《并行计算与分布式计算》

课程设计

题　　目　全主元 Gauss 消元法的并行算法

指导教师　许晓阳

学　　院　计算机科学与技术学院

专业班级　数据科学与大数据技术 1901 班

姓　　名　李鸿杰

学　　号　19408080117

日　　期　2022 年 7 月 15 日

目　录

全主元 Gauss 消元法并行算法的 MPI 并行实现

1、课程设计任务

①了解 Linux 操作系统、并行平台及常用远程登陆软件，掌握并行计算的基本原理，了解通信模式。

②掌握 MPI 编程、并行性能的评价指标，矩阵相乘、Gauss 消元数值算法的并行实现。

③掌握分布式计算的理论基础、核心思想，能独立开展一些分布式算法设计、分析与应用方面的工作。

2、课程设计目的

①通过课程设计使学生熟练掌握并行计算及分布式计算技术方面的基本概念

②熟悉并行计算机系统及其结构模型，了解如何进行并行计算的性能评测

③熟练掌握并行算法设计的基础知识

④了解现有的并行计算模型以及并行算法的设计方法、设计技术和设计过程

⑤熟练掌握 MPI 并行编程方法

3、理论算法

3.1 高斯（Gauss）消去法

①高斯消去法的基本思想

先逐级消去变量，将原方程变换为同解的上三角方程组，此过程称为消元过程。然后按方程组的相反顺序求解上三角方程组便得到原方程组的解，此过程称为回代过程。

②高斯消去法算法流程

消元过程(第k次消元)：

$$\begin{cases} a_{ij}^{(k)} = a_{ij}^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} a_{kj}^{(k-1)} = a_{ij}^{(k-1)} - l_{ik} a_{kj}^{(k-1)}, \\ i = k+1, k+2, ..., n; j = k+1, k+2, ..., n \\ \\ b_i^{(k)} = b_i^{(k-1)} - l_{ik} b_k^{(k-1)}, i = k+1, k+2, ..., n \\ \\ l_{ik} \triangleq \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}, i = k+1, k+2, ..., n \end{cases}$$

回代过程：

$$\begin{cases} x_n = \dfrac{b_n^{(n-1)}}{a_{nn}^{(n-1)}}, \\ x_k = \dfrac{b_k^{(k-1)} - \sum\limits_{j=k+1}^{n} a_{kj}^{(k-1)} x_j}{a_{kk}^{(k-1)}}, k = n-1, n-2, ..., 2, 1 \end{cases}$$

