软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(Suppl.(2)):163−171

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于 OpenCL 的归约算法优化*

颜深根[1,2,3+], 张云泉[1,2], 龙国平[1], 李 焱[1,2,3]

[1](中国科学院 软件研究所 并行软件与计算科学实验室,北京 100190)

[2](中国科学院 软件研究所 计算机科学国家重点实验室,北京 100190)

[3](中国科学院 研究生院,北京 100190)

Reduction Algorithm Optimization Based on the OpenCL

YAN Shen-Gen[1,2,3+], ZHANG Yun-Quan[1,2], LONG Guo-Ping[1], LI Yan[1,2,3]

[1](Laboratory of Parallel Software and Computational Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](State Key Laboratory of Computing Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[3](Graduate University, The Chinese Academy of Sciences, Beijing 100190, China)

+ Corresponding author: E-mail: yanshengen@gmail.com

Yan SG, Zhang YQ, Long GP, Li Y. Reduction algorithm optimization based on the OpenCL. *Journal of Software*, 2011,22(Suppl.(2)):163−171. http://www.jos.org.cn/1000-9825/11037.htm

Abstract: Reduction algorithm has a wide range of applications in areas such as scientific computing and image processing. This paper systematically studies the reduction algorithm optimization on the GPU's cross-platform performance optimization based on the OpenCL framework. Previous research has generally focused on a single hardware architecture, however, this paper based on the OpenCL, studies various kinds of optimization methods, such as using vector, on-chip memory bank conflict, threads organization, instruction selection and so on. The research takes the minMax function for example, dilatationed each optimization method for develep the performance, and detailed the reason. The study tests the algorithm both on AMD GPU and NVIDIA GPU platforms. The test results show that the optimized algorithm on both platforms has achieved good performance. In the AMD ATI Radeon HD 5850 platform, Int and Float types of data bandwidth utilization up to 89%. In the NVIDIA GPU Tesla C2050 platform, the performance has reached 1.3 to 1.9 times compare to appropriate function version of CUDA.

Key words: GPU; parallel reduction; OpenCL; CUDA

摘 要: 归约算法在科学计算和图像等领域有着广泛应用,系统研究了在 OpenCL 框架下,归约算法在 GPU 上的跨平台性能优化.已有研究工作一般只侧重单个硬件架构,基于 OpenCL 从向量化、片上存储体冲突、线程组织方式和指令选择优化等多个优化角度系统考察了不同优化方法在 GPU 硬件平台的影响.具体以 minMax 函数为例,对每种优化方法进行了详细的性能分析,并给出了提高性能的原因.在 AMD GPU 和 NVIDIA GPU 平台分别测试的结

* 基金项目: 国家自然科学基金(60303020, 60533020); 国家高技术研究发展计划(863) (2006AA01A102); ISCAS-AMD 联合 fusion 软件中心资助项目

收稿时间: 2011-07-15; 定稿时间: 2011-12-02

果表明,优化后的算法在两个平台上都能实现很好的性能加速.在 AMD ATI Radeon HD 5850 平台上,Int 和 Float 类型数据带宽利用最高达到了实测带宽的 89%.在 NVIDIA GPU Tesla C2050 平台上,性能也达到了 CUDA 版本的相应函数性能的 1.3~1.9 倍.

关键词: GPU;并行归约;OpenCL;CUDA

GPU 作为一种强大的、高度并行的、可编程的计算部件[1],近年来,已经突破了图形处理的限制,正在逐渐向通用计算(GPGPU)的方向发展,越来越多的 CPU 算法被移植到 GPU 上,并取得了非常好的性能.归约算法在科学计算和图像处理等领域有着非常广泛的应用,是并行计算程序的基本算法之一[2],因此,将归约算法在 GPU 上进行优化具有非常重大的意义.

目前在 GPU 上编程的两个主要框架一个是 CUDA,另外一个是 OpenCL(open computing language),OpenCL 是第一个面向异构系统通用目的的并行编程框架.OpenCL 具有免费、开发的特点,也是一个统一的编程环境,OpenCL 能适用于多核心处理器(CPU)、图形处理器(GPU)、Cell 类型架构以及数字信号处理器(DSP)等其他并行处理器,在游戏、娱乐、科研、医疗等各个领域都有广阔的发展前景.OpenCL 工作组的成员包括:AMD,苹果,ARM,Nvidia,Inter,IBM 等,得到了广泛的支持,因此,OpenCL 的未来非常诱人[3].

