计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2015.20131919
52(4):843-850, 2015

GPU 加速不完全 Cholesky 分解预条件共轭梯度法

陈 尧[1,2] 赵永华[1] 赵 慰[1] 赵 莲[1]
[1](中国科学院计算机网络信息中心 北京 100190)
[2](中国科学院大学 北京 100190)
(cheny_1988@126.com)

GPU-Accelerated Incomplete Cholesky Factorization Preconditioned Conjugate Gradient Method

Chen Yao[1,2], Zhao Yonghua[1], Zhao Wei[1], and Zhao Lian[1]
[1](*Computer Network Information Center, Chinese Academy of Sciences, Beijing* 100190)
[2](*University of Chinese Academy of Sciences, Beijing* 100190)

Abstract Incomplete Cholesky factorization preconditioned conjugate gradient (ICCG) method is effective to solve large sparse symmetric positive definite linear systems. However, ICCG method requires solving two sparse triangular linear systems during each iteration. The inherent serialism of solving sparse triangular becomes a bottleneck which prevents high efficient parallelization of ICCG method on GPU platform. In this paper, an effective method to accelerate solving sparse triangular on GPU platform is proposed. In order to increase the multi-thread parallelism of solving sparse triangular on GPU platform, level scheduling is exploited for the sparse triangular matrixes which incomplete Cholesky factorization generates. For further improving the parallel performance of solving sparse triangular, approximate minimum degree (AMD) algorithm is used to reorder the coefficient matrix before level scheduling. Moreover, a novel method, taking advantage of the level information to reorder the sparse triangular matrices after level scheduling, is applied. These two methods can decrease the number of levels during level scheduling and optimize GPU memory access pattern to utilize memory coalescing in solving sparse triangular, respectively. Numerical experiments indicate that compared with ICCG method implemented with NVIDIA CUSPARSE, applying the above methods can obtain more than 100% performance improvement on average.

Key words incomplete Cholesky factorization; preconditioner; conjugate gradient method; reordering; graphic processing unit (GPU)

摘 要 不完全 Cholesky 分解预条件共轭梯度(incomplete Cholesky factorization preconditioned conjugate gradient, ICCG)法是求解大规模稀疏对称正定线性方程组的有效方法.然而 ICCG 法要求在每次迭代中求解 2 个稀疏三角方程组,稀疏三角方程组求解固有的串行性成为了 ICCG 法在 GPU 上并行求解的瓶颈.针对稀疏三角方程组求解,给出了一种利用 GPU 加速的有效方法.为了增加稀疏三角方程组求解在 GPU 上的多线程并行性,提出了对不完全 Cholesky 分解产生的稀疏三角矩阵进行分层调度(level scheduling)的方法.为了进一步提高稀疏三角方程组求解的并行性能,提出了在分层调度前

收稿日期:2013-12-18;**修回日期**:2014-05-14
基金项目:国家自然科学基金项目(60873113);国家"九七三"重点基础研究发展计划基金项目(2011CB309702);国家"八六三"高技术研究发展计划基金项目(2012AA01A309);国家自然科学基金重大研究计划项目(91430214);数学工程与先进计算国家重点实验室开放基金项目(2014A03)
通信作者:赵永华(yhzhao@sccas.cn)

通过近似最小度(approximate minimum degree, AMD)算法对系数矩阵进行重排序、在分层调度后对稀疏三角矩阵进行层排序的方法,降低了分层调度过程中产生的层数,优化了稀疏三角方程组求解的GPU内存访问模式.数值实验表明,与利用NVIDIA CUSPARSE实现的ICCG法相比,采用上述方法性能可以获得平均1倍以上的提升.

关键词 不完全Cholesky分解;预条件;共轭梯度法;重排序;图形处理器

中图法分类号 TP338.6

在科学与工程计算中,求解大型稀疏线性方程组 $\boldsymbol{Ax}=\boldsymbol{b}$ 是一个基本问题,包括共轭梯度法(conjugate gradient method, CG)、稳定双共轭梯度法(biconjugate gradient stabilized method, BiCGSTAB)和广义极小残差算法(generalized minimal residual algorithm, GMRES)在内的许多方法常被用来求解此类问题.但在实际应用中,直接使用这些方法的收敛速度较慢,因此需要与预条件技术相结合来改善系数矩阵的条件数,加快迭代求解的收敛速度.预条件技术可以分为隐式预条件和显式预条件,包括IC/ILU和SSOR在内的隐式预条件主要通过计算 $\boldsymbol{A}$ 的近似分解得到;而显式预条件通常需要计算 $\boldsymbol{A}$ 的近似逆[1-3].隐式预条件具有良好的健壮性并且可以极大减少迭代求解的次数,但其固有的串行性使得它难以实现高效的并行.

