基于MIC集群平台的GMRES算法并行加速

王明清[1] 李 明[2] 张 清[1] 张广勇[1] 吴韶华[1]
(浪潮集团高效能服务器和存储技术国家重点实验室 济南 250101)[1]
(太原理工大学数学学院 太原 030024)[2]

摘 要 广义极小残量法(GMRES)是最常用的求解非对称大规模稀疏线性方程组的方法之一,其收敛速度快且稳定性良好。Intel Xeon Phi众核协处理器(MIC)具有计算能力强、易编程、易移植等特点。采用MPI+OpenMP+offload混合编程模型将GMRES算法移植到MIC集群平台上。采用进程间集合通信异步隐藏、数据传输优化、向量化以及线程亲和性优化等多种手段,大幅提升了GMRES算法的求解效率。最后将并行算法应用到"局部径向基函数求解高维偏微分方程"问题的求解中。测试表明,CPU节点集群上开启32个进程,并行效率高达71.74%,4块MIC卡的最高加速性能可达单颗CPU的7倍。

关键词 广义极小残量法,MIC,MPI,大规模线性方程组
中图法分类号 TP391,O246 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.04.043

Speedup of GMRES Based on MIC Heterogeneous Cluster Platform

WANG Ming-qing[1] LI Ming[2] ZHANG Qing[1] ZHANG Guang-yong[1] WU Shao-hua[1]
(National Key Laboratory for High-efficient Server and Storage Technology, Inspur, Jinan 250101, China)[1]
(School of Mathematics, Taiyuan University of Technology, Taiyuan 030024, China)[2]

Abstract Generalized minimal residual method (GMRES) is the most commonly used method for solving asymmetric large-scale linear algebraic equations, and it has fast convergence and stable property. Intel many integrated co-processors (MIC) has strong computing power and it can program easily. In this paper, MPI+OpenMP+offload hybrid programming paradigm was used to port GMRES algorithm to the MIC heterogeneous cluster platform. The perfor-mance of GMRES parallel algorithm was greatly improved by using kinds of optimization methods, such as hiding collective communications using asynchronous execution model, vectorization optimization, data transfer optimization, extensibility of MIC thread optimization, etc. Finally, GMRES parallel algorithm was used to improve the perfomance of solving high dimensional PDEs by the localized radical basis functions (RBFs) collocation methods. Results from tests indicate that the parallel efficiency can be up to 71.74% when using 32 processes in cluster, and the maximum speedup ratio of 4 MICs to 1 CPU can be up to 7.

Keywords GMRES, MIC, MPI, Large-scale linear algebraic equations

1 引言

随着计算机水平的不断提高,有限差分(FD)、有限元(FEM)、边界元(BEM)、无网格方法(Meshless Method)等一系列的数值计算方法相继诞生,理论体系逐步完善且在众多的科学研究以及工程应用领域得到广泛的应用。这些应用都有一个共同的特点:将数学物理模型的求解转化成一个稀疏线性方程组的求解。随着科技的不断进步和所求问题规模的增大,模型结构越来越复杂且对计算精度的要求也日益提高,使得线性方程组的规模急剧增加,因此高效地求解大规模线性方程组愈发重要。

Saad Y等[1-2]于1986年首次提出的广义极小残量法(Generalized Minimal Residual Method, GMRES)是目前最常用的求解非对称大规模线性方程组的算法之一,也是Krylov子空间中经典的算法之一。该方法通过子空间中矢量的最小残量来迭代求解,具有收敛速度快、稳定性好的优点。为进一步提高收敛速度,N. M. Nachtigal[3],M Bellalij[12]和全忠[5]等国内外许多学者对GMRES算法做了各种变种与预处理工作并取得一定的成果。此外,另一部分科研人员及工程师对GMRES算法的可并行性做了讨论与分析。例如,Ghae-

到稿日期:2016-02-27 返修日期:2016-06-15
王明清(1989—),男,硕士,工程师,主要研究方向为工程数值计算、高性能计算,E-mail:wmingqing2011@126.com;李 明(1982—),男,教授,博士生导师,主要研究方向为科学数值计算、机器学习;张 清 男,硕士,工程师,主要研究方向为高性能计算;张广勇 男,硕士,工程师,主要研究方向为高性能计算;吴韶华 男,博士,高级工程师,主要研究方向为流体力学及高性能计算。

mian[5]，Wang[6]以及柳有权[7]和 He[13]等曾先后将 GMRES 算法移植到 GPU 卡上加速计算，均取得了不错的加速效果。但之前的大部分工作是在单节点或单块 GPU 上完成的，受 GPU 内存所限，当线性方程组增大到一定规模时，求解工作难以进行。而基于分布式内存的消息传递模型并行实现时，Krylov 子空间算法需要大量的全局通信，因此往往难以实现高效率的并行。如何有效地规避或隐藏全局通信时间将成为提高粗粒度并行扩展性的关键。基于 GHYSELS P 等[8]对 GMRES 算法 MPI 通信的讨论、测试与分析，本文对经典的 GMRES 算法的流程进行调整，以便通过通信与计算异步执行的方式隐藏集合通信。

