一种高效直方图生成算法在GPU上的实现

狄 鹏 胡长军 李建江
(北京科技大学计算机与通信工程学院 北京 100083)

摘 要 直方图生成算法(Histogram Generation)是一种顺序的非规则数据依赖的循环运算,已在许多领域被广泛应用。但是,由于非规则的内存访问,使得多线程对共享内存访问会产生很多存储体冲突(Bank Conflict),从而阻碍并行效率。如何在并行处理器平台,特别是当前最先进的图像处理单元(Graphic Processing Unit,GPU)实现高效的直方图生成算法是很有研究价值的。为了减少直方图生成过程中的存储体冲突,通过内存填充技术,将多线程的共享内存访问均匀地分散到各个存储体,可以大幅减少直方图生成算法在GPU上的内存访问延时。同时,通过提出有效可靠的近似最优配置搜索模型,可以指导用户配置GPU执行参数,以获得更高的性能。经实验验证,在实际应用中,改良后的算法比原有算法性能提高了42%~88%。

关键词 图像处理单元,计算设备统一构架,直方图生成,内存填充
中图法分类号 TP312 **文献标识码** A

Efficient Method for Histogram Generation on GPU

DI Peng HU Chang-jun LI Jian-jiang
(School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China)

Abstract Histogram generation is an inherently sequential loop computation with irregular data-dependence, which has a full range of applications in diverse fields. However, the presence of irregular memory access in histogram loop nest poses an obstacle to its paralleled execution using a massive number of fine-grained threads due to access latency leaded by bank conflicts. It is non-trivial to accelerate histogram generation algorithm on parallel platform, particularly on the state-of-the-art parallel platform, graphics processing unit (GPU). For reducing bank conflicts, utilization of padding technique can evenly distribute shared memory access of multiple threads to different banks and largely exploit GPU's potential on accelerating histogram generation. Moreover, efficient near-optimal configuration search model can guide programmers choosing appropriate GPU execution parameters for higher performance. Experimental result demonstrates the improved histogram generation algorithm has approximate 42% to 88% speedups than traditional histogram generation algorithm on GPU.

Keywords GPU, CUDA, Histogram, Padding

1 引言

直方图生成算法是一种图形学常用计算,它需要统计每一个图像像素颜色,并将其记录在不连续的分类项(Bin)中。这种算法需要评估像素颜色变量的分布可能性,并且需要经常分析其分布密度。颜色直方图被广泛应用于许多领域,特别是图像处理和模式识别领域[1]。直方图生成属于计算密集型,在当前的计算机发展中,单线程串行程序已经无法满足其对实际应用的需求。因此,需要利用并行处理器实现直方图的生成。

在当前的高性能计算领域中,由NVIDIA所引导的多核处理器技术是一大研究热点[2]。NVIDIA的图像处理单元(Graphic Processing Unit,GPU)作为一种专为计算机密集型、高度并行化应用而设计的高性能计算平台,其运算能力和存储器带宽上相对于中央处理单元(Central Processing Unit,CPU)有明显的优势。通过计算统一设备构架(Compute Unified Device Architecture,CUDA)[3],GPU可以在单指令多数据(Single Instruction Multiple Data,SIMD)编程模型下发挥其强大的计算能力。因此,高效的直方图生成需要依托这些并行处理器平台。

直方图生成算法的并行化,其难点在于如何减少数据分布的不规则所产生的写冲突。目前,CUDA软件开发包(Software Development Kit,SDK)实现了32-Bin和256-Bin两种基础的直方图生成算法[4]。两者的实现主要依靠于对直方图进行多次复制,即为每个线程或每个Warp生成一个私有直方图副本(Sub-histogram),以减少写冲突。Shams等将这种基础直方图生成并行算法扩展到任意数量的Bin中[5]。最新的研究利用了GPU排序算法,这种成熟高效的算法在

