文章编号:1671-6833(2022)06-0015-07

基于 CPU+GPU 异构并行的广义共轭余差算法性能优化

黄东强, 黄建强, 贾金芳, 吴 利, 刘令斌, 王晓英

(青海大学 计算机技术与应用系,青海 西宁 810016)

摘 要: 为了提高 GRAPES 数值天气预报模式的计算效率,改善动力框架部分的性能,针对广义共轭余差算法(GCR)求解赫姆霍兹方程在 GRAPES 模式中耗时较大的问题,提出了一种基于 CPU+GPU 异构并行的预处理广义共轭余差算法。采用不完全 LU 分解对系数矩阵进行预处理来减少迭代次数,在此基础上实现了 OpenMP 的细粒度并行和 MPI 粗粒度并行,OpenMP 并行主要是采用循环展开的方式对程序中无数据依赖的循环体使用编译制导来提高程序的性能;MPI 并行主要是将数据划分给各个进程,采用非阻塞通信和优化进程通信数据量的方式来提高并行程序的可拓展性。实现了 MPI+CUDA 异构并行,MPI 负责节点间进程通信以及迭代控制,CUDA 负责处理计算密集型任务,将 GCR 中耗时较大的矩阵计算部分移植到 GPU 上处理,采用访存优化和数据传输优化来减少 CPU 和 GPU 间的数据传输开销。实验结果表明:与串行程序相比,OpenMP 并行加速比为 2.24,MPI 并行加速比为 3.32,MPI+CUDA 异构并行加速比为 4.69,实现了异构平台上的广义共轭余差算法性能优化,提高了程序的计算效率。

关键词: GRAPES; 广义共轭余差算法; GPU; 异构并行

中图分类号: TP311.1;O244 **文献标志码**: A **doi**:10.13705/j.issn.1671-6833.2022.03.018

0 引言

天气预报是人类农业生产、工作生活中必不可少的一部分,数值天气预报模式是目前气候预测的主要手段。中国从 2001 年开始自主研发 GRAPES (global/regional assimilation and prediction system) 数值天气预报系统[1]。

由于气象预报模式的计算量巨大,并且随着模式分辨率的提升,数值模式中积分计算量和计算耗时成倍增加。如何提高模式的计算效率是气象科研人员一直以来的研究重点。科研工作者们对数值模式并行优化的方向主要分为 2 种:一种是使用不同的体系架构并行,例如 CPU 多核并行[2]、GPU 并行[3]以及国产申威异构众核并行[4-5]等;另一种是优化模式中的算法,例如采用预处理策略减少求解赫姆霍兹方程的迭代次数[6]、采用通信避免策略减少进程通信的开销[7]等。近年来,硬件架构不断升级,GPU 浮点计算能力远超于 CPU,因此 CPU+GPU 异构并行为数值模式计算提供了进一步优化的思路。

GRAPES 模式中,广义共轭余差法(generalized conjugate residual method,GCR)求解赫姆霍兹方程是动力框架模块的计算核心[8]。主流的 CPU 并行编程方法在一定程度上为 GCR 求解赫姆霍兹方程提供了良好的性能[9],但是随着数值模式分辨率进一步提高,模式计算量倍增。MPI 的进程通信开销限制了 GCR 并行的性能,为了提高 GCR 并行的计算效率和可拓展性,本文采用优化进程间数据通信量和非阻塞通信方式来减少进程间的通信开销。此外,稀疏矩阵向量乘法(sparse matrix-vector multiplication,SpMV)计算占了 GCR 算法的大部分时间。本文采用按行压缩存储(CSR)来减少 SpMV 计算时的访存开销,同时优化了 CPU 和 GPU 之间的数据传输量,实现 CPU+GPU 异构并行。

1 GRAPES 中的赫姆霍兹方程

数值模式的计算效率影响着天气预报的性能,其中动力框架中的赫姆霍兹方程求解占了整个模式计算时间的 1/3,因此,并行优化赫姆霍兹方程求解的计算效率尤为重要。

迭代法求解大规模线性方程组时,矩阵条件数越小迭代算法收敛效果越好。采用不完全 LU

收稿日期:2022-02-13;**修订日期**:2022-06-20

基金项目:青海省科技计划项目-应用基础研究计划(2022-ZJ-701);国家自然科学基金资助项目(62062059,62162053)

