第 38 卷 第 2 期
Vol.38 No.2

计 算 机 工 程
Computer Engineering

2012 年 1 月
January 2012

• 软件技术与数据库 •　　文章编号：1000—3428(2012)02—0042—03　　文献标识码：A　　中图分类号：TP311.52

基于 GPU 的单源最短路径算法设计与实现

郭绍忠[1]，王 伟[1]，周 刚[1]，胡 艳[2]

(1. 解放军信息工程大学信息工程学院，郑州 450002；2. 卫星导航定位总站，北京 100094)

摘 要： 针对目前图形处理器(GPU)上的动态数据处理问题，在分析现有并行单源最短路径(SSSP)算法的基础上，对 GPU 上的 Moore SSSP 算法进行并行化设计与实现。搜索时，综合应用层次化任务分配、层次化工作队列、层次化 Kernel 调用等策略。在不同类型图数据上进行实验测试，实验结果表明，该算法能有效减少空线程开销、访存开销以及同步时间。

关键词： 图形处理器；图论；动态数据；单源最短路径；计算统一设备架构

Design and Implementation of GPU-based Single Source Shortest Path Algorithm

GUO Shao-zhong[1], WANG Wei[1], ZHOU Gang[1], HU Yan[2]

(1. Institute of Information Engineering, PLA Information Engineering University, Zhengzhou 450002, China;
2. Satellite Tracking and Locating Terminal Station, Beijing 100094, China)

【Abstract】 Based on analyzing existing parallel Single Source Shortest Path(SSSP) algorithm, aiming at the problem of dynamic data processing on Graphic Processor Unit(GPU), this paper designs and implements parallel Moore SSSP algorithm on GPU. The algorithm applies strategies like hierarchical task arrangement, hierarchical work queue and hierarchical Kernel invokes in key step. Experimental results indicate that the algorithm can reduce idle thread cost, memory access cost and synchronizing cost.

【Key words】 Graphic Processor Unit(GPU); graphic theory; dynamic data; Single Source Shortest Path(SSSP); Compute Unified Device Architecture(CUDA)

DOI: 10.3969/j.issn.1000-3428.2012.02.013

1 概述

单源最短路径(Single Source Shortest Path, SSSP)问题是图论的一个基本概念，定义为：给定一个具有权值映射关系 $w: E\rightarrow R$ 的图 $G=(V,E)$，从顶点 u 到顶点 v 的最短路径为所有从 u 到 v 路径中权值最小的路径，即 $\delta(u,v)=\min\{w(p): p$ 是从 u 到 v 的路径$\}$，求源点 s 到 G 中其他所有顶点的最短路径权值。SSSP 问题在道路交通、计算机网络等许多领域具有重要应用。

经典串行 SSSP 算法包括 Dijkstra 算法、Moore 算法等，传统并行 SSSP 算法大多为 Dijkstra 算法的并行化，但是在传统的 PRAM 并行计算模型下还未出现高效的 SSSP 并行算法实现[1]。

利用图形处理器(Graphic Processor Unit, GPU)进行通用计算是近年来并行计算领域的一个研究热点，各种基于 GPU 加速的应用研究纷纷涌现[2]。相比而言，图论算法的 GPU 实现较少且加速效果不甚理想，原因在于图论算法具有不规则计算、数据动态产生等特点，使用 GPU 进行并行化的难度较大。文献[3]针对 GPU 上动态数据处理问题提出一种综合层次化队列和层次化调用的新方法，取得了良好效果。本文应用此方法在计算统一设备架构(Compute Unified Device Architecture, CUDA)平台下针对 Moore SSSP 算法进行了 GPU 并行化初步研究。