图 1 高斯消去法流程

3.2 高斯列主元消去法

①高斯列主元消去法基本思想

在进行第 k 步消元时，从第 k 列的 akk 及其以下的各元素中选取绝对值最大的元素，然后通过行变换将它交换到主元素 akk 的位置上，再进行消元。

②高斯列主元消去法算法流程

相比经典的高斯消去法，在消元过程前额外多了**按列选主元步骤**和**交换行步骤**，其余流程是一致的。

按列选主元步骤的算法描述： 选取第k列的主元，即寻找它所在的行q，有：

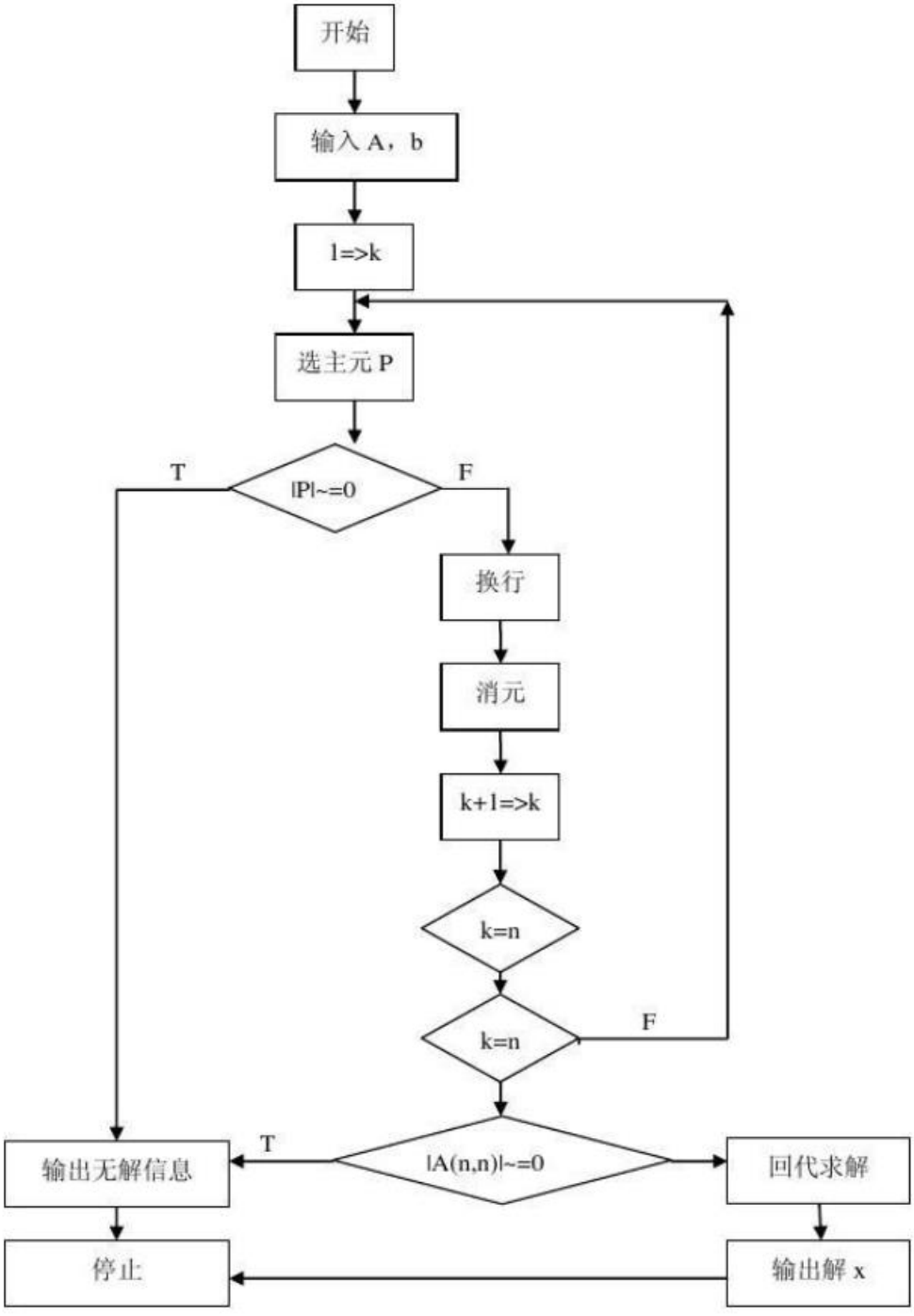
$$q = \arg\max_{i \in [k,n)} |a_{ik}|$$

交换行步骤的算法描述： 找到主元所在行q后，分别交换A和b的第k行和第q行元素，即

$$\begin{cases} \mathrm{swap}(a_{kj}, a_{qj}), j = k, k+1, ..., n \\ \mathrm{swap}(b_k, b_q) \end{cases}$$