MIC (Many Integrated Core)是英特尔至强融核 Xeon Phi 采用的架构，其主要面向高性能计算领域，旨在引领行业进入百亿亿次计算时代。MIC 芯片集成了几十个精简的 x86 核心，每个核心可支持 4 个硬件线程且含有 512bit 的向量位宽，双精度性能达到 1TFlops 以上，具有高度的并行计算能力和一定的逻辑处理能力。MIC 众核计算支持 OpenMP，pThread，OpenCL，MPI 等常见的并行编程模型，可采用 C/C++，Fortran 等常规的编程语言开发设计且移植简单，支持数学库、热点分析、程序调试等丰富的工具链[9-10]。

为充分发挥 CPU 的逻辑处理能力以及 MIC 的强大计算能力，本文采用以 CPU 为主、MIC 众核处理器协同计算的模式，利用 MPI+OpenMP+offload 混合编程模型实现 GMRES 算法的 MIC 版本。首先，利用 MPI+OpenMP 编程模型实现了 CPU 版本的 GMRES 并行；接着，采用 offload 模式进行 MIC 移植，实现了 MIC 版本的 GMRES 并行；然后，采用各种优化手段对 MIC 版本进行算法优化；最后，将算法应用到局部径向基函数求解高维偏微分方程的案例中，与 CPU 并行计算比较，MIC 版的 GMRES 算法有更高的加速效率。

本文的主要工作包括以下几点：

1)通过算法调整隐藏 MPI 全局通信，实现 GMRES 算法的 CPU 并行版本；

2)完成算法的移植和优化，实现 GMRES 算法的 MIC 并行版本；

3)GMRES 并行算法加速了高维偏微分方程的求解。

本文第 2 节阐述了矩阵的存储方式及 GMRES 算法；第 3 节为基于 MIC 集群平台的 GMRES 算法并行实现及优化；第 4 节为 GMRES 并行算法的应用案例及结果分析；最后总结全文。

2 矩阵存储及 GMRES 算法分析

2.1 稀疏矩阵的 CSR 存储方式

稀疏矩阵指矩阵中的非零元素数量远小于零值元素的数量。若将零值元素全部存储起来，不仅会占用大量的存储资源，影响数据的访问效率，还会增加数倍的乘法操作，因此通常采用只存储非零元素值的压缩方式来避免或降低上述问题带来的影响。常用的压缩格式有 DIA，COO，CSR，ELLPACK 等。相比较而言，CSR 存储格式更加简便，运用更广泛，因此本文采用 CSR 压缩格式来存储大规模的稀疏矩阵，以简化矩阵矢量乘的运算。

CSR 压缩格式通常采用 3 个一维数组(val，cidx，rpos)来表示一个稀疏矩阵，如图 1 所示。

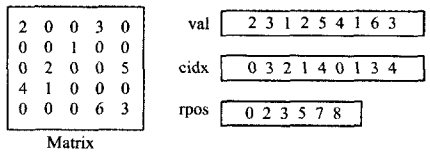

图 1 CSR 压缩存储格式示例

其中，val 存放了稀疏矩阵中所有非零元素的值，长度等于非零元素的个数；cidx 存放了非零元素所在列索引，长度等于非零元素的个数；rpos 存放了矩阵中每行第一个元素在 val 数组中的索引，最后一个元素存放了非零元素的个数，长度为矩阵的行数加 1。

2.2 GMRES 算法

科学研究与工程应用中的数学物理模型经过有限元、有限差分及无网格方法等数值方法离散归结为解稀疏线性方程组：

$$Ax=b$$

其中，$A\in R^{n\times n}$ 为非奇异的稀疏矩阵，$x,b\in R^n$ 分别为未知元与已知向量。当矩阵 A 的规模较大且对称性不能满足时，方程组的求解相对困难。GMRES 算法是求解此类问题的最常用方法之一。

GMRES 算法利用计算中的近似解在相应的 Krylov 子空间中残量范数极小的性质来完成迭代求解，即通过求使残量范数 $\| Ax_m-b \|$ 最小的矢量 $x_m\in K_m$ 来逼近精确解，其中 K_m 为初始残量张成的 m 维 Krylov 子空间。关于 GMRES 算法的具体实现过程可参考文献[1-2]。