在过去的几年中,图形处理器(graphic processing unit, GPU)已经逐渐发展成为灵活强大的众核处理器.而随着NVIDIA CUDA的发布,大规模并行GPU架构所提供的强大性能已经能够为众多的高性能计算应用所利用.很多数值算法都能通过GPU获得明显的加速,其中最典型的例子莫过于稠密线性代数问题.而对于基于GPU的稀疏线性系统的迭代求解器,Li和Saad[4]总结了多种可用的预条件技术;Naumov[5]展示了如何利用CUSPARSE和CUBLAS实现IC/ILU预条件CG/BiCGSTAB算法.在应用方面,Sudan等人[6]将块-ILU预条件GMRES算法应用于油藏数值模拟;Gupta[7]将deflated PCG算法应用于泡状流计算.在国内,张健飞等人[8]研究了在利用CUSPARSE实现的ICCG法中优化稀疏矩阵向量乘的方法;吴长江[9]研究了将CG算法和Jacobi预条件CG算法移植到GPU上的方法.

使用不完全Cholesky分解预条件共轭梯度(incomplete Cholesky factorization preconditioned conjugate gradient, ICCG)法求解稀疏对称正定线性方程组:

$$\boldsymbol{Ax}=\boldsymbol{b}, \tag{1}$$

需要计算稀疏对称正定矩阵 $\boldsymbol{A}$ 的不完全Cholesky分解 $\boldsymbol{LL}^{\mathrm{T}}$,其中 $\boldsymbol{L}$ 是一个稀疏下三角矩阵,而 $\boldsymbol{M}=\boldsymbol{LL}^{\mathrm{T}}$ 通常被称为 $\boldsymbol{A}$ 的不完全Cholesky分解预条件.在得到 $\boldsymbol{A}$ 的不完全Cholesky分解预条件后,ICCG法的每次循环迭代都需要一个预条件操作 $\boldsymbol{LL}^{\mathrm{T}}\boldsymbol{z}=\boldsymbol{r}$,该操作首先通过向前代换求解稀疏下三角方程组 $\boldsymbol{Lu}=\boldsymbol{r}$ 得到向量 $\boldsymbol{u}$,然后利用向后代换求解稀疏上三角方程组 $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{z}=\boldsymbol{u}$ 得到向量 $\boldsymbol{z}$.

由于稀疏三角方程组求解固有的串行性,使得ICCG法在GPU上难以得到好的并行性能.本文提出了一种在GPU上加速ICCG法的有效方法,通过对IC分解产生的稀疏三角矩阵进行分层调度(level scheduling)来增加稀疏三角方程组求解在GPU上的多线程并行性.为了进一步提高稀疏三角方程组求解的性能,本文还提出了在分层调度前和分层调度后分别引入稀疏矩阵的近似最小度(approximate minimum degree, AMD)排序[10]和三角矩阵的层排序,极大地降低了分层调度过程中产生的层数,优化了稀疏三角方程组求解的GPU内存访问模式.数值实验表明,相对于利用NVIDIA CUSPARSE实现的ICCG法,本文提出的方法可以获得平均1倍以上的性能提升.

1 基于CPU+GPU的IC分解预条件CG方法

在GPU上使用IC预条件CG方法求解大型稀疏对称正定方程组时,稀疏三角方程组求解是影响整个并行性能的关键.为了增加稀疏三角方程组在GPU上求解的并行度,一种有效的方法是使用分层调度算法对稀疏三角矩阵分层,而分层后层数的多少就成为了影响并行效率的关键,通常层数越少在GPU上求解时的并行效率越高.这样,为了减少稀疏三角矩阵在分层过程中产生的层数,可以在分层调度前使用AMD算法对系数矩阵进行重排序.同时,为了优化全局存储器(global memory)的内存访问模式,可以对分层后的稀疏三角矩阵采用层排序,以

此增加在 GPU 上多线程内存访问的连续性. 图 1 给出了本文提出的在 CPU+GPU 上通过 IC 预条件 CG 迭代法求解稀疏对称正定方程组的流程图,图 1 中标有 CPU 的部分可在 CPU 上进行.

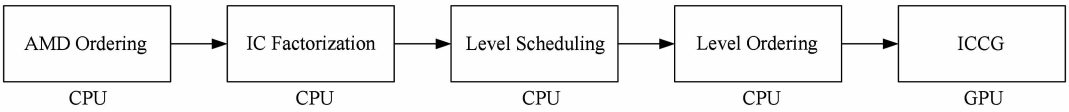


Fig. 1　Flow chart of GPU-accelerated ICCG method.

