并行程序设计第三次作业

> 范吴运维 21373365
>
> JobID: 8455324

错误处理

1. 文件读写：

使用Cannon辅助程序生成矩阵，调整输入输出；

2. MPI通信数据类型：

计算double 型矩阵，将通信数据类型调整为 `MPI_DOUBLE`

3. 维护结束标记：

进程数和 A 矩阵行数不一致时，出现死锁。维护进程结束标记。

```
if (numsend < M)
            { // 如果A还没有分配完,向返回计算结果的从进程再分配A中的一行
                MPI_Send(A[numsend], N, MPI_DOUBLE, sender, 99,
MPI_COMM_WORLD);
                numsend++;
            }
            else
            {
                MPI_Send(A[0], N, MPI_DOUBLE, sender, 0, MPI_COMM_WORLD);
                done[sender] = 1; // 标记进程已经结束计算
            }
```

4. 调整矩阵分配行数

分配的矩阵行数依赖于已发送的数目，将A 下标改为 `numsend`

```
MPI_Send(A[numsend], N, MPI_DOUBLE, sender, 99, MPI_COMM_WORLD);
```

5. B 矩阵类型调整

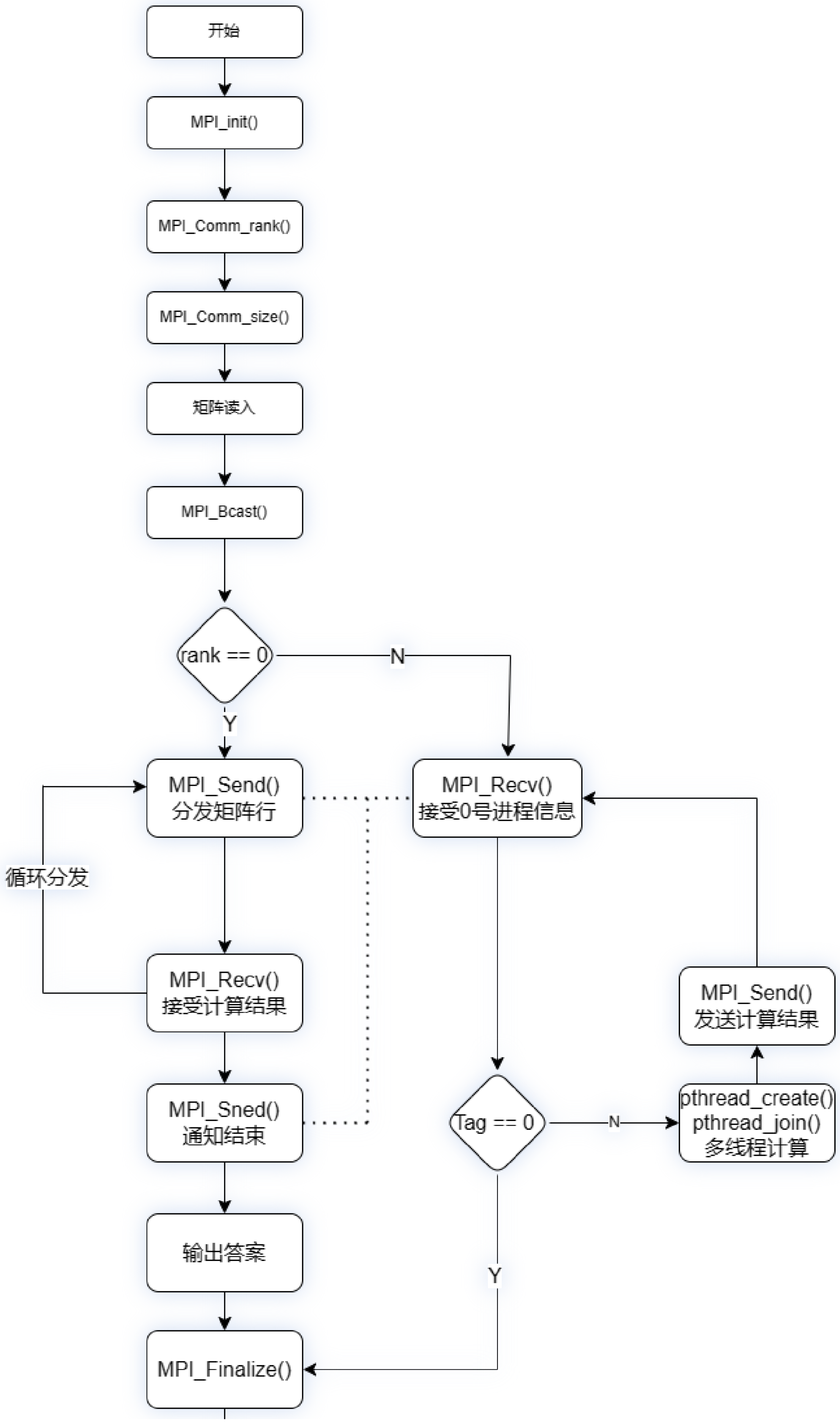
使用指针访问 B矩阵元素

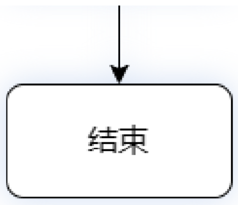
```
myarg->C_row[i] += myarg->A_row[j] * myarg->B[j * P + i];
```

6. MPI_Finalize()

缺少 `MPI_Finalize()` 函数

流程图


开始
MPI_init()
MPI_Comm_rank()
MPI_Comm_size()
矩阵读入
MPI_Bcast()
rank == 0
N
Y
MPI_Send()
分发矩阵行
MPI_Recv()
接受0号进程信息
循环分发
MPI_Recv()
接受计算结果
MPI_Send()
发送计算结果
MPI_Sned()
通知结束
Tag == 0
N
pthread_create()
pthread_join()
多线程计算
Y
输出答案
MPI_Finalize()



程序源码

solve.cxx

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <mpi.h>
#include <pthread.h>

#define NumOfCores 8

struct threadArg
{
    int tid;
    double *B;
    double *A_row;
    double *C_row;
    int numthreads;
};

int M, N, P;

void *worker(void *arg)
{
    int i, j;
    struct threadArg *myarg = (struct threadArg *)arg;
    for (i = myarg->tid; i < P; i += myarg->numthreads)
    {
        // 平均分配B的所有列
        myarg->C_row[i] = 0.0;
        for (j = 0; j < N; j++)
        {
            myarg->C_row[i] += myarg->A_row[j] * myarg->B[j * P + i];
            // B中的一列与A行相乘，计算结果存入C一行中的对应位置
        }
    }
    return NULL;
}

int main(int argc, char *argv[])
{
    int i, j, myid, numprocs, numsend, sender, numthreads;
    double *A_row, *C_row;
    pthread_t *tids;
    struct threadArg *targs;
    MPI_Status status;
    FILE *matrixA, *matrixB, *matrixC;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    if (!myid)