GMRES 算法的流程如算法 1 所示。

算法 1 标准 GMRES

1. $r_0\leftarrow b-Ax_0, v_1\leftarrow r_0/\| r_0 \|_2$
2. for j=1,…,m do
3. $w=Av_j$
4. $h_{i,j}=wv_i, w=w-h_{i,j}v_i, i=1,2,\cdots,j$
5. $h_{j+1,j}=\| w \|_2$
6. $v_{j+1}=w/h_{j+1,j}$
7. endfor
8. $y_m\leftarrow \mathrm{argmin}\| (H_{m+1,m}y_m-\| r_0 \|_2 e_1) \|_2$
9. $x\leftarrow x_0+V_m y_m$

文献[8]详细研究了 GMRES 算法在并行集群中大量集合的通信可能造成严重的通信延迟的问题。尤其是在算法 1 中，每次执行第 3 行之前都要收集矢量 v_i 并广播给所有的进程，通信量非常大并且基于算法 1 的流程难以规避或隐藏。本文对文献[8]中的 p1-GMRES 做了适当的调整，得到改进的 GMRES 算法，如算法 2 所示。

算法 2 改进的 GMRES

1. $r_0\leftarrow b-Ax_0, v_1\leftarrow r_0/\| r_0 \|_2; z_0\leftarrow v_0$
2. for i=0,…,m do
3. $w\leftarrow Av_j$
4. if i>1 then
5. $v_{i-1}\leftarrow v_{i-1}\leftrightarrow h_{i-1,i-2}$
6. $z_i\leftarrow z_i/h_{i-1,i-2}$

7. $w \leftarrow w/h_{i-1,i-2}$
8. $h_{j,i-1} \leftarrow h_{j,i-1}/h_{i-1,i-2}, j=0,1,\cdots,i-2$
9. $h_{i-1,i-1} \leftarrow h_{i-1,i-1}/h_{i-1,i-2}^2$
10. end if
11. if $i<m$ then
12. $z_{i+1} \leftarrow w-\sum_{j=0}^{i-1} h_{j,i-1} z_{j+1}$
13. endif
14. if $i>0$ then
15. $v_i \leftarrow z_i-\sum_{j=0}^{i-1} h_{j,i-1} v_j$
16. $h_{i,i-1} \leftarrow \| v_i \|_2$
17. end if
18. if $i<m$ then
19. $h_{j,i} \leftarrow \langle z_{i+1}, v_j \rangle, j=0,1,\cdots,i$
20. end if
21. $v_m \leftarrow v_m/h_{m,m-1}$
22. end for
23. $y_m \leftarrow \text{argmin} \| (H_{m+1,m} y_m - \| r_0 \|_2 e_1) \|_2$
24. $x \leftarrow x_0+V_m y_m$

经过对算法 2 的分析可知，计算过程主要包括稀疏矩阵矢量乘、矢量加减、标量与矢量乘、矢量内积与范数以及 $O(m)$ 阶稠密矩阵与矢量的相关运算。由文献[7]中的计算复杂度分析可知，算法的热点为大规模稀疏矩阵与矢量的相关运算，因此可将其放到 MIC 卡上计算。而 m 为 Krylov 子空间的维数，m 取值越大，收敛速度越快，精度越高，但也会相应增加每次迭代操作的运算量以及存储量，因此 m 取值不宜过大。当前常用的 MIC 协处理器具有 61 个核心，每个计算核心可支持 4 个硬件线程，而 m 的取值通常远小于 $60*4$，不能充分利用 MIC 卡的加速资源。因此可将稠密矩阵与矢量的计算放到 CPU 上完成。

3　基于 MIC 集群平台的 GMRES 并行算法

本文采用 MPI＋OpenMP＋offload 混合编程模型实现 GMRES 算法的并行。假设集群有 N 个计算节点，每个计算节点内配置 M 块 MIC 卡，那么需要开启 $M*N$ 个进程，其中每个进程控制一块 MIC 卡的计算。采用 offload 模式实现 CPU 与 MIC 卡的数据传输，在 MIC 卡内部采用 OpenMP 多线程并行。

3.1　基于 MPI 的 GMRES 并行算法

本文中 MPI 的实现采用按行划分的方式进行数据分配，如图 2 所示。假设开启 l 个进程 $P_0, P_1, \cdots, P_{l-1}$，可将系数矩阵 A 与矢量 b 按行划分成 l 块，即 $A=[A_0^{\mathrm{T}}, A_1^{\mathrm{T}}, \cdots, A_{l-1}^{\mathrm{T}}]^{\mathrm{T}}$，$b=[b_0^{\mathrm{T}}, b_1^{\mathrm{T}}, \cdots, b_{l-1}^{\mathrm{T}}]^{\mathrm{T}}$。将数据块 $A_0 \sim A_{l-1}$ 及 $b_0 \sim b_{l-1}$ 分别分配给进程 $P_0 \sim P_{l-1}$，而矢量 x 为所有进程共享。