计算机中的归约算法,是指将大量数据压缩成一个或少量数据的一种操作.比如对数组求和、求最大值、最小值等.归约的串行操作非常简单,如下所示即为求 Int 类型数组最大值操作的串行伪码:

```
Max=MIN_INT;
For(int I=0; I<len; i++)
    If(data[i]>Max)
        Max=data[i];
```

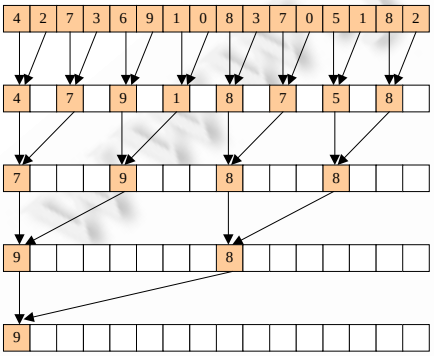

Fig.1 Binary reduction algorithm
图 1 二分法并行归约算法示意图

串行代码中,每一次循环都需要利用上一次循环的结果,所以并不能直接用于并行程序.以求最大值为例,一种常见的并行方式是将原始数据分为多个组,对各个组分别进行求最大值操作,最后再对各个组的结果求最大值,或者是对各个组的结果再分组进行求最大值操作,如此递归,得到最终结果,如图 1 所示.

目前 GPU 领域有两个主流平台,一个是 AMD 的 ATI GPU,另外一个是 NVIDIA 的 GPU.这两个平台各具特点,AMD 是基于向量的 APP 平台,NVIDIA 是基于标量的 CUDA 平台[5].以往的归约算法优化基本上都是针对其中的一个平台[2]而进行的.本文的主要贡献在于:使用多种存储方式,针对全部数据类型对归约算法在 OpenCL 框架下进行了研究.具体包括:(1) 研究了向量化对性能的影响;(2) 提出了消除存储体冲突的方法;(3) 在两个不同的平台都得到了很好的加速.

本文第 1 节介绍 OpenCL 框架及基本优化策略.第 2 节分 5 个步骤在 OpenCL 框架下针对 minMax 函数进行优化.第 3 节对第 2 节提供的每种优化方法进行详细的性能分析并给出提高性能的原因.第 4 节分别在 AMD GPU 和 NVIDIA GPU 平台上进行测试,测试结果表明,优化后的算法在两个平台上都能实现很好的性能加速.其中,在 AMD ATI Radeon HD 5850 平台上,Int 和 Float 类型数据带宽利用最高可达到实测带宽的 89%.在 NVIDIA GPU Tesla C2050 平台上,性能也可达到 CUDA 版本的相应函数的 1.3~1.9 倍.

1 OpenCL 框架及基本优化策略

OpenCL 编程框架主要分为 4 个部分,即平台模型、内存模型、执行模型和编程模型[4].在 OpenCL 平台模

型中,由一个主机连接一个或多个 OpenCL 设备构成,其中一个 OpenCL 设备中可以有多个计算单元(CU),每个计算单元又可以分为多个处理单元(PE),所有的计算操作均由处理单元完成.OpenCL 设备既可以是 GPU,也可以是 CPU,APU 等.OpenCL 的存储模型分为主机和 OpenCL 设备两部分,OpenCL 设备上的内存分为 4 种,全局内存(global memory)、常量内存(constant memory)、局部内存(local memory)和私有内存(private memory).在读写速度上,私有内存>常量内存>局部内存>全局内存.另外,OpenCL 标准里面还定义了图像对象(image object),图像对象属于全局内存的一种,还有一种是缓冲对象(buffer object),OpenCL 设备可以选择是否支持图像对象.OpenCL 的执行模型也分为两部分,一部分是在 host 端的主程序,另外一部分是在 OpenCL 设备上执行的内核程序(kernel),OpenCL 通过主程序来定义上下文并管理内核程序在 OpenCL 设备上的执行.Host 程序需要为设备上的 kernel 创建一个全局索引空间,全局索引空间可以划分为若干个工作组(work-group)空间,OpenCL kernel 会在每一个工作组空间的每一个节点上执行.OpenCL 编程模型支持数据并行和任务并行两种方式,数据并行模型是指同一系列的指令会作用在不同的内存对象上,即不同内存元素按照这个指令序列定义了同一运算.用户可以利用工作节点的 Global ID 或 Local ID 来映射该工作节点所作用的内存元素.另外,OpenCL 支持向量操作.本文算法所使用的均为数据并行方式.

从 OpenCL 框架的 4 种模型,我们可以得出 OpenCL 框架下的基本优化方法[6−9]:

(1) 选择恰当的存储方式,提高访存效率.OpenCL 中定义了 4 种存储方式,不同存储方式之间差别非常大.以 ATI Radeon HD 5850 为例:全局内存的带宽峰值为 128GB/s,常量内存为 3 341GB/s,局部内存为 1 670GB/s,寄存器带宽为 10 022GB/s.其中,全局内存的容量是 1GB,常量内存容量只有 40KB,局部内存容量为 18×32KB,每个 CU 32KB,总共 18 个 CU.寄存器容量为 18×256KB,每个 CU 有 256KB 寄存器.

