并行计算课程

结 题 报 告

报告名称：	MPI 实现矩阵转置算法
姓　　名：	李润泽
学　　号：	3019244266
联系电话：	15942643201
电子邮箱：	Lirz3019244266@163.com
填写日期：	2021 年 4 月 27 日

2020 年制

一、实验内容概述

本实验要求使用 MPI 并行编程技术计算对应的转置矩阵（禁止使用广播通信方式进行矩阵转置操作），并通过不同的转置方式以及不同的进程数来进行性能分析。旨在提升学生对并行计算的理解和认识，培养学生编写并行程序的能力，并巩固和加深学生对 MPI 并行编程的理解和认识。

在进行矩阵转置的过程中，我们分别使用串行算法和并行算法。课程中并行计算提供了两种转置方式：块棋盘划分方法和直角划分方法。实验要求学生编写相应的代码，并通过改变进程数，分别进行不同方式的数据统计、时间记录以及性能分析。（代码中的 N 尽可能大）

1.计算方法：串行算法、并行算法（块棋盘划分方法、直角划分方法）

2.编程语言：C 或 C++

3.并行计算操作系统：天津大学超算平台 CentOS 7.6

4.编译环境：Intel 19.1.0.166

5.脚本编写：系统提交需要编写 PBS 脚本实现

6.数据分析要求：提供实验结果数据、加速比曲线以及效率

二、并行算法分析设计

（一）实现方法

1 生成矩阵（矩阵为 900*900、1800*1800、2700*2700 或 3600*3600）,记录开始时间；

2 将矩阵中分块，用一维数组表示，并进行子块转置；

3 用 MPI_Send 将各个子块矩阵发送给相应进程；

4 所有进程转置各自的子块；

5 其他进程将子块发送给主进程，主进程接收并存放在 txt 文件中；

6 记录结束时间

（二）程序流程图

（三）转置算法

1 块棋盘划分方法

1.1 转置思路

假设线程数为 p，编号为 0,1,…,p-1，则将 n 阶矩阵 A 分成 p 个大小为 m*m 个子块，p 个子块组成一个 √p* √p 的子块阵列，如下图所示：

转置分为两步进行：第一步，子块转置；第二步，处理器内部局部转置。

1.2 核心代码（完整代码位于 matrix_chessboard_MPI.cpp）

```
if(my_rank == 0){
    init();
    getValue();
    t_start = MPI_Wtime();

    for(int i=0;i<sqrt_group_size;i++){
        for(int j=0;j<sqrt_group_size;j++){
            int point = 0;
            for(int k=length*i;k<length*(i+1);k++){
                for(int l=length*j;l<length*(j+1);l++){
                    tmp[point] = matrix[k][l];
                    point++;
                }
            }

            if(i==0 && j==0){
                for(int i=0;i<length;i++){
                    for(int j=0;j<length;j++){
                        out[i][j] = matrix[j][i];
                    }
                }
            }
            else{
    MPI_Send(tmp,length*length,MPI_INT,i*sqrt_group_size+j,i*sqrt_group_size+j,MPI_C
OMM_WORLD);
            }
        }
    }
}

else{
    MPI_Recv(tmp,length*length,MPI_INT,0,my_rank,MPI_COMM_WORLD,&status);
    int t;
    for(int i=0;i<length;i++){
```

```
            for(int j=i+1;j<length;j++){
                t = tmp[i*length+j];
                tmp[i*length+j] = tmp[j*length+i];
                tmp[j*length+i] = t;
            }
        }
        MPI_Send(tmp,length*length,MPI_INT,0,my_rank,MPI_COMM_WORLD);
}

if(my_rank==0){
        for(int i=0;i<sqrt_group_size;i++){
            for(int j=0;j<sqrt_group_size;j++){
                if(i!=0 || j!=0){
        MPI_Recv(tmp,length*length,MPI_INT,i*sqrt_group_size+j,i*sqrt_group_size+j,MPI_C
OMM_WORLD,&status);
                        for(int x=0;x<length;x++){
                            for(int y=0;y<length;y++){
                                out[j*length+x][i*length+y] = tmp[x*length+y];
                            }
                        }
                    }
            }
        }

        t_end = MPI_Wtime();
        printf("Matrix order:%d, Time cost:%lf\n",n,t_end-t_start);
        free(matrix);
}
```

2 并行算法——直角划分方法

2.1 转置思路

转置分为两步进行：第一步，将矩阵划分为大小相近的 p 个子块；第二步，对每一个子块进行转置。

直角划分方法如下图所示：

2.2 核心代码（完整代码位于 matrix_rightAngle_MPI.cpp）

```
for (int i = my_i; i < (myid+1)*n; i++){
    for (int j = 0; j < i; j++){
        matrix[i][j] = matrix[i][j]^matrix[j][i];
        matrix[j][i] = matrix[i][j]^matrix[j][i];
        matrix[i][j] = matrix[i][j]^matrix[j][i];
    }
}

if(myid == 0){
    int temp[num-1][N][N];
    MPI_Status status;
    t_start = MPI_Wtime();

    for (int i = 1; i < num; i++){
        MPI_Recv(&temp[i-1][0][0], N*N, MPI_INT, i, 30, MPI_COMM_WORLD,
&status);
    }
    for (int i = 1; i < num; i++){
        int current_i = i*n;
        for (int m = current_i; m < (i+1)*n; m++){
            for (int j = 0; j < m; j++){
                matrix[m][j] = temp[i-1][m][j];
                matrix[j][m] = temp[i-1][j][m];
            }
        }
    }
```

```
    t_end = MPI_Wtime();
    printf("Matrix order:%d, Time cost:%lf\n",N,t_end-t_start);
}
else{
    MPI_Send(&matrix[0][0], N*N, MPI_INT, 0, 30, MPI_COMM_WORLD);
}
```