    {
        matrixA = fopen("input1.txt", "r");
        matrixB = fopen("input2.txt", "r");
        fread(&M, sizeof(int), 1, matrixA);
        fread(&N, sizeof(int), 1, matrixA);
        fread(&N, sizeof(int), 1, matrixB);
        fread(&P, sizeof(int), 1, matrixB);
    }
    // 广播M, N, P
    MPI_Bcast(&M, 1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(&N, 1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(&P, 1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Barrier(MPI_COMM_WORLD);
    // 声明数组
    double A[M][N], B[N][P], C[M][P];
    int done[1000] = {0};
    if (!myid)
    { // 在0进程内初始化矩阵A和B
        for (i = 0; i < M; i++)
            for (j = 0; j < N; j++)
                fread(&A[i][j], sizeof(double), 1, matrixA);
        for (i = 0; i < N; i++)
            for (j = 0; j < P; j++)
                fread(&B[i][j], sizeof(double), 1, matrixB);
        fclose(matrixA);
        fclose(matrixB);
    }
    MPI_Bcast(B[0], N * P, MPI_DOUBLE, 0, MPI_COMM_WORLD); // 广播矩阵B
    if (!myid)
    { /*0进程：分配任务和回收结果*/
        j = (numprocs - 1) < M ? (numprocs - 1) : M;
        for (i = 1; i <= j; i++)
        {
            MPI_Send(A[i - 1], N, MPI_DOUBLE, i, 99, MPI_COMM_WORLD);
            // 给从进程依次依次分配A的一行
        }
        numsend = j;
        for (i = 1; i <= M; i++)
        {
            sender = (i - 1) % (numprocs - 1) + 1;
            MPI_Recv(C[i - 1], P, MPI_DOUBLE, sender, 100, MPI_COMM_WORLD,
                     &status); // 依次回收计算结果
            if (numsend < M)
            { // 如果A还没有分配完,向返回计算结果的从进程再分配A中的一行
                MPI_Send(A[numsend], N, MPI_DOUBLE, sender, 99,
MPI_COMM_WORLD);
                numsend++;
            }
            else
            {
                MPI_Send(A[0], N, MPI_DOUBLE, sender, 0, MPI_COMM_WORLD);
                done[sender] = 1; // 标记进程已经结束计算
            }
        }
        for (i = 1; i <= numprocs - 1; i++)
        {
            if (done[i] == 0)
            { // 把还未发送结束信号的进程补上
                MPI_Send(A[0], N, MPI_DOUBLE, i, 0, MPI_COMM_WORLD);
            }
        }

