**深 圳 大 学**

**实 验 报 告**

**课程名称****：并行计算**

**实验名称：Cannon矩阵乘法的MPI并行程序**

**姓 名：刘俊楠**

**学 号：2017303010**

**班 级：计科一班**

**实验日期：第16周实验课**

## 一. 实验目的

1. 学会编写Cannon矩阵乘法的MPI程序；

2. 对并行程序进行简单的性能分析。

## 二. 实验环境

1. 软件环境：Microsoft Visual Studio 2013。

## 三. 实验内容

1. 实验要求：用MPI编写两个*n*阶方阵*A*和*B*的Cannon矩阵乘法，结果存放在方阵*C*中。

* *A*、*B*、*C*都采用块棋盘划分存储，每个子块的大小都为，不足的部分用0填充。
* *A*和*B*中的每个数都初始化为一个0到1之间的随机double型值（用rand()/double(RAND\_MAX)实现）。
* 尽可能减少执行时间和存储空间。
* 添加检测计算结果是否正确的代码。
* 计算执行时间用MPI\_Wtime()函数。

2. 程序代码和说明：

#include<iostream>

#include<mpi.h>

#include<math.h>

#include<algorithm>

#include<vector>

**using** **namespace** std**;**

//初始化函数initABC

//一维数组表二维 row为行 cols为列 A为一维数组

int n **=** 1000**;**//矩阵大小

double**\*\*** A **=** **new** double**\*** **[**n**];**//矩阵A

double**\*\*** B **=** **new** double**\*** **[**n**];**//矩阵B

double**\*\*** C **=** **new** double**\*** **[**n**];**//矩阵C 存放并行计算的矩阵数组

double**\*\*** Single\_c **=** **new** double**\*** **[**n**];**//存放串行就按的结果矩阵数组

int**\*\*** sit**;**//sit数组存放当前进程块的排布

int length**;**//表示sit矩阵长度

//初始化矩阵函数

void initABC**(**double**\*\*** A**,** int row**,** int cols**)**

**{**

**for** **(**int i **=** 0**;** i **<** row**;** i**++)** **{**

**for** **(**int j **=** 0**;** j **<** cols**;** j**++)** **{**

A**[**i**][**j**]** **=** **(**double**)**rand**()** **/** double**(**RAND\_MAX**);**

**}**

**}**

**}**

//串行计算函数

void single\_C**(**double**\*\*** A**,** double**\*\*** B**,** double**\*\*** C**)** **{**

//串行计算矩阵乘法

**for** **(**int i **=** 0**;** i **<** n**;** i**++)** **{**

**for** **(**int j **=** 0**;** j **<** n**;** j**++)** **{**

C**[**i**][**j**]** **=** 0**;**

**for** **(**int k **=** 0**;** k **<** n**;** k**++)** **{**

C**[**i**][**j**]** **=** C**[**i**][**j**]** **+** **(**A**[**i**][**k**]** **\*** B**[**k**][**j**]);**

**}**

**}**

**}**

**return** **;**

**}**

//结果检查函数

void compare\_C**(**double**\*\*** c**,** double**\*\*** s\_c**)** **{**

**for** **(**int i **=** 0**;** i **<** n**;** i**++)** **{**

**for** **(**int j **=** 0**;** j **<** n**;** j**++)** **{**

**if** **(**c**[**i**][**j**]** **-** s\_c**[**i**][**j**]>=**1e-6**)** **{**

cout **<<** "i=" **<<** i **<<** " | j=" **<<** j **<<** " | c=" **<<** c**[**i**][**j**]** **<<** " | s\_c=" **<<** s\_c**[**i**][**j**]** **<<** endl**;**

exit**(**0**);**

**}**

**}**

**}**

**return;**

**}**

//计算初始对准需要传递的rank值

int Initial\_Focus**(**int p**,**int row**,**int col**,**bool left\_up**)** **{**