通信作者:黄建强(1985—)男,陕西西安人,青海大学副教授,主要从事高性能计算研究,E-mail:hjqxaly@163.com。

分解(incomplete LU factorization,ILU)[10]对矩阵进行预处理可以通过改善矩阵条件数来减少循环迭代次数。在ILU预处理的基础上使用OpenMP、MPI、MPI+CUDA并行优化GCR来进一步提高程序的计算效率。利用OpenMP对GCR中无数据依赖的循环体进行细粒度优化;使用MPI多进程数据并行,采用非阻塞通信方式完成计算和通信的重叠,通信时只需对相邻进程间重叠区域进行数据交互;对稀疏矩阵按行压缩存储来提高访存效率,将计算密集型的矩阵计算任务移植到GPU上处理,实现CPU+GPU异构并行。

赫姆霍兹方程是由大气基本运动方程组[11]经过时间空间离散化处理后得到的,将其有限差分离散化后得

$$(\zeta_{\Pi 0})_{i,j,k} = B_1(\Pi)_{i,j,k} + B_2(\Pi)_{i-1,j,k} + B_3(\Pi)_{i+1,j,k} + B_4(\Pi)_{i,j-1,k} + B_5(\Pi)_{i,j+1,k} + B_6(\Pi)_{i+1,j+1,k} + B_7(\Pi)_{i+1,j-1,k} + B_8(\Pi)_{i-1,j-1,k} + B_9(\Pi)_{i-1,j+1,k} + B_{10}(\Pi)_{i,j,k-1} + B_{11}(\Pi)_{i-1,j,k-1} + B_{12}(\Pi)_{i+1,j,k-1} + B_{13}(\Pi)_{i,j-1,k-1} + B_{14}(\Pi)_{i,j+1,k-1} + B_{15}(\Pi)_{i,j,k+1} + B_{16}(\Pi)_{i-1,j,k+1} + B_{17}(\Pi)_{i+1,j,k+1} + B_{18}(\Pi)_{i,j-1,k+1} + B_{19}(\Pi)_{i,j+1,k+1}。 \quad (1)$$

式中:i、j、k分别表示经度、纬度以及垂直方向坐标值;其他各系数详见文献[11]。以空间下标形式展开后得到相关系数分布如图1所示[11],每个格点的计算只与其周围19个点有关(包括格点自身)。

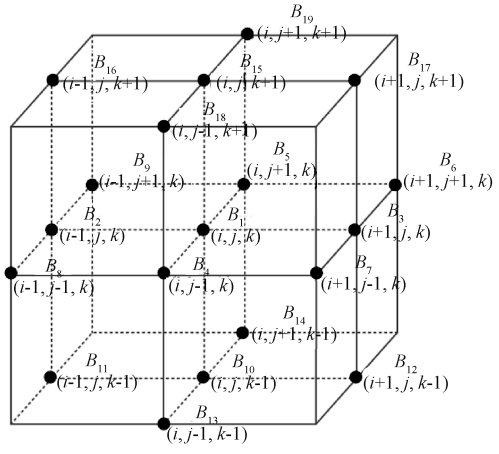

图1 离散点相关系数分布

Figure 1 Discrete point coefficient distribution

以GRAPES模式中1°数据为例,网格数据规模为360×180×38,赫姆霍兹方程未知量个数为总格点数2 462 400,方程总个数为2 462 400,其系数矩阵规模为2 462 400×2 462 400。每个方程最多有19个系数不为零,所以该系数矩阵为大型多对角稀疏矩阵。赫姆霍兹方程可简化为

$$\boldsymbol{Ax} = \boldsymbol{b}。 \quad (2)$$

式中:$\boldsymbol{A}$为赫姆霍兹方程的系数矩阵;$\boldsymbol{x}$为解向量。求解线性方程组的方法包括迭代法和直接法,由于GRAPES模式中的方程组未知量规模与计算量过于庞大,直接法不能有效处理,因此,采用迭代法来求解赫姆霍兹方程。迭代法求解赫姆霍兹方程的算法有广义最小余差法、广义共轭余差法等。本文GRAPES模式中采用广义共轭余差法来进行求解。

2 广义共轭余差算法及优化

2.1 广义共轭余差算法

对于大规模的赫姆霍兹方程来说,其系数矩阵条件数较高,迭代收敛速度较慢,因此,对系数矩阵进行预处理以减少迭代次数。迭代法的预处理子是指构造预处理矩阵$\boldsymbol{M}^{-1}$使得$\boldsymbol{M}^{-1}\boldsymbol{A}$谱分布更加集中,以此来加速迭代收敛。预处理子的构造方法有稀疏近似逆、多项式预处理、ILU分解预处理等。其中ILU分解预处理方法适用范围较广并且在求解大规模线性方程组时有较好的预处理效果[12]。

