小 型 微 型 计 算 机 系 统
Journal of Chinese Computer Systems
DOI: 10.20009/j.cnki.21-1106/TP.2021-0189
2022 年 10 月 第 10 期
Vol.43 No.10 2022

预处理共轭梯度算法异构并行求解及优化

张 琨[1],贾金芳[1],黄建强[1,2],王晓英[1],严文昕[1]

[1](青海大学 计算机技术与应用系,西宁 810016)
[2](清华大学 计算机科学与技术系,北京 100084)
E-mail: qhu_jf@163.com

摘 要: 共轭梯度算法是求解对称正定线性系统的重要方法之一,该算法求解问题通常具有稀疏性.随着问题规模的不断增大,单 CPU 因其存储及计算能力限制已经不能满足大规模稀疏线性方程组求解的实时需求.基于此,本文提出一种基于 CPU + GPU 异构平台的 MPI + CUDA 异构并行求解算法.首先,对共轭梯度算法进行了热点性能分析,说明该算法求解时存在的计算困难及挑战;然后,根据共轭梯度算法特性进行了任务划分,实现异构并行算法设计;最后,针对异构并行算法中存在的通信开销、数据传输开销和存储器访问开销等问题,对异构并行算法进行优化以进一步提升求解效率及性能.实验结果表明,与 MPI 并行和 CUDALib 并行相比,MPI + CUDA 异构混合并行在串行计算部分较少的 Jacobi 预处理共轭梯度算法上分别获得 336% 和 33% 的性能提升,在串行计算部分较多的 ILU 预处理共轭梯度算法上也能分别获得 25% 和 7% 的性能提升,同时结果还显示 MPI + CUDA 混合并行随着节点数目的增加具有一定可扩展性.

关 键 词: 对称正定线性系统;共轭梯度算法;预处理技术;异构并行
中图分类号: TP311 **文献标识码:** A **文 章 编 号:** 1000-1220(2022)10-2040-06

Heterogeneous Parallel Solving and Optimization of Preconditioned Conjugate Gradient Method

ZHANG Kun[1], JIA Jin-fang[1], HUANG Jian-qiang[1,2], WANG Xiao-ying[1], YAN Wen-xin[1]
[1](Department of Computer Technology and Applications, Qinghai University, Xining 810016, China)
[2](Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

Abstract: Conjugate gradient method is one of the important methods for solving symmetric positive definite linear systems. The problem solved by this algorithm is usually sparse. As the scale of the problem continues to increase, a single CPU can no longer meet the real-time requirements for solving large-scale sparse linear equations due to its storage and computing capacity limitations. Based on this, this paper proposes an MPI + CUDA heterogeneous parallel solving algorithm based on CPU + GPU heterogeneous platform. First, the hotspot performance analysis of the conjugate gradient algorithm is carried out to illustrate the computational difficulties and challenges in the solution of the algorithm; Then, the task is divided according to the characteristics of the conjugate gradient algorithm, and the design of heterogeneous parallel algorithms is realized; To solve the problems of communication overhead, data transmission overhead and memory access overhead in parallel algorithms, the heterogeneous parallel algorithms are optimized to further improve the efficiency and performance of the solution. The experimental results show that compared with MPI parallel and CUDALib parallel, MPI + CUDA heterogeneous hybrid parallel achieves 336% and 33% performance improvement on the Jacobi preconditioned conjugate gradient algorithm with less serial calculation part, 25% and 7% performance improvements respectively on ILU preconditioned conjugate gradient algorithm with more serial calculation part. At the same time, the results also show that the MPI + CUDA hybrid parallel has a certain degree of scalability as the number of nodes increases.

Key words: symmetric positive definite linear system; conjugate gradient method; precondition technology; heterogeneous parallel

1 引 言

线性方程组求解是许多科学计算和工程应用中的核心计算内容,其目前主要有直接法和迭代法两种求解方式[1-3].直接法由于计算机资源的限制难以有效求解大规模线性方程组,而迭代法具有存储需求低,系数矩阵在迭代时保持稳定等优点,所以其常用于求解大规模线性方程组.目前迭代法已经由 Jacobi 方法、SOR 方法等经典迭代法过渡到了共轭梯度(CG)方法和广义共轭残差(GCR)方法等效率更高的 Krylov 子空间迭代法[4,5].同时,为了改善迭代法的收敛性,也相继产生了多种多样的预处理技术[6-8].