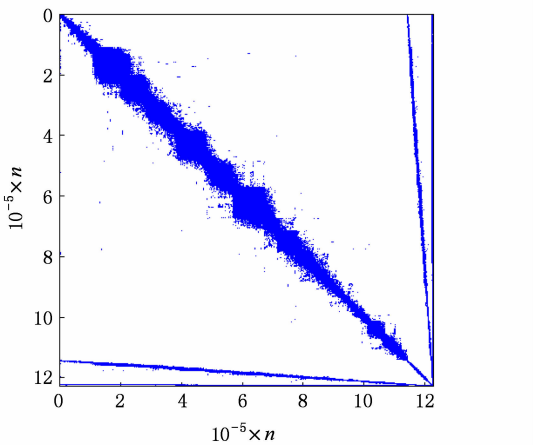
图 1　GPU 加速 ICCG 法的流程图

1.1　稀疏矩阵的 AMD 排序

对系数矩阵采用重排序技术,可以将稀疏矩阵的非零元素集中在一定的区域内,这样能够有效减少分层调度过程中产生的层数,提高并行度. 重排序之后,原有线性系统转换为

$$\boldsymbol{PAP}^{\mathrm{T}} \cdot \boldsymbol{Px} = \boldsymbol{Pb}. \tag{2}$$

重排序是 NP 完全问题,只能采用启发式算法,其中最小度排序[11]是最常用的启发规则,本文采用 AMD 软件包中实现的 AMD 排序算法. 图 2 展示了测试集中的矩阵 ***thermal2*** 在使用近似最小度排序之前和之后的稀疏模式变化示意图,其中 n 为矩阵阶数,横坐标为列,纵坐标为行.



(a) Sparse pattern of ***thermal2*** before AMD ordering

(b) Sparse pattern of ***thermal2*** after AMD ordering

Fig. 2　Sparse pattern of ***thermal2*** before and after AMD ordering.

图 2　AMD 排序之前和之后 ***thermal2*** 的稀疏模式

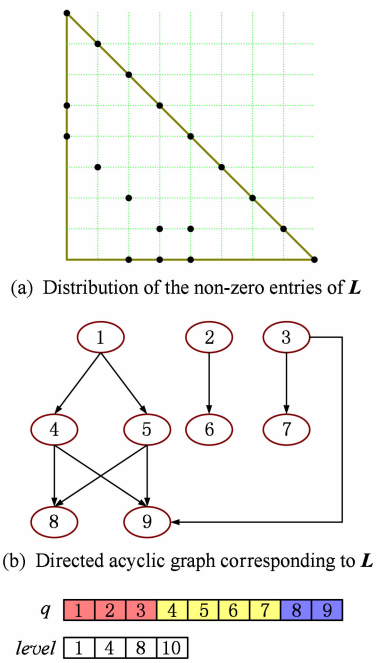
1.2　稀疏三角矩阵分层调度算法

对于稀疏三角方程组 $\boldsymbol{Lx}=\boldsymbol{b}$(这里假设 $\boldsymbol{L}$ 是下三角矩阵),未知量 $\boldsymbol{x}_i$ 的求解对应下列等式:

$$\boldsymbol{x}_i = (\boldsymbol{b}_i - \sum_{k\in S} \boldsymbol{l}_{ik}\boldsymbol{x}_k)/\boldsymbol{l}_{ii}, \tag{3}$$

其中,集合 S 是矩阵 $\boldsymbol{L}$ 的第 i 行中除对角元以外的非零元素所对应的列下标的集合,$\boldsymbol{l}_{ik}$ 表示矩阵 $\boldsymbol{L}$ 的第 i 行、第 k 列的非零元素,$\boldsymbol{l}_{ii}$ 表示第 i 行的主对角元素. 在式(3)中,只有所有其他的未知量 $\boldsymbol{x}_k$ 被确定后,才能求解未知量 $\boldsymbol{x}_i$. 为了在 GPU 上得到更好的多线程并行性能,需要对矩阵 $\boldsymbol{L}$ 进行分层调度.

稀疏三角矩阵分层调度包括分析阶段和求解阶段 2 个过程. 分析阶段用来将未知量分配到不同的层中,使得同一层中的未知量能够被同时求解;而求解阶段用来逐层求解 $\boldsymbol{x}$ 中的各未知量. 图 3 是一个



(a) Distribution of the non-zero entries of $\boldsymbol{L}$

(b) Directed acyclic graph corresponding to $\boldsymbol{L}$

(c) The entries of the same level and the starting indices

Fig. 3　Level scheduling.

图 3　分层调度

稀疏三角矩阵 $\boldsymbol{L}$ 的分层调度示意图. 其中,图 3(a)表示矩阵 $\boldsymbol{L}$ 中各非零元素的分布情况,每个黑点代表矩阵的一个非零元素. 图 3(b)表示对矩阵 $\boldsymbol{L}$ 分层后得到的有向无环图,其中每个带数字的椭圆圈对应一个未知向量,如第 i 个椭圆圈对应未知量 $\boldsymbol{x}_i$;带箭头的有向边表示各未知量之间求解时的依赖关系,如由 j 指向 i 的有向边表示未知量 $\boldsymbol{x}_j$ 必须先于未知量 $\boldsymbol{x}_i$ 被求解. 图 3(c)表示向量 $\boldsymbol{x}$ 中处于同一层的各未知量序号以及各层所对应的位置,其中 q 表示一个整型数组,用来记录向量 $\boldsymbol{x}$ 中各未知量按层排序后的序列,阴影表示处于同一层的各未知量序列号;而层指针数组 $level$ 用来确定数组 q 中各层的起始位置,如 $level(i)$ 表示数组 q 中第 i 层的起始位置.