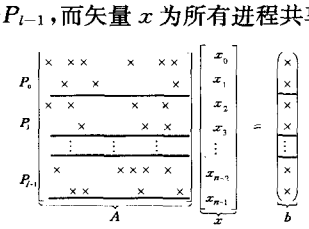

图 2　数据划分示意图

MPI 并行实现的主要思想：开启一个具有 l 个进程的进程组，每个进程根据自己的任务开辟私有存储空间；主进程负责读取数据，并将数据按需广播给进程组中的其他进程；包括主进程在内的所有进程分别执行各自的计算任务，并将计算结果反馈给主进程；主进程处理整合各进程反馈的结果，将必要的结果广播给其他进程；重复前两步直至计算结束。

针对 MPI 通信，在 MPI 并行具体实现中主要采用集合通信的方式完成进程间的消息传递。调用的 MPI 消息传递库函数包括：MPI_Reduce，MPI_ALLReduce，MPI_Bcast 以及 MPI_Allgatherv。在算法 2 中第 3 行矩阵矢量乘法操作之前，主进程首先要收集矢量，之后再将收集的数据广播给组内所有的进程。在程序的运行中执行的次数为 $m*Iter$，$Iter$ 为算法完成的迭代次数。每次通信的数据量大小为 $(4*n)$B。假设求解 5000 万阶的矩阵过程中 Krylov 子空间维数为 30，迭代 20 次满足所需精度，则通信的花费为通信域内分别执行 600 次的收集与广播操作，每次处理的数据量约为 0.186GB，因此通信开销是制约程序性能的重大瓶颈。本文采用计算与集合通信异步隐藏的方式，将处理数据分块，用计算将通信掩藏，其流程如图 3 所示。

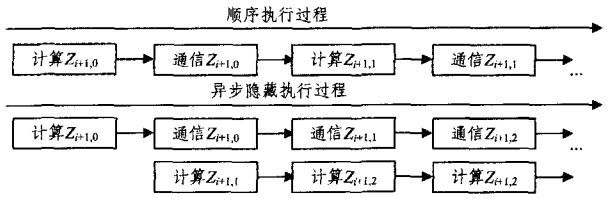

图 3　MPI 通信异步隐藏示意图

3.2　基于 MIC 平台的 GMRES 并行算法

本文采用 offload 编程模式实现 MIC 与 CPU 之间的数据传输与 MIC 内核的调用。指导语句“pragma offload target (mic:num)”将代码段卸载到 MIC 协处理器上运行，关键字 alloc_if 和 free_if 实现 MIC 卡上空间的开辟与释放，关键字 in，out，inout 标示 CPU 端与 MIC 端数据传输长度及传输方向。每个进程控制一块 MIC 卡的数据传输与计算，在 MIC 内部采用共享内存的 OpenMP 多线程并行模式，线程之间的数据采用 OpenMP 自动划分的方式，只需要在待并行的循环语句外加上 omp 编译指导语句即可。MIC 并行算法实现步骤如下：

1）每个进程在 CPU 端开辟自身所需的数组空间，定义相关变量；

2）每个进程在自身控制的 MIC 卡上开辟数组空间，以便数据传输及 MIC 上的计算；

3）当进程遇到以矩阵矢量相乘、矢量与标量相乘、向量内积及向量加减法等 4 类高阶矩阵和矢量运算为主体的计算代码段时，将这 4 类高阶矩阵和矢量运算为主题的计算代码段作为 MIC 内核函数，放到 MIC 卡上计算，此时需要进程控制向 MIC 传输相应的数组元素；

4）当进程进行 MPI 通信时，需由各个进程在 CPU 上完成进程间的通信；MIC 卡上的数据要传输到 CPU 端以便完成消息传递，因此本文算法的 MIC 内核是以 MPI 通信为标志进行划分的；

5)上述 MPI 通信完成之后,程序继续寻找可移植到 MIC 卡上的计算代码段,重复步骤 3)、步骤 4)直至代码全部执行完毕。

将并行度不能满足 MIC 计算的要求的代码段(Krylov 子空间上的矩阵、矢量运算)放在 CPU 端计算。同时为减少数据的传输及代码的复杂度,对于不包含上述任何一个高阶矩阵的矢量运算但规模较大且与高阶矩阵矢量运算相邻的计算代码段,仍将其作为内核的一部分放到 MIC 中进行计算。基于 MPI+offload 模式的 GMRES 并行算法的伪代码如算法 3 所示。