图 2 高斯列主元消去法流程

③高斯列主元消去法程序流程图



图表 3 高斯列主元消去法程序流程图

4、代码实现

4.1 MPI 环境搭建

本次 MPI 环境基于 VS2015+MPI。

①下载并配置 VS

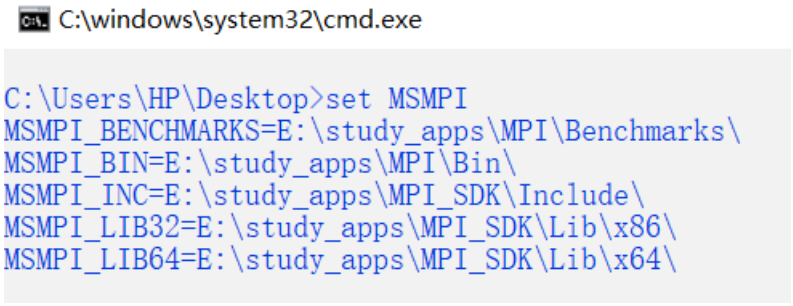


图 4MPI 安装路径

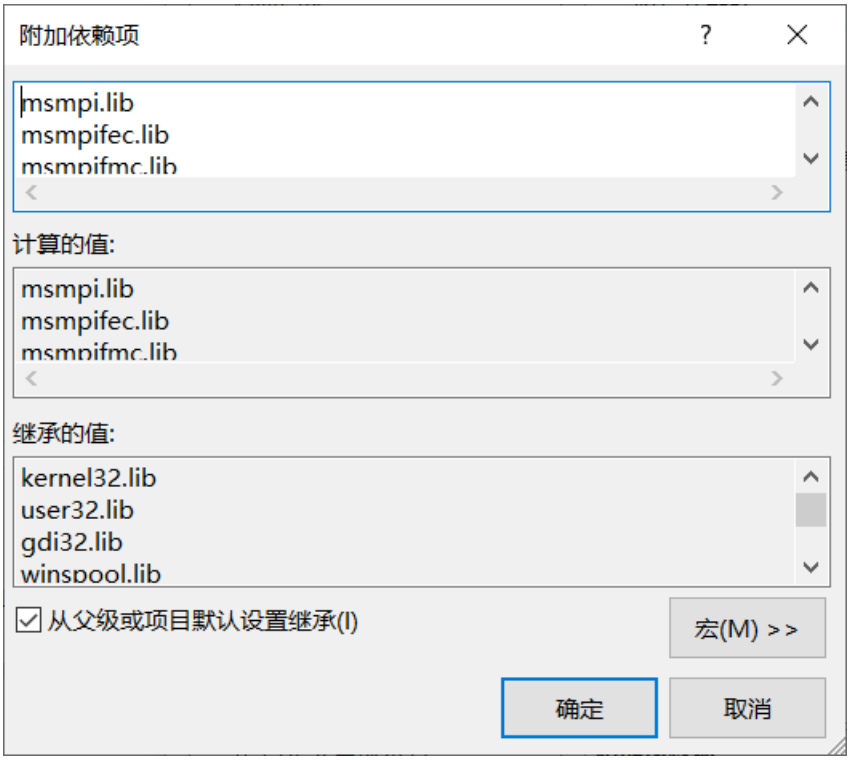


图 5 配置 MPI

②MPI 环境测试

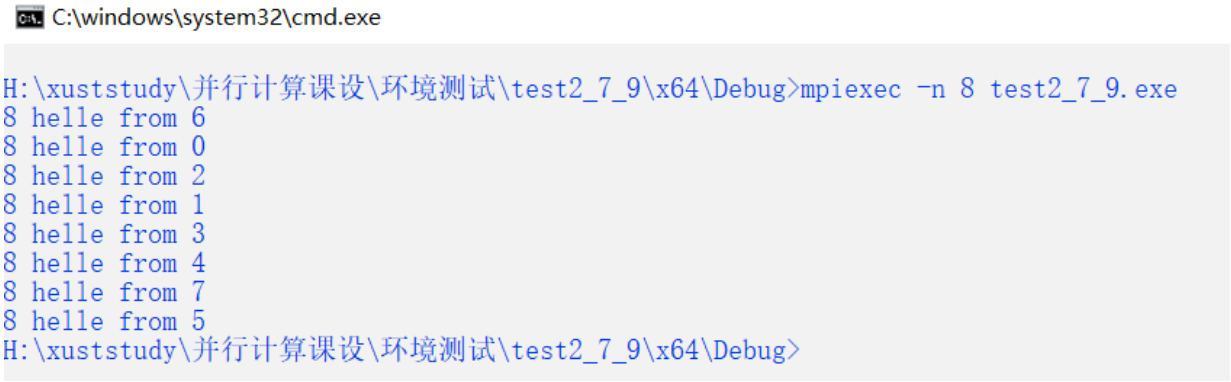


图 6MPI 环境测试结果图

4.2 高斯主元消去法代码实现

4.2.1 代码组成结构

①设置运行进程数

②根据用户输入随机生成线性方程组

③rank0 组内广播数据

④高斯列主元消去法选取主元消去

⑤高斯列主元消去法回代

⑥程序计时部分

⑦计算加速比与效率

4.2.2 完整代码

```
#include "stdio.h"
#include "stdlib.h"
#include "mpi.h"
#include "math.h"
#include <time.h>
#include <stdlib.h>
#include <iostream>
#include <time.h>
using namespace std;
#define a(x,y) a[x*M+y]
#define b(x) b[x]
#define A(x,y) A[x*M+y]
#define B(x) B[x]
```

```
#define floatsize sizeof(float)
#define intsize sizeof(int)
int M;
int N=0;
int m;
float *A;
float *B;
double starttime;
double time1;
double time2;
int my_rank;
int p;
int l;
MPI_Status status;
void Environment_Finalize(float *a, float *b, float *x, float *f)
{
    free(a);
    free(b);
    free(x);
    free(f);
}
int main(int argc, char **argv)
{
    clock_t start,end;
    int i, j, t, k, my_rank, group_size;
    int i1, i2;
    int v, w;
    float temp;
    int tem;
    float *sum;
    float *f;
    float lmax;
    float *a;
    float *b;
    float *x;
    int *shift;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &group_size); //组进程大小
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);   //组进程编号
    p = group_size;//p即为核心数
```

```
if (my_rank == 0)
{
    cout << "-------李鸿杰-19408080117-高斯列主元消去法-------"<<endl;
    cout << "未知数X的个数" << endl;
    cin >> M;
    N = M + 1;
    srand((unsigned)time(NULL));
    A = (float *)malloc(floatsize*M*M);
    B = (float *)malloc(floatsize*M);
    for (i = 0; i < M; i++)
    {
        for (j = 0; j < M; j++)
        {
            A(i, j) = rand();
        }
        B(i) = rand();
    }
}
start = clock();
starttime = MPI_Wtime();
MPI_Bcast(&M, 1, MPI_INT, 0, MPI_COMM_WORLD); /* 0号处理机将M广播给所有处理机 */
m = M / p;
if (M%p != 0) m++;
f = (float*)malloc(sizeof(float)*(M + 1)); /* 各处理机为主行元素建立发送和接收缓冲区(M+1) */
a = (float*)malloc(sizeof(float)*m*M); /* 分配至各处理机的子矩阵大小为m*M */
b = (float*)malloc(sizeof(float)*m); /* 分配至各处理机的子向量大小为m */
sum = (float*)malloc(sizeof(float)*m);
x = (float*)malloc(sizeof(float)*M);
shift = (int*)malloc(sizeof(int)*M);
for (i = 0; i<M; i++)
    shift[i] = i;                    //进行行列变换选取主元时的标记量
                                     /*
                                     0号处理机采用行交叉划分将矩阵A划分为大小为m*M的p块子矩
阵,将B划分为大小