到稿日期:2011-04-15 返修日期:2011-07-14 本文受教育部科学技术研究重点项目(108008),国家"863"计划资助项目(2008AA01Z109)资助。
狄 鹏(1982-),男,博士生,主要研究方向为高性能计算,E-mail:mhyxxcmz@163.com;胡长军(1963-),男,教授,博士生导师,主要研究方向为高性能计算;李建江(1971-),男,博士,副教授,主要研究方向为高性能计算。

生成直方图前引入预排序的方法来加速直方图的生成[6]。

但是，这些工作并没有很好地考虑到 GPU 体系结构下共享内存(Shared Memory)的特性，从而避免 Bank 冲突所带来的内存访问延时。同时，这些工作也没有很好地参数化直方图生成算法，从而建立 GPU 执行参数模型，用户需要利用经验配置 GPU 执行参数。

本文主要引入内存填充技术来减少 GPU 共享内存的 Bank 冲突所带来的内存访问延时。这种方法，尤其在退化分布(Degenerate Distribution)下能减少或消除存储体冲突(Bank Conflict)，大幅提高性能。非退化分布(Non-degenerate Distribution)也能普遍提高性能。同时，通过近似最优配置搜索模型，指导发现最优的 GPU 配置。

2 GPU 体系结构和 CUDA 概述

本节简略介绍 GPU 的内存层次结构和 CUDA 线程组织形式，这些要素将影响直方图生成算法在 GPU 上的实现。

GPU 构建以一个可伸缩的多线程流处理器(Streaming Multiprocessor，SM)阵列为中心，每个多处理器(SM)拥有多个标量处理器(Streaming Processor，SP)。GPU 的内存组成主要有两部分：全局内存(Global Memory)和共享内存(Shared Memory)(其他 GPU 内存，本文介绍的算法不涉及)。SM 之间通过全局内存同步数据，但是全局内存的访问延时高达 400 到 600 时钟周期。为了减小全局内存的访问延时，可以通过结合访问(Coalescing)的方式使带宽利用最大化。但是，即使如此，全局内存依然比共享内存访问延时大得多。共享内存属于每个 SM，同一 SM 上的 SP 可以通过共享内存来同步数据。共享存储器被划分为多个大小相等的存储器模块，称为存储体(Bank)。多个线程同时对不同的 Bank 发送读写请求，可同步实现。如果多个线程同时访问同一存储体，则产生存储体冲突，多个线程间必须顺序完成读写操作。共享内存的访问延时非常小，只要线程间不存在存储体冲突，访问共享存储器的速度就与访问寄存器一样快[3]。可见，多线程合理地共享内存调度机制，对于减少存储体冲突，提高性能是十分必要的。

在 CUDA 编程模型中，GPU 作为 CPU 的协处理器。CPU 通过调用 GPU 内核函数(kernel)来给 GPU 分配计算任务。GPU 产生若干线程，并通过 SIMD 模式运行实现其计算。GPU 的线程按照 3 层进行组织。首先，整个 kernel 产生的线程都属于它自身唯一的 grid。其次，grid 被划分为许多个线程块(Threads Block)，这些线程块被分配给 SM。一个 SM 可以计算很多线程块，但是一个线程块只能被分给一个 SM。同一个线程块内的线程可以同步并共享位于 SM 的共享内存数据。最后，同一线程块内的线程按照 32 个一组组织成 warp。warp 是 GPU 的最小线程调度单元，也就是说 GPU 的流处理器是以 warp 为单位完成操作的，同时访问共享内存的线程不会多于 32 个(Fermi 之前的 NVIDIA GPU，是以半 warp 为单位进行调度，同时访问线程不会多于 16)[7]。

3 高效的直方图生成算法

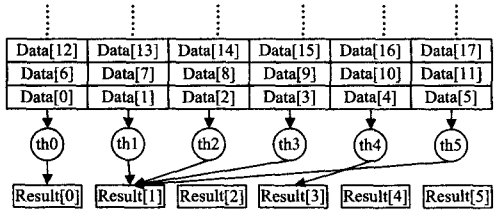
3.1 传统直方图生成算法

如图 1 所示，利用单线程串行程序计算一个直方图生成是相对简单的。

```
for (int i=0; i< BIN_COUNT; i++){
        Result[i]=0;
}for (int i=0; i< data_N; i++){
        bin=Data[i] * (bins-1); //数据标准化
        Result[bin]++;   //对不同的 Bin 进行投票
}
```

