基于 Jacobi 算法并行求解大型稀疏矩阵的计算结构分析

1 大型稀疏矩阵问题简介

数学、物理、流体力学、工程技术和经济学等学科中的许多问题最终都归结为求解大型稀疏矩阵的线性代数方程组。解线性方程组主要有直接法和迭代法。对于大规模线性方程组的求解问题，特别是大规模稀疏线性方程组，迭代法是求解线性方程组的一种有效方法，它有存储空间小，程序简单等特点[1]。使用迭代法求解方程组充分利用了矩阵的稀疏性，从而节省大量计算存储空间，故其在求解大规模计算问题中发挥着重要的作用，成为求解大型稀疏代数方程组的实用方法。

2 大型稀疏矩阵的 Jacobi 迭代求解算法

近年来随着高性能计算机的发展，并行计算技术也越来越普及。并行计算的主要目的为：一是为了提供比传统计算机快的计算速度；二是解决传统计算机无法解决的问题。现今网络技术的飞速发展，PC 机性能快速提高，为并行计算打下了坚实的硬件基础。

Jacobi 迭代法是一种常用的解线性方程组的算法，在计算领域具有广泛的应用。但近年来随着计算技术的飞速发展，单线程串行程序已经无法满足 Jacobi 迭代算法对实际应用的需求，因为其需要处理的数据越来越复杂多样，规模也越来越大，因此提高 Jacobi 迭代算法的计算速度就成为研究 Jacobi 迭代算法的一个重要方向[2]。而并行技术是加快计算速度的重要技术。因此，将 Jacobi 同并行计算结合起来，实现 Jacobi 迭代的并行化。自身的并行特性，应用一种最基本的并行计算方法——MPI 来实现 Jacobi 迭代，加速其收敛速度，为算法的实际应用奠定良好的基础，从而进一步加深对并行计算的认识，对于今后类似实际问题的解决具有重要的意义[3]。

3 Jacobi 迭代算法的算术密度分析

Jacobi 迭代算法的算术密度求解，当完成一个 n*n 的矩阵的时候，需要循环 n*n 次，进行 2n 次赋值操作，n 次减法操作、除法操作，进行了 n*n 次的比较操作、乘法操作、加法操作，所需要的进行的计算次数为（$2n+3n^2$），所需要的内存带宽为（$4n+3n^2$）算数密度为（$2n+3n^2$）/（$4n+3n^2$）。

考察单个 CPU 核心的计算时间和并行的计算时间，相当于 n=1 的情况下，以此作为基准考察并行的结果。随着线程数的增加，计算时间呈下降趋势，当进程数达到 16 时，计算时间最少。然而，进程数增多，数据分配时间却变化甚微。由于机器的资源的限制，没有

尝试更多的线程数。

4 Jacobi 迭代算法现有实现的概述

当阶数较大、系数阵为稀疏阵时，可以采用迭代法求解线性方程组。用迭代法求解线性方程组的优点是方法简单，便于编制计算机程序，但必须选取合适的迭代格式及初始向量，以使迭代过程尽快地收敛。Jacobi 迭代法是其中一种迭代格式，考虑计算机内存和运算这两方面，Jacobi 迭代法可充分利用稀疏矩阵中大量零元素的特点。

Jacobi 迭代的原理是求解 n 阶线性方程组 $Ax=b$，假设系数矩阵 A 的主对角线元素 $a_{ii}\neq 0$，且每行严格对角占优，即：

$$|a_{ii}|>\sum_{j\neq i}^{n}|a_{ii}|\ (i=1,2,...,n)$$

将原方程组的每一个方程 $a_{i1}x_1+a_{i2}x_2+\cdots+a_{in}x_n=b_i$ 改写为未知向量 x 的分量的形式：

$$x_i=(b_i-\sum_{j\neq i}^{n}a_{ij}x_j)/a_{ii}\quad(1\le i\le n)$$

然后使用第 $k-1$ 步所计算的变量 $x_i^{(k-1)}$ 来计算第 k 步的 $x_i^{(k)}$ 的值：

$$x_i^{(k)}=(b_i-\sum_{j\neq i}^{n}a_{ij}x_j^{(k-1)})/a_{ii}\quad(1\le i,\ k\le n)$$

这里 $x_i^{(k)}$ 为第 k 次迭代得到的近似解向量 $x^{(k)}=(x_1^{(k)},x_2^{(k)},\cdots,x_n^{(k)})^T$ 的第 i 个分量。取适当初始解向量 $x^{(0)}$ 代入上述迭代格式中，可得到 $x^{(1)}$，再由 $x^{(1)}$ 得到 $x^{(2)}$，依次迭代下去得到近似解向量序列 $\{x^{(k)}\}$。若原方程组的系数矩阵按行严格对角占优，则 $\{x^{(k)}\}$ 收敛于原方程组的解[4]。