由于求解问题规模不断增大,单处理器已经不能满足大

收稿日期: 2021-03-03 收修改稿日期: 2021-04-06 基金项目: 青海省科技计划项目-应用基础研究计划项目(2019-ZJ-7034)资助;国家自然科学基金项目(61762074,62062059)资助;教育部"春晖计划"科研基金项目(QDCH2018001)资助. 作者简介: 张 琨,男,1997 年生,硕士研究生,CCF 会员,研究方向为高性能计算;贾金芳(通讯作者),女,1991 年生,硕士,讲师,CCF 会员,研究方向为高性能计算;黄建强,男,1985 年生,博士研究生,副教授,CCF 会员,研究方向为高性能计算、大规模图计算;王晓英,女,1982 年生,博士,教授,CCF 会员,研究方向为高性能计算、绿色计算;严文昕,女,1996 年生,硕士研究生,CCF 会员,研究方向为高性能计算.

规模计算的需求,因此研究大规模稀疏线性方程组的快速求解方案具有重要意义. 并行计算是高效求解大规模稀疏线性方程组的重要途经,其主要有多核 CPU、GPGPU、集群 3 种类型[9,10]. 许多学者为探究不同平台的高性能迭代法实现方案展开了相关研究. Bohacek J 等[11] 针对热扩散问题在 GPU 上利用多项式预处理对 CG 法进行并行优化,与传统 CFD 代码相比最高能够获得 16000 倍的加速,同时还指出了高效的并行算法不仅要考虑硬件设备的更新,还要考虑算法本身的改进. Anzt H 等[12] 提出一种应用于线性系统求解的不完全稀疏近似逆(ISAI)预处理方法,该方法可以并行生成预条件子,而且与 Block-Jacobi 预处理方法相比可以获得更好的性能. Jiao Y Y 等[13] 为满足地质领域求解线性方程组的需要,在 MPI、OpenMP、MPI + OpenMP 3 种方案中测试可行且高效的 CG 法并行方案来解决球面非连续变形分析问题,在模拟隧道坍塌的实验中加速比达到 7x 左右. Suryanarayana P 等[14] 基于 PETSc 并行库实现了一种新的 AAR 迭代法,并与 GMRES、Bi-CG-STAB 等多种迭代法进行了对比,在相同的预处理条件下,新方法能够更快地完成求解. Bernaschi M 等[15] 实现了基于 GPU 的包含 AMG 预处理部分和求解部分的线性求解器,其通过一种兼容加权匹配聚合算法实现最优的核函数配置,从而有效利用了 GPU 的性能. Aliaga J I 等[16] 对 ILUPACK 中的 GMRES 算法进行了改进,优化了其中的预处理子应用部分及 MGSO 部分,使这两部分的加速比分别提升至 3x 和 10x. Sanjuan G 等[17] 将 Schwarz 交替域分解技术与 MPI + OpenMP 混合并行技术结合,使风场计算中的 CG 迭代法求解速度提升至近 9x.

综上所述,目前的性能优化技术大多集中在算法改进或者单一并行方式领域,而基于异构平台的多机并行优化相对较少. 本文在 CPU + GPU 异构平台上实现了基于 MPI + CUDA 混合并行的预处理共轭梯度算法,并对实验结果进行了对比分析,发现 MPI + CUDA 混合并行的预处理共轭梯度算法能够更高效地求解大规模对称正定线性系统.

2　线性方程组与预处理共轭梯度算法

2.1　线性方程组

线性方程组的求解广泛存在于工程问题中. 就大规模数值计算而言,使用迭代法求解线性方程组时涉及到的矩阵及向量运算会占据大量的计算时间和硬件资源. 随着求解问题规模的不断增大,如何加快大规模稀疏线性方程组的求解过程显得愈加重要. 一个 n 元线性方程组如公式(1)所示,其中 m 为方程组个数,$a_{m,n}$ 为方程系数、x 为方程解、b 为常数项.

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n = b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n = b_2 \\ \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ a_{m,1}x_1 + a_{m,2}x_2 + \cdots + a_{m,n}x_n = b_m \end{cases} \tag{1}$$

将线性方程组转换为线性代数中的概念则可以表示为 $Ax = b$ 的形式,如公式(2)所示,其中 A 是 $m \times n$ 的系数矩阵,x 是解向量,b 是右端向量.