2 相关工作

2.1 并行 SSSP 算法

文献[4]总结了已出现的各种并行 SSSP 算法，并详细比较了性能。文献[1]指出，基于传统 PRAM 并行计算模型的高效 SSSP 算法尚未出现；目前，利用 GPU 对 SSSP 算法进行并行化的研究很少。文献[5]对这个问题进行了开拓性研究，提出了一种适用于 GPU 架构的算法实现(以下称 IIIT 算法)，取得了一定加速比。文献[6]对 IIIT 算法的实现进行了改进，但是其改进主要是扩展了兼容性，对算法性能并无实质改进，目前尚未出现 Moore SSSP 算法的 GPU 并行化研究。

2.2 动态数据问题的处理

动态数据问题是指在某些应用(如图论算法)中的工作数据是在计算过程中动态产生的，在并行计算中如果采用静态工作分配将导致严重的负载不平衡。传统并行计算采用动态工作分配解决此问题，工作队列是常用的一种动态分配策略。GPU 通用计算中同样存在动态数据问题，但是由于其硬件架构的特殊性，需要进行针对性的特殊处理。文献[3]对此进行了初步研究，提出了一种综合层次化队列和层次化调用的新方法。其主要思想为：针对 GPU 架构中的层次化线程结构和存储模型，将工作队列分为 Grid、Block、Warp 3 级，并由此提出 3 种对应调用策略，从而有效减少访存和 Kernel 调用时间，其实验结果验证了这种方法的有效性。

3 算法的设计与实现

在 GPU 上使用层次化队列和层次化调用的方法对 Moore 算法进行了并行化设计与实现。

3.1 串行 Moore 算法描述

Moore 算法是经典 SSSP 算法之一，最早由文献[7]提出，

作者简介： 郭绍忠(1964－)，女，副教授，主研方向：分布式计算；王 伟，硕士研究生；周 刚，副教授、博士；胡 艳，硕士

收稿日期： 2011-07-13　**E-mail：** xy_gsz@163.com

适用于权值为非负的有向或无向图，描述为：设源点 $s \in V$，使用数组 D 存储从 s 到其他各顶点最短路径长度，使用队列 Q 存储待搜索顶点；初始化时置 $D[s]=0$，$D[v]=\infty$，其中，$v \neq s$ 且 $v \in V$；首次执行时 Q 只包含源点 s，之后每次只要队列非空，就将队头顶点 u 出队并作为本次搜索的顶点，然后检查 u 的所有引出边 $(u,v) \in E$，若 $D[u]+w(u,v)<D[v]$，则此时找到一条从 s 至 v 的更短路径，置 $D[v]=D[u]+w(u,v)$，若 v 不在 Q 中，则将 v 入队；重复以上步骤，Q 为空时算法结束。

3.2　基于 GPU 的并行 Moore 算法的设计

3.2.1　并行化时机选择

考察串行 Moore 算法可知，共有 3 处可并行化之处：(1)数组 D 的并行初始化；(2)主过程中的队列顶点并行搜索；(3)某一顶点在搜索时并行检查所有引出边。其中，第(1)处并行化较容易实现，由于采用了近似广度优先搜索的分层策略，第(2)处比第(3)处能产生更大的加速效果，因此选择前两处进行并行化设计。

3.2.2　并行化具体策略

搜索过程是 Moore 算法的主要执行步骤，对此在任务分配、工作队列和 Kernel 调用等 3 个关键之处应用层次化策略，实现高效并行化。

层次化任务分配：算法全过程中的队列 Q 的顶点可以看作是一个近似广度优先搜索的顶点序列，因此，将队列 Q 的全部顶点按照其搜索顺序分成多个层次，每一个层次对应上个层次在 D 数组中有更新的顶点，并进行一次任务分配，如图 1 所示。这样分配的结果是除了开始几个层次，后面大部分层次都有足够多的任务可以分配，这个策略非常适合 GPU 架构的大规模线程并行执行的特点。