通过预处理子可以将原始线性方程组转化为易于求解的线性方程组:

$$\boldsymbol{A} = \boldsymbol{LU} - \boldsymbol{R}。 \quad (3)$$

如式(3)所示,ILU预处理将系数矩阵$\boldsymbol{A}$分解成上三角矩阵$\boldsymbol{U}$、下三角矩阵$\boldsymbol{L}$和剩余矩阵$\boldsymbol{R}$。计算分解因子$\boldsymbol{L}$和$\boldsymbol{U}$的近似稀疏矩阵$\boldsymbol{L}\approx\boldsymbol{L}^{-1}$,$\boldsymbol{U}\approx\boldsymbol{U}^{-1}$;之后以$\boldsymbol{M}^{-1}=\boldsymbol{L}^{-1}\boldsymbol{U}^{-1}$作为$\boldsymbol{A}$的近似矩阵,也就是系数矩阵的预处理子。通过舍弃策略(预先决定零元素集合P,设置某些位置元素为0)保持上三角矩阵和下三角矩阵的稀疏性并且减少了计算量,同时可以加快迭代法的收敛速度。ILU预处理伪代码如下。

① For $k=1,2,\cdots,n-1$ Do:
② For $i=k+1,k+2,\cdots,n$ And$(i,k)\notin P$ Do:
③ $a_{ik}=a_{ik}/a_{kk}$
④ For $j=k+1,k+2,\cdots,n$ And$(i,j)\notin P$ Do:
⑤ $a_{ij}=a_{ij}/a_{ik}\cdot a_{kj}$
⑥ EndDo
⑦ EndDo
⑧ EndDo

采用ILU分解预处理矩阵,在求解过程中需要进行多次迭代计算,直至收敛到目标精度$norm$。GCR求解赫姆霍兹方程伪代码[13]如下。

① Compute:$\boldsymbol{R}_0=\boldsymbol{b}-\boldsymbol{Ax}_0$ $\hat{\boldsymbol{R}}_0=\boldsymbol{M}^{-1}\boldsymbol{R}_0$ $\boldsymbol{p}_0=\hat{\boldsymbol{R}}_0$
② Do $i=1,2,\cdots,N$ or $norm<1E-10$

③ $\alpha_{i-1}=\langle \boldsymbol{R}_{i-1},\boldsymbol{Ap}_{i-1}\rangle/\langle \boldsymbol{Ap}_{i-1},\boldsymbol{Ap}_{i-1}\rangle$

④ $\boldsymbol{x}_i=\boldsymbol{x}_{i-1}+\alpha_{i-1}\boldsymbol{Ap}_{i-1}$

⑤ $\boldsymbol{R}_i=\boldsymbol{R}_{i-1}-\alpha_{i-1}\boldsymbol{Ap}_{i-1}$

⑥ $\hat{\boldsymbol{R}}_i=\boldsymbol{M}^{-1}\boldsymbol{R}_i$

⑦ Do $j=\mathrm{int}[(i-1)/k]k$ to $i-1$

⑧ $\boldsymbol{\beta}_{ij}=-\langle \boldsymbol{A}\hat{\boldsymbol{R}}_i,\boldsymbol{Ap}_j\rangle/\langle \boldsymbol{Ap}_j,\boldsymbol{Ap}_j\rangle$

⑨ EndDo

⑩ $\boldsymbol{p}_i=\hat{\boldsymbol{R}}_i+\sum_{j=\mathrm{int}[(i-1)/k]}^{i-1}\boldsymbol{\beta}_{ij}\boldsymbol{p}_j$

⑪ $\boldsymbol{Ap}_i=\boldsymbol{A}\hat{\boldsymbol{R}}_i+\sum_{j=\mathrm{int}[(i-1)/k]k}^{i-1}\boldsymbol{\beta}_{ij}\boldsymbol{Ap}_j$

⑫ EndDo

GCR 求解赫姆霍兹方程的具体步骤如下:

Step 1　计算 $\boldsymbol{R}=\boldsymbol{b}-\boldsymbol{Ax}$ 的初始值以及 ILU 预处理子;

Step 2　进入循环迭代,当循环迭代次数超过最大迭代次数 N 或计算精度达到要求时,退出循环;

Step 3　计算 α 搜索步长,需要先进行矩阵向量乘,之后分子分母分别进行点积;

Step 4　α 乘以向量后与 $\boldsymbol{x}$ 进行向量间的加法;

Step 5　α 乘以向量后与 $\boldsymbol{R}$ 进行向量间的减法;

Step 6　ILU 分解,前代回代更新 $\boldsymbol{R}$ 向量的值;

Step 7　根据最近的 k 个搜索矩阵,求 $\boldsymbol{\beta}$ 矩阵修正步长;