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \tag{2}$$

2.2　预处理共轭梯度算法及性能分析

2.2.1　预处理共轭梯度算法

共轭梯度算法是解决对称正定线性系统问题最有效的迭代方法之一,具有存储量要求少、不必事先估计某些参数、编程简单等优点[18]. 为改善迭代法的收敛性,目前主要利用预处理技术将原方程 $Ax = b$ 替换为 $M^{(-1)}Ax = M^{(-1)}b$ 等形式进行求解,其中较为常用的预条件子有不完全分解预条件子、稀疏近似逆预条件子等[19-21]. 预处理共轭梯度算法的具体内容如算法 1 所示.

算法 1. 预处理共轭梯度算法

输入: 系数矩阵 A,预条件子 M,初始解 x_0,右端向量 b.

输出: 近似解 x.

1. $r_0 = b - Ax_0, z_0 = M^{-1}r_0, p_1 = z_0$
2. **for** $i = 1$: Maximum number of iterations **do**
3. $\alpha_i = (r_{i-1}, z_{i-1}) / (Ap_i, p_i)$
4. $x_i = x_{i-1} + \alpha_i p_i$
5. $r_i = r_{i-1} - \alpha_i Ap_i$
6. **if** converged **then** exit
7. $z_i = M^{-1}r_i$
8. $\beta_i = (r_i, z_i) / (r_{i-1}, z_{i-1})$
9. $p_{i+1} = z_i + \beta_i p_i$
10. **end for**

2.2.2　预处理共轭梯度算法性能分析

每次迭代过程中,预处理共轭梯度算法需要进行 1 次稀疏矩阵向量乘,5 次向量内积(包括 1 次残差计算),1 次预条件子应用,3 次向量数乘. 在预处理共轭梯度算法的主要计算量分布测试中,预处理部分使用 ILU 预条件子进行测试,占比 55.51%,稀疏矩阵向量乘占比 40.01%,向量内积占比 2.36%,向量数乘占比 2.12%. 由于使用不同的预条件子的计算开销会有较大差异,本文主要针对其余计算部分进行性能优化. 单机设备的计算性能有限,而且还有可能无法满足大规模数据的存储需求,所以目前通常采用并行的方式对预处理共轭梯度算法进行求解及优化.

3　预处理共轭梯度算法并行优化

3.1　优化框架

并行算法在优化过程中为保证稀疏矩阵存储性能一致,统一使用 CSR(Compressed Sparse Row)格式保存稀疏矩阵[22,23]. 实验在异构平台上进行测试,CPU 端并行技术使用 MPI,GPU 端并行技术使用 CUDA. 异构并行的预处理共轭梯度算法执行步骤为:

1) MPI 进程读取数据并初始化 CPU 和 GPU 内存空间;

2) CUDA 核函数计算 r,然后 MPI 进程进行数据通信;

3) MPI 进程进行预条件子应用,计算 z 和 p;

4) CUDA 核函数计算稀疏矩阵向量乘和向量内积,然后 MPI 进程进行通信得到 α;

5) CUDA 核函数计算向量数乘,更新 x 和 r;

6) CUDA 核函数计算残差 Res,然后 MPI 进程进行数据通信及迭代退出判断;

7) MPI 进程进行预条件子应用,计算 z;

8) CUDA 核函数计算向量内积,然后 MPI 进程进行数据

通信得到 β;

9) CUDA 核函数计算向量数乘,更新 p,然后 MPI 进程进行数据通信并返回步骤 4.

实验方案分为 3 个部分,首先在预处理优化部分,方案对 ILU 预条件子和 Jacobi 预条件子的加速性能进行对比分析;然后在 MPI 并行部分,方案将可以重叠的部分计算操作和通信操作进行了整合;最后在 MPI + CUDA 混合并行部分,方案针对异构并行中存在的数据传输问题、访存问题等,分别通过使用页锁定内存、改变线程任务分配方式及利用共享存储器的措施进行优化.

3.2 预处理优化

本文所测试的预条件子为 ILU 预条件子和 Jacobi 预条件子. Jacobi 预条件子的构造过程相对简单,最后生成的预条件子 M 即为 $DIA(A)$. ILU 预条件子将系数矩阵 A 分解为 LU-R 的形式,分解之后的预条件子可以满足不同的条件 P. ILU 预条件子的构造过程如算法 2 所示.