//向左移的情况

**if** **(!**left\_up**)** **{**

int num**;**

//如果块不在第一行

**if** **(**row **!=** 0**)** **{**

//根据rank以及row映射到对应rank

//当偏移需要循环时

**if** **(**col**-**row**<**0**)** **{**

int diff **=** row**-**col**;**

**return** sit**[**row**][(**length**)-**diff**];**

**}**

**else** **{**

**return** sit**[**row**][**col **-** row**];**

**}**

**}**

**else** **{**

**return** sit**[**row**][**col**];**

**}**

**}**

**else** **{**

//向上移的情况

**if** **(**col **!=** 0**)** **{**

**if** **(**row **-** col **<** 0**)** **{**

int diff **=** col **-** row**;**

**return** sit**[**length **-** diff**][**col**];**

**}**

**else** **{**

**return** sit**[**row **-** col**][**col**];**

**}**

**}**

**else** **{**

**return** sit**[**row**][**col**];**

**}**

**}**

**}**

int main**(**int argc**,** char**\*\*** argv**)** **{**

int rank**,** size**;** //分别是进程号与进程数目

int times **=** 5**;** //执行次数

MPI\_Status status**;** //MPI状态变量

double end1**;** //串行时间

int p **=** 9**;** //当前进程数目

//start、end：并行时间记录

//sum：并行计算总时间，用于计算并行计算平均时间

double start**,** end**,** sum **=** 0**;**

MPI\_Init**(&**argc**,** **&**argv**);**//初始化MPI

MPI\_Comm\_rank**(**MPI\_COMM\_WORLD**,** **&**rank**);**

MPI\_Comm\_size**(**MPI\_COMM\_WORLD**,** **&**size**);**

**while** **(**times**--)** **{**

//根号p

int sqrtp **=** sqrt**(**p**);**

int block\_size **=** n **/** sqrtp**;**//块大小

int block\_large **=** block\_size **\*** block\_size**;** //每个块的double数目

double**\*** a\_temp **=** **new** double**[**block\_large**];** //暂时存放上一个进程发来的a数组

double**\*** b\_temp **=** **new** double**[**block\_large**];** //暂时存放上一个进程发来的b数组

double**\*** a **=** **new** double**[**block\_large**];** //进程块里面的a数组

double**\*** b **=** **new** double**[**block\_large**];** //进程块里面的b数组

double**\*** c **=** **new** double**[**block\_large**];** //进程块里面的c数组

int p\_col **=** rank **%** sqrtp**;** //进程块逻辑上处于的列号

int p\_cow **=** **(**rank **-** p\_col**)** **/** sqrtp**;** //进程块逻辑上处于的行号

int recv**,** recv1**;**

//初始化c数组

**for** **(**int i **=** 0**;** i **<** block\_large**;** i**++)** **{**

c**[**i**]** **=** 0**;**

**}**

//0号进程下 初始化以及串行计算

**if** **(**rank **==** 0**)** **{**

**for** **(**int i **=** 0**;** i **<** n**;** i**++)** **{**

A**[**i**]** **=** **new** double**[**n**];**

B**[**i**]** **=** **new** double**[**n**];**

C**[**i**]** **=** **new** double**[**n**];**

Single\_c**[**i**]** **=** **new** double**[**n**];**

**}**

initABC**(**A**,** n**,** n**);**

initABC**(**B**,** n**,** n**);**

**if** **(**times **==** 4**)** **{**

clock\_t start1 **=** clock**();**

single\_C**(**A**,** B**,** Single\_c**);**

clock\_t end11 **=** clock**();**

end1 **=** **(**double**)(**end11 **-** start1**)** **/** 1000**;**

cout **<<** "串行时间为：" **<<** end1 **<<** "s" **<<** endl**;**

**}**

**}**

//计算矩阵块的排布 sit数组

length **=** sqrtp**;**

int q **=** 0**;**

sit **=** **new** int**\*** **[**length**];**

**for** **(**int i **=** 0**;** i **<** length**;** i**++)** **{**

sit**[**i**]** **=** **new** int**[**length**];**

**for** **(**int j **=** 0**;** j **<** length**;** j**++)** **{**

sit**[**i**][**j**]** **=** q**;**

q**++;**

**}**

**}**

//0号进程进行分块处理

**if** **(**rank **==** 0**)** **{**

//开始记录并行时间

start **=** MPI\_Wtime**();**

int imin**,** jmin**,** imax**,** jmax**,** l **=** 0**;**

**for** **(**int k **=** 0**;** k **<** p**;** k**++)** **{**

//因为是一维数组存放矩阵块，所以需要l来给temp矩阵赋值，imin是块内行号最小值，jmin是块内列号最小值

jmin **=** **(**k **%** sqrtp**)** **\*** block\_size**;**

jmax **=** **(**k **%** sqrtp **+** 1**)** **\*** block\_size **-** 1**;**

imin **=** **(**k **-** **(**k **%** sqrtp**))** **/** sqrtp **\*** block\_size**;**