                                     为m的p块子向量，依次发送给1至p-1号处理机
                                     */
if (my_rank == 0)
{
    for (i = 0; i<m; i++)
        for (j = 0; j<M; j++)
            a(i, j) = A(i*p, j);
```

```
    for (i = 0; i<m; i++)
        b(i) = B(i*p);
}
if (my_rank == 0)
{
    for (i = 0; i<M; i++)
        if ((i%p) != 0)
        {
            i1 = i%p;
            i2 = i / p + 1;
            MPI_Send(&A(i, 0), M, MPI_FLOAT, i1, i2, MPI_COMM_WORLD);
            MPI_Send(&B(i), 1, MPI_FLOAT, i1, i2, MPI_COMM_WORLD);
        }
} /* my_rank==0 */
else /* my_rank !=0 */
{
    for (i = 0; i<m; i++)
    {
        MPI_Recv(&a(i, 0), M, MPI_FLOAT, 0, i + 1, MPI_COMM_WORLD, &status);
        MPI_Recv(&b(i), 1, MPI_FLOAT, 0, i + 1, MPI_COMM_WORLD, &status);
    }
}
time1 = MPI_Wtime(); /* 开始计时 */
for (i = 0; i<m; i++) /* 消去 */
    for (j = 0; j<p; j++)
    {
        if (my_rank == j) /* j号处理机负责广播主行元素 */
        {
            v = i*p + j; /* 主元素在原系数矩阵A中的行号和列号为v */
            lmax = a(i, v);
            l = v;
            for (k = v + 1; k<M; k++) /* 在同行的元素中找最大元，并确定最大元所在的列l */
                if (fabs(a(i, k))>lmax)
                {
                    lmax = a(i, k);
                    l = k;
                }
            if (l != v) /* 列交换 */
            {
                for (t = 0; t<m; t++)
                {
```

```
                temp = a(t, v);
                a(t, v) = a(t, l);
                a(t, l) = temp;
            }
            tem = shift[v];
            shift[v] = shift[l];
            shift[l] = tem;
        }
        for (k = v + 1; k<M; k++) /* 归一化 */
            a(i, k) = a(i, k) / a(i, v);
        b(i) = b(i) / a(i, v);
        a(i, v) = 1;
        for (k = v + 1; k<M; k++)
            f[k] = a(i, k);
        f[M] = b(i);
        /* 发送归一化后的主行 */
        MPI_Bcast(&f[0], M + 1, MPI_FLOAT, my_rank, MPI_COMM_WORLD);
        /* 发送主行中主元素所在的列号 */
        MPI_Bcast(&l, 1, MPI_INT, my_rank, MPI_COMM_WORLD);
    }
    else          //从进程
    {
        v = i*p + j;
        MPI_Bcast(&f[0], M + 1, MPI_FLOAT, j, MPI_COMM_WORLD);
        MPI_Bcast(&l, 1, MPI_INT, j, MPI_COMM_WORLD);
        if (l != v)
        {
            for (t = 0; t<m; t++)
            {
                temp = a(t, v);
                a(t, v) = a(t, l);
                a(t, l) = temp;
            }

            tem = shift[v];
            shift[v] = shift[l];
            shift[l] = tem;
        }
    }
    if (my_rank <= j)
        for (k = i + 1; k<m; k++)
```

```
