基于OpenCL的直方图生成算法优化方法研究

安小景 张云泉 贾海鹏
(中国科学院计算技术研究所体系结构国家重点实验室 北京 100190)

摘 要 随着GPU计算能力及可编程性的不断增强,采用GPU作为通用加速器对应用程序进行性能加速已经成为提升程序性能的主要模式。直方图生成算法是计算机视觉的常用算法,在图像处理、模式识别、图像搜索等领域都有着广泛的应用。随着图像处理规模的扩大和实时性要求的提高,通过GPU提升直方图生成算法性能的需求也越来越强。在GPU计算平台关键优化方法和技术的基础上,完成了直方图生成算法在GPU计算平台上的实现及优化。实验结果表明,通过使用直方图备份、访存优化、数据本地化及规约优化等优化方法,直方图生成算法在AMD HD7850 GPU计算平台上的性能相对于优化前的版本达到了1.8~13.3倍的提升;相对于CPU版本,在不同数据规模下也达到了7.2~210.8倍的性能提升。

关键词 GPGPU,OpenCL,数据本地化,直方图生成
中图法分类号 TP302.7 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.11.005

Research on Histogram Generation Algorithm Optimization Based on OpenCL

AN Xiao-jing ZHANG Yun-quan JIA Hai-peng
(State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China)

Abstract Application developers increasingly adopt GPUs as standard computing accelerators to improve application performance with their easier programmability and increasing computing power. The histogram generation algorithm is a common algorithm of computer vision, and is widely used in image processing, pattern recognition and image search. With the scale enlargement of image processing and the demand of real-time, improving the performance of histogram generation algorithm by GPU is in increasingly high demand. We introduced the realization and optimization on GPU of histogram to research the major optimization methodologies and technologies. Experimental results show that the applications of access optimization of histogram backup, memory optimization, data localization and mergence optimization, and some other optimization strategies, bring about a 1.8 ~ 13.3 times speedup for the algorithm on AMD HD 7850, than versions before optimization, and brings about a 7.2~ 210.8 times speedup than CPU versions.

Keywords GPGPU, OpenCL, Data localization, Histogram generation

1 引言

当前,GPU通用计算发展如火如荼,它作为提高算法性能的一种成熟有效的解决方案,已融入各行各业。应用开发人员也已经越来越重视通过利用GPU强大的计算能力来满足相关领域的性能要求。各大硬件厂商也在竞相开发能够更灵活高效地支持GPU通用计算且更易编程的GPU设备;与此同时,GPU通用计算的编程模型也日益成熟,本文采用的OpenCL编程模型具有跨厂商和跨平台性的特点,可通过统一的编程接口同时发挥系统中所有计算单元的计算能力,这些都使得GPU程序的实现变得越来越简单。但是,为使GPU程序能够更加有效地利用设备的计算能力,还需要对GPU程序进行优化。相对于GPU程序的实现,优化过程要复杂得多,因为程序员不仅要考虑算法特征,还要了解底层硬件架构的特征,以获得这两种特征的高效映射[1]。

直方图生成算法是计算机视觉的常用算法,在图像处理、模式识别、图像搜索等领域都有着广泛的应用。随着图像处理规模的扩大和实时性要求的提高,通过GPU提升直方图生成算法性能的需求也越来越强。一方面,直方图生成算法具有高度并行性,适合通过GPU计算平台进行加速;另一方面,直方图生成算法的GPU实现存在并发访问冲突、原子操作的使用导致算法并发性降低等特点,严重影响算法性能。本文通过直方图备份、访存优化、数据本地化、规约优化以及最优work-group数目选取等优化策略的使用,使得直方图生成算法在AMD HD7850 GPU计算平台上的性能相对于优化前的版本达到了1.8~13.3倍的提升,相对于CPU版本,在不同数据规模下也达到了7.2~210.8倍的性能提升。

本文的贡献如下:

到稿日期:2014-10-11 返修日期:2014-12-30 本文受国家自然科学基金(61272136),国家自然科学基金创新群体(61221062)资助。
安小景(1991-),女,硕士生,主要研究方向为高性能计算,E-mail:thefighteran@163.com(通信作者);张云泉(1973-),男,博士,博士生导师,CCF会员,主要研究方向为高性能计算及并行数值软件、并行计算模型;贾海鹏(1983-),男,博士后,主要研究方向为高性能计算、众核编程方法。