(2) 隐藏访存延迟.通过设置恰当的工作组数量和工作组大小,来隐藏访存延迟.在 ATI Radeon HD 5850 上,一个 CU 有 16 个核,每个核是 5 发射超长指令字流水线,工作频率为 725M,一个 wavefront 可以发射 64 个线程,因此,我们可以设置一个工作组为 256 个线程,这样一个工作组可以分成 4 个 wavefront,从而达到隐藏访存延迟的目的.

(3) 使用向量操作.OpenCL 框架提供了向量操作,通过使用向量操作既可以提高访存带宽利用率,也可以提高计算效率.

2 归约算法的优化策略

本节将以 minMax(同时求数组最大值和最小值)为例,详细介绍归约算法的优化策略.

2.1 读取连续数据

如图 1 所示,并行归约一般可以用二分法的方式来实现,考虑到在大部分 OpenCL 设备上相邻线程读取连续数据时会自动地对读取操作进行合并,因此,我们可以利用求最大值、最小值来满足交换律和结合律的特点,让相邻线程尽量读取连续数据,如图 2 所示.

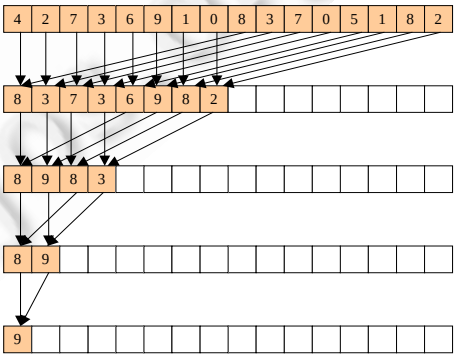

Fig.2 Continous read data binary reduction algorithm

图 2 读取连续数据二分法并行归约算法示意图

第 1 节已指出,选择恰当的存储方式可以提高访存效率.Image 对象是 OpenCL 框架里针对图像算法设计的一种存储方式,该存储方式的特点是对读写的数据会自动地进行缓存.同时,该存储方式读写数据是按照像素的方式来进行读取,一个像素最多可以有 4 个通道,因此,该存储方式一次可以读取一个长度为 4 的向量.与主存交换数据的有 Buffer 对象和 Image 对象两种方式,因此,在我们的算法中,将分别考察 Buffer 和 Image 作为基本存储方式对函数性能的影响.同时,由于进行归约操作时线程之间需要

共享数据,所以我们选择利用局部内存(local memory)来存储中间结果[9],从而实现线程之间的数据共享(图像对象虽然可以自动缓存,但是只有在数据是只读或者只写的情况下才可行).使用该方法后,minMax 函数的性能如图 3 所示.

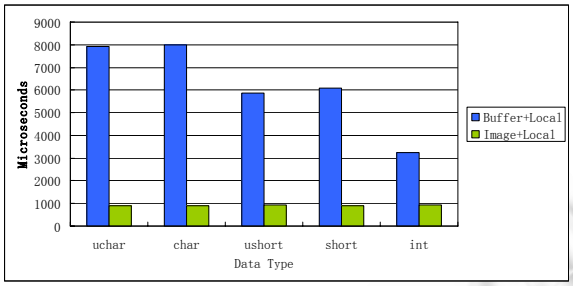

Fig.3 The effect of using Buffer object or Image object for basic memory to minMax function

图 3 分别使用 Buffer 对象和 Image 对象作为基本存储方式对 minMax 函数性能的影响

2.2 向量化及不同向量长度性能对比

OpenCL 框架支持向量操作,向量操作的一个好处在于读写数据时,可以提高带宽利用率.在 ATI 5850 显卡上,尤其是在向显存写数据的时候,当一个线程写入数据长度不是 32 位整数倍时,将走 complete path,这将大大降低写数据的速度,我们可以通过使用向量来弥补这个缺陷.虽然由于归约算法的特点,不需要向显存进行大量的写操作,但是使用向量化来提高计算部件的使用效率还是值得尝试的.我们针对长度为 4,8,16 这 3 种类型的向量均进行了尝试.使用该方法后,minMax 函数的性能如图 4 所示.