        matrixC = fopen("solve_out.txt", "w");
        int fsizec = sizeof(int) * 2 + sizeof(double) * M * P;
        char *fstreamc = (char *)malloc(fsizec);
        ((int *)fstreamc)[0] = M;
        ((int *)fstreamc)[1] = P;
        double *pc = (double *)(fstreamc + sizeof(int) * 2);
        for (i = 0; i < M; i++)
            for (j = 0; j < P; j++)
                *(pc + i * P + j) = C[i][j]; // 输出矩阵C
        fwrite(fstreamc, sizeof(char), fsizec, matrixC);
        fclose(matrixC);
        free(fstreamc);
    }
    else
    { // 从进程
        /*从进程(myid > 0)：接收0进程发来的任务， 计算完毕发回主进程*/
        numthreads = NumOfCores;                                    // 从进程所在
节点的CPU数
        tids = (pthread_t *)malloc(numthreads * sizeof(pthread_t)); // tids数组
用来存放线程ID
        A_row = (double *)malloc(N * sizeof(double));               // 存放主进程
分配的A中一行
```

```
        C_row = (double *)malloc(P * sizeof(double));               // 存放计算结果C中的一行
        targs = (struct threadArg *)malloc(numthreads * sizeof(struct threadArg)); // 线程参数，由于传入参数比较多，采用结果传递
        for (i = 0; i < numthreads; i++)
        {