1. Atomic-free：充分利用数据本地化并结合线程调度策略，通过直方图备份，避免了全局原子操作；

2. Balance：实现了最优 work-group 数目确定策略，获得统计与规约计算的最佳平衡。

本文第 2 节介绍相关工作；第 3 节简要介绍了 GPU 框架及直方图生成算法；第 4 节介绍了直方图生成算法的基本实现及优化措施；第 5 节对直方图算法的不同实现进行性能评测及性能分析；最后对全文进行了总结。

2 相关工作

近年来，关于直方图生成算法在 GPU 计算平台上的实现和优化，已有许多类似的工作。

Shams[2] 等引入了虚拟原子更新的方式来减小原子操作对直方图生成带来的延时，但这种方法仍然不是一种理想的直方图生成算法。北京科技大学的狄鹏等[3] 利用共享内存避免了全局内存的原子操作，并通过填充技术来避免 Bank 冲突，取得了较高的加速比，然而却没有提出对直方图规约过程的优化，并且内存填充技术也并不是最佳的避免 Bank 冲突的方式。Juan Gómez-Luna 等的工作[4] 与狄鹏等的类似，即增加了交错全局内存读取的方式，以提高访存带宽，但现在的大部分 GPU 都已将此优化措施直接在硬件实现了。

与以上工作不同，本文从建立直方图备份、数据本地化、规约树优化、最优 work-group 数目的测定 4 个方面完成了直方图算法在 GPU 计算平台上的实现和完整优化。

3 背景介绍

3.1 GPU 架构

经过几年的发展，虽然 GPU 计算平台呈现出日益多样化的特点，主流硬件厂商如 AMD 和 NVIDIA 也都发布了不同架构的 GPU 产品，但是 GPU 计算平台依然遵循着最基本的层次架构特征。GPU 层次架构体现在 3 方面：计算单元的组织、内存架构以及优化。

计算单元的组织方面：多个计算元素（Processing Element，PE）组成一个计算单元（Compute Unit，CU），多个计算单元组成一个计算设备，而多个计算设备可组成一个计算平台[5]。这种组织结构与线程的层次调度相对应，为实现任务的灵活分配、线程间同步等提供条件。

内存架构方面：内存架构分为片上和片外两部分：片上有仅被一个线程访问的私有内存和能够被一个 work-group 内所有线程访问的本地内存（LDS）。片外有能够被所有线程访问的全局内存。内存的这种层次式架构为访存优化提供了可能，尽可能地充分利用访存带宽和数据本地化是最常用的优化方法。

优化方面：结合硬件架构的层次性，优化方法也由此具有层次性。通用的优化方法包括优化访存、优化计算资源的使用率、数据本地化等。

3.2 直方图生成算法

直方图（Histogram）生成算法用于统计图像中像素强度分布，即统计图像中不同像素值的个数。

例如，对于一个黑白图像，直方图生成算法就是统计各个像素值（0～255）在该黑白图像中出现的次数，最终生成一个包含 256 个元素的整型数组。本节研究的是黑白图像的直方图生成，其中，将图像数据视为一个一维的整型（integer）数组，每个整型元素包括 4 个像素值。故算法输入是一个整型一维数组，输出是一个长度为 256 的整型数组，数组的每个元素代表了各个像素值在图像中出现的次数。

直方图生成算法是一种顺序的非规则数据依赖的循环计算[3]。对于单线程的实现，如本次实验的 CPU 实现，算法较为简单，而且对图像及存储直方图的数组的数据访问总能达到 cache 命中，获得较高的性能，图 1 所示为直方图生成算法的 CPU 实现。

```
Image：源图像数据
IMAGE_LENGTH：源图像数据量
Histogram：直方图
1. Call GetImageData(SOURCEIMAGE,image);
2. Histogram ← 0;
3. For i from 0 to IMAGE_LENGTH step 1
4.      histogram [image[i]] ← histogram [image[i]]+1;
5. End for
```

