南開大學
Nankai University

计算机学院

并行程序设计期末开题报告

高斯消去的并行化

姓名：高祎珂

学号：2011743

专业：计算机科学与技术

2022 年 4 月 2 日

目录

1 问题描述

1.1 数学阐述

数学上，高斯消元法 (Gaussian elimination) 是求解线性方阵组的一种算法，它也可用来求矩阵的秩，以及求可逆方阵的逆矩阵。它通过逐步消除未知数来将原始线性系统转化为另一个更简单的等价的系统。它的实质是通过初等行变化 (Elementary row operations)，将线性方程组的增广矩阵转化为行阶梯矩阵 (row echelon form)。

1.2 原理分析

对于高斯消去法而言，n 个方程 n 个未知数的消去计算，可以在 $\frac{2}{3}n^3 + \frac{1}{2}n^2 - \frac{7}{6}n$ 次操作后完成，在高斯消去完成之后，矩阵就会变成上三角形式：

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ 0 & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} & bn \end{array}\right]$$

图 1.1: Matrix

这样，n 个方程 n 个未知数的回代过程可以使用 n^2 次操作完成。将线性方程组表示为矩阵方程 Ax=b ，将系数矩阵表示为 A ，未知数为 x ，等号右边为 b，LU 分解即将系数矩阵 A 分解为上三角矩阵和下三角矩阵的乘积。那么就有：

$m \times n$ 矩阵 L 是下三角矩阵，如果其对应元素满足 $l_{ij} = 0, i < j$,$m \times n$ 矩阵 U 是下三角矩阵，如果其对应元素满足 $u_{ij} = 0, i > j$ 。那么一旦完成 LU 分解，Ax=b 就转化为 LUx=b ，定义向量 c=Ux ，那么求解方程组的回代过程就为：对于方程 Lc=b ，求解 c；对于方程 Ux=c ，求解 x 。因为 L 和 U 是三角阵，因此运算变得较为简单直接。

2 前人研究

2.1 全主元高斯消去

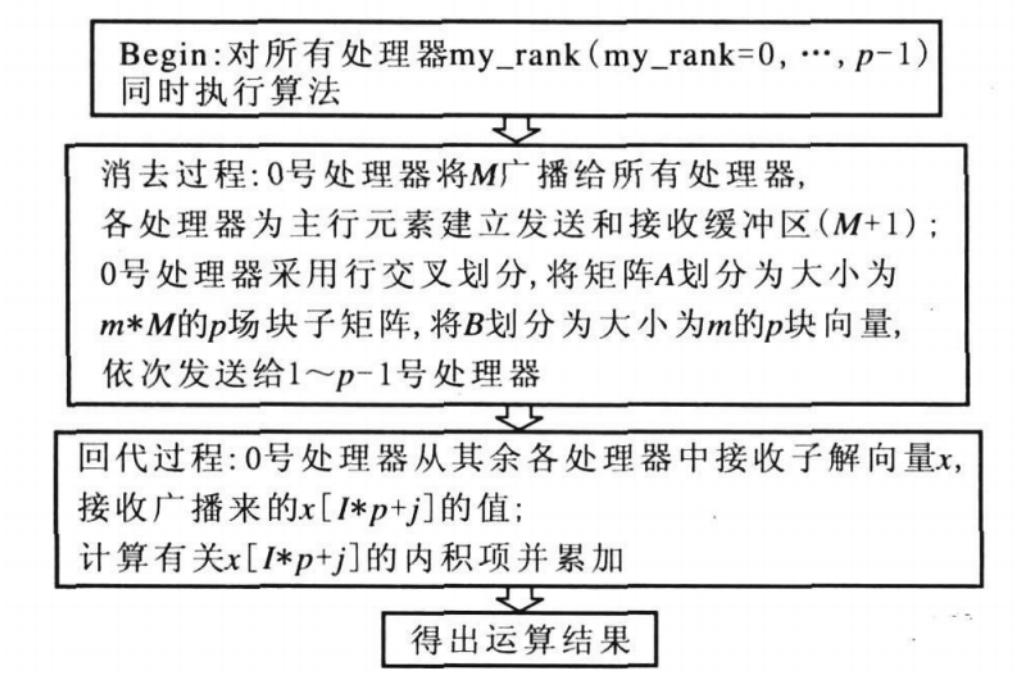
高斯消去法是利用主行 i 对其余各行 j(j >i), 作初等行变换, 各行计算之间没有数据相关关系, 因此可以对矩阵 A 按行划分. 考虑到在计算过程中处理器之间的负载均衡, 对 A 采用行交叉划分. 设处理器个数为 p , 矩阵 A 的阶数为 n , m =[n/ p] , 对 A 行交叉划分后, 编号为 i(i =0 , 1 , ···, p -1) 的处理器含有 A 的第 i , i +p , ···, i +(m -1)p 行和 B 的第 i , i +p , ···, i +(m -1)p 一共 m 个元素. 每个线程控制一个处理器进行子线程的运算, 最后把运算结果发送回主机进行处理.