为了确定各未知量所在的层,首先通过有向图得到图中不依赖于任何未知量的未知量,即 $\boldsymbol{L}$ 中非对角元素都为零的行所对应的未知量,这些未知量构成了分层后的第 1 层. 接下来,确定仅依赖于第 1 层中未知量的未知量,这些未知量对应分层后的第 2 层. 以此类推,第 l 步确定分层后的第 l 层所包含的未知量,即图中仅依赖于前 $l-1$ 层的未知量的未知量. 上述过程实质上为每个未知量 $\boldsymbol{x}_j$ 确定了一个深度 $depth(j)$,其中第 k 层的所有未知量的深度都为 k.

这样,图 3 中的每一层未知量是具有相同深度顶点的集合. 下面定义了 2 个数组说明分层后的数据结构:一个是整型数组 q,用于记录 $\boldsymbol{x}$ 中各未知量按层排序后的序列;另一个是层指针数组 $level$,其中 $level(i)$ 指向数组 q 中第 i 层的起始位置. 假设三角矩阵 $\boldsymbol{L}$ 采用 CSR 格式(数组 al, jal 和 ial)存储,$nlev$ 表示层数. 算法 1 给出了采用上述数据结构的分层调度算法:

算法 1. 带有分层调度的向前代换.

① for $lev=1,\cdots,nlev$ do
② $\quad j_1=level(lev)$;
③ $\quad j_2=level(lev+1)-1$;
④ $\quad$ for $k=j_1,\cdots,j_2$ do
⑤ $\quad\quad i=q(k)$;
⑥ $\quad\quad$ for $j=ial(i),\cdots,ial(i+1)-1$ do
⑦ $\quad\quad\quad \boldsymbol{x}(i)\leftarrow\boldsymbol{x}(i)-al(j)\times\boldsymbol{x}(jal(j))$;
⑧ $\quad\quad$ end for
⑨ $\quad$ end for
⑩ end for

1.3 分层后稀疏三角矩阵的层排序算法

分层后稀疏三角矩阵的层排序算法是利用分析阶段产生的层信息对三角矩阵进行层排序,将同一层中分散的各行集中在一起. 由于分层后稀疏三角矩阵同一层中的各行通常是不连续的,而相对于 CPU 上的高速缓存(cache),GPU 上每个流多处理器(streaming multiprocessor, SM)的 cache 较小,这样 GPU 更多地依赖于 NVIDIA 公司的并行计算架构 CUDA(compute unified device architecture)的合并存储器访问来提升数据访问的性能. 在 NVIDIA 公司的 Fermi 架构中,内存合并可以概括为:同一个线程束(warp)中线程的并发内存访问将会合并为一些内存事务,而内存事务的数目等于向该 warp 中的所有线程提供服务所需要的 cache line 的数目[13],所以将同一层中的行集中起来可以减少求解阶段所需要的内存事务的数目,如图 4 所示,同时也可以提高 cache 命中率.

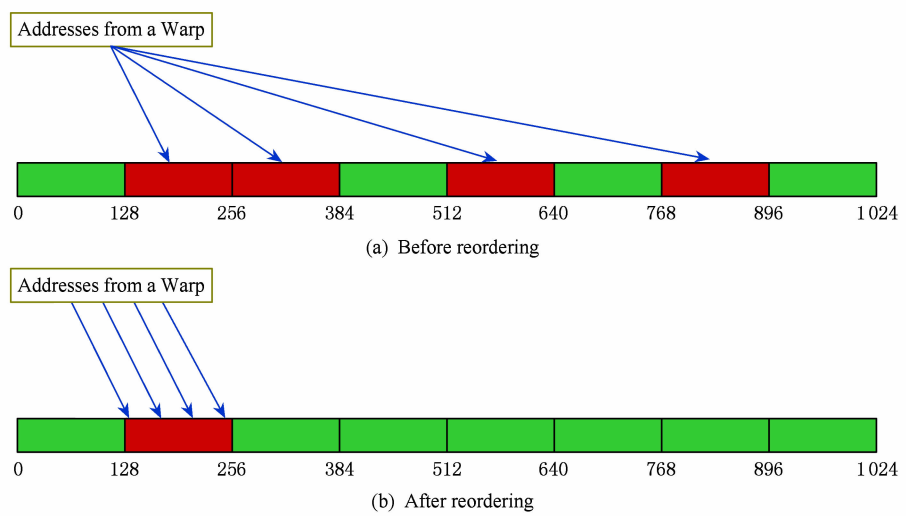


Fig. 4 Reordering (8 threads per row, 128 B cache line).