（四）运行脚本（以 test_matrix_chessborad_MPI.pbs，p=4，N=900 为例）

```
#!/bin/bash
#PBS -N test
#PBS -q qstudent
#PBS -l nodes=1:ppn=4
#PBS -j oe

cd $PBS_O_WORKDIR
procs=$(cat $PBS_NODEFILE | wc -l)
mpirun ./matrix_chessboard_MPI 900
```

三、实验数据分析

（一）实验环境

CPU：Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz　　2.59 GHz

内存：16.0GB

互联网络参数：172.23.80.20（用的是校园网）

（二）实验数据综合分析

1 实验数据

1.1 块棋盘划分方法

1.1.1 计算时间

ppn / N	1	4	16
900	0.016775	0.015209	0.023157
1800	0.076600	0.051678	0.043946
2700	0.188044	0.114146	0.091105
3600	0.360282	0.203548	0.166940

1.1.2 加速比

ppn / N	1	4	16
900	1	1.103	0.724
1800	1	1.482	1.743
2700	1	1.647	2.064
3600	1	1.770	2.158

1.1.3 效率

ppn / N	1	4	16
900	1	0.276	0.045
1800	1	0.371	0.109
2700	1	0.412	0.129
3600	1	0.443	0.135

1.2 直角划分方法

1.2.1 计算时间

ppn	1	4	16

N			
900	0.059193	0.022472	0.077205
1800	0.072301	0.063850	0.230804
2700	0.105122	0.131865	0.505565
3600	0.140364	0.233189	0.899268

1.2.2 加速比

ppn N	**1**	**4**	**16**
900	1	2.634	0.767
1800	1	1.132	0.313
2700	1	0.797	0.208
3600	1	0.602	0.156

1.2.3 效率

ppn / N	1	4	16
900	1	0.659	0.048
1800	1	0.283	0.020
2700	1	0.199	0.013
3600	1	0.150	0.010

可以看出，随着数据规模的增大，程序耗费时间也在增大，而在同等规模的情况下，随着进程的增多，一般而言所需时间在不断减小，加速比较高，但效率较低。

四、实验总结

根据分析我们可以得出以下结论：

1、程序在进程为 4、16 时效果较好（进程数为 9 的情况在进行实验时效果不好，故在报告里并未体现）同等进程数以及数据规模的情况下，运行时间并无较大波动；

2、随着数据规模的增大，加速比上升，但效率降低；

3、随着进程数的增加，加速比上升，但效率也降低。

在本次实验中，我了解到 MPI 编程与多线程有所不同，多线程编程中，当某一个线程改变全局变量是，所有线程都可以收到；但对于 MPI 编程，全局变量的变化只有在使用 MPI_Send 和 MPI_Recv 时才可以将其传输出去。

在编程的过程中，我也遇到了一些问题。首先我需要将读取的矩阵分成子块，并把子块的首地址进行转置。一开始始终没能成功，是由于我读取的矩阵与开的矩阵规模不同导致。

通过编写两种不同的并行算法来进行矩阵的转置，我对计算机并行计算有了更加深刻的认识。通过对实验中串行、并行结果的分析，我也更加清晰认识到并行计算的重要性。这会对我未来编写有效高速的代码奠定了更加坚实的基础。

五、课程总结

本次实验在有了前两次实验的基础上，继续在 Windows 系统和天津大学超算平台的环境下进行 MPI 编程。有了前两次次实验的经验以及教训，我在编程过程中变得更加轻松，尽管在过程中遇到了或多或少的错误，但在自己的不断修改、优化以及与同学的沟通下，我算是顺利地完成了此次实验。就授课的内容而言，授课教师和助教对我的帮助很大，实验指导书讲解详细，图文并茂，在编程前可以起到很好的帮助，可谓是事半功倍。

在实验过程中，请允许我在此提出一下建议：

1，实验指导书中可以适当添加一些与并行计算相关联知识的网络链接，可以提供一个让同学们自主获取知识的渠道，这样可以更加高效率地进行该学科的学习。

2，建议实验期间进行开放式问题的探究（可以选做），这样可以促进同学们进行深入挖掘，对并行计算有更多了解。

在做实验之前，我认为我们必须要将课程上学到的理论知识完全吸收，这是进行实践过程中最为重要的基石。至此，第三次实验顺利完成，这对我在并行计算方面的学习有巨大的帮助。

附：上机实验与课程知识点分析

序号	上机实验内容	理论知识点	分析总结
1	矩阵转置的方式	两种并行计算方式详见实验指导书	在块棋盘划分方法中，注意进程数需要是整数的平方，否则转置过程中易出现错误。
2	加速比	S(n)=ts/tp	加速比等于串行计算时间与多线程计算时间的比值
3	效率	E=S(n)/n	效率等于加速比与线程数的比值
4			