Proc of IEEE Swarm Intelligence Symposium Indian-apolis. 2006: 651-656.

[9] KARABOGA D, AKAY B B. Artificial bee colony algorithm on training artificial neural networks [C]//Proc of the 15th IEEE Signal Processing and Communications Applications Conference. 2007: 1-4.

[10] KARABOGA D, BASTURK B. Artificial bee colony (ABC) optimization algorithm for solving constrained optimization problems [C]// Proc of the 12nd International Fuzzy Systems Association World Congress on Foundations of Fuzzy Logic and Soft Computing. Berlin: Springer-Verlag, 2007: 789-798.

[11] KARABOGA D, BASTURK B. On the performance of artificial bee colony (ABC) algorithm [J]. Applied Soft Computing, 2008, 8(1): 687-697.

[12] 暴励，曾建潮. 自适应搜索空间的混沌蜂群算法 [J]. 计算机应用研究，2010，27(4): 1330-1334.

(上接第 1684 页) 法思想的 MST 算法，并使用了自己开发的 GPU min-reduction 数据并行原语对算法的关键步骤进行了表示。与 CPU 串行程序和非原语 GPU 程序的对比表明，使用原语的 GPU 程序明显提高了算法性能。笔者认为，在使用 GPU 设备解决图论、计算几何等非规则问题时，数据并行原语的使用对于提高性能可以起到关键的帮助作用。在下一步工作中，将研究 GPU 上数据并行原语在其他图论算法中的应用。

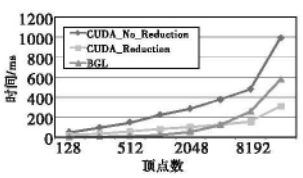

图5 算法运行时间对比

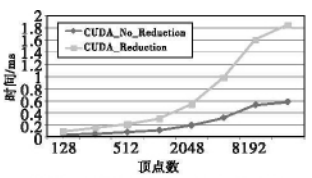

图6 两种CUDA Prim算法加速比对比

参考文献：

[1] ROHIT S, ARUN N, SHANKAR B. A new parallel algorithm for minimum spanning tree problem [C]//Proc of International Conference on High Performance Computing. 2009: 1-5.

[2] EKATERINA G, LAXMIKANT V K. Parallel Prim's algorithm on dense graphs with a novel extension [R]. Urbana: University of Illinois at Urbana-Champaign, 2007.

[3] BADER D A, CONG Guo-jing. Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs [J]. Journal of Parallel and Distributed Computing, 2006, 66(11): 1366-1378.

[4] VUDUCY R, CHANDRAMOWLISHWARANY A, CHOI J, *et al*. On the limits of GPU acceleration [EB/OL]. (2010). http://www.usenix.org/event/hotpar10/tech/full_papers/Vuduc.pdf.

[5] HARRIS M. Optimizing parallel reduction in CUDA [EB/OL]. (2007). http://developer.download.nvidia.com/compute/cuda/1_1/Website/Data-Parallel Algorithms.html.

[6] VINEET V, HARISH P, PATIDAR S, *et al*. Fast minimum spanning tree for large graphs on the GPU [C]//Proc of Conference on High Performance Graphics. New York: ACM Press, 2009: 167-171.

[7] BULLUC A. Linear algebraic primitives for parallel computing on large graphs [D]. Santa Barbara: University of California, 2010.