图 4 重排序(每行 8 线程、128 B 缓存行)

再次重排序后,$q(i)$表示矩阵的第 i 行在原矩阵中的行号,$level(i)$表示矩阵的第 i 层的起始行号.算法 2 给出了三角矩阵在层排序后的向前代换算法.算法 2 和算法 1 的唯一区别在于将行⑥中索引 i 替换为索引 k.

算法 2. 三角方程组重排序后带有分层调度的向前代换.

① for $lev=1,\cdots,nlev$ do
② $\quad j_1=level(lev)$;
③ $\quad j_2=level(lev+1)-1$;
④ $\quad$ for $k=j_1,\cdots,j_2$ do
⑤ $\quad\quad i=q(k)$;
⑥ $\quad\quad$ for $j=ial(k),\cdots,ial(k+1)-1$ do
⑦ $\quad\quad\quad \boldsymbol{x}(i)\leftarrow\boldsymbol{x}(i)-al(j)\times\boldsymbol{x}(jal(j))$;
⑧ $\quad\quad$ end for
⑨ $\quad$ end for
⑩ end for

2　稀疏三角方程组的 GPU 并行

在三角矩阵进行分层调度和重排序后,稀疏三角方程组在 GPU 上的多线程并行求解需要更多地考虑适合 GPU 操作的稀疏矩阵存储格式以及针对 CUDA 架构的并行优化.稀疏三角方程组在 GPU 上的并行求解对应着分层调度中求解阶段的并行,即算法 2 在 GPU 上的并行.不同于三角矩阵分层调度中分析阶段所采用的列压缩(compressed sparse column, CSC)存储格式,稀疏三角方程组的求解阶段需要采用行压缩(compressed sparse row, CSR)存储格式,这样可以避免 CSC 格式所导致的原子操作和拷贝开销.此外,在 GPU 上并行求解时需要在求解每层后进行同步.CUDA 架构提供了块内同步的轻量级方法 *_syncthreads*(),而块间同步通常需要重启 *kernel* 来实现. *_syncthreads*()尽管开销较少,但参与并行计算的线程数受到限制,而块间同步由于需要重启 *kernel*,开销较大(一般为几微秒).而大规模稀疏矩阵分层后通常有几百层甚至上千层,完全使用全局同步会带来很大的同步开销.为了获得更多的并行线程数同时减少同步开销,一个折中方案是混合使用 2 种方法,如图 5 所示,当一个单独的层有很多行时,启动一个多块 *kernel* 来保证足够的并行性,而当连续的层仅有少数几行时,启动一个单块 *kernel* 来减少同步开销.

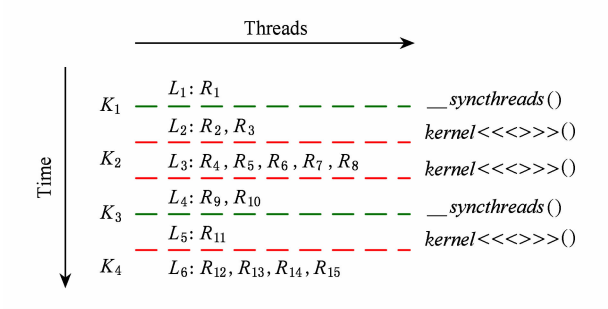


Fig. 5　Combined approach in synchronization between levels.

图 5　层间同步的混合方法

通过程序优化可以进一步挖掘 GPU 的性能潜力.CUDA 优化可以应用在 3 个方面:内存优化、指令优化和执行参数优化.1)在内存优化方面,已经引入了层排序技术来提高空间局部性.2)在指令优化方面,注意求解阶段开销最大的算数指令是浮点除,它可以用浮点乘替换,只要在对角元中存储原始值的倒数即可.此外,整数除和模运算都可以用位运算替换.在并行归约中,还可以移除 if 语句来避免分支歧义,但循环展开的效果不明显,因为编译器已经自动作了分支预测.3)在执行参数优化方面,需要确定的参数有块大小、块数目以及向量长度(每行有多少线程).NVIDIA 建议设置块大小为 64 的倍数来避免寄存器存储体冲突(bank conflict)[14].在实践中,64 线程通常也是最佳的块大小,因为在单指令多数据流(single instruction multiple data, SIMD)架构中,使用短向量可以减少底层开销[15].块数目可利用 GPU 硬件参数和层信息确定(不超过硬件可同时执行的最大块数),然后使用固定的块在行上

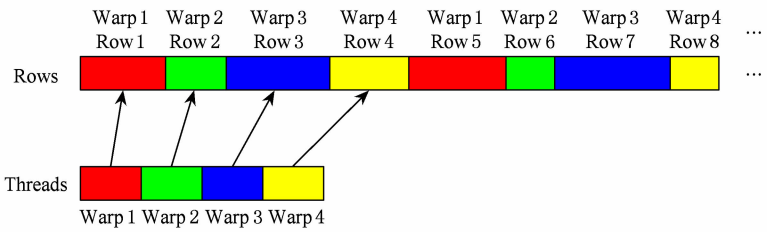


Fig. 6　Warp-oriented programming method.