图 1 直方图生成算法的 CPU 实现伪代码

4 直方图生成算法的 GPU 实现与优化

4.1 并行性分析及基本实现

直方图生成算法具有良好的并行性，每个像素值的统计是相互独立的，可以并行执行，对应于 GPU 的众多处理单元，可使每个线程分别处理不同像素值的统计。

在直方图生成算法的 GPU 基本实现中，由于直方图的计算密度较低，如果一个线程只处理一个像素，执行时间将大量用于数据访存，从而无法有效利用 GPU 的计算能力；同时，GPU 对一个整型数据的访存及计算速度，要快于对 4 个单独的 char 类型的数据的访存和计算，所以我们将图像存储为整型数组，并令每个线程处理 *PERWORKITEM_PROCESS_INT* 个整型数据，*PERWORKITEM_PROCESS_INT*≥1。在全局内存的访问上，应保证线程的连续访存，即使连续的 16 个线程访问连续的 16 个 4 字节的数据（64 字节是全局内存的访存单位[6]），实现带宽利用率最大化。在 GPU 上绘制直方图，使用原子操作解决当多线程同时将统计结果写至直方图数组时导致的访存冲突问题。

4.2 算法优化

直方图生成算法的 GPU 基本实现的性能并不理想，其主要原因是全局内存的原子操作加重了访存负担，使访存操作序列化，从而严重影响了算法性能。本文将通过建立直方图备份避免全局内存的原子操作，通过本地访存优化避免 Bank 冲突，通过规约树优化提升规约效率，通过最优 work-group 数目的选择来获得最佳统计规约计算比，从而大大提升直方图生成算法在 GPU 计算平台上的性能。

4.2.1 全局内存访存优化

全局内存访问是非常昂贵的，本地内存的访问可能只需几个时钟周期，而全局内存则需要数百个时钟周期[7]。在直方图生成算法的 GPU 初步实现中，因为线程对直方图数组（全局内存数据）的访问是原子操作，且访问重复率高，这使得线程对直方图数组的访问顺序化，并行性大大降低，并使得全

局内存的带宽大大降低，大量线程在等待资源，造成了处理器闲置，计算能力利用率下降。

为解决这个问题，本文在每个 work-group 的局部内存中开辟一份直方图备份，每个 work-group 处理原图像数据的一部分并将统计结果存储于局部直方图备份中，work-group 间并行处理图像数组中的数据，然后将这些局部的直方图进行规约。统计与规约过程如图 2 所示。

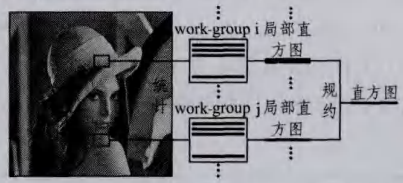

图 2　GPU 直方图生成算法的统计与规约过程

在统计过程的每个 work-group 中，线程内部是顺序执行的，线程间是并发的，所有线程均访问同一个直方图数组，因此存在数据访问冲突，故需要局部内存的原子操作。规约过程可以直接在 CPU 上进行。这样，全局内存的访存中不再有冲突，也就不需要全局内存的原子操作，而作用于局部内存的原子操作产生的延时要远小于作用于全局内存上的。这个方法虽然增加了全局内存数据的传输，但是却极大提高了计算的并行性，算法性能也得到明显的提升。

4.2.2　数据本地化

在统计过程中，每个 work-group 从全局内存中读取一部分数据，通过原子操作对这部分数据进行统计，得到局部直方图，但是会出现大量的 Bank 冲突。可以通过在每个 work-group 中创建多个直方图备份来避免 Bank 冲突。

线程对 Banks 的访存是以连续的 32 个线程为单位的[8]，最好的情况是这连续的 32 个线程所访问的数据分别在 32 个 Banks 中，或者均访问一个地址，这样这 32 个线程就能实现在一个访问周期内，所有 32 个线程均访问到所需数据，否则这 32 个线程间将存在 Bank 冲突。在局部直方图生成过程中，每个线程对局部内存中的直方图的访问与其对应的图像数据有关，故访问具有随机性。随机访问是不可预测的，即连续 32 个线程对 Banks 的一个轮次的访问不能保证是均匀分布在每个 Bank 中的，也不能保证都访问一个 Bank 中的数据，相反这两种情况都是非常不常见的，因此，局部内存的访问中会出现大量 Bank 冲突，这使得数据访问的并行性大大降低，严重影响算法性能。