算法 2. ILU 分解

输入: 系数矩阵 A,条件 P.

输出: 预条件子 M.

```
for i = 2,3,…,n do
    for k = 1,2,…,i-1 and (i,k) ∉ P do
        A_ik = A_ik / A_kk
        for j = k+1,…,n and (i,j) ∉ P do
            A_ij = A_ij - A_ik × A_kj
        end for
    end for
end for
```

3.3 MPI 并行通信优化

MPI 并行方式中各个进程间的通信开销会随着进程规模的增大而增大[24]. 在预处理共轭梯度算法中,并行任务主要在每一个迭代步内执行,其中 MPI 主要负责任务的粗粒度划分、进程间数据通信、进程内部数据拷贝及非并行部分的计算.

在 MPI 并行算法中,每次迭代都会进行数据通信,但是除计算 Ap 所进行的稀疏矩阵向量乘(SpMV)通信之外,其他向量内积(Dot)的部分通信结果可以延迟参与计算. 为减少算法的通信开销,每次迭代步内将部分计算操作与通信操作进行重叠. 由于计算 Ap 的 SpMV 操作需要保证 p 向量在各个进程中是完整的,而且也不存在可以与通信进行重叠的计算任务,所以在 SpMV 操作之前使用阻塞的 Allgather 函数进行数据通信. 在后续 3 组需要通信的 Dot 操作中,通信结果存在可以延迟计算的部分,同时有相应的计算任务可以与通信进行重叠,所以使用非阻塞的 Iallreduce 函数进行数据通信. 在将计算操作和通信操作进行重叠之后,通信优化算法可以减少使用阻塞通信函数带来的额外开销.

3.4 GPU 并行优化

3.4.1 数据传输优化

CPU 与 GPU 协同计算时最主要的开销在于两者之间的数据传输,并行程序只对部分必须进行全局更新的向量进行数据传输. 在每次迭代过程中无法避免的数据传输内容包括向量 p、向量 r、向量 z、残差 Res 以及向量内积的中间结果 $subDot$. 对于这些需要在 CPU 与 GPU 间频繁传输的数据在开辟内存空间时采用页锁定内存(pinned memory),因为操作系统不会对页锁定内存执行分页和交换操作,使得该内存能够始终驻留在物理内存中,而 GPU 可以通过直接内存访问方式直接在 CPU 和 GPU 间拷贝数据,从而提高数据传输效率.

3.4.2 访存优化

将算法主要计算操作移植到 GPU 上之后,全局存储器的合并访存(coalesced access)特性也会对程序性能造成一定影响. CUDA 模型的最小执行单位为线程束(warp),当同一线程束内的线程访问连续的存储空间时,能够减少访存操作的次数. 向量计算中的一维数据能够保证线程束中的线程访存是连续的,但在矩阵计算中则需要根据使用的稀疏矩阵存储格式调整线程计算任务的分配策略. 实验采用的稀疏矩阵存储格式为 CSR,该格式使用 3 个数组分别保存稀疏矩阵每行的偏移量、列索引及非零元素值. 计算任务以线程为最小单位分配时,会出现同一线程束内的线程访问不连续内存空间的情况,从而造成访存性能下降. 为保证矩阵计算过程中的访存连续性,计算任务以线程束为最小单位进行分配. 以线程束为最小单位调度任务之后,同一线程束内的线程访问连续的内存空间,从而可以利用全局存储器的合并访存特性提高访存性能.

3.4.3 多级存储器优化

在利用 GPU 执行并行任务时,可以使用共享存储器保存只需要各线程块内部的线程访问的数据. 共享存储器(Shared Memory)的访存开销没有全局存储器(Global Memory)大,在计算时线程块内部的线程如果能避免频繁访问全局存储器,就可以降低访存开销. MPI + CUDA 混合并行方式中向量内

