GPU 上稀疏矩阵与矢量乘积运算的一种改进①

马　超[1]　韦　刚[1]　裴颂文[2]　吴百锋[1]

(1.复旦大学 计算机科学技术学院 上海 200433;2.上海理工大学 计算机科学工程系 上海 2000933)

摘　要： 稀疏矩阵和矢量的乘积运算在工程实践及科学计算中经常用到，随着矩阵规模的增长，大量的计算限制了整个系统的性能，因此可以利用 GPU 的高运算能力加速 SpMV。分析了现有 GPU 上实现的 SpMV 存在的问题，并设计了行分割优化和 float4 数据类型优化两种方案。实验表明，该方案可以使性能提升 2～8 倍。

关键词： GPU；稀疏矩阵；CSR；CUDA

Improvement of Sparse Matrix-Vector Multiplication on GPU

MA Chao[1], WEI Gang[1], PEI Song-Wen[2], WU Bai-Feng[1]

(1. School of Computer Science, Fudan University, Shanghai 200433, China; 2. Department of Computer Science and Engineering, University of Shanghai for Science and Technology, Shanghai 200093, China)

Abstract: Sparse Matrix-vector multiplication (SpMV) is one of the most frequently used kernels in engineering practice and scientific computing. With the growth of the scale matrix, a large number of calculations restrict the performance of system, so SpMV can be accelerated by utilizing the high computing power of GPU. In this paper, the problem of existing SpMV on GPU is analyzed. Besides, row partition optimization and float4 optimization are designed. Experimental results demonstrate that the proposed approach can enhance the performance by 2-8 times.

Keywords: GPU; sparse matrix; CSR; CUDA

近年来，受消费者对实时、高清晰度 3D 图形需求的驱使，可编程的图形处理器(GPU)的峰值处理能力以每年 2.5~3 倍的速度增长，远高于 CPU 的增长速度，已经发展成为一种具有巨大的计算能力和存储器带宽的高度并行的多线程处理器。正由于 GPU 内部拥有大量的高度并行化的运算资源和高的存储带宽，加上 AMD 公司的 Steam SDK,NVIDIA 公司的 CUDA 模型及 OPENCL[1]标准等对 GPU 的支持，其可编程性逐渐增强，所以被广泛的用于非图形处理的领域，尤其是科学计算领域。

稀疏矩阵与矢量的乘积运算(SpMV)的性能在计算科学中具有十分重要的地位，它的时间开销是整个系统性能的关键。稀疏矩阵的存储格式是影响 SpMV 性能的一个关键因素，最经常用到的稀疏矩阵的存储格式是行压缩[2](Compressed Sparse Row，CSR)格式，CSR 格式是把一个二维矩阵压缩到一维数组中，如图 1 所示，对于 5X5 的矩阵，可以把全部非 0 元素记录在数组 elem 中，然后用数组 rowptr 记录每行上第一个非零元素在数组 elem 中的索引，每个非零元素对应的列坐标记录在 col 数组中。基于 CSR 格式的 SpMV 具有高度的并行性，因此可以使用 GPU 对 SpMV 进行并行加速。但是 GPU 只适合于计算密集型的应用，而 SpMV 中访问显存的指令数目与计算指令数目的比值却比较高，所以基本的 GPU 实现的 SpMV 相对于串行实现的加速比并不高。

文献[2,3]针对 CSR 压缩格式的稀疏矩阵，从多个方面对并行的 GPU 实现的 SPMV 进行了优化，使其性能得到提升，本文在文献[3]的基础上进行再优化，使

① 收稿时间:2009-09-10;收到修改稿时间:2009-10-25

性能提升 2~8 倍。

$$A=\begin{bmatrix}1&-3&0&-1&0\\0&0&-2&0&3\\2&0&0&0&0\\0&4&0&-4&0\\5&0&-5&0&6\end{bmatrix}$$

下标	0	1	2	3	4	5	6	7	8	9	10
rowptr	0	3	5	6	8	11					
col	0	1	3	2	4	0	1	3	0	2	4
elem	1	-3	-1	-2	3	2	4	-4	5	-5	6