一个解决方案是为每个线程创建一个直方图副本，这个副本存储在线程的私有寄存器中。这样避免了 Bank 冲突，因为统计过程不再访问本地内存，而是一直在访问私有内存，但是仍需原子操作，因为要将每个线程的直方图规约到本地内存的局部直方图。而且由于每个线程都开辟了一个直方图，总体开辟空间增多，导致空间利用率下降，进而导致每个处理核心上可以同时并行的线程数减少，并行性降低，同时，私有直方图规约到局部直方图也相当耗时。

另一个方案充分利用了本地内存的硬件结构特征及线程调度策略，无需开辟太多空间就可以极大地减少 Bank 冲突。具体实现是在本地内存中开辟 32 个直方图的数据空间，每个 Bank 中存放一个直方图子图，则连续的 32 个线程分别访问这 32 个直方图，这样就可以尽量避免 Bank 冲突，同时也基本避免了线程间由于原子操作而造成的资源等待。这一方案与只有一个直方图子图的方案的本地内存存储访问的对比如图 3 所示。

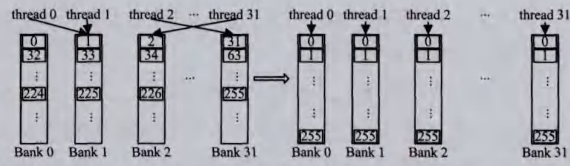

图 3　直方图生成 GPU 算法的数据本地化优化

此方案利用了本地内存是以连续 32 个线程的访问为单位，使统计过程获得了与前一个方案几乎一样的性能，但是却不用开辟那么多的空间，而且 work-group 内部的直方图规约的工作量也要少很多。实验证明，这一方案的确优于前一个方案。

方案二的 work-group 的统计过程中，需要首先将本地的 32 个直方图的数据初始化为 0，最后还需对 32 个直方图进行规约，在这两个过程中，通过有效地控制线程与数据的对应关系，避免 Bank 冲突，实现性能进一步提升。

4.2.3　全局规约优化

在 4.2.2 节中讨论的是统计过程，其结果是统计图像的像素值，并生成局部直方图集，本小节是对规约过程进行优化。

由于统计后的数据量很小，可以将这一局部直方图集合的数据读至 CPU 端，用 CPU 进行规约。但是通常处理的图片数据都不会太小，所以这里还是讨论再实现一个 kernel 来规约。为提高规约过程的并行性，可以采用规约树，规约树的数组实现实例如图 4 所示。

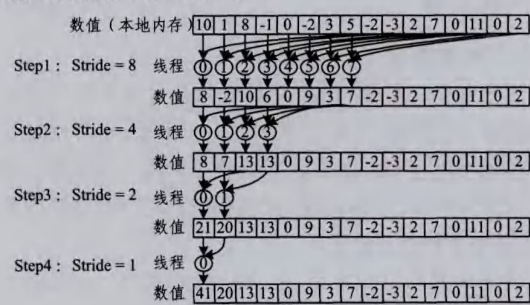

图 4　数组求和的规约树过程

图 4 所示的规约树实例是用 8 个线程对 16 个元素进行求和，按照规约树来进行规约可以达到最高 8 个线程的并行度，从而有效地实现规约的并发执行。

直方图生成的规约的处理对象是包含多个直方图子图的集合，目标是最终生成一个直方图。本次实验中，我们在规约过程中定义 256×256 个线程，也即 256 个 work-group，并让 work-group X 规约直方图的第 X 个元素，work-group 内规约第 X 个元素的过程与图 4 类似，此处不再赘述。

4.2.4　最优 work-group 数目确定

确定最优的统计阶段的 work-group 数量是为了均衡统计阶段和规约阶段用时（其中，每个 work-group 中有 256 个线程）。因为如果 work-group 数量少，在统计阶段每个 work-group 内数据计算量大，并行粒度增大，总的计算量也相对较大，但是规约阶段所需处理的数据量就小，速度也就快。同时，在 work-group 较少时，片上及片下内存的冗余数据传输也就少，这也会节约时间；相反，如果 work-group 数量大，那