                {
                    for (w = v + 1; w<M; w++)
                        a(k, w) = a(k, w) - f[w] * a(k, v);
                    b(k) = b(k) - f[M] * a(k, v);
                }

            if (my_rank>j)
                for (k = i; k<m; k++)
                {
                    for (w = v + 1; w<M; w++)
                        a(k, w) = a(k, w) - f[w] * a(k, v);
                    b(k) = b(k) - f[M] * a(k, v);
                }
        } /* for i j */
    for (i = 0; i<m; i++)
        sum[i] = 0.0;

    for (i = m - 1; i >= 0; i--) /* 回代 */
        for (j = p - 1; j >= 0; j--)
            if (my_rank == j)
            {
                x[i*p + j] = (b(i) - sum[i]) / a(i, i*p + j);

                MPI_Bcast(&x[i*p + j], 1, MPI_FLOAT, my_rank, MPI_COMM_WORLD);
                for (k = 0; k<i; k++)
                    sum[k] = sum[k] + a(k, i*p + j)*x[i*p + j];
            }
            else
            {
                MPI_Bcast(&x[i*p + j], 1, MPI_FLOAT, j, MPI_COMM_WORLD);
                if (my_rank>j)
                    for (k = 0; k<i; k++)
                        sum[k] = sum[k] + a(k, i*p + j)*x[i*p + j];

                if (my_rank<j)
                    for (k = 0; k <= i; k++)
                        sum[k] = sum[k] + a(k, i*p + j)*x[i*p + j];
            }
    if (my_rank != 0)
        for (i = 0; i<m; i++)
            MPI_Send(&x[i*p + my_rank], 1, MPI_FLOAT, 0, i, MPI_COMM_WORLD);
```

```
    else
        for (i = 1; i<p; i++)
            for (j = 0; j<m; j++)
                MPI_Recv(&x[j*p + i], 1, MPI_FLOAT, i, j, MPI_COMM_WORLD, &status);
    end = clock();
    time2 = MPI_Wtime();//结束计时
    if (my_rank == 0)
    {
        cout << "方程组的解" << endl;
        for (k = 0; k<M; k++)
        {
            for (i = 0; i<M; i++)
            {
                if (shift[i] == k) cout << "x[" << k << "]=" << x[i] << endl;
            }
        }
    }
    cout << endl;
    cout << "输入数组大小：" << M << "*" << N << endl;
    cout <<  endl;
    cout << "并行计算计算的核数" << p << endl;
    if (my_rank == 0)
    {
        cout << "时间:" << (double)(end - start) / CLOCKS_PER_SEC << endl;
        cout << "并行计算时间情况" <<endl;
        cout << "总共用时= " << time2 - starttime << endl;//Whole running time
        cout << "分发数据用时 = " << time1 - starttime << endl;//Distribute data time
        cout << "并行计算用时 = " << time2 - time1 << endl;//Parallel compute time
    }
    MPI_Finalize();
    Environment_Finalize(a, b, x, f);
    system("pause");
    return(0);
}
```