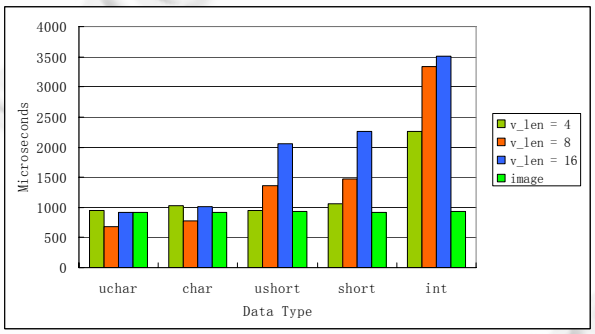

Fig.4 MinMax function performance used vector

图 4 使用向量化之后的 minMax 函数性能

2.3 减少存储体冲突及Local memory使用

从图 4 中可以发现,性能有了很大的提高,但是也发现 Uchar 数据类型在向量长度为 8 时效果最好,不到 700ms.随着向量长度的增加,Ushort 和 Short 类型的性能逐渐变差.在使用局部内存时,有一个非常影响性能的因素,即存储体冲突.ATI Radeon HD 5850 中一个 CU 有 32 个存储体,每个存储体可以读写 32 位数据,ATI Radeon HD 5850 中局部内存中读取数据是以 16 个线程为 1 组,当多个线程读取同一个存储体数据时,就会造成存储体冲突.产生存储体冲突有两种情况,第 1 种是存储体并没有用满而产生的存储体冲突,如使用 Uchar,Char,Ushort, Short 类型进行操作的时候,由于 Uchar 长度不足 32 位,因此会导致相邻线程使用同一个存储体而造成冲突.另外一种情况是因为存储体数量的限制,由于一个 CU 里面只有 32 个存储体,在 ATI 5850 上,当正在运行的 16 个线程同时访问超过 128 个字节的时候,无论如何均会造成存储体冲突.表 1 表示出向量化之后的存储体冲突占 kernel 总运行时间的比例.

从表 1 可以看出,数据类型长度和向量长度的增加均会造成存储体冲突时间的增加.Uchar 和 Char 类型在向量长度为 4 时,没有发生存储体冲突.冲突最严重的是 Int 类型向量长度为 16 时,达到了 33.75%.要减少存储体

冲突,可以通过控制向量长度来实现,但从表 2 可以看出,即使在向量长度为 4 时,Int 类型的冲突时间也超过了 10%.因此,即使控制了向量长度,Int 类型也不一定有好的效果.而且控制向量长度将影响算法的性能.实际上,仔细分析后可以发现,在归约的过程中,之所以需要将数据写入局部内存,是因为在二分法归约时,一个工作组里前一半线程需要使用后一半线程的计算结果,因此只需要将后一半线程的计算结果保存到局部内存即可,前一半线程的计算结果仍然保存在寄存器中,这样即可以降低一半局部内存读写.在使用 Image 对象作为基本存储方式时,局部内存也有非常严重的存储体冲突,该方法也非常有效.减少存储体冲突之后,minMax 函数的性能如图 5 所示.

Table 1 Percentage of bank conflict time in total time (%)

表 1 存储体冲突延迟占总时间比例(%)

向量长度\数据类型	Uchar	Char	Ushort	Short	Int
4	0	0	3.89	3.62	10.11
8	2.78	2.38	8.62	8.03	17.69
16	7.06	5.92	13.04	12.45	33.75

Table 2 Device parameters for ATI Radeon HD 5850 and NVIDIA Tesla C2050

表 2 ATI Radeon HD 5850 和 NVIDIA Tesla C2050 的主要参数

OpenCL device	CU	Cores	Local memory limit (K)	Memory band width (GB/s)	Memory speed (GHZ)	Single precision floating point performance (Tflops)
HD 5850	18	288×5	32	128	1	2.09
Tesla C2050	14	488	48	144	1.5	1.03

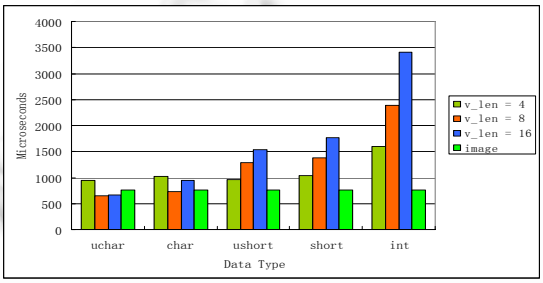

Fig.5 MinMax function performance reduced bank conflict

图 5 减少存储体冲突之后的 minMax 函数性能

2.4 固定启动线程数量

从图 3~图 5 可以看出,到目前为止,一直是使用 Image 对象作为基本存储方式,比 Buffer 对象作为基本存储方式效果好很多.那么为什么还要保留 Buffer 的版本来进行分析呢?实际上,使用 Image 作为基本存储方式有两个缺点,一是 Image 对象即使在计算 Uchar 类型时,也是将 Uchar 转换成一个 Int 来进行计算,非常浪费带宽.另外一个是 Image 对象只能使用长度为 4 的向量.在带宽利用率不是很高的情况下,Image 对象作为基本存储方式的缺点没有体现出来,但是在继续优化之后,就会发现 Buffer 对象作为基本存储方式比 Image 对象作为基本存储方式的速度更快.