imax **=** **((**k **-** **(**k **%** sqrtp**))** **/** sqrtp **+** 1**)** **\*** block\_size **-** 1**;**

l **=** 0**;**

//给块赋值

**for** **(**int i **=** imin**;** i **<=** imax**;** i**++)** **{**

**for** **(**int j **=** jmin**;** j **<=** jmax**;** j**++)** **{**

//多余部分取0

**if** **(**i **>=** n **||** j **>=** n**)** **{**

a\_temp**[**l**]** **=** 0**;**

b\_temp**[**l**]** **=** 0**;**

l**++;**

**}**

**else** **{**

a\_temp**[**l**]** **=** A**[**i**][**j**];**

b\_temp**[**l**]** **=** B**[**i**][**j**];**

l**++;**

**}**

**}**

**}**

//分发块给每个进程，0号进程直接拷贝自己的块到a、b数组

**if** **(**k **==** 0**)** **{**

**for** **(**int i **=** 0**;** i **<** l**;** i**++)** **{**

a**[**i**]** **=** a\_temp**[**i**];**

b**[**i**]** **=** b\_temp**[**i**];**

**}**

**}**

**else** **{**

MPI\_Send**(**a\_temp**,** block\_large**,** MPI\_DOUBLE**,** k**,** 1**,** MPI\_COMM\_WORLD**);**

MPI\_Send**(**b\_temp**,** block\_large**,** MPI\_DOUBLE**,** k**,** 2**,** MPI\_COMM\_WORLD**);**

**}**

**}**

**}**

**else** **{**

//其余进程直接接收0号进程发来的块

MPI\_Recv**(**a**,** block\_large**,** MPI\_DOUBLE**,** 0**,** 1**,** MPI\_COMM\_WORLD**,** **&**status**);**

MPI\_Recv**(**b**,** block\_large**,** MPI\_DOUBLE**,** 0**,** 2**,** MPI\_COMM\_WORLD**,** **&**status**);**

**}**

//处室对准计算的进程 计算预计发往的进程号 0为左移 1为上移

recv **=** Initial\_Focus**(**p**,** p\_cow**,** p\_col**,** 0**);**

recv1 **=** Initial\_Focus**(**p**,** p\_cow**,** p\_col**,** 1**);**

//左循环i步

//各个进程将自己的a数组发送block\_large的大小，以double形式发送给recv进程，tag为1

//同时自己的a\_temp数组接收表达式长串表示的进程，共计block\_large的double大小，

//tag为1，域为MPI\_COMM\_WORLD，状态码为status

MPI\_Sendrecv**(**a**,** block\_large**,** MPI\_DOUBLE**,** recv**,** 1**,**

a\_temp**,** block\_large**,** MPI\_DOUBLE**,** **((**p\_cow **+** sqrtp**)** **%** sqrtp**)** **\*** sqrtp **+** **(**p\_col **+** p\_cow **+** sqrtp**)** **%** sqrtp**,** 1**,** MPI\_COMM\_WORLD**,** **&**status**);**

//各个进程将自己的temp数组转移到a数组中 推迟做是为了防止死锁

**for** **(**int i **=** 0**;** i **<** block\_large**;** i**++)** **{**

a**[**i**]** **=** a\_temp**[**i**];**

**}**

//上循环j步

//各个进程将自己的b数组发送block\_large的大小，以double形式发送给recv1进程，tag为1

//同时自己的b\_temp数组接收表达式长串表示的进程，共计block\_large的double大小，

//tag为1，域为MPI\_COMM\_WORLD，状态码为status

MPI\_Sendrecv**(**b**,** block\_large**,** MPI\_DOUBLE**,** recv1**,** 1**,**

b\_temp**,** block\_large**,** MPI\_DOUBLE**,** **((**p\_cow **+** p\_col **+** sqrtp**)** **%** sqrtp**)** **\*** sqrtp **+** **(**p\_col **+** sqrtp**)** **%** sqrtp**,** 1**,** MPI\_COMM\_WORLD**,** **&**status**);**