5、结果分析

5.1 矩阵 50*51 核数：1,5,10

①1 核运行

```
输入数组大小：50*51

----------------------------------------
并行计算计算的核数1
时间:0
并行计算时间情况
总共用时= 0.0002342
分发数据用时 = 1.31e-05
并行计算用时 = 0.0002211
----------------------------------------
```

图 7 50*51 矩阵 1 核运行

②5 核运行

```
输入数组大小：50*51

----------------------------------------
并行计算计算的核数5
时间:0.001
并行计算时间情况
总共用时= 0.0018409
分发数据用时 = 0.0005322
并行计算用时 = 0.0013087
----------------------------------------
```

图表 8 50*51 矩阵 5 核运行

③10 核运行

```
输入数组大小：50*51

----------------------------------------
并行计算计算的核数10
时间:0.003
并行计算时间情况
总共用时= 0.002918
分发数据用时 = 0.0010664
并行计算用时 = 0.0018516
----------------------------------------
```

图 9 50*51 矩阵 10 核运行

④加速比与运行效率

```
矩阵大小：50 * 51
核数：5
加速比：0.127283
并行效率：0.0254565
```

```
矩阵大小：50 * 51
核数：10
加速比：0.0802605
并行效率：0.00802605
```

图 10 50*51 矩阵 5 核、10 核加速比与效率

5.2 矩阵 200*201 核数 1,5,10

①1 核运行

```
输入数组大小：200*201

----------------------------------------
并行计算计算的核数1
时间:0.009
并行计算时间情况
总共用时= 0.0089487
分发数据用时 = 0.0001265
并行计算用时 = 0.0088222
----------------------------------------
```

图 11 矩阵 200*201 1 核运行

②5 核运行

```
输入数组大小：200*201

----------------------------------------
并行计算计算的核数5
时间:0.005
并行计算时间情况
总共用时= 0.0049092
分发数据用时 = 0.0007822
并行计算用时 = 0.004127
----------------------------------------
```

图 12 矩阵 200*201 5 核运行

③10 核运行

```
输入数组大小：200*201

----------------------------------------
并行计算计算的核数10
时间:0.007
并行计算时间情况
总共用时= 0.0062702
分发数据用时 = 0.0010636
并行计算用时 = 0.0052066
----------------------------------------
```

图 13 矩阵 200*201 10 核运行

④加速比与运行效率

```
矩阵大小：200 * 201
核数：5
加速比：1.8
并行效率：0.36
```

```
矩阵大小：200 * 201
核数：10
加速比：1.42722
并行效率：0.142722
```

图 14 200*201 矩阵 5 核、10 核加速比与效率

5.3 矩阵 500*501 核数 1,5,10

①1 核运行

```
输入数组大小：500*501

----------------------------------------
并行计算计算的核数1
时间:0.13
并行计算时间情况
总共用时= 0.130251
分发数据用时 = 0.0007123
并行计算用时 = 0.129539
----------------------------------------
```