算法 3 基于 MPI+offload 模式的 GMRES 并行算法

```
1. #pragma offload target(mic:num)
2.   ‖b‖2<-sqrt((b,b));
3.  MPI_Reduce(‖b‖2);
4.  for(It=0;It<MAX;It++){
5.  #pragma offload target(mic:num)
6.  temp<-A*x;r0<-b-temp;beta<-‖r0‖2;
7.  MPI_Allreduce(beta);
8.  #pragma offload target(mic:num)
9.  v0<-r0/bata;z0=v0;
10.  MPI_Allgatherv(z0);
11.  for(i=0;i<m+1;++i){
12.  #pragma offload target(mic:num)
13.  w<-A*vj;
14.  if(i>1){
15.  #pragma offload target(mic:num)
16.    Update vi-1,zi,w;
17.      for j=0,1,2,…,i-2,Update hj,i-1 and hi-1,i-1;
18.  }
19.    if(i<m){
20.  #pragma offload target(mic:num)
21.    Generate zi+1;
22.    MPI_Allgatherv(zi+1);
23.  }
24.  if(i>0){
25.  #pragma offload target(mic:num)
26.    Generate vi and hi,i-1
27.    MPI_Allreduce(hi,i-1);
28.  }
29.  if(i<m){
30.  #pragma offload target(mic:num)
31.  for j=0,1,…,i,generate hj,i;
32.  MPI_Allreduce(hi(:));
33.  }
34.}
35.    #pragma offload target(mic:num)
36.    vm<-vm/hm,m-1;
37.  }
38.  ym<-arg min‖(hm+1,mym-‖r0‖2e1)‖2;
39.  x<-x0+Vmym;
40.  MPI_Allgatherv(x);
41.}
```

3.3 程序优化方法

(1)MIC 空间开辟优化

为了加快 MIC 卡上空间的首次开辟速度,通常使用 2MB 的大页面取代 4kB 的页面进行存储。这是因为使用 2MB 大页面可以减少 TLB miss 以及页面访问失效的次数,从而减少空间开辟的时间花费。通过环境变量的设置可以利用 2MB 大页面开辟 MIC 卡上规模较大的数组或缓冲区空间:

export MIC_USE_2MB_BUFFERS=?

上述环境变量值"?"的取值为 interger B|K|G|T,表示基于指针的变量大小超过设定值时被放入 2MB 大页中可以提升 MIC 开辟空间的效率。但是,若把所有的基于指针的变量都放入大页中,则会一定程度地降低访存性能。因此,可根据处理矩阵的阶数以及设定的 Krylov 子空间的维数估算出数组开辟空间的大小。设定的阈值要满足小于待处理数组所占空间的大小,但又大于其他任何数组占用空间的大小。通过优化,可使空间开辟性能提高 60%以上。

(2)CPU 端与 MIC 端数据传输优化

CPU 与 MIC 之间通过 PCI-E 进行通信,PCI-E 的速度相对较慢,当数据传输次数较多且数据量较大时,其成为性能制约的瓶颈。本文主要利用 nocopy 技术有效地减少了 CPU 与 MIC 之间的通信次数。在有迭代的 MIC 应用中,充分利用重复利用的数据和空间,减少空间开辟及数据传输的花费。

将算法 3 中的 9 个"offload"语句依次编号为 Part1~Part9,表 1 列出了各部分内核计算必须要传入或传出 MIC 端的数组变量,而对于其他数组变量,采取 nocopy 技术,重复利用。

表 1 变量传输表

PART	1	2	3	4	6	8	9
IN	b	A,x		v_j			
OUT			z_0,v		z_{i+1}	h_i	v_m

(3)向量化优化

鉴于 MIC 卡支持 512 位宽的 Knights Corner 指令,即能同时处理 16 个单精度浮点数,向量化可大大提高计算速度,因此向量化对 MIC 的性能极为重要。

本文主要采用自动向量化的方式,通过增加编译选项"-vec-report"生成向量化报告。对于未向量化的代码,在调整代码结构或确定无数据依赖的情况下,通过添加"#pragma simd"及"#pragma ivdep"等编译指导语句的方式向量化。

(4)线程亲和性优化

线程根据亲和性设置的不同,被分配到不同的逻辑核心上,如果线程间有数据相关,当逻辑核心处于同一核心时,线程间可以利用同一物理核心上的 Cache,提高计算速度。本文将亲和性设置为 compact 模式,可使相邻线程尽量分配到同一物理核心,对于共享数据的利用,可以尽可能地利用 Cache,设置方式为:

kmp_set_defaults("KMP_AFFINITY=compact")