//各个进程将自己的temp数组转移到b数组中 推迟做是为了防止死锁

**for** **(**int i **=** 0**;** i **<** block\_large**;** i**++)** **{**

b**[**i**]** **=** b\_temp**[**i**];**

**}**

//在根号p的循环中，循环计算块内的c矩阵，具体操作是加乘运算

**for** **(**int l **=** 0**;** l **<** sqrtp**;** l**++)** **{**

**for** **(**int i **=** 0**;** i **<** block\_size**;** i**++)** **{**

**for** **(**int j **=** 0**;** j **<** block\_size**;** j**++)** **{**

**for** **(**int k **=** 0**;** k **<** block\_size**;** k**++)** **{**

//每个块把自己的块内的c矩阵算出来

c**[**i **\*** block\_size **+** j**]** **+=** a**[**i **\*** block\_size **+** k**]** **\*** b**[**k **\*** block\_size **+** j**];**

**}**

**}**

**}**

//recv为发送的进程号 recv1为接收的进程号

int recv **=** **((**p\_cow **+** sqrtp**)** **%** sqrtp**)** **\*** sqrtp **+** **(**p\_col **-** 1 **+** sqrtp**)** **%** sqrtp**;**

int recv1 **=** **((**p\_cow **+** sqrtp**)** **%** sqrtp**)** **\*** sqrtp **+** **(**p\_col **+** 1 **+** sqrtp**)** **%** sqrtp**;**

//每个进程将自己的a矩阵右移一位交给recv进程，并且接收recv1的a\_temp矩阵

MPI\_Sendrecv**(**a**,** block\_large**,** MPI\_DOUBLE**,** recv**,** 1**,**

a\_temp**,** block\_large**,** MPI\_DOUBLE**,** recv1**,** 1**,** MPI\_COMM\_WORLD**,** **&**status**);**

//将自己的a矩阵更新为a\_temp矩阵

**for** **(**int i **=** 0**;** i **<** block\_large**;** i**++)** **{**

a**[**i**]** **=** a\_temp**[**i**];**

**}**

//每个进程同理将自己的b矩阵上移一位给表达式进程，接受表达式进程的temp矩阵

MPI\_Sendrecv**(**b**,** block\_large**,** MPI\_DOUBLE**,** **((**p\_cow **-** 1 **+** sqrtp**)** **%** sqrtp**)** **\*** sqrtp **+** **(**p\_col **+** sqrtp**)** **%** sqrtp**,** 1**,**

b\_temp**,** block\_large**,** MPI\_DOUBLE**,** **((**p\_cow **+** 1 **+** sqrtp**)** **%** sqrtp**)** **\*** sqrtp **+** **(**p\_col **+** sqrtp**)** **%** sqrtp**,** 1**,** MPI\_COMM\_WORLD**,** **&**status**);**

//更新

**for** **(**int i **=** 0**;** i **<** block\_large**;** i**++)** **{**

b**[**i**]** **=** b\_temp**[**i**];**

**}**

**}**

//0号进程进行综合，将每个块的计算结果综合起来

**if** **(**rank **==** 0**)** **{**

//将0号进程的块内的c矩阵重新计算到C中

**for** **(**int i **=** 0**;** i **<** block\_size**;** i**++)** **{**

**for** **(**int j **=** 0**;** j **<** block\_size**;** j**++)** **{**

C**[**i**][**j**]** **=** c**[**i **\*** block\_size **+** j**];**

**}**

**}**

//接受其余进程的c矩阵到自己的c矩阵中

**for** **(**int k **=** 1**;** k **<** p**;** k**++)** **{**

int imin**,** jmin**,** imax**,** jmax**,** l **=** 0**,** l2 **=** 0**;**

jmin **=** **(**k **%** sqrtp**)** **\*** block\_size**;**

jmax **=** **(**k **%** sqrtp **+** 1**)** **\*** block\_size **-** 1**;**

imin **=** **(**k **-** **(**k **%** sqrtp**))** **/** sqrtp **\*** block\_size**;**

imax **=** **((**k **-** **(**k **%** sqrtp**))** **/** sqrtp **+** 1**)** **\*** block\_size **-** 1**;**