            targs[i].tid = i;
            targs[i].B = B[0];
            targs[i].A_row = A_row;
            targs[i].C_row = C_row;
            targs[i].numthreads = numthreads;
        }
        while (1)
        {
            MPI_Recv(A_row, N, MPI_DOUBLE, 0, MPI_ANY_TAG, MPI_COMM_WORLD,
                     &status);       // 接收0进程发送来A的一行
            if (status.MPI_TAG == 0) // 若接收到标识为0的消息则退出执行
                break;
            for (i = 0; i < numthreads; i++)
            {
                pthread_create(&tids[i], NULL, worker, &targs[i]); // 创建线程执行计算
            }
            for (i = 0; i < numthreads; i++)
            {
                pthread_join(tids[i], NULL); // 等待线程内的计算完成
            }
            MPI_Send(C_row, P, MPI_DOUBLE, 0, 100, MPI_COMM_WORLD);
        }
        // 释放内存
        free(tids);
        free(A_row);
        free(C_row);
        free(targs);
        // 其他进程接收任务、计算、返回计算结果
    } /*从进程结束*/
    MPI_Finalize();
    return 0;
}
```

solve_run.sh

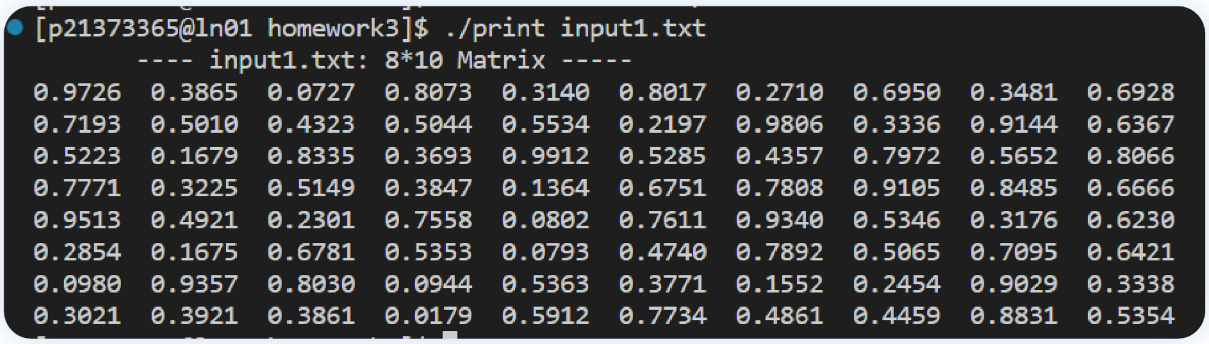
```
#!/bin/bash
#SBATCH -J waysome_hw3   #作业名
#SBATCH -p cpu-quota
#SBATCH -N 4            #4 节点
#SBATCH -n 8           #8 核
#SBATCH -o waysome_hw3.out # 将标准输出结果
#SBATCH -e waysome_hw3.err # 将错误输出结果

srun hostname | sort > machinefile.${SLURM_JOB_ID}
NP=`cat machinefile.${SLURM_JOB_ID} | wc -l`
module load intel/19.0.5.281
export I_MPI_HYDRA_TOPOLIB=ipl
mpirun -genv I_MPI_FABRICS shm:dapl -np ${NP} -f ./machinefile.${SLURM_JOB_ID}
./solve
```

run.sh