4 算例与测试

4.1 测试环境

本节测试均是在浪潮倚天 NF5580M3 小规模双路服务

器集群上完成的。集群共有 16 个计算节点，计算节点 CPU 使用 Intel(R) Xeon(R) E5-2650 v2，8 个核心，主频为 2.6GHz，内存大小为 128GB。每个节点配有两块 Intel Xeon Phi 7120P，每块卡有 61 个核心，主频为 1.25GHz，内存大小为 16GB。节点之间采用 FDR Infiniband 高速网络互联，带宽为 56Gb/s。集群软件环境搭建：操作系统为 Redhat 6.4；编译器为 Intel (R) 64，Version 14.0.2；MPI 库为 Intel(R) MPI Library for Linux* OS，Version 4.1；MIC 驱动版本为 MPSS：3.2.1-1，Flash Version：2.1.02.0390。

4.2 测试算例

高维偏微分方程广泛应用于分子动力学、量子力学、金融学等科研及工程应用领域。有限元、有限体积等常规的数值方法求解高维微分方程时通常会面临极大的挑战，这是因为当所描述的数学物理模型超过 3 维时，高维度问题域的离散将变得非常困难，而无网格方法可以规避离散高维问题域带来的困难，因此常常运用无网格方法代替传统的网格方法求解高维问题。

LI 和 CHEN 等[11]提出的“基于局部径向基函数求解高维偏微分方程”的方法不仅操作更加简便，同时生成的线性方程组的系数矩阵具有比其他全局无网格方法更少的条件数，线性方程组求解的收敛性更好。如文献[11]所述，若高维偏微分方程的问题域为 $\Omega=[0,1]^d \subset \mathbf{R}^d$，其中 d 为偏微分方程的维度，每个坐标轴上分 p 个部分，那么问题的自由度及系数矩阵的规模为 $N_f=p^d$，非零元素的个数为 $N_{non}=2d(p-2)^d+p^d$。假设偏微分方程维度为 8，每个坐标轴分成 10 份，那么线性方程组的规模为 $10^8 \times 10^8$，非零元素的个数为 368435456。利用基于 MIC 异构平台的 GMRES 并行算法对不同维度、不同规模的问题进行加速求解。本节测试矩阵均根据文献[11]生成，形函数以及相关参数的选择均按照文献中的方法选取。

4.3 测试结果分析

本小节的测试矩阵及矢量元素均为单精度浮点数，设定迭代残差的阈值为 $\varepsilon=10^{-7}$，子空间的维数为 $m=32$。表 2 列出了测试用例矩阵的特征，d 为求解微分方程的维度，p 为每个坐标上点的个数，N_f 为求解问题的自由度（矩阵规模），N_{non} 为稀疏矩阵非零元素的个数（矩阵稀疏程度）。

表 2　测试矩阵列表

序号	d	p	N_f	N_{non}
1#	7	10	10000000	39360128
2#	6	15	11390625	69312333
3#	6	18	34012224	235338816
4#	6	20	64000000	472146688
5#	8	10	100000000	368435456

采用大规模矩阵 4# 测试出 CPU 集群平台上 MPI 程序的扩展情况。为更直观地比较 MPI 程序的加速扩展情况，固定 GMRES 算法迭代运行 20 次。表 3 列出了当进程数由 1 扩展到 32（计算节点数由 1 扩展为 16）时的加速比和并行效率。

表 3 所列测试结果表明，当进程数小于 16 时，并行算法具有良好的 MPI 扩展性，并行效率接近 100%；当进程数大于 16 时，该并行算法的 MPI 并行效率有所下降，这是由于 GMRES 算法具有大量的集合通信，当进程数增加到一定程度时，MPI 集合通信相对计算部分的比例逐步增大，从而 MPI 的通信瓶颈效应凸显出来。总体来看，并不具有良好天然并行的 GMRES 并行算法在 MPI 进程数达到 32 时并行效率仍可达到 71.47%，效果良好。

表 3　GMRES 并行算法 MPI 扩展性测试结果

进程数	运行时间/s	加速比	MPI 并行效率/%
1	4391	1	100
2	2201	2	99.75
4	1149	3.82	95.50
8	592	7.43	92.75
16	310	14.16	91.25
32	192	22.87	71.47