Step 8　通过修正步长 $\boldsymbol{\beta}$ 更新 $\boldsymbol{p}$ 向量的值以及 $\boldsymbol{Ap}$ 向量的值。

设 n 阶系数矩阵 $\boldsymbol{A}$ 每行非零元素个数为 k,Krylov 子空间维度为 m,每次迭代中计算次数以及计算步复杂度如表 1 所示。其中矩阵向量乘计算占据了 GCR 大部分时间,因此采用 CPU+GPU 异构并行来提高整体计算效率。

表 1　GCR 中计算步复杂度

Table 1　Complexity of computational step in GCR

计算步	复杂度	计算次数
矩阵向量乘	$O(kn)$	2
向量点积	$O(n)$	$m+3$
向量加减	$O(n)$	$m+2$
ILU 前代回代	$O(kn)$	1

2.2　迭代算法的比较

迭代法求解大规模线性方程组的方法有广义共轭余差法(GCR)、共轭梯度法(conjugate gradient method,CG)[14]、广义极小残余法(generalized minimal residual method,GMRES)[15]等。实验使用 GRAPES 模式 1°分辨率数据测试不同迭代算法到达收敛精度(1E-10)时所需要的迭代次数,结果如表 2 所示。其中 GCR、CG、GMRES 未进行预处理,ILU-GCR 采用 ILU 分解进行预处理,Jacobi-GCR 采用 Jacobi 矩阵进行预处理。

表 2　不同算法的迭代次数

Table 2　Number of iterations of different algorithms

算法	迭代次数/次	算法	迭代次数/次
GCR	134	ILU-GCR	21
CG	210	Jacobi-GCR	68
GMRES	904		

GCR 算法相比于 CG 以及 GMRES 算法,其迭代次数更少。GMRES 和 CG 算法只比 GCR 算法少一次矩阵向量乘操作,从整体的计算量角度来看,GCR 算法更具有优势。对 GCR 算法进行 ILU 预处理和 Jacobi 预处理,迭代次数由 134 次分别减少到 21、68 次。可以看出,ILU 预处理赫姆霍兹方程效果优于 Jacobi 预处理。与 GCR 程序相比,ILU-GCR 加速比达到了 3.78(加速比=原程序的计算耗时/优化后程序的计算耗时)。因此,本文基于 ILU 预处理的 GCR 算法进行 CPU+GPU 异构并行。

2.3　GCR 性能分析

在搭载 Intel Xeon Gold 6242 处理器的服务器上使用 vtune 对串行 ILU-GCR 程序进行测试分析,测试数据为 GRAPES 模式 1°分辨率数据,结果如图 2 所示。

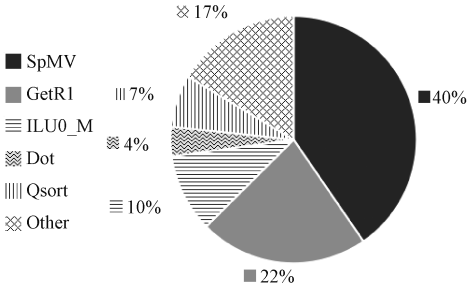

图 2　ILU-GCR 串行程序各个模块占比

Figure 2　Proportion of ILU-GCR modules

在 GCR 中,稀疏矩阵向量乘 SpMV 是算法主要耗时模块,占了整个 GCR 算法时间的 40%;Qsort 是对 CSR 格式系数矩阵数据进行快速排序,以提高稀疏矩阵计算的访存效率;ILU0_M 是预处理子;GetR1 是 ILU 前代回代计算。ILU0_M 和 GetR1 中存在数据依赖,无法很好地实现 CUDA 并行优化,因此,将矩阵计算部分移植到 GPU 上处理以充分发挥 GPU 的计算能力,提高程序的性能。

3 CPU+GPU 异构并行优化 GCR

CPU+GPU 异构并行结构如图 3 所示。MPI 启动多进程将数据划分为多个区域块,每个进程负责各自区域块上的计算,当进行迭代控制与 MPI 通信任务时,进程之间需要进行全局通信以及 Halo 区(数据重叠区域)数据交互[16]。对于计算密集型的任务(矩阵向量乘、向量点积等)采用 CUDA 并行,先通过 PCIE(peripheral component interconnect express)将计算数据从 CPU 端拷贝到 GPU 端,在 GPU 端调用多个线程计算完成后,再将数据拷贝回 CPU 端,以此来实现 CPU+GPU 异构并行。