图 15 矩阵 500*501 1 核运行

②5 核运行

```
输入数组大小：500*501

----------------------------------------
并行计算计算的核数5
时间:0.041
并行计算时间情况
总共用时= 0.0402404
分发数据用时 = 0.0017687
并行计算用时 = 0.0384717
----------------------------------------
```

图 16 矩阵 500*501 5 核运行

③10 核运行

```
输入数组大小：500*501

----------------------------------------
并行计算计算的核数10
时间:0.036
并行计算时间情况
总共用时= 0.0363118
分发数据用时 = 0.0020795
并行计算用时 = 0.0342323
----------------------------------------
```

图 17 矩阵 500*501 10 核运行

④加速比与运行效率

```
矩阵大小：500 * 501
核数：5
加速比：3.23685
并行效率：0.647371

矩阵大小：500 * 501
核数：10
加速比：3.58702
并行效率：0.358702
```

图 18 500*501 矩阵 5 核、10 核加速比与效率

5.4 矩阵 1000*1001 核数 1,5,10

①1 核运行

```
输入数组大小：1000*1001

----------------------------------------
并行计算计算的核数1
时间:1.243
并行计算时间情况
总共用时= 1.24312
分发数据用时 = 0.0032815
并行计算用时 = 1.23984
----------------------------------------
```

图 19 矩阵 1000*1001 1 核运行

②5 核运行

```
输入数组大小：1000*1001

----------------------------------------
并行计算计算的核数5
时间:0.357
并行计算时间情况
总共用时= 0.356703
分发数据用时 = 0.0024471
并行计算用时 = 0.354255
----------------------------------------
```

图 20 矩阵 1000*1001 5 核运行

③10 核运行

```
输入数组大小：1000*1001

----------------------------------------
并行计算计算的核数10
时间:0.252
并行计算时间情况
总共用时= 0.251663
分发数据用时 = 0.0028145
并行计算用时 = 0.248848
----------------------------------------
```

图 21 矩阵 1000*1001 10 核运行

④加速比与运行效率

```
矩阵大小：1000 * 1001
核数：5
加速比：3.48179
并行效率：0.696359
```

```
矩阵大小：1000 * 1001
核数：10
加速比：4.93254
并行效率：0.493254
```

图 22 1000*1001 矩阵 5 核、10 核加速比与效率

5.5 不同核数加速比并行效率分析

本次课程设计为探讨并行性能的评价指标，我采取了不同矩阵下不同核数的加速比并行效率的对比。并行计算的核数为 5 核、10 核。矩阵大小为 50, 200, 500, 1000。