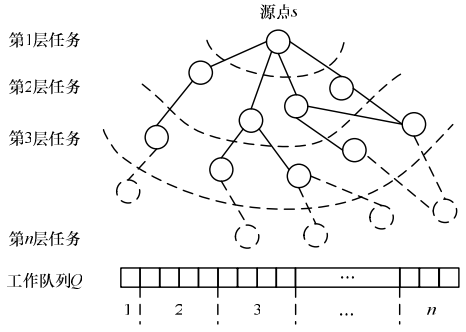

图 1　层次化任务分配

层次化工作队列：在每一次的任务分配中，多个线程同时对队列进行操作，为避免同时入队带来的锁开销，也为了充分利用 GPU 的层次化存储模型，将队列 Q 分为 3 个层次，即 Grid 层次(以下简称 G-Q)、Block 层次(以下简称 B-Q)、Warp 层次(以下简称 W-Q)，如图 2 所示。

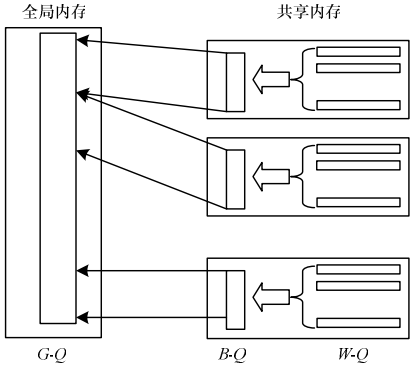

图 2　层次化工作队列

3 个层次的队列分别位于全局内存、共享内存、共享内存，队列内容是顺次包含的关系，访问速度是逐步递增的关系。在每一次分配开始时，由 1 号线程将所有任务分配至各 W-Q，在每一次执行结束后，由 1 号线程将新任务复制至 B-Q 或 G-Q。

层次化 Kernel 调用：GPU 架构中的 Kernel 调用是一个开销很大的操作，应尽量避免，因此，每个层次的任务分配并不对应于一次 Kernel 调用。将 Kernel 调用分为 3 类：单 Block 调用，多 Block 调用，无限制 Block 调用，前 2 种调用可以跨越多次任务分配。其中，单 Block 调用中多次任务分配间采用共享内存的_Syncthreads()函数同步；多 Block 调用中多次任务分配之间采用文献[8]的全局内存同步技巧。3 级调用在下文中分别称为 $K1$、$K2$、$K3$。

3.2.3　并行化性能比较分析

将上述算法(以下称 GPU Moore 算法)与 IIIT 算法进行综合比较，可以得出其理论上的性能改进之处：(1)线程划分方面，IIIT 算法在整个算法中将线程数固定为图顶点数，而 GPU Moore 算法在不同层次的 Kernel 调用中采用不同数量的线程划分，可有效减少空线程的额外开销；(2)访存开销方面，IIIT 算法使用掩码方法确定每个线程的任务分配，但是其掩码数组位于全局内存，每次分配都要访问全局内存，开销较大，而 GPU Moore 算法设置了 3 级队列，很大一部分的访存位于共享内存，可有效节约访存开销；(3)同步开销方面，IIIT 算法的每次任务分配都要对应于一次 Kernel 调用，利用开销极大的重调用进行同步，而 GPU Moore 算法的多次连续任务分配可以共用一次 Kernel 调用，利用 GPU 内存进行同步，从而明显提高同步性能。

3.3　基于 GPU 的并行 Moore 算法实现

在完成算法设计的基础上，在 Nvidia GPU 上使用 CUDA 平台进行了算法实现。

3.3.1　GPU 上的图数据结构

CUDA 平台上使用较多的图数据结构是压缩邻接表，既保证了存储效率，又利用了 CUDA 平台对数组的高效访问。为充分利用其内置向量的效率，对压缩邻接表进行改进，将一维数组改进为二维数组，如图 3 所示。改进后的压缩邻接表包含 V 和 E 2 个 int2 向量类型的数组，int2 向量类型是 CUDA 的内建向量类型，可高效读取。V 的长度为 $|V|$，其中，每个元素都保存其连接顶点的数目并指向在 E 中的开始位置。由于选择无向图进行实现，因此 E 的长度为 $2|E|$，其中，每个元素保存相关联的顶点号及相应边的权值。