图 1 直方图生成算法串行程序(伪码)

如果使用 x 个线程并行实现 y-Bins 的直方图生成，如图 2 所示，6 个线程(th0 到 th5)并行实现 6-Bins 的直方图生成。线程通过迭代完成对输入数据的遍历，然后对数组 Result 进行投票更新(Vote)。这种投票更新是一种非规则的数据依赖，线程对哪个 Result 地址投票，取决于图像输入数组 Data 的值。多个线程可能对 Result 的同一地址进行投票更新，如 Result[1]，这样就产生了写冲突(Write Collision)。这种同时访问所造成的写冲突，CUDA 并不能保证其访问的正确性。在计算能力 1.0 (capability 1.0)的 CUDA 版本中，NVIDIA 并不提供原子操作，因此这种写操作只能通过信号量的方法来保证其计算的正确性。即使在之后更高的版本中，原子操作也严重影响到 GPU 的计算效率。可见，在传统直方图生成算法中，其性能和图像颜色数据分布有很大的关系。数据的集中分布，尤其是退化分布，使所有投票更新发生在同一 Result 地址，它会产生大量的写冲突，从而造成性能的大幅下降。如何减少投票更新所造成的写冲突，是实现高效直方图生成算法的关键。



6 个线程更新 6-Bins 直方图

图 2 多线程直方图生成图示

3.2 利用共享内存建立子直方图

经过研究，可以发现传统的直方图生成需要对 GPU 进行原子操作。Shams 等引入了虚拟原子更新的方式去减小原子操作对计算带来的延时，但这种方法仍然不是一种最理想的直方图生成算法[5]。相对来说，零写冲突更新算法是一种更高效的算法，它可以忽视数据分布对性能的影响。零写冲突更新算法是针对每一个 thread 在全局内存建立一个私有子直方图，每个线程对自己的私有子直方图进行数据更新，最后将所有子直方图数据结合成一个直方图。

这种算法的优点是，对于任何数量的 Bins，都可以通过一次遍历输入数据完成直方图生成。并且，由于这种算法的各个线程在投票更新阶段只访问自身的子直方图，不存在对同一内存地址的多线程并行更新访问，并且没有线程同步需求，因此算法的性能和图像输入数据分布本身无关。

由于直方图的投票更新都是在全局内存完成的，使得每次读写操作的延时较大。同时内存访问相对分散，CUDA 很难实现结合访问全局内存来减小延时。为了避免过多的非结合(Non-coalescing)全局内存更新，使用共享内存作为子直方

图的存储载体。

使用共享内存建立子直方图的方法，可以极大地减少投票更新的操作延时，并以结合的方式在共享内存和全局内存进行数据交换，高效利用内存带宽，加速数据存取。但是，这种算法也存在两个缺点：一方面，由于共享内存的存储体是一个内存访问的瓶颈，因此有可能产生大量的存储体冲突，导致性能下降；另一方面，共享内存相对较小，如果 kernel 产生很多活动线程，GPU 很难对所有线程分配足够的共享内存空间以储存完整的子直方图。