整个算术步骤如下所示：

$$\begin{cases}a_{11}x_1+\cdots+a_{1n}x_n=b_1\\ \vdots\\ a_{n1}x_1+\cdots+a_{nn}x_n=b_n\end{cases}\qquad(1)$$

有（1）可得各 X 的值为

$$\begin{cases} x_1 = \frac{-1}{a_{11}}(a_{12}x_2 + \cdots + a_{1n}x_n - b_1) \\ x_2 = \frac{-1}{a_{22}}(a_{21}x_1 + a_{23}x_3 + \cdots + a_{1n}x_n - b_2) \\ \vdots \\ x_n = \frac{-1}{a_{\mathrm{nn}}}(a_{n1}x_1 + \cdots + a_{nn-1}x_{n-1} - b_n) \end{cases} \quad (2)$$

所以

$$\begin{cases} x_{1^{(k+1)}} = \frac{-1}{a_{11}}\left(a_{12}x_{2^{(k)}} + \cdots + a_{1n}x_{n^{(k)}} - b_1\right) \\ x_{2^{(k+1)}} = \frac{-1}{a_{22}}\left(a_{21}x_{1^{(k)}} + a_{23}x_{3^{(k)}} + \cdots + a_{1n}x_{n^{(k)}} - b_2\right) \\ \vdots \\ x_{n^{(k+1)}} = \frac{-1}{a_{\mathrm{nn}}}\left(a_{n1}x_{1^{(k)}} + \cdots + a_{nn-1}x_{n-1^{(k)}} - b_n\right) \end{cases} \quad (3)$$

最后可得

$$x_{i^{(k+1)}} = \frac{-1}{a_{\mathrm{ii}}}\left(\sum_{j=1}^{i-1} a_{\mathrm{ij}}x_{j^{(k)}} + \sum_{j=i+1}^{n} a_{\mathrm{ij}}x_{j^{(k)}} - b_i\right) \quad (4)$$

Jacobi 迭代是一种比较常见的迭代方法。Jacobi 迭代的局部性很好，传统的串行方法效率不高。

5 Jacobi 迭代算法理想的计算结构

5.1 MPI 并行方法

并行计算就是研究如何把一个需要非常巨大计算能力才能解决的问题分成许多小的部分，然后将这些小部分分给许多计算机进行并行处理，最后将这些计算结果综合起来得到最终的结果。用一句话说，就是为了加快运算速度和解决大主存容量的求解问题的多计算机/多处理机的并行编程技术[5]。MPI(Message Passing Interface)是消息传递函数库的标准规范，其是一种消息传递编程模型，也是现今最流行的并行计算编程方法。

5.2 Jacobi 迭代并行算法性能

采用 MPI 编程技术实现雅可比迭代法解线性方程组。若处理器个数为 p，则对矩阵 A 按行划分。设矩阵被划分为 p 块，每块含有连续的 m 行向量，这里 $m=\lceil n/p \rceil$，编号为 i 的处理器含有 A 的第 im 至第 $(i+1)m-1$ 行数据，同时向量 b 中下标为 im 至 $(i+1)m-1$ 的元素也被分配至编号为 i 的处理器($i=0,1,\cdots,p-1$)，初始解向量 x 被广播给所有处理器。

迭代计算中，各处理器并行计算解向量 x 的各分量值，编号为 i 的处理器计算分量 x_{im} 至 $x_{(i+1)m-1}$ 的新值[6]。并求其分量中前后两次值的最大差 localmax，然后通过归约操作的求最大值运算求得整个 n 维解向量中前后两次值的最大差 max 并广播给所有处理器。最后通

过扩展收集操作将各处理器中的解向量按处理器编号连接起来并广播给所有处理器，以进行下一次迭代计算，直至收敛。

5.2.1 矩阵数据的分割

对高阶矩阵的运算处理采用分块方法，将矩阵分割，分配给多个进程并行执行，从而提高运算速度。为减少进程之间的通信开销，应选择合理的数据块划分方式。因为本文采用 C 语言实现程序设计且 C 语言采用按行优先存储。为了实现并行计算，首先将参加迭代的数据按行分割。假设 P 个进程并行执行，需要迭代的数据是 M * M 的二维数组 A(M , M)，令 M = p * N，将数组 A(M , M)按行划分为 p 个数据块，每个数据块的大小为 N * M[7]。为了编程简便和形式上的一致，因此进程与数据块的对应关系：进程 0，A（0：N-1，M）；进程 1，A（N：2N-1，M）；同理，进程 P - 1，A（（p - 1）* N：p * N – 1，M）。