消去过程的并行是依次以第 0 , 1 , ···, n -1 行作为主行进行消去计算, 由于对行的交叉划分与分布, 这实际上是由各处理器轮流选出主行. 在每次消去计算前, 各处理器并行求其局部存储器中右下角子阵的最大元. 若以编号为 my_rank 的处理器的第 i 行作为主行, 则编号在 my_rank 后面的处理器 (包括 my_rank 本身) 求其局部存储器中第 i 行至第 m -1 行元素的最大元, 并记录其行号、列号及所

在处理器编号; 编号在 my_rank 前面的处理器求其局部存储器中第 i +1 行至第 m -1 行元素的最大元, 并记录其行号、列号及所在处理器编号.

然后通过扩展收集操作, 将局部存储器中的最大元按处理器编号连接起来, 并广播给所有处理器, 各处理器以此求得整个矩阵右下角阶子阵的最大元 (maxvalue) 及其所在行号、列号和处理器编号. 若 maxvalue 的列号不是原主元素 akk 的列号, 则交换第 k 列与 maxvalue 所在列的两列数据; 若 maxvalue 的处理器编号不是原主元素 a_{kk} 的处理器编号, 则在处理器间的进行行交换; 若 maxvalue 的处理器编号是原主元素 a_{kk} 的处理器编号, 但行号不是原主元素 a_{kk} 的行号, 则在处理器内部进行行交换.

在消去计算中, 首先对主行元素作归一化操作 $a_{kj} = a_{kj}/a_{kk}, b_k = b_k/a_{kk}$, 然后将主行广播给所有处理器, 各处理器利用接收到的主行元素对其部分行向量做行变换. 若以编号为 my_rank 的处理器的第 i 行作为主行, 并将它播送给所有的处理器. 则编号在 my_rank 前面的处理器 (包括 my_rank 本身) 利用主行对其第 i +1 , ···, m -1 行数据和子向量 B 做行变换. 编号在 my_rank 后面的处理器利用主行对其第 i , ···, m -1 行数据和子向量 B 做行变换, 回代过程的并行是按 $X_n, X_{n-1}, ..., X_1$ 的顺序由各处理器依次计算 X($i \times p$ +my_rank), 一旦 X($i \times p$ +my_rank), 被计算出来就立即广播给所有处理器, 用于与对应项相乘并做求和



2.2 部分选主元的高斯消去

前述的消去过程中，未知量是按其出现于方程组中的自然顺序消去的，所以又叫顺序消去法。实际上已经发现顺序消去法有很大的缺点。设用作除数的 a_{kk} 为主元素，首先，消元过程中可能出现 a_{kk} 为零的情况，此时消元过程亦无法进行下去；其次如果主元素 a_{kk} 很小，由于舍入误差和有效位数消失等因素，其本身常常有较大的相对误差，用其作除数，会导致其它元素数量级的严重增长和舍入误差的扩散，使得所求的解误差过大，以致失真。

高斯消去法的一个简单变形———部分选主元高斯消去法，可以产生可靠的结果。在部分选主元的高斯消去法的第 i 步，我们在第 i 行到第 n-1 行中寻找第 i 列元素的绝对值最大的行并将这一行与第 i 行交换（变成主元）。在部分主元消去法中，未知数仍然是顺序地消去的，只是选各方程中要消去的那个未知数的系数的绝对值最大的作为主元素，然后用顺序消去法的公式求解。例：用部分选主元

高斯消去法求解方程

$$\begin{cases} 2x_1 - x_2 + 3x_3 = 1 \\ 4x_1 + 2x_2 + 5x_3 = 4 \\ x_1 + 2x_2 = 7 \end{cases}$$

由于解方程组取决于它的系数，因此可用这些系数（包括右端项）所构成的“增广矩阵”作为方程组的一种简化形式。对这种增广矩阵施行消元手续：

$$\begin{pmatrix} 2 & -1 & 3 & 1 \\ 4^* & 2 & 5 & 4 \\ 1 & 2 & 0 & 7 \end{pmatrix}$$