3.3 填充技术解决存储体冲突

如果同一 warp 的两个或多个线程同时访问同一共享内存存储体，将会产生存储体冲突，并导致这些线程间的顺序执行，如图 3 所示。

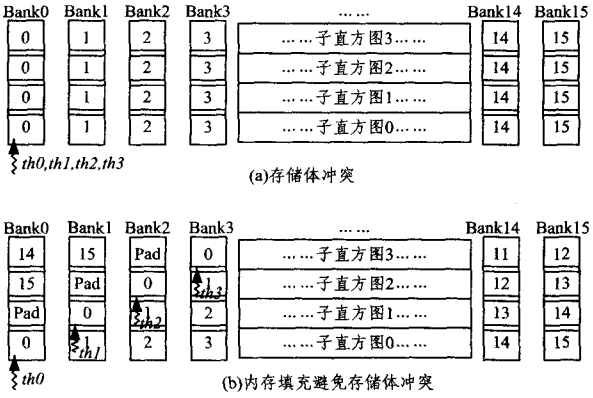


图 3 退化分布产生存储体冲突及内存填充避免

一个退化分布的图像，其在投票更新时，达到直方图生成算法最坏的情况，所有的线程都将投票更新同一 Bin。在这种情况下，所产生的存储体冲突也达到最大。如图 3(a)所示，4 个子直方图的起始地址位于同一存储体 Bank0，退化分布要求 4 个线程同时对自己的子直方图起始位置进行投票更新，这样就产生了存储体冲突。内存填充算法此时可以发挥重大作用，可以完全避免存储体冲突。如图 3(b)所示，通过填充 Pad，使子直方图起始地址错位。这时再发生同时访问，起始地址将不存在于同一存储体内，避免了存储体冲突。即使在大多数普通图片中，如标准 Lenna 灰阶图像，这种内存填充算法依然能提高效率。

3.4 近似最优配置模型

直方图生成算法的性能主要依靠一些 GPU 硬件特性，如设备特征、图像数据分布、直方图 Bin 的数目和投票更新前的预运算多少。只有合适并充分地利用资源，才能最大效力发挥 GPU 的运算能力[8]。这一节主要提出一个准则，用于指导直方图的算法在 GPU 上的配置，以发挥 GPU 的最大计算能力。GPU 配置主要包括两个参数：每个 block 中的线程数 K 和 grid 中的 block 数 L。

3.4.1 隐藏流处理器指令延时

每个块的线程数目应该根据隐藏 SP 指令延迟的线程最小数目选取。一般来说，GPU 的流处理器会存在因为寄存器读取而产生的延时。如果有足够的活动线程，这种延时可以完全被流水线操作所隐藏。文献[9,10]中指出，对于计算能力 1.x 的 GPU，每个 SM 中需要多于 192 个活动线程才可以完全隐藏 SP 指令延时。这就是说，如果 SM 中拥有 3 个活动的线程块，那么对于 Tesla C1060，则需要每个块多于 64 个线程，才能屏蔽延时。

3.4.2 SM 中的最大活动线程块数目

SM 同时最多可以执行 B 个线程块，B 的大小由 GPU 硬件限制和 kernel 程序两方面决定，具体计算通过表 1 所列完成。直方图生成是非寄存器瓶颈的计算应用，在一般情况下，B 的选取由共享内存大小限制，即 $B_S \leqslant B_R$，故可以暂时忽略寄存器的影响。如果每个 block 有 K 个线程，即其在共享内存中拥有 K 个子直方图，且直方图大小选用 T 个 bins，则对于 C1060，每个线程块所需共享内存大小为

$$S_{TB} = K \times T \times 4\text{bytes} \tag{1}$$

如果使用内存填充优化，式(1)改进为

$$S_{TB} = (K \times T + \left\lfloor \frac{K \times T}{Bank\_NUM} \right\rfloor) \times 4\text{bytes} \tag{2}$$

最大 SM 共存线程块数目可以通过表 1 得到。

表 1 Tesla C1060 设备参数及最大活动线程块计算

参数描述	参数命名和参数值
最高计算能力(+)	1.3
SM 数(+)	SM_NUM=30
每个 SM 的 SP 数(+)	SP_NUM=8
存储体数目(+)	Bank_NUM=16
Warp 大小(+)	W=32
每个 SM 最大活动 Warp 数(+)	$W_{SM}=32$
每个 SM 最大活动线程数(+)	$T_{SM}=1024$
每个 SM 最大活动线程块数(+)	$B_{SM}=8$
每个 SM 共享内存大小(+)	$S_{SM}=16$kB
每个 SM32 位寄存器大小(+)	$R_{SM}=16$k
每个线程块的线程数(*)	K
线程块数(*)	L
每个线程块所需寄存器大小(*)	R_{TB}
每个线程块所需共享内存大小(*)	S_{TB}
每个线程块 Warp 数(!)	$W_{TB}=\lceil K/W \rceil$
由活动 Warp 数所限定的线程块数(!)	$B_W=\min(\lfloor W_{SM}/W_{TB} \rfloor, B_{SM})$
由寄存器需求所限定的线程块数(!)	$B_R=\lfloor R_{SM}/R_{TB} \rfloor$
有共享内存需求限定的线程块数(!)	$B_S=\lfloor S_{SM}/S_{TB} \rfloor$
最大活动的线程块数(!)	$B=\min(B_R, B_S, B_W)$