图 6　面向 warp 的编程方法

迭代,这是一种面向 warp 的编程方法,如图 6 所示,可以避免块调度开销.而每行线程个数的选取应该遵循 2 条原则:1)线程个数是 2 的幂,并且小于等于 32,这样可以在并行归约中利用 warp 内部的硬件同步;2)线程个数接近 $(nnz+n)/n$,这样可以保证即使是非零元素最多的行也有足够的线程.实验表明,该方法在大多数情况下都能选取出最佳的向量长度.

3 数值实验

实验环境为中国科学院网络中心深腾 7 000 GB 的一个节点,CPU 为 AMD Phenom 9850(4 核、2.5 GHz、4 MB 缓存),GPU 为 NVIDIA Tesla C2075(448 核、1.15 GHz、4 GB 显存),操作系统为 RHEL 5.3×64 Linux,CUDA Toolkit 版本为 4.2.CPU 代码用 G++编译,使用默认优化级别;CUDA 代码用 nvcc 编译,使用-arch sm_20 标志开启双精度浮点数支持.

3.1 测试矩阵

表 1 给出了测试使用的对称正定矩阵的详细信息,测试矩阵中的前 7 个是佛罗里达大学稀疏矩阵集 UF Sparse Matrix Collection[16] 中收集的各种实际应用产生的矩阵,最后 1 个来源于泊松方程的离散化.其中,n 为矩阵阶数,nnz 为非零元素个数,*cg-iter* 为采用 CG 方法求解的迭代次数.矩阵 ***inline_1*** 和 ***Flan_1565*** 为通过三维有限元法求解结构问题时所产生的矩阵,该矩阵平均每行具有相对较多的非零元素.矩阵 ***offshore*** 和 ***inline_1*** 是具有很大条件数的病态矩阵,从表 1 中的 *cg-iter* 列可以看出,使用不带预条件的 CG 方法求解矩阵 ***offshore*** 和 ***inline_1*** 时的迭代次数达到 10 000 以上.

Table 1 SPD Matrices

表 1 对称正定矩阵

ID	Name	n	nnz	nnz/n	*cg-iter*	Kind
1	***offshore***	259 789	4 242 673	16.3	18 413	Electromagnetics Problem
2	***inline_1***	503 712	36 816 342	73.1	14 537	Structural Problem
3	***af_shell3***	504 855	17 562 051	34.8	1 438	Subsequent Structural Problem
4	***ecology2***	999 999	4 995 991	5.0	5 490	2D/3D Problem
5	***thermal2***	1 228 045	8 580 313	7.0	2 590	Thermal Problem
6	***Flan_1565***	1 564 794	117 406 044	75.0	8 527	Structural Problem
7	***G3_circuit***	1 585 478	7 660 826	4.8	3 180	Circuit Simulation Problem
8	***Poisson***	4 840 000	24 191 200	5.0	3 146	Poisson's Equation

在 3.2 节给出的 ICCG 法在 GPU 上的性能测试中,设置的相对误差为 10^{-6},最大迭代次数为 5000,稀疏线性方程组的右端项 $\boldsymbol{b}$ 为系数矩阵 $\boldsymbol{A}$ 与向量 $(1,\cdots,1)^{\mathrm{T}}$ 的乘积.

3.2 GPU 性能测试

基于不完全 Cholesky 分解预条件,表 2 给出了在应用近似最小度排序算法前利用 NVIDIA CUSPARSE 实现的 ICCG 法和本文提出的方法的比较结果.其中,*iter* 为迭代次数,*residual* 为残差.从表 2 可以看出,由于采用更高效的并行稀疏三角方程组求解方法,本文提出的方法比利用 CUSPARSE 实现的 ICCG 法快 25%.根据分析,其中 20%的性能提升来自于分层后稀疏三角矩阵的层排序算法,这也说明了内存优化在 CUDA 程序优化中的重要作用,尤其是对于稀疏矩阵这样的不规则问题.