图 1　5X5 的矩阵及其 CSR 格式

本文其余部分组织如下：第 1 节介绍现有对并行 SpMV 优化的相关工作；第 2 节提出优化的方案；第 3 节进行性能分析。最后是对全文的总结及对今后研究工作的展望。

1　相关工作

多核平台的普及使人们对 SpMV 优化的方向发生了变化，更多的针对多核平台的特性进行优化。GPU 由多个流式多处理器(Streaming Multiprocessor, SM)组成，每个多处理包含 8 个标量处理器(Scalar Processor, SP)核心和 2 个特殊功能单元。SM 利用了单指令多线程的架构来管理 GPU 上的多个线程，SM 将各个线程映射到 SP 上，SP 使用自己的指令地址和寄存器状态独立执行。多处理器以 warp 为单进行线程的创建、管理、调度和执行，其中每个 warp 含有 32 个并行的线程。GPU 的存储结构复杂，各级存储访问延时差别很大，如图 2 所示，它包含：寄存器、共享存储器、纹理和常量缓存、纹理长常量存储器、全局存储器。而其中的寄存器、共享存储器和纹理常量的缓冲位于片内，访问速度比片外的存储器快很多。对全局存储器的访问非常耗时，但是一个 half-warp(16 个线程)内线程对全局内存并行访问在一定条件下可以被合并成对整块内存区域的访问。

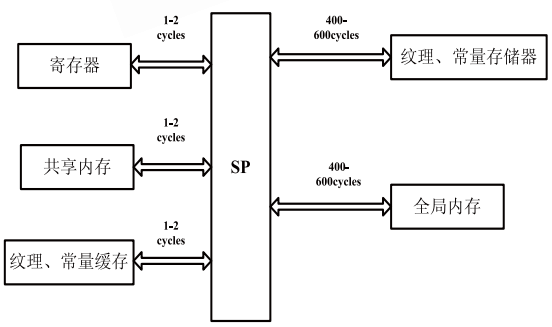


图 2　GPU 的存储器延时

在稀疏矩阵以 CSR 为存储格式的 SpMV 中，由于矩阵的每个非零元素的计算都需要从存储器中读出其对应的值、向量元素和列值，而且相对于算术指令 4～20 个时钟周期的延时，访问全局内存的延时高达 400～600 个时钟周期，所以优化的一个突破点就是降低访问全局内存的时间开销。CUDA 模型中，一个 half-warp 内的线程对全局内存的并行访问可以通过合并全局内存访问(coalesced Global Memory Access)的方式，即把连续的数据组织成块进行存取，这种方式使得整个 half-warp 的线程对全局内存访问的延时降低为原来的 1/16 左右,但是它的启用必须要满足以下几个条件[4]：1)half-warp 内的线程访问的数据长度为 4，8 或 16 字节；2)被访问地址构成一片连续的内存空间；3)起始全局内存地址对齐到访问的数据长度的 16 倍；4)第 n 个线程访问其对应的第 n 个地址。为了满足上述条件，文献[2]通过为每行分派一个 Warp 的方式，即 32 个线程共同计算目标向量的一个元素，使得部分的全局内存访问满足了合并的条件，但是这产生了两个新问题：首先，对于大多数行的非零元素个数都远低于 32 的矩阵，如第 4 节性能分析用到的矩阵 G2_circuit.mtx，大多数的 SP 处于空转状态，浪费大量的计算资源；其次，除第一行而外，其他所有行对应的 half-warp 都可能不满足条件 3)，只能进行非合并的全局内存访问。