```
#!/bin/bash
if [ $# -lt 3 ];
then
    echo "wrong arguments"
else
    ./matgen $1 $2 input1.txt
    ./print input1.txt
    ./matgen $2 $3 input2.txt
    ./print input2.txt
    ./serial input1.txt input2.txt serial_out.txt
    ./print serial_out.txt
fi
```

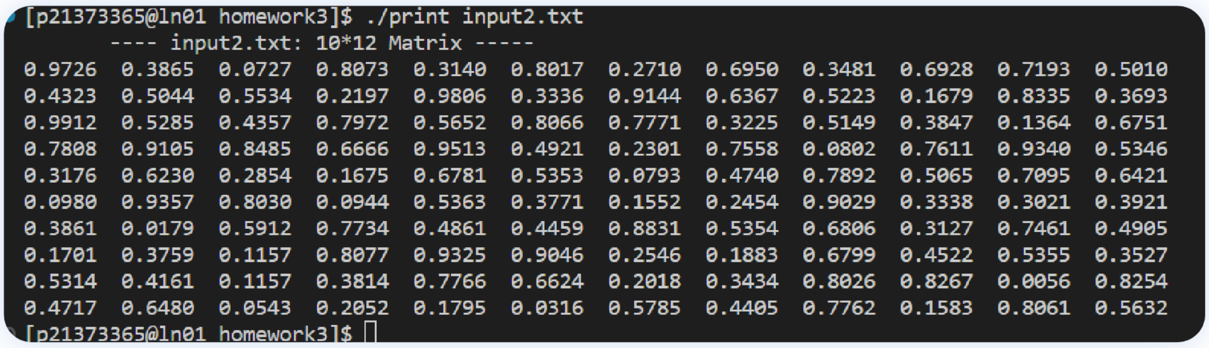
程序输出

input1.txt (A : 8 * 10)



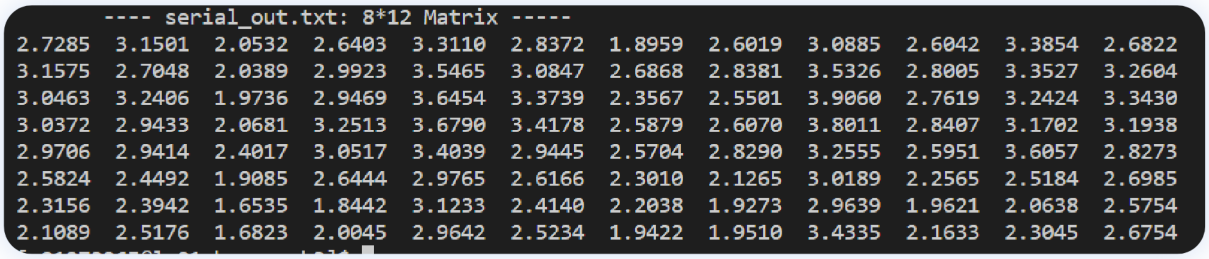
```
[p21373365@ln01 homework3]$ ./print input1.txt
        ---- input1.txt: 8*10 Matrix -----
 0.9726  0.3865  0.0727  0.8073  0.3140  0.8017  0.2710  0.6950  0.3481  0.6928
 0.7193  0.5010  0.4323  0.5044  0.5534  0.2197  0.9806  0.3336  0.9144  0.6367
 0.5223  0.1679  0.8335  0.3693  0.9912  0.5285  0.4357  0.7972  0.5652  0.8066
 0.7771  0.3225  0.5149  0.3847  0.1364  0.6751  0.7808  0.9105  0.8485  0.6666
 0.9513  0.4921  0.2301  0.7558  0.0802  0.7611  0.9340  0.5346  0.3176  0.6230
 0.2854  0.1675  0.6781  0.5353  0.0793  0.4740  0.7892  0.5065  0.7095  0.6421
 0.0980  0.9357  0.8030  0.0944  0.5363  0.3771  0.1552  0.2454  0.9029  0.3338
 0.3021  0.3921  0.3861  0.0179  0.5912  0.7734  0.4861  0.4459  0.8831  0.5354
```

input2.txt (B : 10 * 12)



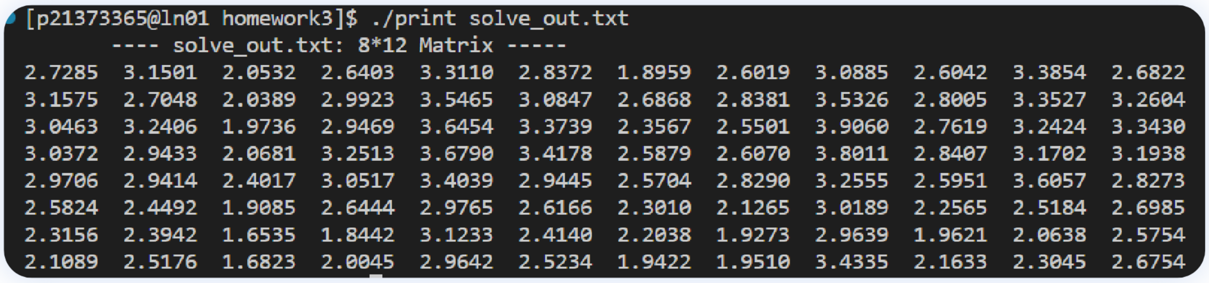
```
[p21373365@ln01 homework3]$ ./print input2.txt
        ---- input2.txt: 10*12 Matrix -----
0.9726  0.3865  0.0727  0.8073  0.3140  0.8017  0.2710  0.6950  0.3481  0.6928  0.7193  0.5010
0.4323  0.5044  0.5534  0.2197  0.9806  0.3336  0.9144  0.6367  0.5223  0.1679  0.8335  0.3693
0.9912  0.5285  0.4357  0.7972  0.5652  0.8066  0.7771  0.3225  0.5149  0.3847  0.1364  0.6751
0.7808  0.9105  0.8485  0.6666  0.9513  0.4921  0.2301  0.7558  0.0802  0.7611  0.9340  0.5346
0.3176  0.6230  0.2854  0.1675  0.6781  0.5353  0.0793  0.4740  0.7892  0.5065  0.7095  0.6421
0.0980  0.9357  0.8030  0.0944  0.5363  0.3771  0.1552  0.2454  0.9029  0.3338  0.3021  0.3921
0.3861  0.0179  0.5912  0.7734  0.4861  0.4459  0.8831  0.5354  0.6806  0.3127  0.7461  0.4905
0.1701  0.3759  0.1157  0.8077  0.9325  0.9046  0.2546  0.1883  0.6799  0.4522  0.5355  0.3527
0.5314  0.4161  0.1157  0.3814  0.7766  0.6624  0.2018  0.3434  0.8026  0.8267  0.0056  0.8254
0.4717  0.6480  0.0543  0.2052  0.1795  0.0316  0.5785  0.4405  0.7762  0.1583  0.8061  0.5632
[p21373365@ln01 homework3]$
```

serial_out.txt (C1 : 8 * 12)



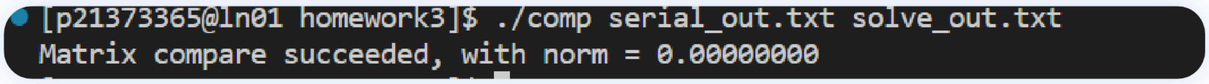
```
        ---- serial_out.txt: 8*12 Matrix -----
2.7285  3.1501  2.0532  2.6403  3.3110  2.8372  1.8959  2.6019  3.0885  2.6042  3.3854  2.6822
3.1575  2.7048  2.0389  2.9923  3.5465  3.0847  2.6868  2.8381  3.5326  2.8005  3.3527  3.2604
3.0463  3.2406  1.9736  2.9469  3.6454  3.3739  2.3567  2.5501  3.9060  2.7619  3.2424  3.3430
3.0372  2.9433  2.0681  3.2513  3.6790  3.4178  2.5879  2.6070  3.8011  2.8407  3.1702  3.1938
2.9706  2.9414  2.4017  3.0517  3.4039  2.9445  2.5704  2.8290  3.2555  2.5951  3.6057  2.8273
2.5824  2.4492  1.9085  2.6444  2.9765  2.6166  2.3010  2.1265  3.0189  2.2565  2.5184  2.6985
2.3156  2.3942  1.6535  1.8442  3.1233  2.4140  2.2038  1.9273  2.9639  1.9621  2.0638  2.5754
2.1089  2.5176  1.6823  2.0045  2.9642  2.5234  1.9422  1.9510  3.4335  2.1633  2.3045  2.6754
```

solve_out.txt (C2 : 8 * 12)