采用不同规模、不同稀疏程度的大规模稀疏矩阵 1# ～ 5# 测试 MIC 程序的并行加速情况。为使测试更贴近实际应用情况，当迭代残差满足预先设定的阈值时，计算完成；为更为直观地表述 MIC 的加速效果，测试出 1 颗 CPU(8 核)的最佳性能与 MIC 加速性能进行比较。表 4 列出了不同规模矩阵的 MIC 加速测试结果，第 2 列为开启 1 个进程、1 个线程的运行时间，第 3 列为 1 颗 CPU(8 个核心)的运行时间，第 4—6 列分别为 1、2、4 块 MIC 卡的运行时间。测试每颗 CPU 性能时，开启 1 个进程，进程内开启 8 个 OpenMP 线程；测试 MIC 性能时，每个进程控制 1 块 MIC 卡，MIC 内部开启 240 个线程。由于单块 MIC 卡的内存为 16GB，在设定 Krylov 子空间维数为 32 时，单 MIC 卡或双 MIC 卡的显存不能满足矩阵 4# 及矩阵 5# 的计算要求。测试结果表明，GMRES MIC 并行算法比串行及 CPU 并行具有更好的收敛性，这是由于并行设计时采用的归约操作不仅可以提高计算性能，还可提高计算精度，从而加快算法的收敛。为更公平、更直观地比较 MIC 与 CPU 的加速效果，本文测试出了单次迭代的计算时间，并计算出了单次迭代（包括计算以及 MPI 通信）的加速比。测试结果表明，单块 MIC 卡相对于单颗 CPU 的加速比为 1.6～1.9，两块 MIC 卡相对于单颗 CPU 的加速比为 2.6～3.5，4 块 MIC 卡相对于单颗 CPU 的加速比为 3.95～7；矩阵规模越大，非零元素越多，加速效果越好。

表 4　不同规模矩阵的 MIC 加速效果

矩阵序号	1CPU core	1CPU (8cores)	1MIC	2MICs	4MICs
1#	迭代次数	7	5	3	3	3
	单次迭代时间/s	25.79	6.12	3.82	2.32	1.55
	迭代加速比	1	4.2	6.8	11.1	16.6
2#	迭代次数	18	6	6	4	4
	单次迭代时间/s	29.93	7.43	4.68	2.48	1.73
	迭代加速比	1	4	6.4	12	17.3
3#	迭代次数	>100	34	17	17	16
	单次迭代时间/s	91.58	26.71	14.10	7.38	4.77
	迭代加速比	1	3.4	6.5	12.4	19.2
4#	迭代次数	>100	>100	—	26	26
	单次迭代时间/s	179.76	49.24	—	13.88	8.51
	迭代加速比	1	3.7	—	13	21.1
5#	迭代次数	>100	11	—	—	3
	单次迭代时间/s	288.51	85.43	—	—	12.87
	迭代加速比	1	3.4	—	—	22.4

（下转第 240 页）

tions in EFSM testing[J]. IEEE Transactions on Software Engineering, 2004, 30(1): 29-42.

[12] DERDERIAN K, HIERONS R M, HARMAN M, et al. Estimating the feasibility of transition paths in extended finite state machines[J]. Automated Software Engineering, 2010, 17(1): 33-56.

[13] YANG R, CHEN Z, ZHANG Z, et al. EFSM-Based Test Case Generation: Sequence, Data, and Oracle[J]. International Journal of Software Engineering and Knowledge Engineering, 2015, 25(4): 633-667.

[14] WALKINSHAW N, TAYLOR R, DERRICK J. Inferring extended finite state machine models from software executions[J]. Empirical Software Engineering, 2016, 21(3): 811-853.

[15] HUANG C M, JANG M Y, LIN Y C. Executable EFSM-based data flow and control flow protocol test sequence generation using reachability analysis[J]. Journal of the Chinese Institute of Engineers, 1999, 22(5): 593-615.

[16] ZHANG J, YANG R, CHEN Z, et al. Automated EFSM-based test case generation with scatter search [C] // International Workshop on Automation of Software Test (AST). IEEE Press, 2012: 76-82.

[17] GAROFALAKIS M N, RASTOGI R, SHIM K. SPIRIT: Sequential pattern mining with regular expression constraints: VLDB, 1999[C] // Edinburgh. Scotland: Morgan Kaufmann Publishers, 1999: 7-10.

[18] LIU P, MIAO H. Theory of Test Modeling Based on Regular Expressions[M] // Structured Object-oriented Formal Lanuage and Method. 2013: 17-31.

[19] BELLI F, GROSSPIETSCH K E. Specification of fault-tolerant system issues by predicate/transition nets and regular expressions-approach and case study[J]. IEEE Transactions on Software Engineering, 1991, 17(6): 513-526.