编译时优化(Compile－time optimizations)[3]通过为每行分派一个 half-warp 的方式映射线程，并且对 CSR 格式进行适当的修改，把每个非零行补零对齐，使每行元素个数为 16 的倍数，任意一个 half-warp 内的线程对全局内存的访问都满足合并访问的条件。在 SpMV 中，不同的线程对向量元素的访问可能会重复，因此将向量绑定到纹理存储上，利用纹理存储器的缓存(Texture Memory Cache)，通过数据复用的方式进一步优化。上述的这些工作使 SpMV 的在 Geforce 8800GTX 上的性能比 CUDPP(CUDA Data Parallel Primitives Library)和分段扫描执行(segmented scan implementation)[5]方式都提升了 2～8 倍，相对于 NVIDIA SpMV 库也有最高为 15%的性能提升。

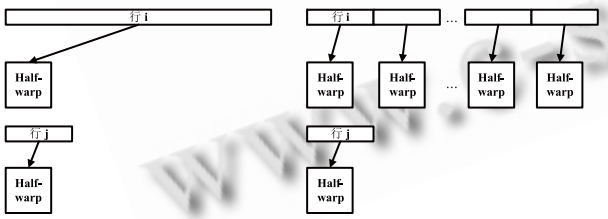
2　优化方案

本文主要针对第 2 节中提到的两种情况进行优

化：1、非零元素最多的行与最少的行的非零元素个数相差很大；2、大多数行的非零元素个数都很少。

2.1 行分割优化

行分割优化是通过均衡负载的方式，为每个 SP 分配等量的计算任务，减少因负荷不均衡造成的等待时间，进而达到缩短整个系统的运行时间的目的。基于第 2 节中对 SpMV 的分析可以知道，对于非零元素最多的行与最少的行的非零元素个数相差很大的情况，为非零行分派固定数量的线程，则非零元素多的行对应的线程的时间开销要高于非零元素少的行对应线程的时间开销，大量的 SP 处在空转状态，整体性能受限。本方案对分配到不同 SP 上的不均匀任务进行重新分配，以文献[3]为基础，在已经进行补零对齐的情况下，为每一行的计算分派不同数量的 SP，任意行分配的 SP 数量依赖于该行的非零元素个数。这种方式是把非零元素较多的行分割成多个等长的计算任务进行计算，然后进行合并。图 3 所示为两行计算任务采用行分割优化进行负载均衡前后线程分配的对比情况，均衡负载前，第 i 行的计算任务被分配到一个 half-warp 上，而负载均衡后，第 i 行的计算任务被分割成多个等长的计算任务并把每个计算任务分配到一个 half-warp 上。任务分割的依据是计算任务最小的行，这里假定第 j 行的非零元素个数最少，为计算任务最小的行，那么第 i 行对应计算任务的时间开销接近于第 j 行对应的计算任务的时间开销，差值仅仅为少量合并操作的时间开销。



(a) 均衡负载前线程分配情况　　(b) 均衡负载后线程分配情况

图 3　均衡负载前后的线程分配对比

假设补零后元素个数最少的行元素个数为 16，如图 4 所示为一个补零对齐后元素个数为 32 的行，该行的计算需要分配一个 warp，每个线程计算对应元素与向量元素的乘积，用 SM 片上的共享内存存储结果，然后通过并行求和的方式对连续的 16 个线程的计算结果进行累加，写入全局内存。Kernel 2 就是对 kernel 1 中对应于每一行的计算结果进行再次累加，求得目标向量。