MPI\_Recv**(**c**,** block\_large**,** MPI\_DOUBLE**,** k**,** 1**,** MPI\_COMM\_WORLD**,** **&**status**);**

//其余进程的c矩阵转换到C结果矩阵中

**for** **(**int i **=** imin**;** i **<=** imax**;** i**++)** **{**

l2 **=** 0**;**

**for** **(**int j **=** jmin**;** j **<=** jmax**;** j**++)** **{**

C**[**i**][**j**]** **=** c**[**l **\*** block\_size **+** l2**];**

l2**++;**

**}**

l**++;**

**}**

**}**

//计算并行时间，总和时间

end **=** MPI\_Wtime**()-**start**;**

sum **+=** end**;**

cout **<<** p **<<** "个进程的时间为：" **<<** end **<<** "秒" **<<** endl**;**

**if(**times**==**4**)**

//比较检查并行结果

compare\_C**(**C**,** Single\_c**);**

**else** **if** **(**times **==** 0**)** **{**

//输出加速比和平均时间

cout **<<** "平均时间为：" **<<** sum **/** 5 **<<** "秒" **<<** endl**;**

cout **<<** "加速比为：" **<<** end1 **/** **(**sum **/** 5**)** **<<** endl**;**

**}**

**}**

**else** **{**

//其余进程发送c矩阵给0号进程

MPI\_Send**(**c**,** block\_large**,** MPI\_DOUBLE**,** 0**,** 1**,** MPI\_COMM\_WORLD**);**

**}**

//删除指针数组节省空间

**delete[]**a\_temp**;**

**delete[]**b\_temp**;**

**delete[]**a**;**

**delete[]**b**;**

**delete[]**c**;**

MPI\_Barrier**(**MPI\_COMM\_WORLD**);**

**}**

MPI\_Finalize**();** // 并行结束

**}**

3. 实验结果和分析：测试并行程序在不同进程数下的执行时间和加速比（串行执行时间/并行执行时间），并分析实验结果。其中，*n*固定为1000，进程数分别取1、4、9、16、25、36、49、64。为了减少误差，每项实验进行5次，取平均值作为实验结果。

表1 并行程序在不同进程数下的执行时间（秒）和加速比

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 线程数  执行时间 | 1 | 4 | 9 | 16 | 25 | 36 | 49 | 64 |
| 第1次 | 2.13 | 0.36 | 0.20 | 0.19 | 0.25 | 0.34 | 0.49 | 0.23 |
| 第2次 | 2.38 | 0.37 | 0.21 | 0.24 | 0.42 | 0.75 | 1.15 | 0.74 |
| 第3次 | 2.22 | 0.39 | 0.21 | 0.25 | 0.46 | 0.81 | 1.21 | 0.93 |
| 第4次 | 2.09 | 0.35 | 0.20 | 0.25 | 0.45 | 0.80 | 1.21 | 0.96 |
| 第5次 | 2.16 | 0.35 | 0.21 | 0.24 | 0.45 | 0.80 | 1.31 | 0.94 |
| 平均值 | 2.20 | 0.36 | 0.21 | 0.23 | 0.41 | 0.70 | 1.07 | 0.76 |
| 加速比 | 1.13 | 6.06 | 9.33 | 8.21 | 4.61 | 2.50 | 1.86 | 2.56 |

### 实验结果分析

1. 根据数据结果可以看出，多个处理器并行的cannon算法比串行处理快得多，串行处理时间复杂度为O（n3），而并行的cannon算法有三个阶段：初始对准，结果矩阵的初始化以及移位乘加运算。所以时间复杂度为O（n3/P）
2. 由于本电脑cpu数目为16，所以加速比到16进程时已经到了顶峰开始下降，后续进程加速效果越来越不明显，还有一个原因是因为本题n=1000，可能在计算的时候进程数目一多，步长过小，容易出现空间局部性的问题。
3. 正常情况下，根据时间复杂度，并行的时间应该是串行的p倍，多进程后时间表达式中T=n3/P+2\*,其中第一项是计算的时间， 第二项是对齐循环移位时间，第三项是单步移位时间。