么统计阶段就会相对较快，而规约阶段耗时增多，同时片上片下内存的冗余传输增多。所以不同的 work-group 的数量对直方图生成算法的性能还是有较大的影响。进行某一数据量的图像处理时，改变 work-group 的数量，并统计不同 work-group 下的算法性能，可以获得一个最优的 work-group 数量。对于不同的图像数据量，会有不同的最优 work-group 数量。本文算法测得当图片大小为 8192×8192(int * int)时，最优 work-group 数量为 1024，考察过程详见 4.2.1 小节。

5 性能评估

5.1 测试环境搭建

本次实验选择 Intel(R)Core(TM)i5-3470 CPU(以下简称 Intel CPU)、AMD Radeon™ HD 7850 GPU(以下简称 AMD7850)作为硬件，硬件平台的性能参数如表 1[9]所列。

表 1 实验硬件平台参数对比

	AMD HD7850	Intel i5-3470 CPU
计算单元时钟频率	0.84 GHz	3.2GHz
处理核心数目	1761	4
单精度浮点数峰值	1761GFlops	57.76GFlops
片外访存峰值带宽	154GB/s	25GB

选取的直方图生成算法的 CPU 版本为 OpenCV 2.4 中的代码，OpenCV(Open Source Computer Vision Library)[10]是计算机视觉领域应用最为广泛的开源代码库，其中的算法和实现都经过了深度优化，与这个版本进行性能比较，更具有说服力。

5.2 性能分析

5.2.1 最优 work-group 数量测定

在确定要处理的图像的大小后，通过改变 work-group 的数量，测定直方图生成过程的时间，找到最优的 work-group 数量，以 8192×8192(int * int)为例，测定不同的 work-group 数量下的性能，结果如图 5 所示。

观察实验结果可以很容易地找到最优 work-group number，即 1024，此时总体执行时间最短。还可以发现，随着 work-group number 的增长，规约时间明显地呈指数递增。

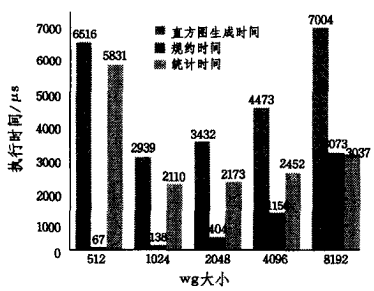

图 5 最优 work-group number 测定

5.2.2 优化后性能提升分析

直方图生成算法在 4 步优化操作后，与优化前相比，不再包含对全局内存的原子操作，并行性得到极大地提高，而局部内存访问中也不存在 Bank 冲突。图 6 所示为优化各阶段直方图生成算法性能对比。

直方图备份后，避免了全局内存的原子操作，使算法性能相对优化前有 2.85～4.13 倍的提升，由此可知全局内存的原子操作对直方图算法的性能有巨大的影响。本地访存优化后，增加了本地内存的使用，限制了每个计算单元上的 work-group 数，同时也增加了计算，但是算法性能仍有 1.02～1.34 倍的提升，可知本地访存优化措施确实避免了大部分的 Bank 冲突。规约树优化后，避免了在数据量较大时规约时间的指数级增长，提升了规约效率，对算法性能有明显的提升效果即 0.64～3.15 倍，其中，在数据量较小时，另启用 kernel 来进行规约操作时的性能反不及直接用 CPU 进行规约，故存在性能负增长的情况。

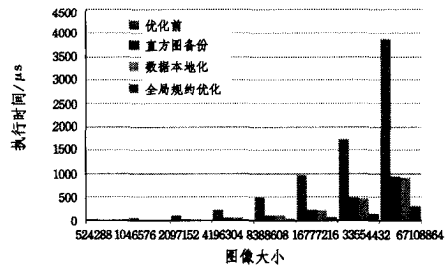

图 6 优化各阶段性能对比

经过一系列优化措施后，性能相对优化前获得了 1.8～13.3 倍的提升，随着图像大小的增大，性能提升倍数也逐渐扩大。

5.2.3 直方图生成算法在 GPU 与 CPU 上的性能对比

对比 GPU 和 CPU 上不同图像大小下 GPU 与 CPU 的性能，从 256×256(int * int)大小开始测试，依次成二倍增长，GPU 相对 CPU 获得了 7.2～210.8 倍的性能提升。图 7 为所得统计数据。