在以上的优化中,一直采用的是根据数据量大小来启动线程的数量.这种方式的一个最大的好处是非常直观.实际上,在归约的过程中,有一个非常重要的操作是同步.当数组长度为 2560×2560 时,工作组大小为 256,工作组的数量是 25 600 个,每一个工作组需要进行 7 次同步,总共进行的同步操作是 179 200 次,这是一个非常耗时的操作,因此需要想办法减少同步次数.要减少同步操作的次数,有两种方法,一是不使用二叉树而是使用四叉树来进行归约,从而减少同步数量.二是减少工作组的数量.其实,在 ATI GPU 5850 上,只有 18 个 CU,由于 OpenCL 的一个工作组只能在一个 CU 上运行,因此在真实运行的过程中,同时运行的只有 18 个工作组.所以可以考虑将工作组数量固定为 18,工作组大小为 256.在计算过程中,通过对数据进行折叠,将数据规模降低到 18×256 个线程一次可以运行完成的大小.

这种方法的另外一个好处是,在 GPU 端第 1 次归约完成之后,只剩下少量的数据.第 1 轮归约完成之后,有 3 种方法可以最终完成归约,第 1 种是将第 1 轮归约剩下的数据,重新启动 kernel,继续进行归约,直至归约完成.由于启动 kernel 是一个非常耗时的操作,因此这种方法并不划算.第 2 种方法是使用原子操作来完成最后的归约.第 3 种方法是将数据下载到 CPU 端进行最后的归约,但是下载数据需要消耗时间,因此只适合第 1 轮归约之后数据量非常小的情况.在此之前,在第 1 轮归约完成之后使用的都是原子操作来完成最后的归约,原子操作除了时间非常长以外,还有一个缺点是不能对浮点类型进行操作.在使用固定工作组数量的方法之后,第 1 轮归约完毕,剩下的数据非常少,因此可以将数据下载到 CPU 端进行最后的归约.由于不需要进行原子操作,因此现在可以将 Float 和 Double 类型也加上.如图 6 所示.

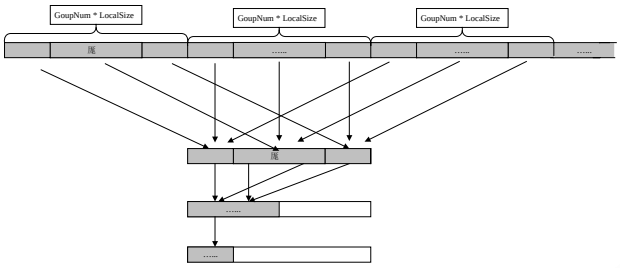

Fig.6 Fix work group reduction algorithm in OpenCL

图 6 固定工作组数量归约算法示意图

另外,考虑到隐藏全局内存访问延迟,尝试着将工作组数量设置为 CU 个数的整数倍.使用该方法后,程序的运行性能如图 7 和图 8 所示.

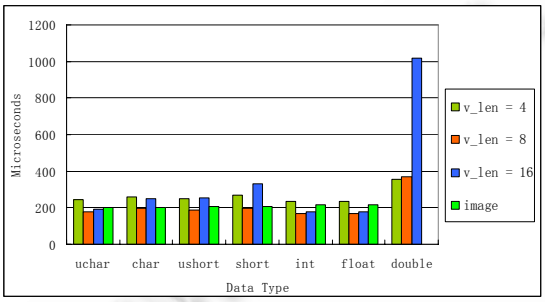

Fig.7 MinMax function performance used 18 work groups

图 7 工作组数量为 18 时 minMax 的性能

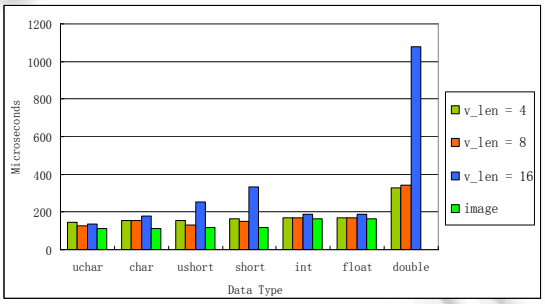

Fig.8 MinMax function performance used 36 work groups

图 8 工作组数量为 36 时 minMax 的性能

2.5 使用内建函数

根据 AMD 优化手册的建议,能使用内建函数的地方尽量使用内建函数[9].OpenCL 框架提供了支持向量的比较函数 min 和 max.图 9 和图 10 展示了使用内建函数之后的性能提升效果,但是该方法只有在 OpenCL 提供相应的内建操作时才有效.此外,我们还尝试了在归约过程中将二分法部分进行循环展开的操作,不过,在 ATI 5850 上,这样做并没有效果.