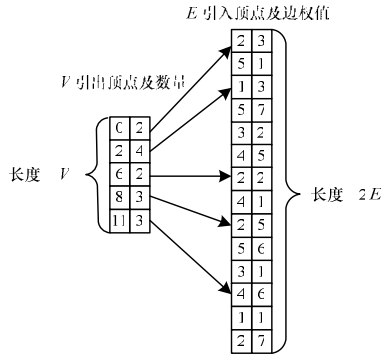

图 3　改进后的压缩邻接表

3.3.2　重要变量及函数

程序中用到的重要变量和函数如表 1 所示，按功能分为 3 个方面：用于图表示，用于工作任务分配，用于结果存储。

表 1 重要变量及功能

名称	类型	功能
V, *E*	int2 型向量数组	顶点集、边集、权值集
G-Q, *B-Q*, *W-Q*, *G-Q'*, *B-Q'*	一维数组	Grid、Block、Warp 级别的工作队列及中间结果
Distance	一维数组	存储最短路径中间结果及最终结果
Num	int 型变量	用于表示当前待分配任务数量，为 0 时算法结束
*Limit*1, *Limit*2	常量	*K*1 和 *K*2 任务容量限制

3.3.3 程序完整流程描述

程序 CUDA_MOORE_SSSP

(1)从输入文件中读取图数据，初始化数组 *V*, *E*；

(2)在 CPU 内存中建立变量 *Num*，数组 *Distance*；

(3)初始化 CPU 内存变量：*Num*=1，*Distance*[*s*]=0，*Distance*[*v*]=∞，其中，*v*≠*s* 且 *v* ∈ *V*；

(4)将 *V*、*E*、*Num*、*Distance* 从 CPU 内存传输至 GPU 显存；

(5)在 GPU 显存中初始化各级别的工作队列 *G-Q*, *B-Q*, *W-Q*, *G-Q'*, *B-Q'*；

(6)根据 *Num* 配置 CUDA GRID 维度及尺寸，并调用相应的 CUDA Kernel 函数；

(7)在每次 Kernel 执行中，首先根据 *G-Q'*或 *B-Q'*决定 3 级工作队列的任务分配，然后执行主搜索过程，之后将产生的下一层次任务写入 *G-Q'*或 *B-Q'*；

(8)若任务数量<*Limit*1，则在 *K*1 中执行多层搜索，若任务数量>*Limit*1，则写入 *Num*，并传回 CPU 内存，结束 *K*1，转步骤(5)；

(9)若任务数量>*Limit*1 且<*Limit*2，则在 *K*2 中执行多层搜索，若任务数量>*Limit*2，则写入 *Num*，并传回 CPU 内存，结束 *K*2，转步骤(5)；

(10)若任务数量>*Limit*2，则在 *K*3 中执行一次搜索，结束 *K*3，转步骤(5)；

(11)若 *Num*=0，则结束 CUDA Kernel 函数调用；

(12)将 GPU 显存内的 *Distance* 数组传回 CPU 内存，作为算法结果写入输出文件。

4 实验测试及分析

4.1 实验环境及实验数据

实验环境为：Intel Pentium 4 3 GHz CPU，2 GB 内存，NVIDIA GeForce GTX260 GPU，896M 显存，OS 为 Linux RedHat 5，CUDA 2.3 平台。实验数据来自 GTgraph 套件的 Random 与 SSCA#2 2 种图生成工具，图数据的顶点数目限定为 128~16 384，边权值范围限定为 1~1 000。Random 数据的边数设定为顶点数的 10 倍。在生成数据后，需要对结果进行一定的格式转换。对比算法为文献[5]中的 IIIT 算法实现。

4.2 实验结果

实验结果如图 4 和图 5 所示。分析可知，与 IIIT 算法相比，GPU Moore 算法在 SSCA#2 数据上取得了良好效果，平均加速比为 2.91，最大加速比为 4.42(|*V*|=2 048 时)；在 Random 数据上取得了一定效果，平均加速比为 1.38，最大加速比为 1.58(|*V*|=512 时)。实验结果验证了本文方法在解决 GPU 上动态数据问题方面的有效性。