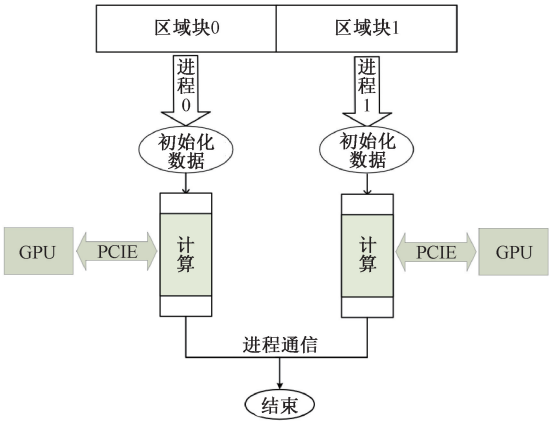

图 3 CPU+GPU 异构并行结构

Figure 3 CPU+GPU heterogeneous parallel structure

3.1 MPI 优化

3.1.1 数据并行划分区域块

通过数据并行方式优化 GCR 算法,根据处理器的数量对计算数据进行切分,将数据分配给各个进程进行计算,每个进程只需要对自己独占的区域块进行计算。假设 MPI 启动 N 个进程,将总数据划分为规模大致相同的 N 个区域块,每个进程处理其中一块区域。

MPI 进程划分区域块,如图 4 所示,考虑左右边界情况将首进程与尾进程相连成环状,相邻的 2 个进程间需要进行通信。传统的数据并行方式是各进程完成计算之后进行全局数据汇总,这样各个进程间都需要进行数据交互。但是对于迭代法来说,每次迭代时进程只需计算自己所负责的区域块以及区域块的边界部分(即相邻进程之间数据重叠区域),各个进程只需对 Halo 区数据进行交互。这样进程间数据交互就从一对多(全局数据交互)变成一对二(相邻进程数据交互)。由此可以进一步减少节点的通信开销,提高程序的可拓展性。

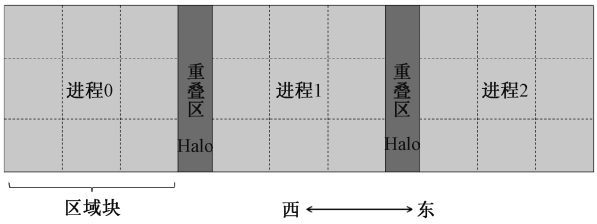

图 4 进程区域块示意图

Figure 4 Schematic diagram of process area block

3.1.2 非阻塞通信

通过 MPI_Isend 和 MPI_Irecv 实现进程间异步通信,这使得进程在通信过程中仍然可以进行其他的计算操作。每个进程开辟了 2 块用于数据通信缓存的内存空间:一个是用于数据发送的空间,另一个是用于数据接收的空间。

相邻进程之间对 Halo 区进行数据交互。进程将数据发送到目标进程的消息缓冲区后即可继续执行下一步计算,无须等待发送和接收成功,当需要缓冲区的数据时,调用 MPI_Wait 函数来检测或者等待进程完成,如图 5 所示,由此实现计算和通信重叠,提高程序并行效率。

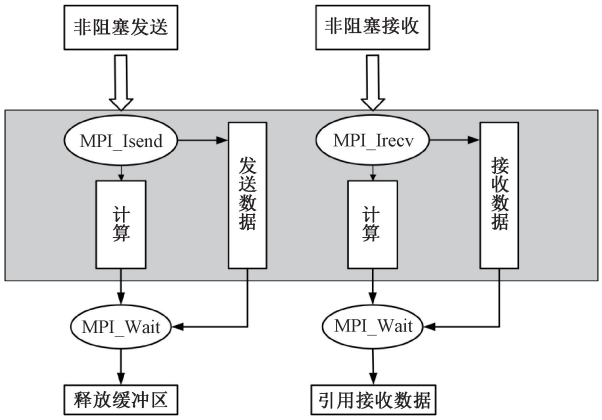

图 5 MPI 非阻塞通信结构

Figure 5 MPI non-blocking communication structure

3.2 CUDA 优化

将大规模矩阵计算移植到 GPU 上,主要是对 GCR 算法中的矩阵向量乘 SpMV 进行计算。CUDA 优化 GCR 算法可分为以下 3 个基本步骤:

Step 1 MPI 进程 cudaMalloc 申请所要计算的空间,cudaMemcpy 将数据从 CPU 端复制到 GPU 端;

Step 2 分配线程块个数及线程数量,调用核函数在 GPU 上进行计算;