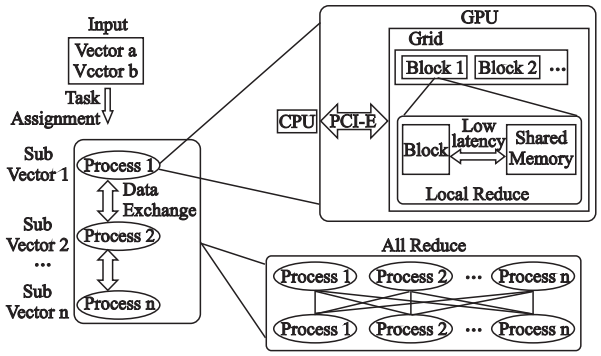

图 1 内积并行

Fig. 1 Dot parallel mode

积的并行结构如图 1 所示,其计算公式可以表示为 $\sum_{i=1}^{n} a_i b_i$. Vector a 和 Vector b 首先根据 MPI 进程数粗粒度划分为 Sub-Vector,然后移植到 GPU 上的向量任务利用多线程并行执行. 为避免修改同一内存地址的数据,并行算法先利用共享存储器进行局部规约,最后各个 MPI 进程再进行全局规约.

4 实验与分析

4.1 实验环境和数据

实验在 4 个高性能节点上进行测试,每个节点两块 CPU,型号为 Intel(R) Xeon(R) CPU E5-2692 v2 @ 2.20GHz,内存

为 64GB; 节点配备 GPU 为 Tesla T4,显存为 16GB; CUDA 模型版本为 v10.1. 实验所测试的数据来自 SuiteSparse Matrix Collection 数据集[25],该数据集包含大量来自实际工程中的稀疏矩阵数据. 如表 1 所示,实验选用了计算流体力学问题、电路仿真问题、结构性问题等 8 个领域共 11 个测试数据,同时测试数据的元素量从 3087898 到 117406044 不等,具有一定数据规模跨度.

表 1　矩阵数据

Table 1　Matrix data

Matrix	Num_row	Num_element	Density
af_shell7	504855	17588875	34.84
audikw_1	943695	77681847	82.28
BenElechi1	245874	13150496	53.48
boneS10	914898	55468422	60.63
cfd2	123440	3087898	25.02
Emilia_923	923136	41005206	44.42
Flan_1565	1564794	117406044	75.03
G3_circuit	1585478	7660826	4.83
offshore	259789	4242673	16.33
PFlow_742	742793	37138461	50.00
thermal2	1228045	8580313	6.99

注: Density = Num_element / Num_row

4.2　实验结果与分析

4.2.1　预处理结果

对于条件数过大的系数矩阵,使用迭代法进行求解时通常会出现收敛速度慢甚至不收敛的情况. 虽然使用预处理技术会增加部分计算开销,但为了提高迭代法的求解效率,利用预处理技术来改善迭代法的收敛性是有必要的.

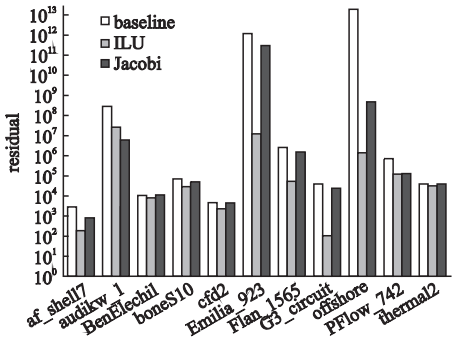

图 2　预条件子对比

Fig. 2　Comparison of preconditioner

实验测试了 ILU 预条件子和 Jacobi 预条件子两种方式对 CG 算法收敛性的影响. 由于不同领域问题对误差精度要求不一致,将问题精度设置为统一标准会出现不同测试数据差异过大的情况,因此实验对相同迭代次数下的误差变化进行对比. 在迭代 200 次的条件下,无预处理算法和两种预处理算法的最终残差如图 2 所示. 从图中可以看出,在加速迭代法收敛的方面 ILU 预条件子比 Jacobi 预条件子效果更好. ILU 预条件子构造和使用的成本都比 Jacobi 预条件子更高,而且 ILU 预条件子在应用时也没有与 Jacobi 预条件子相当的并行性. 因此对于预处理子的选择需要综合考虑应用成本和误差精度要求两个方面,当误差精度要求较低且需要控制应用成本时可以选择使用 Jacobi 预条件子,当对误差精度要求较高时则需要使用 ILU 预条件子.