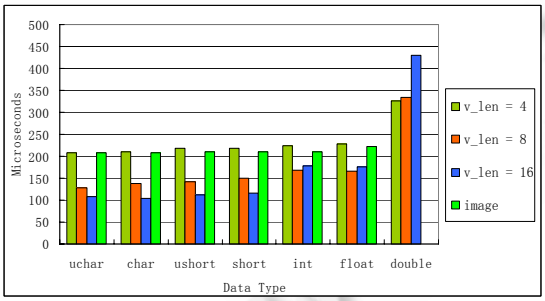

Fig.9 Used build-in function work groups 18

图 9 工作组数量 18 时使用内建函数的 minMax 性能

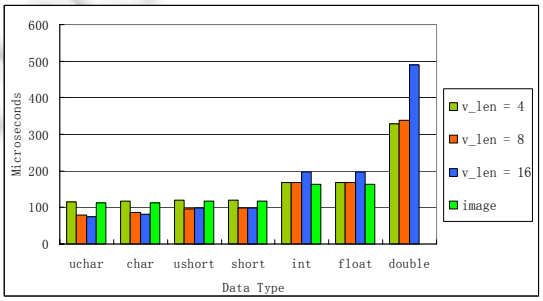

Fig.10 Used build-in function work groups 36

图 10 工作组数量 36 时使用内建函数的 minMax 性能

3 性能分析

本节测试数据规模为 2560×2560 长度的数组,测试平台为 ATI Radeon HD 5850,平台主要参数见表 2.

3.1 读取连续数据

图 3 展示了按图 2 方式分别以 Buffer 对象作为基本存储方式和 Image 对象作为基本存储方式实现的 minMax 函数的性能.图中时间单位为微秒(ms),从图 3 可以看出,随着数据类型的变化,数据类型字节数的增加,时间反而越来越短,Int 类型所花费的时间只有 Uchar 类型所花费时间的 40%左右.实际上,这主要是因为局部内存的存储体冲突导致的.通过 AMD 自带的 sprofile 分析工具可以看出,采用 Uchar,Char 类型时,存储体冲突在 3.6%左右;采用 Ushort,Short 类型时,存储体冲突在 2%左右;而采用 Int 类型时,存储体冲突为 0.当使用 Image 对象作为基本存储方式时,数据类型之间耗时差别并不明显,这是由于 Image 对象读取时,不论是读取 Uchar,Char,Ushort,Short,还是 Int,都将返回一个 Int4 类型的向量.因此,在局部内存里存储的均为 Int 类型的数据.同时可以看到,使用 Image 对象作为基本存储方式时所花费的时间比使用 Buffer 对象时大为减少.观察 Int 类型可以发现,使用 Image 对象作为基本存储方式时大概只有使用 Buffer 对象所花时间的 28%左右,速度提高将近 4 倍.之所以使用 Image 对象会比 Buffer 对象快这么多,除了 Image 对象有缓存以外,还有一个原因是因为 Image 对象使用向量来进行存储和计算,从而大大提高了带宽利用率和计算部件的利用率.由下面的第 3.2 节我们将看到以 Buffer 对象作为基本存储方式加上向量化之后的性能.

3.2 向量化及不同向量长度性能对比

图 4 展示了使用 Buffer 对象作为基本存储方式加上向量化之后 minMax 函数的性能.使用向量之后,Uchar,Char,Ushort,Short 类型的时间大为减少,减少最多的是 Uchar 类型,时间从接近 8 000μs.降低到了不到 1 000μs,速度提升超过 8 倍.对 Int 类型,当向量长度为 4 时,速度提升也非常明显.我们还可以发现,当使用向量长度为 4 时,Uchar,Char,Ushor,Short 类型耗时和使用 Image 对象作为基本存储方式耗时相差不大,但是 Int 类型却比使用 Image 对象时差很多,这可能是因为 Int 类型的数据量相对比较大,因此 Image 对象的缓存对提高访存速度有较大的帮助.

3.3 减少存储体冲突及Local Memory的使用

图 5 展示了减少存储体冲突之后的 minMax 函数性能,对比图 4 和图 5 可以看出,该方法对减少 Int4 和 Uchar16 的存储体冲突效果最为明显.Int4 类型的执行时间从 2 250ms 降到了 1 600ms 左右,减少将近 29%.Uchar16 类型的执行时间从 920ms 降到了 670ms 左右,减少 27%左右,其他类型也均有所减少.使用 Image 对象作为基本存储方式减少存储体冲突之后,时间从 920ms 左右降到了 760ms 左右,时间减少了 17%.通过 sprofile 分析可以发现,使用 Image 对象作为基本存储方式的存储体冲突从 22.5%降到了 1%左右.之所以减少存储体冲突之后,不同类型的速度的提升效果不一致,主要是因为有的类型冲突其程度非常严重,即使使用了该方法之后,仍然存在比较严重的冲突.该方法充分利用了硬件寄存器,在减少局部内存访问的同时也节省了局部内存空间.ATI Radeon HD 5850 的一个 CU 中寄存器大小是 256K,而每个 CU 局部内存大小为 32K,该方法也给我们的算法优化提供了一条思路,即尽量多地使用寄存器来减少访存.当然,如果一个 CU 里使用寄存器的大小超过了 256K,反而会极大地影响函数性能.