(+)表示该参数由硬件设备决定；(*)表示该参数由程序决定；(!)表示该参数由两者共同决定

3.4.3 每个 block 中的线程数 K

根据以上提出的原则构建近似最优配置模型。通过一组限制条件，选取合适的线程块内线程数 K。首先，K 的选取必须满足 $B_S \leqslant B_W$，以保证最大化利用共享内存资源。其次，K 必须是 32 的倍数，以保证块内的线程可以满足形成完整的 warp。再次，$K \leqslant T_{SM}$，保证不会超出 GPU 硬件限制。最后，K 必须保证 $K \times B > 192$，可以隐藏延时。

3.4.4 grid 中的 block 数 L

线程块必须平均分配给 SM。例如，Tesla C1060 拥有 30 个 SM，最优的配置需要 grid 中的块数目是 30 的倍数。如果每个 SM 所能激活的最大块数目为 B，block 数应该选取 $B \times 30$ 为最佳。少于这个值，GPU 的 SM 无法填装满，无法发挥它的最大计算能力。多于这个数，SM 将不得不通过移出移入线程块的方法来完成全部线程块运算。

4 实验结果与分析

本节主要分析直方图生成算法的实现。首先比较分析两

种极端图像分布(均匀分布(Uniform Distribution)和退化分布(Degenerate Distribution)下内存填充技术所起到的效果。其次,通过实验分析,评估近似最优配置模型。本文的实验平台是 NVIDIA Tesla C1060。其设备参数如表 1 所列。

本文实验选用两种极端图像分布。其他图像分布介于两者之间,其算法的性能表现也在两者之间。

1)均匀分布:每一个 Bin 接受到同样数目的投票更新,并且不存在写冲突和存储体冲突。

2)退化分布:所有线程投票更新同一 Bin,写冲突和存储体冲突达到最大。

4.1 内存填充技术实验结果及分析

图 4 和图 5 所示为 16-Bin 到 112-Bin 均匀分布和退化分布使用和不使用内存填充技术的实验结果。输入数据大小为 10^8。先对所有可能的 GPU 配置进行穷举,运行所有可能的配置得出实验结果,并从中选取最优结果得出图 4 和图 5。

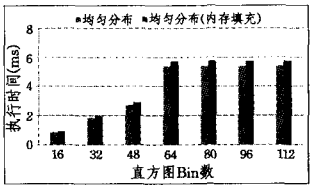


图 4 均匀分布内存填充方法实验结果

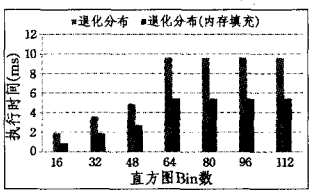


图 5 退化分布内存填充方法实验结果

由于原始的均匀分布并没有写冲突,也没有发生存储体冲突,因此内存填充的方法并不能使其性能进一步提高。但是,尽管加入填充计算语句,使其计算量轻微增加,但其内存填充后的性能并没有明显下降,如图 4 所示。因此,对于均匀分布,不应使用内存填充。反观退化分布,由于大量的写冲突和存储体冲突发生,内存填充方法能明显帮助其减少冲突,极大程度上提高了性能,如图 5 所示。

64-Bin 到 112-Bin,共享内存成为程序的主要瓶颈。利用共享内存,其所能激活的最大线程数目相同。因此,在输入数据规模相同的情况下,其所产生的更新投票运算和延时等待是相同的。因此,执行时间近似。