对于高度病态矩阵 ***offshore*** 和 ***inline_1***,我们发现对其应用 IC 预条件后,迭代次数分别从 18 413 次和 14 537 次下降到 8 次和 1 214 次.从表 2 可以看出,对于高度病态矩阵来说 IC 预条件表现出良好的健壮性和预条件效果.对于三维有限元法导出的稠密度相对较高的矩阵 ***inline_1*** 和 ***Flan_1565***,由于这类矩阵所对应的有向图的结构比较复杂,因此在分层调度过程中产生的层数会比较多,从而导致平均每层中的行数较少,使得行间并行度不高.但是,由于这类矩阵每行非零元素个数比较多,使得同一行内具有较高的并行度,所以总体上分层调度算法仍然可以带来较高的并行性.数值实验表明,我们提出的分层调度后利用层排序算法优化内存访问的方法仍然有效,对于这 3 个矩阵,我们的实现相对于 CUSPARSE 实现分别有 29%,16%和 12%的性能提升.

Table 2　CUSPARSE-ICCG vs Our ICCG

表 2　CUSPARSE-ICCG 与本文 ICCG 的比较

ID	*nlev*	CUSPARSE-ICCG			Our ICCG		
		iter	*residual*	Solve-time/s	*iter*	*residual*	Solve-time/s
1	3 452	8	7.79E−7	0.48	8	7.79E−7	0.34
2	1 752	2 314	9.89E−7	140.81	2313	9.96E−7	117.69
3	3 725	395	9.86E−7	24.21	395	9.86E−7	19.47
4	1 999	1 601	9.92E−7	51.40	1601	9.90E−7	40.45
5	1 239	1 264	9.97E−7	43.17	1265	9.87E−7	33.56
6	7 806	1 214	9.96E−7	216.93	1216	9.40E−7	189.36
7	2 594	153	9.12E−7	7.22	153	9.11E−7	5.46
8	4 399	932	9.85E−7	108.78	932	9.85E−7	81.78

基于 IC 预条件,表 3 给出了 NVIDIA CUSPARSE 实现的 ICCG 并行算法和本文在采用 AMD 排序后给出的 ICCG 并行算法的比较结果. 从表 3 可以看出,AMD 排序可以明显减少分层调度过程中产生的层数,进而提升了稀疏三角方程组求解的并行度. 尽管在 AMD 排序后,表 3 中某些矩阵的迭代次数有所增加,但由于 AMD 排序明显提高了稀疏三角方程组求解的并行性能,使得预条件开销的下降完全抵消了迭代次数的增加. 也就是说,通过 AMD 排序整体上降低了稀疏方程组的求解时间. 数值实验表明,运用本文提出的 2 种重排序技术的 AMD-ICCG 算法,相对于仅利用 CUSPARSE 实现的 CUSPARSE-ICCG 算法,性能可以获得平均 1 倍以上的提升.

Table 3　CUSPARSE-ICCG vs Our AMD-ICCG

表 3　CUSPARSE-ICCG 与本文 AMD-ICCG 的比较

ID	AMD-time/s	CUSPARSE-ICCG			Our AMD-ICCG		
		iter	*nlev*	Solve-time/s	*iter*	*nlev*	Solve-time/s
1	1.25	8	3 452	0.48	10	352	0.12
2	3.01	2 314	1 752	140.81	1 187	600	43.21
3	0.88	395	3 725	24.21	639	597	16.54
4	1.13	1 601	1 999	51.40	2 738	6	23.78
5	3.39	1 264	1 239	43.17	955	36	14.23
6	8.26	1 214	7 806	216.93	1 691	1 434	174.79
7	3.30	153	2 594	7.22	285	12	4.16
8	5.40	932	4 399	108.78	1 582	6	64.48

表 3 同时给出了 AMD 排序的时间. 对于矩阵 ***offshore***,由于迭代求解时间较少(仅为 0.12 s),使得 AMD 排序时间在整体计算时间中所占的比例较高. 但对于大部分矩阵而言,本文提出的 ICCG GPU 求解时间与 AMD 排序时间之和,仍然明显优于 NVIDIA CUSPARSE 的求解时间. 另外,在实际应用中往往需要求解具有相同系数矩阵和不同右端项的多个方程组,对于此类问题,矩阵重排序时间在整体计算时间中所占的比例将大大降低.

4　结　　论

GPU 加速 ICCG 法的性能瓶颈在于并行稀疏三角方程组求解. 由于稀疏三角方程组求解固有的串行性,使得 ICCG 法难以在 GPU 上得到好的并行性能. 为了增加稀疏三角方程组在 GPU 上求解的并行度,本文提出的一种有效方法是使用分层调度算法对稀疏三角矩阵分层. 而为了进一步减少稀疏三角矩阵在分层过程中产生的层数,在进行分层调度前又使用了近似最小度(AMD)算法对系数矩阵进行重排序. 同时,为了优化 global memory 的内存访问模式,本文对分层后的稀疏三角矩阵的各层又采用了层排序策略,提高多线程内存访问的连续性. 通过测试表明,相对于利用 NVIDIA CUSPARSE 实现的 ICCG 法,本文提出的方法可以获得平均 1 倍以上的性能提升. 对于实际应用中比较难解的高度

病态矩阵和三维有限元法导出的稠密度较高的矩阵,本文提出的方法同样具有比较好的加速效果.