3.4 固定启动线程数量

上面的图 7 和图 8 展示了改进线程组织方式之后 minMax 函数的性能.从图中可以看出,采用固定工作组数量之后,性能改进得非常明显.当工作组数量为 36 时,Uchar 类型在向量长度为 8 时最好,耗时只需 126ms 左右.比之前算法的最佳性能 647ms 提高了将近 5 倍,其他类型的时间也大为减少.采用固定线程方法之后,除 Double 类型以外,其他类型之间的时间差别已经不是太大.Double 类型在向量长度为 4 时,性能最好,耗时 326ms.各数据类型 CPU 端归约时间在 3ms~8ms 左右.

3.5 使用内建函数

图 9 和图 10 向我们展示了使用内建函数之后的 minMax 函数性能.Uchar,Char,Ushort,Short 和 Double16 这几种类型速度提升得非常明显,分别提升了 40%~50%左右的性能.因此,在优化过程中,能使用内建函数的地方尽量使用内建函数.

4 跨平台性能评测

表 2 展示了本节用到的两个测试平台 ATI Radeon HD 5850 和 NVIDIA Tesla C2050 的主要参数.

4.1 AMD平台测试结果

通过图 10 可以发现,在工作组数量为 36,向量长度为 8 时,以 Buffer 作为基本存储方式各个类型的性能均比较好,而且在这时它的性能已经超过了以 Image 对象作为基本存储方式的性能.因此我们选择工作组数量为 36,工作组大小为 256,向量长度为 8,以 Buffer 对象作为基本存储方式来作为我们在 ATI Radeon HD 5850 上的最终性能.时间测量使用的是 OpenCL 框架提供的测时接口函数.ATI Radeon HD 5850 显存的理论带宽峰值是 128GB/s,见表 3.但从表 4 可以发现,我们选择的部分类型的带宽利用实际上超过了理论带宽,这主要是因为该型显卡使用的是 DDR5 的内存,DDR5 内存的实际工作频率可以高于理论频率[10].根据 AMD SDK 自带的带宽测试工具,我们测得的该型显卡实际带宽峰值为 164GB/s,因此,在数组长度为 2560×2560 时,最常用的 Int 和 Float 类型的带宽利用分别达到了实测带宽的 88.8%和 88.7%.

Table 3 Time cost optimized minMax function on ATI Radeon HD 5850 (ms)

表 3 经过优化后的 minMax 在 ATI Radeon HD 5850 上的耗时(微秒)

数组长度\数据类型	Uchar	Char	Ushort	Short	Int	Float	Double
1280×1280	30.49	30.58	31.47	32.45	46.77	46.43	102.67
1280×2560	46.77	49.65	54.56	56.27	90.91	91.58	199.89
2560×2560	74.84	86.84	95.30	98.37	167.59	167.65	372.44

Table 4 Bandwidth cost optimized minMax function on ATI Radeon HD 5850 (GB/s)

表 4 经过优化后的 minMax 在 ATI Radeon HD 5850 上的带宽使用量(GB/s)

数组长度\数据类型	Uchar	Char	Ushort	Short	Int	Float	Double
1280×1280	50.04	49.89	96.97	94.04	130.50	131.46	118.89
1280×2560	65.25	61.47	110.87	108.47	134.28	133.29	122.13
2560×2560	81.55	70.28	128.09	124.09	145.67	145.62	131.10

4.2 NVIDIA平台测试结果

在 NVIDIA Tesla C2050 上的测试,我们选择了开源计算机视觉库 OpenCV 中的 CUDA 版本 minMax 函数作为对比.测试数组长度为 2560×2560,以 Buffer 对象作为基本存储方式时选择的向量长度为 8(double 类型向量长度为 4),启动工作组数量为 28,工作组大小为 512,以 Image 作为基本存储方式时启动的工作组数量为 28,工作组大小为 512.测试结果见表 5 和表 6.