第一步将 4 选为主元素，并把主元素所在的行定为主元行，然后将主元行换到第一行得到

$$\begin{pmatrix} 4 & 2 & 5 & 4 \\ 2 & -1 & 3 & 1 \\ 1 & 2 & 0 & 7 \end{pmatrix} \xrightarrow{\text{第一步消元}} \begin{pmatrix} 1 & 0.5 & 1.25 & 1 \\ 0 & -2^* & 0.5 & -1 \\ 0 & 1.5 & -1.25 & 6 \end{pmatrix} \xrightarrow{\text{第二步消元}} \begin{pmatrix} 1 & 0.5 & 1.25 & 1 \\ 0 & 1 & -0.25 & 0.5 \\ 0 & 0 & -0.875 & 5.25 \end{pmatrix} \xrightarrow{\text{第三步消元}} \begin{pmatrix} 1 & 0.5 & 1.25 & 1 \\ 0 & 1 & -0.25 & 0.5 \\ 0 & 0 & 1 & -6 \end{pmatrix}$$

消元过程的结果归结到下列三角形方程组：

$$\begin{cases} x_1 + 0.5x_2 + 1.25x_3 = 1 \\ x_2 - 0.25x_3 = 0.5 \\ x_3 = -6 \end{cases} \quad \text{回代，得} \quad \begin{cases} x_1 = 9 \\ x_2 = -1 \\ x_{-3} = -6 \end{cases}$$

相应的并行算法需要根据穿行的算法进行修改优化

2.3 高斯并行算法基于云平台的研究

研究者程序中使用的并行线程系统为 Linux 下的 pthread。thread_id 为线程的 id，thread_count 为总的线程数。算法的第 2 到 4 行为部分选主元（partial pivoting），仅由 0 号线程执行。7 到 11 行在矩阵 A 上进行高斯消元（Gaussian Elimina- tion），由所有线程执行。14 到 17 行为处理向量 B，仅由 0 号线程执行。

并行算法伪代码

Algorithm1: Gaussian – elimination – parallel

1: for $k = 1, \cdots, n-1$

2: if (thread_id = = 0)

3: find $i_k \geqslant k$ such that $|a_{i_k k}| = \max_{k \leqslant i \leqslant n} |a_{ik}|$

4: if $i_k > k$, interchange $a_{kj} \leftrightarrow a_{i_k j}$ for $j = k, \cdots, n$

5: barrier_wait();

6:

7: for $i = k+1, \cdots, n$

8: if((i % thread_count) = = thread_id)

9: $a_{ik} \leftarrow - a_{ik}/a_{kk}$

10: for $j = k+1, \cdots, n$

11: $a_{ij} \leftarrow a_{ij} + a_{ik}a_{kj}$

12: barrier_wait();

13:

14: if (thread_id = = 0)

15: if $i_k > k$, interchange $b_k \leftrightarrow b_{i_k}$

16: for $i = k+1, \cdots, n$

17: $b_i \leftarrow b_i + a_{ik}b_k$

整个算法的计算量主要集中在 7 到 11 行的高斯消元，因此使用多线程，其余部分使用单线程

研究者对这个并行多线程的高斯消元程序进行了编译、运行和验证。做系统优化和 CPU 优化后可以看到运行时间有了明显的提升。

3 研究计划

3.1 研究目标

总体研究将分四个阶段进行，分别进行 SIMD 技术实验，多线程技术实验，和 MPI，以及最终的综合性优化。

3.2 个人安排

SIMD	两周4.10	Pthread	两周4.24	MPI	两周5.8	GPU	两周5.22	整理优化

在 SIMD 实验中，初步实现高斯消去的并行化算法，在 ARM 平台和 X86 平台上测试代码，测试全主元与部分主元算法的优劣。

在多线程实验中使用合适的数据划分方式，使用多个线程同时进行运算，找出最匹配的数据划分方式，考虑和 SIMD 的配合优化，在 MPI 实验中不同平台进行测试，独立进程对高斯消去并行化的影响。之后进行综合性优化没考虑之前做的小实验，主公特殊的高斯消去，利用倒排索引方式进行存储优化等等，查阅资料找到更好的优化方式进行实验。

利用性能剖析工具对每个实验进行性能测试，记录数据，并利用相关数据进行实验改进，撰写实验报告。

参考文献

[1] 陈国良. 并行算法实践 [M] . 北京: 高等教育出版社, 2004.

[2] 潘晓辉. 并行高斯消去法在云计算平台上的研究 [J]. 计算机技术与发展,2014,24(05):125-128+133.

[3] 田希山. 用部分选主元的高斯消去法并行求解线性方程组 [J]. 电脑知识与技术,2011,7(16):3960-3963+3966.

[4] [1] 石虎, 熊健民, 宋庭新. 全主元高斯消去法在有限元并行计算中的应用 [C]