并行计算的加速比，随着矩阵规模的增加，加速比逐渐接近与计算核数。如何图 23 所示。

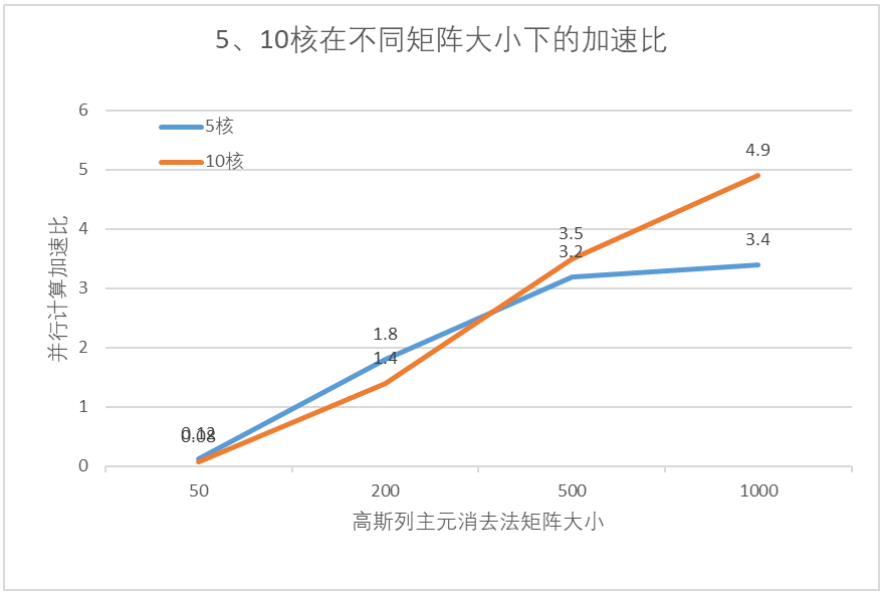


图 22 不同矩阵 5 核、10 核加速比

并行计算的效率，随着矩阵规模的增加，并行效率不断提高。

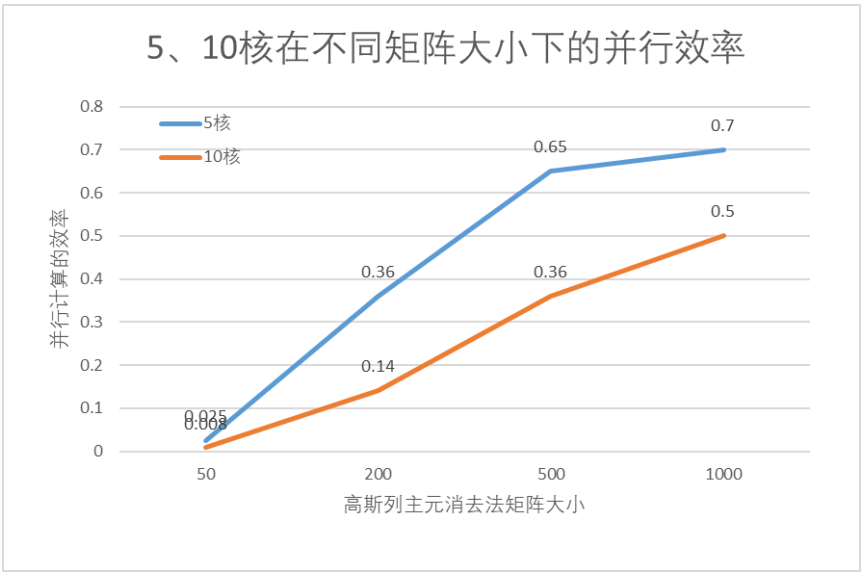


图 23 不同矩阵 5 核、10 核加速比

由于本次并行计算规模没有太大，所以加速比和效率都是 5 核较优，但可以推断出如果计算规模继续扩大，那么 10 核的优势将逐渐扩大。所以在并行计算时，针对不同大小的计算规模我们要选择合适的计算核数，使得并行计算的加速比和效率达到一个合理的范围当中避免出现浪费核心数的情况。

6、个人心得体会

通过本次并行计算与分布式计算课程设计， 我对并行计算有了更加深入的了解。充分掌握了 MPI 编程和数值分析高斯列主元消去法的程序算法。同时对于并行计算性能的评价指标有了更加深入的认识。

本次 MPI 环境是在 Windows 的操作系统下的 MPI+C++的编程环境。同时也使用了老师提供的 Linux 环境下的 MPI 环境。将程序放在两个不同编程环境下运行后，我发现相同运算核的前提下，老师的环境运行速度更快。这里的原因可能是因为笔记本的 CPU 的电压比主机的 CPU 电压更低，所以从运行速度上，笔记本天然会比主机更慢。

我的程序是实现高斯主列元消去法，所以 MPI 程序的并行思路是不同核处理不同的行，最终合并结果。实现算法后，我将不同规模的矩阵输入进程序，以此来分析不同核数与不同规模数据的并行计算的加速比与效率。最终得出结论，并行计算的核数需要与数据规模呈一定比例增长，这样才能保证并行计算的加速比与效率达到合理的范围中。

并行计算与分布式计算在大数据时代的应用十分广泛，大规模科学与工程计算应用对并行计算的需求十分巨大。本次课程设计，让我对并行计算的实际应用有了深刻的认识，自己动手实践能力大大增强。我的课程设计所有源码都已开源，Github 地址如下：https://github.com/LHJ205/MPI_GAUSS。