对于图 5 中|*V*|=256、512 这 2 处加速比较低的问题，经详细分析，原因为这 2 项数据在算法执行时的任务层次数较少(都只有 3 层)，使得层次化 Kernel 调用在同步方面的优势无法体现，导致执行效率不高。对于导致不同类型图数据的加速效果不同的根本原因，有待深入研究。

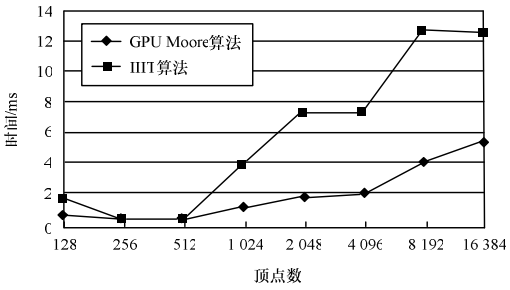

图 4 SSCA#2 数据上的算法时间对比

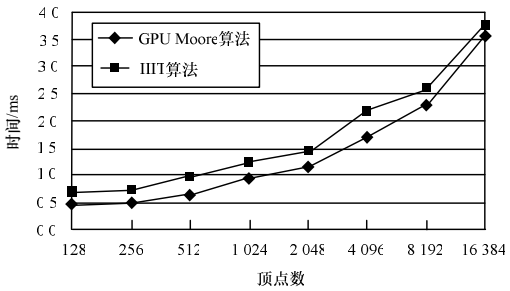

图 5 Random 数据上的算法时间对比

5 结束语

本文使用层次化工作队列和层次化 Kernel 调用的方法对 Moore SSSP 算法在 GPU 上进行了实现，并在不同类型图数据上进行了实验测试，实验结果取得了较好的加速比，验证了上述方法的有效性。此方法在解决 GPU 上动态数据的问题上具有一定的通用性，有助于提高图论等算法的性能。下一步工作将对更多类型及更大规模的图数据进行测试，从图数据特点的角度研究影响加速比的关键因素，探究 GPU Moore 算法的最佳适用范围。

参考文献

[1] Meyer U, Sanders P. Δ-Stepping: A Parallel Single Source Shortest Path Algorithm[C]//Proc. of the 6th Annual European Symposium on Algorithms. London, UK: Springer-Verlag, 1998: 393-404.

[2] 程 豪, 张云泉, 张先轶, 等. CPU-GPU 并行矩阵乘法的实现与性能分析[J]. 计算机工程, 2010, 36(13): 24-26.

[3] Luo Lijuan, Wong M, Hwu W M. An Effective GPU Implementation of Breadth First Search[C]//Proc. of the 47th Design Automation Conference. New York, USA: ACM Press, 2010: 52-55.

[4] Meyer U. Design and Analysis of Sequential and Parallel Single Source Shortest Paths Algorithms[D]. Saarbrucken, Germany: Universität des Saarlandes, 2002.

[5] Harish P, Narayanan P J. Accelerating Large Graph Algorithms on the GPU Using CUDA[C]//Proc. of the 14th International Conference on High Performance Computing. Berlin, Germany: Springer-Verlag, 2007: 197-208.

[6] Pedro J M, Robert T, Antonio G. CUDA Solutions for the SSSP Problem[C]//Proc. of the 9th International Conference on Computational Science. Berlin, Germany: Springer-Verlag, 2009: 904-913.

[7] Moore E F. The Shortest Path Through a Maze[C]//Proc. of the International Symposium on Theory of Switching. Cambridge, UK: Harvard University Press, 1959: 285-292.

[8] Xiao Shucai, Feng Wu-Chun. Inter-block GPU Communication via Fast Barrier Synchronization[R]. Dept. of Computer Science, Virginia Tech, Tech. Rep.: TR-09-19, 2010.

编辑 顾逸斐