5.2.2 数据通信

首先对数组赋初值。注意对不同的进程，赋值是不同的。然后便开始进行 Jacobi 迭代，在迭代之前，每个进程都需要从相邻的进程得到数据块，同时每一个进程也都需要向相邻的进程提供数据块[8]。由于每一个新迭代点的值是根据上一次迭代的值的出来的，所以这里引入一个中间数组，用来记录临时得到的新值，一次迭代完成后，再统一进行更新操作。

Jacobi 迭代过程中需要数据区块之间的通信[9]。对于最上方的数据块，只需向下方数据块发送数据，从下方数据块接收数据；对于中间的数据块，需要向上、下相邻数据块发送数据，同时从上、下相邻数据块接收数据；对于最下方的数据块，只需向上方数据块发送数据，从上方数据块接收数据。

将 Jacobi 同并行计算结合起来，实现 Jacobi 迭代的并行化。自身的并行特性，应用一种最基本的并行计算方法——MPI 来实现 Jacobi 迭代，加速其收敛速度，为算法的实际应用奠定良好的基础，从而进一步加深对并行计算的认识，对于类似实际问题的解决具有重要的意义[10]。

附录

通过基于消息传递的并行编程模型 MPI，设计和实现了一种基于 MPI 的并行 Jacobi 算法求解大型稀疏矩阵问题，其结果如下表所示。

表 1 基于 MPI 的 Jacobi 求解线性方程组实验结果

线程数	1	4	8	16
运行时间	49.127553 s	47.115856 s	46.267590 s	46.386513 s
数据分配	45.251523 s	46.135080 s	45.751325 s	46.011767 s
并行时间	3.876030 s	0.980776 s	0.516265 s	0.374746 s

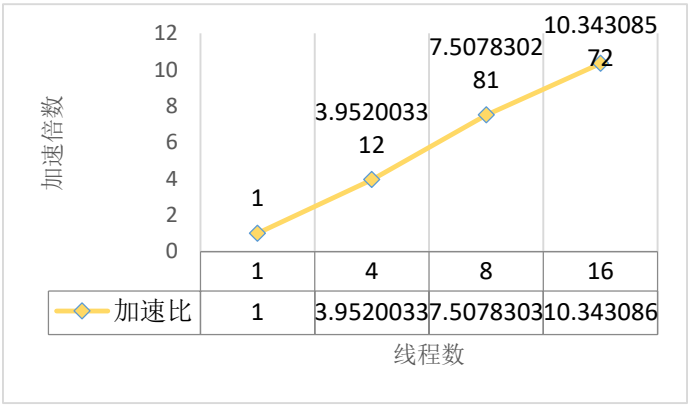

图 1 不同线程数的加速比

可以看出，该并行策略可以获得较好的解，随着线程数的增加，运行时间不断减小，加速比呈线性不断增大，4 线程的加速倍数近似 4 倍，8 线程的加速倍数约为 7.5 倍，16 线程数的加速倍数约为 10.3 倍，跟理想的 8 倍、16 倍还是有差距的，这主要因为进程数越多，则占用的内存越多，所耗费的内存分配时间、内存回收时间、寻址时间等都会大大增加，进而加速倍数达不到理想值。同时，在实验中也可以通过做多次实验求平均数，这样可以有效的缩小误差。

参考文献

[1]赵博颖,肖鹏,张力.基于 Docker 的 MPI 和 OpenMP 混合编程[J].计算机与现代化,2018(05):60-64.

[2]向宇,刘芳,万传棕,彭露,奉丽薇,王冰清.一种基于Linux集群和MPI编程环境的并行计算方法[J].甘肃科技,2019,35(17):26-28.

[3]闵瑞高.基于分布式并行计算的高性能演化算法研究[D].江南大学,2019.

[4]卢可佩,祝永志.基于 MPI 的 Jacobi 迭代算法的并行化[J].电脑知识与技术, 2014, 10(31): 7485-7487.

[5]刘广西,张衡.一种改进的变预处理 SOR-BICR 算法[J].福建师大福清分校学报, 2018 (02):1-6.

[6]张海龙,张萌,王杰,冶鑫晨,王万琼,朱艳.基于 MPI 和 Taurus 高性能计算系统的 Jacobi 并行迭代算法[J].吉林大学学报(工学版),2019,49(02):606-613.

[7]张维儒,潘无名.基于MPI的并行计算实现Jacobi迭代[J].软件导刊,2008(09):16-17.

[8]陈国良.并行算法的设计与分析[M].北京:高等教育出版社,2002:364-367.

[9]Alberto Arenas,Óscar Ciaurri,Edgar Labarga. A weighted transplantation theorem for Jacobi coefficients[J]. Journal of Approximation Theory,2019,248.

[10]袁云,王传美,童恒庆.基于混合编程模式的 Jacobi 迭代并行算法[J].武汉理工大学学报(信息与管理工程版),2014,36(01):18-20.