Step 3 cudaMemcpy 将计算结果从 GPU 端拷贝回 CPU 端。

3.2.1 访存优化

对于稀疏矩阵向量乘计算,一般的二维数组格式存储稀疏矩阵所占的存储空间以及访存开销

较大。因此采用 CSR 格式存储,可以有效提高矩阵计算效率以及减少内存空间开销,如图 6 所示。

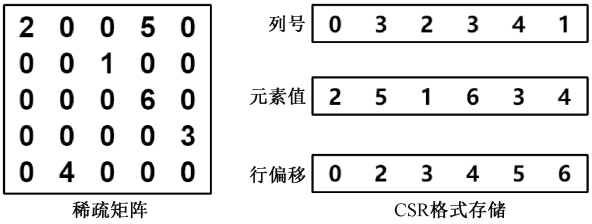

图 6　CSR 格式存储示意图

Figure 6　CSR format storage diagram

CSR 存储矩阵非零元素所在的列号、元素值以及每一行的行偏移。通过行偏移 $ptr[i]$ 确定第 i 行首元素序号,通过 $ptr[i+1]$ 确定下一行首元素序号,以此确定矩阵中一行所有的非零元素,根据线程块 id 和线程 id 来划分每个线程所需计算的行,以减少矩阵向量乘的计算耗时。

3.2.2　传输量优化

在 GPU 上计算 SpMV 时都要进行 CPU 与 GPU 之间的数据传输,计算前通过 cudaMemcpy 将系数矩阵以及向量从 CPU 拷贝到 GPU 上,计算完成后将解向量拷贝回 CPU。在 GCR 算法中每一次迭代时矩阵向量乘的系数矩阵不变,因此系数矩阵在 GPU 显存中初始化之后保持常驻。

每次进行 SpMV 计算时 CPU 与 GPU 之间数据传输量为 $\boldsymbol{A}_{\mathrm{CSR\_array}}+\boldsymbol{b}+\boldsymbol{x}$。经过传输量优化之后除了第 1 次初始化 SpMV,每次 CPU 与 GPU 之间数据传输量为 $\boldsymbol{b}+\boldsymbol{x}$。以此进一步减少 CPU 与 GPU 之间的数据传输开销。

4　实验与结果分析

4.1　实验环境与测试用例

实验中节点操作系统为 Linux Ubuntu(内核版本为 4.15.0-91-generic),处理器为 Intel Xeon Gold 6242 CPU,其主频为 2.8 GHz,每个节点上有 2 块 CPU(共 32 个核心);2 块 Tesla T4 GPU,显存为 32 GB。实验环境中有 CUDA、MPI 及 vtune 测试工具、gcc 编译器等。测试的算例是 GRAPES 模式 1°分辨率数据(360×180×38 规模的网格点),格点总数为 2 462 400。

4.2　串行及 OpenMP 优化分析

测试 ILU-GCR 串行程序以及 OpenMP 并行 ILU-GCR 程序的计算耗时如图 7 所示。通过 ILU 预处理来进一步减少迭代次数,预处理后的串行程序收敛加速效果明显,ILU-GCR 加速比为 3.78。OpenMP 并行主要是针对无数据依赖的循环体,通过调用多个线程,加速循环体的计算效率。随着 OpenMP 线程数的不断增加,ILU-GCR 计算时间不断减少,在线程数为 32 时,与串行程序相比,OpenMP 并行加速比达到了 2.24。

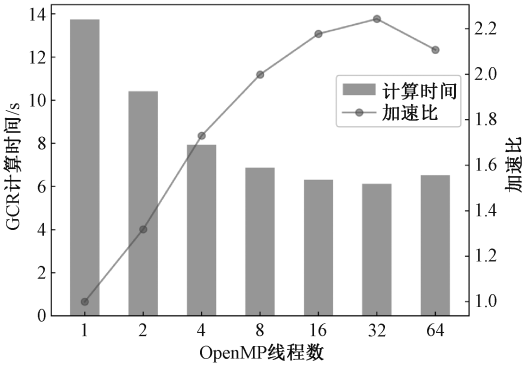

图 7　不同线程下 OpenMP 优化 ILU-GCR

Figure 7　OpenMP optimized ILU-GCR in different threads

单核 CPU 同一时间只能处理一个线程,OpenMP 并行是通过 CPU 调度多线程,线程过多则需要排队等待 CPU 执行。因此,在本实验环境中当线程数超过 32 之后(一个节点上只有 32 核),OpenMP 加速效果变差。

4.3　MPI 优化分析

在不同进程下,MPI 优化 ILU-GCR 的加速比如图 8 所示。在进程数为 16 时,进程通信开销成为 MPI 并行的性能瓶颈。随着进程数的增多,通信开销逐渐大于计算开销,导致程序的可拓展性较差。本文对进程通信数据量进行优化以及采用非阻塞通信方式优化之后,随着进程数的增加,ILU-GCR 计算时间呈线性递减,提高了程序的可拓展性和计算效率。