(上接第 201 页)

结束语 本文基于 Intel Xeon Phi 协处理器集群平台实现了 GMRES 算法的移植工作。为克服加速设备内存不能满足计算要求的问题，本文首先对数据进行划分，采用分布式内存模型设计；为克服 GMRES 算法过多的集合通信带来的影响，对算法结构进行调整以便集合通信的隐藏。整个移植工作采用 MPI+OpenMP+offload 混合编程的模式：首先运用消息传递接口编程模型完成了数据任务的划分及并行设计，并利用计算与集合通信异步隐藏的方式提升并行算法的可扩展性，当进程数为 32 时，MPI 的并行效率达到 71.74%；接着运用 offload 编程模式，将计算任务卸载到 MIC 卡上，在 MIC 卡上通过开启 OpenMP 线程的方式实现并行；然后从 CPU 端与 MIC 端的数据传输、向量化及线程亲和性设计等方面对并行算法进行进一步优化；最后将并行算法应用到“局部径向基函数求解高维偏微分方程”的求解中。最终完成了对不同维度线性方程组的求解，测试结果表明，GMRES MIC 算法对于高维线性偏微分方程的求解具有收敛速度快、计算高效等特点。单块 MIC 卡的加速效果可达单颗 Xeon(R) E5-2650 v2 CPU 的 1.6 倍以上，4 块 MIC 卡的最高加速性能可达单颗 CPU 的 7 倍。因此，基于 MIC 异构集群平台的 GMRES 并行算法适用于超大规模线性方程组的求解，并有良好的并行加速效率和扩展性。

参 考 文 献

[1] SAAD Y, SCHULTZ M H. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems[J]. Siam Journal on Scientific & Statistical Computing, 1986, 7(3): 856-869.

[2] SAAD Y. Iterative methods for sparse linear systems(2nd ed)[M]. Philadelphia: SIAM, 2003: 151-126.

[3] NACHTIGAL N M, REICHEL L, TREFETHENK L NL N. A hybrid GMRES algorithm for nonsymmetric linear systems[J]. Siam Journal on Scientific & Statistical Computing, 1992, 13(3): 796-825.

[4] QUAN Z, XIANG S H. A GMRES based polynomal preconditioning algorithm[J]. Mathematical Numerical Sinica, 2006, 28(4): 365-376. (in Chinese).
全忠，向淑晃. 基于 GMRES 的多项式预处理广义极小残差法[J]. 计算数学，2006，28(4)：365-376.

[5] GHAEMIAN N, ABDOLLAHZADEH A, HEINEMANN Z. Accelerating the GMRES Iterative Linear Solver of an Oil Reservoir Simulator using the Multi-Processing Power of Compute Unified Device Architecture of Graphics Cards[C] // Proceedings of the 9th International Workshop on State of the Art in Scientific and Parallel Computing. Heidelberg: Springer, 2008: 156-159.

[6] WANG M L, KLIE H, PARASHAR M, et al. Solving Sparse Linear Systems on NVIDIA Tesla GPUs[C] // Computational Science-ICCS 2009 Lecture Notes in Computer Science. 2009, 5544: 864-873.

[7] LIU Y Q, YIN K X, WU E H. Fast GMRES-GPU Solver Large Scale Sparse Linear Systems[J]. Journal of Computer-Aided Design & Computer Graphics, 2011, 23(4): 553-560. (in Chinese)
柳有权，尹康学，吴恩华. 大规模稀疏线性方程组的 GMRES-GPU 快速求解算法[J]. 计算机辅助设计与图形学学报，2011，23(4)：553-560.

[8] GHYSELS P, ASHBY T J, MEERBERGEN K, et al. Hiding global communication latency in the GMRES algorithm on massively parallel computers[J]. Siam Journal on Scientific Computing, 2013, 35(1): 48-71.

[9] 王恩东，张清，等. MIC 高性能计算编程指南[M]. 北京：中国水利水电出版社，2012.

[10] JEFFERS J, REINDERS J. Intel Xeon Phi Coprocessor High Performance Programming[R]. morgan kaufmann. 2013.

[11] LI M, CHEN W, CHEN C S. The localized RBFs collocation methods for solving high dimensional PDEs[J]. Engineering Analysis with Boundary Elements, 2013, 37(10): 1300-1304.

[12] BELLALIJ M, REICHEL L, SADOK H. Some properties of range restricted GMRES methods[J]. Journal of Computational & Applied Mathematics, 2015, 290: 310-318.

[13] HE K, TAN X D, ZHAO H Y, et al. Parallel GMRES solver for fast analysis of large linear dynamic systems on GPU platforms [J]. Integration the VLSI Journal, 2016, 52(c): 10-22.