4.2.2　无预处理算法并行结果

无预处理算法中的主要计算内容为向量数乘、向量内积、稀疏矩阵向量乘. 串行算法和 3 种并行算法的运行性能对比如图 3 所示. 为与 MPI + CUDA 混合并行进程数量保持一致,MPI 并行一共使用 4 个节点,每个节点启动一个进程. 由于使用 MPI 并行会存在部分额外的开销,实验所测试的矩阵数据平均加速比为 2.99x(最高 3.41x). MPI 并行的开销主要来自于各进程间的数据通信,不同算例的通信次数是一致的,通信开销有差异主要是因为测试算例的数据规模不同. 当测试矩阵非零元素数量不多而阶数较大时,会出现矩阵每行计算量(Density)过小的情况,最后导致 MPI 并行性能提升不明显. MPI 并行中矩阵每行计算量较小的算例如 G3_circuit(4.83)和 thermal2(6.99)加速比只有 2.5x 左右,而矩阵每行计算量较大的算例如 Emilia_023(44.42)、BenElechi1(53.48)加速比能达到 3x 以上.

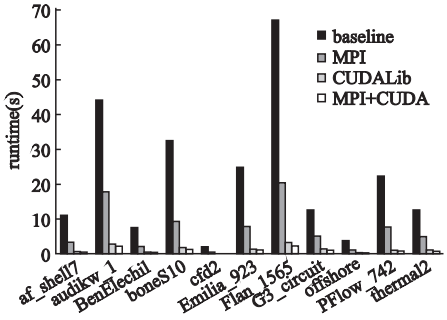

图 3　无预处理算法性能对比

Fig. 3　Performance comparison of original algorithm

MPI + CUDA 混合并行在 MPI 并行的基础上将计算任务移植到 GPU 上执行,该方式并没有更改 MPI 并行各进程间的通信模式,但是可以提高各进程计算部分的性能. MPI + CUDA 混合并行将计算任务进行 GPU 移植之后充分发挥了 GPU 的多线程能力,在测试数据上的平均加速比为 17.25x(最高 27.12x). MPI + CUDA 混合并行的性能变化趋势与 MPI 并行基本一致,但有一点区别是总的计算量不能过小,否则不能发挥 GPU 的多线程优势,甚至可能由于 CPU 与 GPU 之间数据传输开销造成性能下降的情况. 如测试数据 cfd2(25.02),虽然矩阵每行都有一定的数据量,但由于总的计算量过小,最后 MPI + CUDA 混合并行的加速比不足 10x. CUDALib 并行算法使用 Nvidia 并行计算库中的 cuSPARSE 和 cuBLAS 提供的代数计算接口函数进行实现,在各个测试数据上的平均加速比为 13.13x(最高 19.42x).

4.2.3　ILU 预处理算法并行结果

ILU 预处理算法在无预处理算法的基础上增加了预条件子应用的计算内容. ILU 预条件子应用需要在每次迭代过程中增加两次三角方程求解. 因为三角方程求解计算具有明显的串行性质,实验将这一部分计算内容按照串行的方式执行. 由于存在一部分串行计算内容,ILU 预处理算法与无预处理算法相比整体性能出现下降的现象. 串行算法、MPI 并行算

法、CUDALib 并行算法及 MPI + CUDA 混合并行算法的性能对比如图 4 所示. 在实验测试的矩阵数据上 MPI 并行算法性能平均提升 47%(最高 54%),CUDALIB 并行算法性能平均提升 71%(最高 92%),MPI + CUDA 混合并行算法性能平均提升 84%(最高 97%). ILU 预处理算法在应用预条件子时的计算量与 SpMV 相当,而且由于预条件子应用没有进行并行处理,所以当算法中其他部分如 SpMV、Dot、AXPY 等通过并行方式提高计算性能时,预条件子应用会占据程序执行的主要时间.