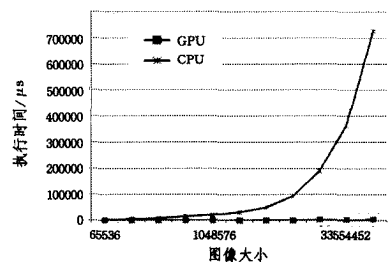

图 7 GPU 与 CPU 平台测试

由图 7 可以发现，CPU 执行时间随着图像大小的变化呈指数级增长，而 GPU 虽然开始时的性能与 CPU 基本相同，但是随着图像的增大，总的执行时间并未出现明显变化。因为 GPU 程序执行时间中包含 kernel 启用的时间，所以在图像数据较小时，GPU 就有一个较长的执行时间，但是随着图像的增大，kernel 启用时间基本保持不变，而 GPU 的强大的计算能力使得算法执行时间也无较大变化，所以在数值较大时，GPU 运行时间要远少于 CPU。图 7 中的统计时间仅为 kernel 执行时间。因为在实际项目中，平台布置时间和数据传输时间都是可以减少甚至避免的。在系统初始化中完成平台布置，并把平台数据置于缓存中，之后的计算都无须再因为平台布置而耗时。对于数据传输，因为现在的硬件设备一直在改进，使 GPU 与 CPU 使用同一块地址空间，可逐渐避免数据的冗余传输。

结束语 本次试验详细分析并完成了直方图生成算法在 GPU 计算平台上的实现及优化。直方图生成算法在 GPU 计算平台上的实现和优化中，通过设置直方图备份以避免全局内存的原子操作，通过本地访存优化以避免 Bank 冲突，通过规约树优化以提升规约效率，通过最优 work-group 数选取以获得统计与规约计算的最佳平衡，实现了较为完整、全面的优

化，相对于优化前版本，整体取得了 1.8～13.3 倍的性能提升。但是由于时间所限，还有很多工作需要完善，如算法的指令级优化、访存锁优化等。由于 NVIDIA 690 的 OpenCL 原子操作是由软件实现的，导致本算法的实现未能在 NVIDIA 690 上获得理想的性能，之后也将对此做进一步研究。

参 考 文 献

[1] Jia Hai-peng. Research of Parallel Optimization Technicals on GPU Computing Platforms[D]. Qingdao: Ocean University of China, 2013

[2] Shame R, Kennedy R A. Efficient histogram algorithms for NVIDIA CUDA compatibledevice [C] // ICSPCS2007. New York: IEEE, 2007: 418-422

[3] Di Peng, Hu Chang-jun, Li Jian-jiang. Efficient Method for Histogram Generetionon GPU[D]. Beijing: University of Science and Technology, 2011

[4] Gómez-Luna J, González-Linares J M, Benavides J I, et al. An optimized approach to histogram computation on GPU[J]. Machine vision and applications, 2013, 24(5): 899-908

[5] Zhang Yuan-quan, ZhangXian-yi, Jia Hai-peng, et al. Heterogeneous Computing with OpenCL [M]. Tsinghua University press, 2012

[6] AMD GRAPHICS CORES NEXT(GCN) Architecture Whitepaper [J/OL]. https://www.amd.com/Documents/GCN_Architecture_whitepaper.pdf

[7] Munshi A, Gaster B, Mattson T G, et al. OpenCL programming guide[M]. Pearson Education, 2011

[8] AMD R & D center in Shanghai. Cross platform multicore and manycore Programming Notes——int the way of OpenCL[OL]. http://down.51cto.com/data/964762

[9] AMD. AMD Accelerated Parallel Processing OpenCLTM Programing Guide[OL]. http://developer.amd.com/wordpress/media/2013/07/AMD_Accelerated_Parallel_Processing_OpenCL_Programming_Guide-rev-2.7.pdf

[10] Zhang Jing. OpenCV2 Computer vision programming manual [M]. Science Press Limited liability company, 2013