```
[p21373365@ln01 homework3]$ ./print solve_out.txt
        ---- solve_out.txt: 8*12 Matrix -----
 2.7285  3.1501  2.0532  2.6403  3.3110  2.8372  1.8959  2.6019  3.0885  2.6042  3.3854  2.6822
 3.1575  2.7048  2.0389  2.9923  3.5465  3.0847  2.6868  2.8381  3.5326  2.8005  3.3527  3.2604
 3.0463  3.2406  1.9736  2.9469  3.6454  3.3739  2.3567  2.5501  3.9060  2.7619  3.2424  3.3430
 3.0372  2.9433  2.0681  3.2513  3.6790  3.4178  2.5879  2.6070  3.8011  2.8407  3.1702  3.1938
 2.9706  2.9414  2.4017  3.0517  3.4039  2.9445  2.5704  2.8290  3.2555  2.5951  3.6057  2.8273
 2.5824  2.4492  1.9085  2.6444  2.9765  2.6166  2.3010  2.1265  3.0189  2.2565  2.5184  2.6985
 2.3156  2.3942  1.6535  1.8442  3.1233  2.4140  2.2038  1.9273  2.9639  1.9621  2.0638  2.5754
 2.1089  2.5176  1.6823  2.0045  2.9642  2.5234  1.9422  1.9510  3.4335  2.1633  2.3045  2.6754
```

comp



```
[p21373365@ln01 homework3]$ ./comp serial_out.txt solve_out.txt
Matrix compare succeeded, with norm = 0.00000000
```