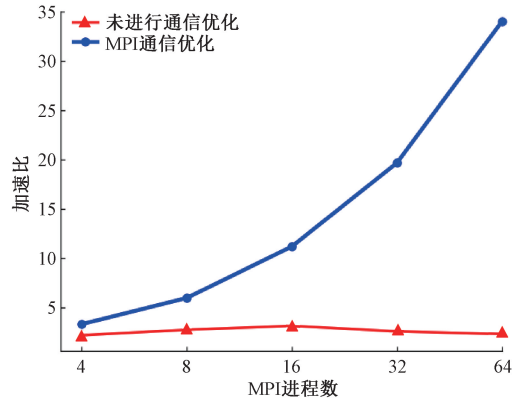

图 8　不同进程下 MPI 优化 ILU-GCR

Figure 8　MPI optimized ILU-GCR in different processes

MPI 粗粒度并行比 OpenMP 细粒度优化效果更好,它从程序初始化阶段就已进入了并行模式,

各个进程共同执行程序并各自计算所负责的区域块。与串行程序相比,4进程下并行ILU-GCR的加速比为3.32,64进程下并行ILU-GCR的加速比为34.01。

4.4 CPU+GPU并行分析

实验将ILU-GCR算法中的大规模矩阵计算部分移植到GPU上,通过CPU进程调用GPU上的kernel计算函数来完成MPI+CUDA混合并行,采用压缩矩阵行方式来减少计算量以及提高访存效率。将系数矩阵常驻在GPU显存中来减少迭代过程中CPU和GPU之间的数据传输量。

CPU-GPU异构并行ILU-GCR算法后,在进程数为4时,MPI+CUDA并行ILU-GCR加速比相较于MPI并行ILU-GCR为1.41,相较于串行程序为4.69。

5 结论

本文使用CPU和GPU上的3种并行编程方式(OpenMP、MPI、CUDA)并行优化GRAPES模式中的计算核心GCR,以求解赫姆霍兹方程。采用不完全LU分解预处理减少迭代次数,在此基础上,使用OpenMP结合共享内存细粒度并行ILU-GCR算法;在MPI并行中采用非阻塞通信和优化进程通信量的方式使ILU-GCR程序性能有良好的可拓展性,随着进程数的增多,GCR的计算时间呈线性减少;通过访存优化和CPU/GPU间数据传输量优化使MPI+CUDA异构并行ILU-GCR的计算效率进一步提高。实验结果表明:在Intel Xeon Gold 6242 CPU、Tesla T4 GPU的平台上与ILU-GCR串行程序相比,OpenMP、MPI以及MPI+CUDA异构并行ILU-GCR的加速比分别达到了2.24、3.32、4.69。

参考文献:

[1] SHEN X S, WANG J J, LI Z C, et al. Research and operational development of numerical weather prediction in China[J]. Journal of meteorological research, 2020, 34(4): 675-698.

[2] 张志远,周宇峰,刘利,等. MASNUM海浪模式的性能特点分析与并行优化[J]. 计算机研究与发展, 2015, 52(4): 851-860.
ZHANG Z Y, ZHOU Y F, LIU L, et al. Performance characterization and efficient parallelization of MASNUM wave model[J]. Journal of computer research and development, 2015, 52(4): 851-860.

[3] 肖洒,魏敏,邓帅,等. 基于GPU-OpenACC的气候模式加速优化研究[J]. 气象, 2019, 45(7): 1001-1008.
XIAO S, WEI M, DENG S, et al. Research on accelerated optimization of climate models based on GPU-OpenACC[J]. Meteorological monthly, 2019, 45(7): 1001-1008.

[4] 魏敏,王彬,何香,等. BCCAGCM模式在神威·太湖之光系统的优化[J]. 应用气象学报, 2019, 30(4): 502-512.
WEI M, WANG B, HE X, et al. Optimizing BCCAGCM on sunway TaihuLight[J]. Journal of applied meteorological science, 2019, 30(4): 502-512.

[5] DIWAN G C, MOHAMED M S. Iterative solution of Helmholtz problem with high-order isogeometric analysis and finite element method at mid-range frequencies[J]. Computer methods in applied mechanics and engineering, 2020, 363: 112855.

[6] 金之雁,杨磊,林隽民,等. 广义共轭余差法的通信避免算法[J]. 计算机工程与应用, 2020, 56(3): 74-79.
JIN Z Y, YANG L, LIN J M, et al. Communication avoiding algorithm of generalized conjugate residual method[J]. Computer engineering and applications, 2020, 56(3): 74-79.

[7] LI L F, XUE W, RANJAN R, et al. A scalable Helmholtz solver in GRAPES over large-scale multicore cluster[J]. Concurrency and computation: practice and experience, 2013, 25(12): 1722-1737.