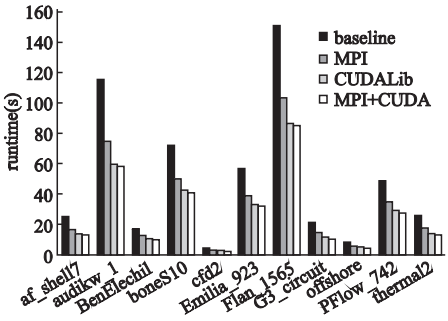

图 4 ILU 预处理算法性能对比

Fig. 4 Performance comparison of ILU preconditioned algorithm

4.2.4 Jacobi 预处理算法并行结果

Jacobi 预条件子的应用比 ILU 预条件子的应用计算量更少,每次迭代过程中应用 Jacobi 预条件子的计算量与一次向量数乘相当,所以在 Jacobi 预处理并行算法中主要的计算开销还是集中在稀疏矩阵向量乘部分. 因为串行任务所占据的计算量较少,Jacobi 预处理算法在各矩阵数据上的运行时间与无预处理算法整体接近,能获得较好的加速效果. 串行算法、MPI 并行算法、CUDALib 并行算法及 MPI + CUDA 混合并行算法的性能对比如图 5 所示. 在实验测试的矩阵数据上 MPI 并行算法平均加速比为 2.92x(最高 3.46x),CUDALib 并行算法平均加速比为 9.56x(最高 14.3x),MPI + CUDA 混合并行算法平均加速比为 12.76x(最高 19.45x).

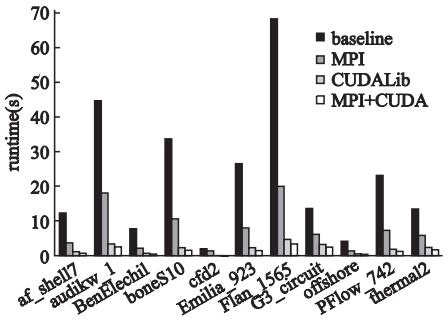

图 5 Jacobi 预处理算法性能对比

Fig. 5 Performance comparison of Jacobi preconditioned algorithm

4.2.5 可扩展性分析

如图 6 所示,实验测试了不同并行算法加速比随着节点数目增加的变化趋势. 在使用 MPI 并行方式的算法中,随着节点数目的增加,无预处理算法和 Jacobi 预处理算法的加速比保持上升趋势,但 ILU 预处理算法的加速比提升并不明显,这是由于 ILU 预处理算法中的预条件子应用会增加较多额外的开销,从而限制了整体加速比的提升. 在使用 MPI + CUDA 混合并行方式的算法中,随着节点数目的增加,无预处理算法和Jacobi预处理算法的加速比同样保持上升趋势,并

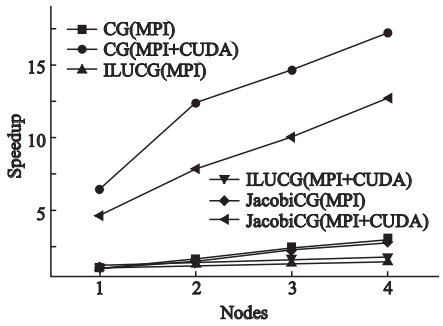

图 6 可扩展性对比

Fig. 6 Comparison of scalability

且与 MPI 并行方式的对应算法加速比相比还有大幅度的提升,这得益于 GPU 的多线程优势,而 ILU 预处理算法的加速比由于预条件子应用开销的存在依然没有明显提升.

5 结 语

大规模稀疏线性方程组的求解问题一直都是人们研究的重点方向. 随着各种工程问题的求解规模增加,单机设备已经难以高效完成大规模计算任务. 本文首先实现了 3 种不同的预处理共轭梯度并行算法,然后对结果进行了对比分析. 实验结果表明,在 MPI 并行、CUDALib 并行和 MPI + CUDA 混合并行 3 种方式中,MPI + CUDA 混合并行方式能够充分发挥多机异构设备的性能,达到更高的计算效率. 在下一步工作中,我们将进行更多预条件子异构性能优化的研究.

References:

[1] Simoncini V, Szyld D B. Recent computational developments in Krylov subspace methods for linear systems [J]. Numerical Linear Algebra with Applications, 2007, 14(1): 1-59.

[2] Ahamed A K C, Magoules F. Iterative methods for sparse linear systems on graphics processing unit [C]//IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems, IEEE, 2012: 836-842.

[3] Varga R S. Iterative analysis [M]. Prentice Hall, Englewood Cliffs, NJ, 1962.

[4] Kohno T, Kotakemori H, Nini H, et al. Improving the modified gauss-seidel method for Z-matrices [J]. Linear Algebra and its Applications, 1997, 267: 113-123.

[5] Van Der Vorst H A. Krylov subspace iteration [J]. Computing in Science and Engineering, 2000, 2(1): 32-37.