[11] Jia Hai-peng, Zhang Yun-quan, Long Guo-ping, et al. GPURoofline: A Model for Guiding Performance Optimizations on GPUs [C]// Proceeding of International European Conference on Para-llel and Distributed Computing. Rhodes Island, Greece, 2012: 920-932

[12] Jia H, Zhang Y, Wang W, et al. Accelerating viola-jones facce detection algorithm on gpus[C]// 2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS). IEEE, 2012: 396-403

[13] NVIDAI. GPU-ACCELERATED APPLICATIONS [OL]. http://www.nvidia.com/object/media-and-entertainment.html

[14] NVIDIA. NVIDIA's Next Generation CUDA™ Compute Achitecture: Kepler GK110 [OL]. http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf

(上接第 31 页)

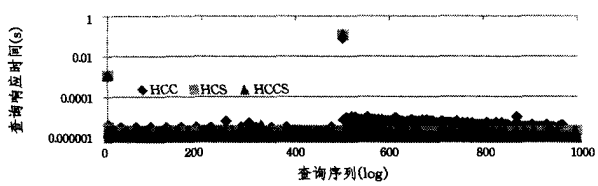

图 11 SeqZoomIn 查询负载下的结果

结束语 在海量数据下，分析型数据库系统面临索引构建和维护、查询效率等一系列新的挑战。为了应对挑战，数据库分裂技术、自适应合并技术及一系列的混合算法被提出并成为近期关注的热点。

本文在对已有算法分析的基础上，给出了一个定量的代价模型。已有的研究没有量化的代价模型，导致对于查询执行过程中自适应索引的构建和查询的响应无法进行量化分析，难以对动态负载下的算法进行优化选择。基于代价模型，我们给出一个新的自适应索引算法。新的算法更灵活，同时具备已有算法的优点。在不同的查询工作负载下的实验表明，新的混合算法具有很好的性能表现。

参 考 文 献

[1] Chaudhuri S, Weikum G. Rethinking database system architecture: Towards a self-tuning risc-style database system [C] // Proceedings of the 26th In VLDB. 2000: 1-10

[2] Graefe G, Idreos S, Kuno H, et al. Benchmarking adaptive indexing[C]// TPCTC. 2010: 169-184

[3] Graefe G, Kuno H. Adaptive indexing for relational keys[C]// ICDE. 2010: 69-74

[4] Graefe G, Kuno H. Self-selecting, self-tuning, incrementally optimized indexex[C]// EDBT. 2010: 371-381

[5] Idreos S, Kersten M L, Manegold S. Database cracking[C]// CIDR. 2007: 68-78

[6] Idreos S, Kersten M L, Manegold S. Updating a cracked database[C]// SIGMOD. 2007: 413-424

[7] Idreos S, Kersten M L, Manegold S. Self-organizing tuple reconstruction in column stores[C]// SIGMOD. 2009: 297-308

[8] Idreos S, Manegold S, Kuno H, et al. Merging what's cracked, cracking what's merged: adaptive index in main-memory column-stores[J]. PVLDB, 2011, 4(9): 585-597

[9] Halim F, Idreos S, Karras P, et al. Stochastic database cracking: Towards robust adaptive indexing in main-memory Column-stores[J]. PVLDB, 2012, 5(6): 502-513

[10] Kersten M, Manegold S. cracking the database store[C]// CIDR. 2005: 213-224

[11] Hoare C A R. Algorithm 64: Quicksort[J]. Communications of the ACM, 1961, 4(4): 319-320

[12] Schuhknecht F M, Jindal A, Dittrich J. The Uncracked Pieces in Database Cracking[C]// VLDB. 2013

[13] Pirk H, Petraki E, Idreos S, et al. Database Cracking: Fancy Scan, not Poor Man's Sort! [J/OL]. http://oai.cwi.nl/oai/asset/22474/22474D.pdf

[14] Graefe G, Halim F, Idreos S, et al. Transactional support for adaptive indexing[J]. The VLDB Journal, 2014, 23(2): 303-328

[15] Alvarez V, Schuhknecht F M, Dittrich J, et al. Main Memory Adaptive Indexing for Multi-core Systems[J/OL]. http://arxiv.org/abs/1404.2034