IC 预条件的优势在于良好的健壮性和预条件效果,GPU 加速 ICCG 法的适用范围包括:1)采用其他预条件较为难解的方程组,这时只能利用 IC 预条件的健壮性;2)层数较少或平均每行非零元素个数较多的方程组,这时稀疏三角方程组求解可以有比较好的并行性. 对于其他类型的方程组而言,IC 预条件并不一定优于具有内在并行性的显式预条件,适用于 GPU 的显式预条件技术也是将来的工作方向之一.

参 考 文 献

[1] Ament M, Knittel G, Weiskopf D, et al. A parallel preconditioned conjugate gradient solver for the poisson problem on a multi-gpu platform [C] //Proc of the 18th Euromicro Conf on Parallel, Distributed and Network-Based Processing. Piscataway, NJ: IEEE, 2010: 583-592

[2] Benzi M, Tuma M. A comparative study of sparse approximate inverse preconditioners [J]. Applied Numerical Mathematics, 1999, 30(2): 305-340

[3] Helfenstein R, Koko J. Parallel preconditioned conjugate gradient algorithm on GPU [J]. Journal of Computational and Applied Mathematics, 2012, 236(15): 3584-3590

[4] Li R, Saad Y. GPU-accelerated preconditioned iterative linear solvers [J]. The Journal of Supercomputing, 2013, 63(2): 443-466

[5] Naumov M. Incomplete-LU and Cholesky preconditioned iterative methods using CUSPARSE and CUBLAS [R]. Santa Clara, CA: NVIDIA Corporation, 2011

[6] Sudan H, Klie H, Li R, et al. High performance manycore solvers for reservoir simulation [C] //Proc of the 12th European Conf on the Mathematics of Oil Recovery. Berlin: Springer, 2010 [2013-09-20]. http://www-users.cs.umn.edu/~saad/PDF/A044.pdf

[7] Gupta R. A GPU implementation of a bubbly flow solver [D]. Delft, Holland: Delft University of Technology, 2009

[8] Zhang Jianfei, Shen Defei. Preconditioned conjugate gradient method for sparse linear systems based on GPU [J]. Journal of Computer Applications, 2013, 33(3): 825-829 (in Chinese)
(张健飞, 沈德飞. 基于 GPU 的稀疏线性系统的预条件共轭梯度法[J]. 计算机应用, 2013, 33(3): 825-829)

[9] Wu Changjiang. The design of large sparse linear system solvers based on CUDA [D]. Chengdu: University of Electronic Science and Technology of China, 2013 (in Chinese)
(吴长江. 基于 CUDA 的大规模线性稀疏方程组求解器的设计[D]. 成都: 电子科技大学, 2013)

[10] Amestoy P R, Davis T A, Duff I S. An approximate minimum degree ordering algorithm [J]. SIAM Journal on Matrix Analysis and Applications, 1996, 17(4): 886-905

[11] George A, Liu J W H. The evolution of the minimum degree ordering algorithm [J]. SIAM Review, 1989, 31(1): 1-19

[12] Saad Y. Iterative Methods for Sparse Linear Systems [M]. Philadelphia, PA: SIAM, 2003

[13] NVIDIA Corporation. CUDA C Programming Guide [M/OL]. Santa Clara, CA: NVIDIA Corporation, 2012 [2013-07-25]. http://docs.nvidia.com/cuda/cuda-c-programming-guide

[14] NVIDIA Corporation. CUDA C Best Practices Guide [M/OL]. Santa Clara, CA: NVIDIA Corporation, 2012 [2013-07-25]. http://docs.nvidia.com/cuda/cuda-c-best-practices-guide

[15] Volkov V, Demmel J. LU, QR and Cholesky factorizations using vector capabilities of GPUs, UCB/EECS-2008-49 [R]. Berkeley, CA: Electrical Engineering of Computer Science Department, University of California, Berkeley, 2008

[16] Davis T A, Hu Y. The University of Florida sparse matrix collection [J]. ACM Trans on Mathematical Software (TOMS), 2011, 38(1): 1-28

Chen Yao, born in 1988. Received his master degree from the Computer Network Information Center, Chinese Academy of Sciences in 2014. His main research interests include parallel computing and preconditioners for sparse linear systems.

Zhao Yonghua, born in 1966. Professor at the Computer Network Information Center, Chinese Academy of Sciences. His main research interests include parallel algorithm and software, parallel programming model and method, and parallel solvers for scalable nonlinear/linear algebra problems.

Zhao Wei, born in 1990. Received his master degree from the Computer Network Information Center, Chinese Academy of Sciences in 2014. His main research interests include CUDA and DFT (zhaowei@sccas.cn).

Zhao Lian, born in 1989. PhD candidate at the University of Chinese Academy of Sciences. Her main research interests include parallel computing and algebraic multi-grid (zhaolian@sccas.cn).