[6] Pearson J W, Pestana J. Preconditioners for Krylov subspace methods: an overview [J]. GAM-Mitteilungen, 2020, 43(4): e202000015.

[7] Anzt H, Gates M, Dongarra J, et al. Preconditioned Krylov solvers on GPUs [J]. Parallel Computing, 2017, 68: 32-44.

[8] Gao Jia-quan, Wang Zhi-chao. Research of the SSOR sparse approximate inverse preconditioner on GPUs [J]. Journal of Zhejiang University of Technology, 2016, 44(2): 140-145.

[9] Davina A L, Roman J E. MPI-CUDA parallel linear solvers for

block-tridiagonal matrices in the context of SLEPc's eigensolvers [J]. Parallel Computing, 2018, 74: 118-135.

[10] Geist A, Reed D A. A survey of high-performance computing scaling challenges [J]. International Journal of High Performance Computing Applications, 2017, 31(1): 104-113.

[11] Bohacek J, Kharicha A, Ludwig A, et al. A GPU solver for symmetric positive-definite matrices vs. traditional codes [J]. Computers & Mathematics with Applications, 2019, 78(9): 2933-2943.

[12] Anzt H, Huckle T K, Brackle J, et al. Incomplete sparse approximate inverses for parallel preconditioning [J]. Parallel Computing, 2018, 71: 1-22.

[13] Jiao Y Y, Zhao Q, Wang L, et al. A hybrid MPI/OpenMP parallel computing model for spherical discontinuous deformation analysis [J]. Computers and Geotechnics, 2019, 106: 217-227.

[14] Suryanarayana P, Pratapa P P, Pask J E. Alternating anderson-richardson method: an efficient alternative to preconditioned Krylov methods for large, sparse linear systems [J]. Computer Physics Communications, 2019, 234: 278-285.

[15] Bernaschi M, D'ambra P, Pasquini D. AMG based on compatible weighted matching for GPUs [J]. Parallel Computing, 2020, 92: 102599.1-102599.13.

[16] Aliaga J I, Dufrechou E, Ezzatti P, et al. An efficient GPU version of the preconditioned GMRES method [J]. The Journal of Supercomputing, 2019, 75(3): 1455-1469.

[17] Sanjuan G, Margalef T, Cortes A. Wind field parallelization based on Schwarz alternating domain decomposition method [J]. Future Generation Computer Systems, 2018, 82: 565-574.

[18] Kameari A. Improvement of ICCG convergence for thin elements in magnetic field analyses using the finite-element method [J]. IEEE Transactions on Magnetics, 2008, 44(6): 1178-1181.

[19] Delgado C, Catedra M F. Sparse approximate inverse preconditioner with parametric sparsity pattern applied to the macrobasis function methods [J]. IEEE Antennas and Wireless Propagation Letters, 2018, 17(5): 849-852.

[20] Kang S, Choi H, Chung W, et al. Performance comparison of various parallel incomplete LU factorization preconditioners for domain decomposition method [J]. Journal of Mechanical Science and Technology, 2018, 32(11): 5315-5323.

[21] Anzt H, Dongarra J, Flegar G, et al. Adaptive precision in block-Jacobi preconditioning for iterative sparse linear system solvers [J]. Concurrency and Computation: Practice and Experience, 2019, 31(6): e4460.1-e4460.12.

[22] Kreutzer M, Hager G, Wellein G, et al. A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units [J]. SIAM Journal on Scientific Computing, 2014, 36(5): 401-423.

[23] Zardoshti P, Khunjush F, Sarbzai-Azad H. Adaptive sparse matrix representation for efficient matrix-vector multiplication [J]. The Journal of Supercomputing, 2016, 72(9): 3366-3386.

[24] Chunduri S, Parker S, Balaji P, et al. Characterization of mpi usage on a production supercomputer [C] //International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2018: 386-400.

[25] Kolodziej S P, Aznaveh M, Bullock M, et al. The suitesparse matrix collection website interface [J]. Journal of Open Source Software, 2019, 4(35): 1244.1-1244.4.

附中文参考文献:

[8] 高家全, 王志超. 基于 GPU 的 SSOR 稀疏近似逆预条件研究 [J]. 浙江工业大学学报, 2016, 44(2): 140-145.