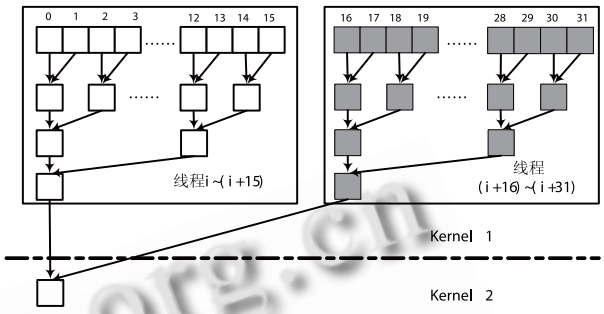


图 4　补零对齐后元素个数为 32 的行的计算过程

行分割优化是通过均衡负载的方式间接提升 SpMV 的性能，均衡负载的优化方式不仅适用于本文的 SpMV，它是利用多核平台对算法进行并行加速时的重要优化策略之一。

2.2 float4 数据类型优化

对于大多数行的非零元素都很少的情况，采用编译时优化和行分割优化都需要对非零行进行 16 的倍数补零对齐，而此时因补零对齐造成的额外计算对性能影响很大，因此在需要考虑补零对齐的规模，如 8 的倍数或 4 的倍数补零对齐方式。而且从前文的分析可知，采用 16 的倍数补零对齐方式的原因是在于映射线程的需要，所以若采用了 8 或 4 的倍数的补零对齐方式，就需要改变线程的映射方式。

本方案利用 CUDA 对 float4 类型数据的支持，把连续的 4 个 float 类型的数据组织成一个，通过为每个非零行分配多个线程，每个线程负责对应一个 float4 数据与对应矢量元素的乘积并求和，再调用另外一个内核函数求得目标矢量。这种线程映射方式使所有的 half-warp 对全局内存的访问满足合并访问全局内存条件，并且由于 float4 数据类型的引入，每个线程的计算量提升为原来的四倍，而访问全局内存次数并没有增加，所以计算指令相对于全局内存访问指令的比例有所提高，更利于存储器访问延时的隐藏。如图 5 所示为 8 的倍数补零对齐后元素个数为 16 的行的计算过程，kernel 1 中为其分配 4 个线程，每个线程把对应的 float4 类型数据的计算结果写入片上共享内存，由于行间采用的是 8 对齐方式，所以需要进

行一次合并，然后写入全局内存，kernel 2 根据每行在修改的 CSR 格式中的位置进行累加，求得结果。

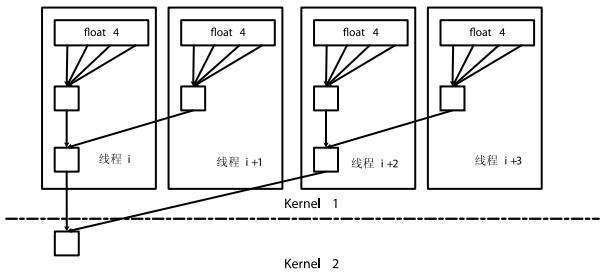


图 5　补零对齐后元素个数为 16 的行的计算过程

行分割优化方案和 float4 数据类型优化方案都是通过行补零对齐，改变线程映射的方式提升性能，各自有其适用的范围，行分割优化适用于行非零元素个数相差较大的矩阵，而 float4 数据类型优化适用于行非零元素个数普遍少于 8 或 4 的矩阵，因此需要动态的进行选择。本文是通过判断行平均非零元素个数的方法选择合适的方案。

3　性能分析

本文的实验是在型号为 NVIDIA GeForce 9800 GT 的 GPU 上进行的， NVIDIA GeForce 9800 GT 拥有 14 个 SM，每一个多处理器含有 8 个 SP，8096 个寄存器和 16KB 的片上共享内存。它拥有峰值为 378GFLOPS 的计算能力，57.6GB/s 的存储器带宽和 512MB 的存储容量。测试所用的矩阵选自各个领域的真实数据[6]，矩阵数据的规模如表 1 所示。

表 1　测试矩阵集

矩阵名	行数	列数	非零元素个数
rajat27	20640	20640	99777
Pre_poisson	14822	14822	365313
Wang3	26064	26064	177168
Cage12	130228	130228	2032536
G2_circuit	150102	150102	438388
baumann	112211	112211	760631