现实中的图像分布介于均匀分布和退化分布之间,因此内存填充方法所产生的性能提升也在其两者之间。我们测试了两组普遍的视频例子,每组视频由大约 3000 帧 600×480 像素组成,其性能的提升在 42%到 88%之间。

4.2 近似最优配置模型实验结果及分析

根据之前的穷举实验结果,分析近似最优配置模型。表 2 的测试配置中,列出 K 从 64 到 224 的配置(且为 32 的倍数)。最大的线程块 L 数量由 3.4 节内容算出,以确保其最优。

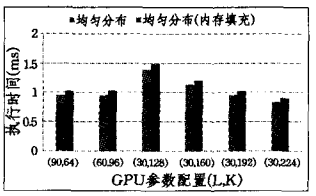


图 6 均匀分布近似最优模型评估

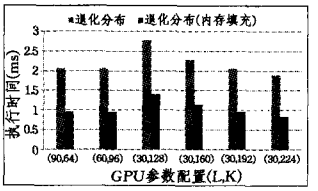


图 7 退化分布近似最优模型评估

图 6 和图 7 为 16-Bin 直方图生成,使用不同配置的实验结果。按照近似最优算法,挑选活动线程为 224 的配置作为近似最优的可能配置,即表 2 中最后一项配置。实验结果显示,近似最优配置确实是所穷举的配置中最优的。

表 2 一组 16-Bins 直方图生成 GPU 配置参数

线程块数	每个线程块中线程数	SM 活动线程块数	SM 活动线程数
90	64	3	192
60	96	2	192
30	128	1	128
30	160	1	160
30	192	1	192
30	224	1	224

结束语 近几年,GPU 作为一个强大的数据并行协处理器,已经不仅仅作用于图像处理,它越来越受到高性能计算领域的关注,被用于完成数据密集运算。本文基于当今最热的 GPU 并行计算平台,通过内存填充技术,提出一种新的直方图生成加速算法,大幅提高了这一被广泛应用的图像基础算法在 GPU 上的性能,并建立了近似最优搜索模型,以寻找最优的 GPU 执行配置。经过实验测试,新算法在通用图像实例中,性能比原有算法提高了 42%到 88%。

参 考 文 献

[1] Pal N R, Pal S K. A review on image segmentation techniques [J]. Pattern Recognition, 1993, 26(9): 1277-1294

[2] 白洪涛,欧阳丹彤,李熙铭,等. 基于 GPU 的稀疏矩阵向量乘优化[J]. 计算机科学,2010,37(8):168-171

[3] NVIDIA. CUDA C Programming Guide 3.2[EB/OL]. http://developer.download.nvidia.com/compute/cuda/3_2/toolkit/docs/CUDA_C_Programming_Guide.pdf, 2011-03-30

[4] Podlozhnyuk V. Histogram calculation in CUDA(White paper) [EB/OL]. http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/histogram256/doc/histogram.pdf, 2011-03-30

[5] Shams R, Kennedy R A. Efficient histogram algorithms for NVIDIA CUDA compatible devices[C]// ICSPCS2007. New York: IEEE, 2007: 418-422

[6] Shams R, Sadeghi P, Kennedy R A, et al. Parallel computation of mutual information on the GPU with application to real-time registration of 3D medical images[J]. Computer Methods and Programs in Biomedicine, 2010, 99(2): 133-146

[7] NVIDIA. Fermi compute architecture(White paper)[EB/OL]. http:// www.nvidia.com/content/PDF/fermi white papers/, 2011-03-30

[8] Ryoo S, Rodrigues C I, Stone S S, et al. Program optimization carving for GPU computing[J]. 2008, 68(10): 1389-1401

[9] NVIDIA. CUDA C Best Practices Guide 3.2[EB/OL]. http://developer.download.nvidia.com/compute/cuda/3_2/toolkit/docs/CUDA_C_Best_Practices_Guide.pdf, 2011-03-30

[10] Wong H, Papadopoulou M M, Sadooghi-Alvandi M, et al. Demystifying GPU Microarchitecture Through Microbenchmarking[C]//ISPASS2010. New York: IEEE, 2010: 235-246