Table 5 Time cost optimized minMax function on NVIDIA Tesla C2050 (ms)

表 5 经过优化后的 minMax 在 NVIDIA Tesla C2050 上的耗时(微秒)

测试类型\数据类型	Uchar	Char	Ushort	Short	Int	Float	Double
以 Buffer 对象作为基本存储方式	199.67	199.96	200.99	201.34	313.18	314.81	607.36
以 Image 对象作为基本存储方式	126.65	126.62	192.67	194.68	380.23	381.10	不支持
CUDA	242.14	242.56	269.98	278.88	405.37	404.52	627.98

Table 6 Bandwidth cost optimized minMax function on NVIDIA Tesla C2050 (GB/s)

表 6 经过优化后的 minMax 在 NVIDIA Tesla C2050 上的带宽使用量(GB/s)

测试类型\数据类型	Uchar	Char	Ushort	Short	Int	Float	Double
以 Buffer 对象作为基本存储方式	30.56	30.52	60.73	60.62	77.96	77.55	80.39
以 Image 对象作为基本存储方式	48.19	48.20	63.36	60.70	64.21	64.06	不支持
CUDA	25.20	25.16	45.21	43.77	60.22	60.35	77.75

从表 5 和表 6 可以看出,即使在 Nvidia 的平台上,我们的性能也完全超过了 CUDA,其中以 Image 对象作为基本存储方式在 Uchar 类型的时候,耗时只有 CUDA 的 52%,性能是 CUDA 的 1.9 倍左右.以 Buffer 对象作为基本存储方式在 Int 和 Float 类型的时候,耗时分别只有 CUDA 的 77%和 78%,性能是 CUDA 的 1.3 倍.

本文所述算法也可以用于其他归约函数,我们在 Sum 和 CountNonZero(计算数组中非零元素个数)这两个函数中采用本文所述方法,效果均很理想.Sum 函数在 ATI 平台上,数组长度为 2560×2560,当数据类型为 Uchar,Char 时,时间大概在 60ms 左右;当数据类型为 Short,Ushort 时,时间大概在 90ms 左右;当数据类型为 Int,Float 时,时间在 160ms 左右,均略快于 minMax.

5 结束语

本文以 minMax 函数为例,详细叙述了在 OpenCL 框架下,归约函数的优化策略,并最终获得了非常好的性能.在 ATI GPU 5850 平台上,本文算法在常用的 Int 和 Float 类型上带宽利用率达到了实测带宽的 89%左右,即使进行改进,空间也不大.在 NVIDIA Tesla C2050 平台上,性能也达到了 OpenCV 官方库中 minMax 函数的 1.3~1.9 倍.

在实验过程中我们还发现一个现象,即当使用 Image 对象作为基本存储方式的时候,在 ATI 平台,数据类型对时间的影响并不明显,但在 Nvidia 平台上,随着数据类型字节数的增多,会导致时间开销变长.之所以会出现这种现象,可能是由于 ATI 显卡和 Nvidia 显卡 Image 对象的实现方法不同所致,具体细节还有待研究.本文主要阐述的是以二分法为基础的归约,实际上,我们也可以尝试用其他,如四分法等方式来进行归约,这样可以有效地减少同步,但是由于时间限制,我们暂时没有进行尝试.未来我们希望能将本文所阐述的算法在其他 OpenCL 平台,如 CPU,APU,Cell 等处理器上进行测试,进一步验证本算法的性能.

References:

[1] Owens JD, Houston M, Luebke D, Green S, Stone JE, Phillips JC. GPU computing. In: Proc. of the IEEE 96. 2008. 879−899.

[2] Harris M. Optimizing parallel reduction in CUDA. Nvidia Developer Technology. 2007.

[3] Stone JE, Gohara D, Shi G. OpenCL: A parallel programming standard for heterogeneous computing systems. Comput. Sci. Eng., 2010,12:66−73.

[4] 陈钢,吴百锋.面向 OpenCL 模型的 GPU 性能优化.计算机辅助设计与图形学学报,2011,23(4).

[5] Karimi K, Dickson NG, Hamze F. A performance comparison of CUDA and OpenCL. 2010. http://arxiv.org/abs/1005.2581

[6] AMD 上海研发中心.跨平台的多核与众核编程讲义——OpenCL 的方式.上海,2010.

[7] NVIDIA. NVIDIA OpenCL Best Practice Guide Version 1.0. 2010.

[8] Khronos OpenCL Working Group. Aaftab Munshi. The OpenCL Specification v1.1. 2011.

[9] Advanced Micro Devices. AMD Accelerated Parallel Processing Programming Guide OpenCL. 2011.

[10] Kho R, *et al*. A 75 nm 7 Gb/s/pin 1 Gb GDDR5 graphics memory device with bandwidth-improvement techniques. In: Proc. of the IEEE ISSCC Dig.Tech. 2009. 134−135.

颜深根(1986－),男,湖南湘潭人,硕士,CCF 学生会员,主要研究领域为异构并行编程.

龙国平(1982－),男,博士,助理研究员,CCF 会员,主要研究领域为计算机体系结构,并行编程,性能模型和评估.

张云泉(1973－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为大型并行数值软件,并行程序设计和性能评价,并行计算模型,并行数据挖掘.

李焱(1985－),男,博士,CCF 学生会员,主要研究领域为自适应性能优化,异构并行编程.