优化后的性能与编译时优化[3]性能比较如图 6 所示，本文采用的两种优化方案：行分割优化和 float4 数据类型的优化都比编译时优化在运行时间上都有所减少，而且二者之间的相对性能因不同测试数据也有所区别。这两种方案的适用范围分别为：1、非零元素最多的行与最少的行的非零元素个数相差很大的情况；2、大多数行的非零元素个数都很少的情况。然而这两个情况存在冲突，可能因为一个条件的满足而限制了另外一种优化方案的性能，如测试矩阵 rajat27.mtx，它的非零元素分布极不均匀，16 倍数补零对齐后，元素个数最少行的元素个数为 16，而最多行的元素个数 1680，但非零行元素个数普遍低于 8，只有少量非零元素较多的行，采用方案一会因过多的额外计算限制性能，采用方案二会因负载极不均衡导致性能下降，所以相对编译时优化的性能只能达到 2 倍左右的加速比。测试矩阵 Cage12.mtx、Wang3.mtx 和 Pre_poisson.mtx 的特点是补零对齐后元素最少行和最多行元素个数相差不大，而且行均非零元素个数普遍高于 8，所以本文采用方案相比编译时优化的加速比并不高，为 2~4 倍。测试矩阵 G2_circut.mtx 和 baumann.mtx 的特点是每行的非零元素都很少，补零对齐前行平均非零元素个数小于 8，本文优化方案相对编译时优化达到 6~8 倍的加速比，其中的测试矩阵 G2_circut.mtx 因行非零元素个数普遍低于 4 使 float4 数据类型优化能达到 8 倍左右的加速比。

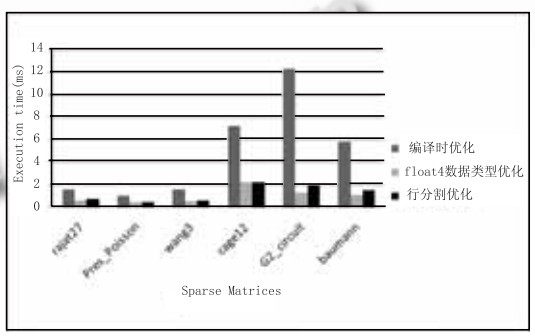


图 6　性能比较

尽管本文采用的优化方案是为特殊情况设计，但是在两种情况都不满足时，如测试矩阵 Cage12.mtx 等，两种方案相对于编译时优化仍然有一定的性能提升，这是因为于编译时优化中一个 half-warp 对应一行的计算任务，这个 halfwarp 内所有线程要访问 CSR 格式中每行第一个非零元素的索引值的两次，并且由于访问的位置相同导致必须顺序执行，而本文的两个方案都因其特定的线程映射方式在上述问题上有所改善。

4 总结与展望

本文是利用 CUDA 编程模型特殊的多级存储结构和多种访问模式，对科学计算中经常遇到的 SpMV 问题进行分析，针对不同种类的稀疏矩阵，优化已有 GPU 上实现的并行 SpMV 优化算法，优化的主要工作集中在修改稀疏矩阵的存储格式和改进线程映射的方式两方面。理论分析和实验都表明了所述方案对 SpMV 性能提升上的有效性。科学计算领域有很多问题受限于计算复杂度，尽管其中不少已经通过多核平台进行加速，但加速比并不理想，是因为多核平台的编程受限于其特性。因此深入的了解 GPU 的体系结构对算法的优化很有帮助。今后将会优化科学计算领域其他的算法。

参考文献

1 Open Computing Language (OpenCL). [2009-08-05] http:www.khronos. org/opencl/.

2 Bell N, Garland M. Efficient sparse matrix-vector multiplication on CUDA. NVIDIA Technical Report NVR-2008-004, December 2008.

3 Muthu Manikandan Baskaran, Rajesh Bordawekar. Optimizing Sparse Matrix-Vector Multiplication on GPUs, IBM Technical Report RC24704. 2008.

4 Harris M. High Performance Computing with CUDA-Optimizing CUDA, Super-computing Tutorials (2007) [2009-08-05]. http://gpgpu.org/sc2007.

5 Sengupta S, Harris M, Zhang Y, Owens JD. Scan primitives for gpu computing. GH'07: Proc. of the 22nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics hardware, 2007. 97－106.

6 Davis T. The University of Florida Sparse Matrix Collection.[2009-08-05] http://www.cise.ufl.edu/research/sparse/matrices/