[8] 刘钊. 基于国产高性能计算机的GRAPES性能优化研究[D]. 上海: 上海交通大学, 2014.
LIU Z. Study of GRAPES numerical weather prediction system optimization on domestic high performance computers[D]. Shanghai: Shanghai Jiao Tong University, 2014.

[9] 王卓薇,许先斌,赵武清,等. 基于GPU的GRAPES模型并行加速及性能优化[J]. 计算机研究与发展, 2013, 50(2): 401-411.
WANG Z W, XU X B, ZHAO W Q, et al. Parallel acceleration and performance optimization for GRAPES model based on GPU[J]. Journal of computer research and development, 2013, 50(2): 401-411.

[10] 王克文,冶梦雨,刘艳红. 建立电力系统状态空间方程的并行方法[J]. 郑州大学学报(工学版), 2021, 42(1): 15-20.
WANG K W, YE M Y, LIU Y H. Parallel method for establishing state space equation of power system[J]. Journal of Zhengzhou university (engineering science), 2021, 42(1): 15-20.

[11] 薛纪善,陈德辉. 数值预报系统GRAPES的科学设计与应用[M]. 北京: 科学出版社, 2008.

XUE J S, CHEN D H. Scientific design and application of numerical prediction system GRAPES [M]. Beijing: Science Press, 2008.

[12] 李建斌,王鹏程,傅侃,等. 基于预处理共轭梯度迭代法的电力系统状态估计算法[J]. 电力系统自动化, 2021, 45(14): 90-96.
LI J B, WANG P C, FU K, et al. State estimation algorithm of power system based on preconditioned conjugate gradient iteration[J]. Automation of electric power systems, 2021, 45(14): 90-96.

[13] 刘宇,曹建文. 适用于 GRAPES 数值天气预报软件的 ILU 预条件子[J]. 计算机工程与设计, 2008, 29(3): 731-734.
LIU Y, CAO J W. ILU preconditioner for NWP system: GRAPES[J]. Computer engineering and design, 2008, 29(3): 731-734.

[14] LI Y, XIE H H, XU R, et al. A parallel generalized conjugate gradient method for large scale eigenvalue problems[J]. CCF transactions on high performance computing, 2020, 2(2): 111-122.

[15] GANDER M J, GRAHAM I G, SPENCE E A. Applying GMRES to the Helmholtz equation with shifted Laplacian preconditioning: what is the largest shift for which wavenumber-independent convergence is guaranteed[J]. Numerische mathematik, 2015, 131(3): 567-614.

[16] HUANG J Q, XUE W, BIAN H D, et al. Helmholtz solving and performance optimization in global/regional assimilation and prediction system[J]. Tsinghua science and technology, 2020, 26(3): 335-346.

Performance Optimization of GCR in GRAPES Based on CPU+GPU Heterogeneous Parallel

HUANG Dongqiang, HUANG Jianqiang, JIA Jinfang, WU Li, LIU Lingbin, WANG Xiaoying

(Department of Computer Technology and Application, Qinghai University, Xining 810016, China)

Abstract: In order to improve the computational efficiency of the GRAPES(global/regional assimilation and prediction system) numerical weather prediction model, and to improve the performance of the dynamic framework, In order to solve the problem that the GCR algorithm was time-consuming in GRAPES mode, a CPU+GPU heterogeneous parallel preprocessing GCR algorithm was implemented. Firstly, incomplete LU decomposition was used to preprocess the coefficient matrix to reduce the number of iterations. On this basis, fine-grained parallelism of OpenMP and coarse-grained parallelism of MPI were implemented. OpenMP parallelism was mainly used to improve the performance of the program by using compiler guidance to the loop body without data dependence in the way of loop unrolling. MPI parallelism was used to divide the data into various processes and improve the scalability of parallel programs by non-blocking communication and optimizing the amount of communication data. MPI was responsible for process communication and iterative control between nodes, while CUDA was responsible for processing computation-intensive tasks. The time-consuming matrix calculation part of GCR was transferred to GPU for processing, and memory optimization and data transmission optimization were adopted to reduce the data transmission overhead between CPU and GPU. The experimental results showed that the parallel acceleration ratio of OpenMP was 2.24 times that of the serial program, the parallel acceleration ratio of MPI was 3.32 times that of the serial program, and the parallel acceleration ratio of MPI+CUDA was 4.69 times that of the serial program. The performance optimization of the generalized conjugate redundancy algorithm on the heterogeneous platform was realized, and the computational efficiency of the program was improved.

Keywords: GRAPES; generalized conjugate residual method; GPU; heterogeneous parallel