多核平台下的并行计算课程实验报告

班级： 15级1班

学号： 201500301125

姓名： 马嫣然

1.**填写下表，给出所使用的软硬件环境参数**

|  |  |
| --- | --- |
| 操作系统版本 | Ubuntu 16.04 |
| 并行编程模型 | MPICH-3.2 |
| 编译器版本 | gcc 5.4.0 |
| CPU 型号、主频及核数 | Intel(R) Core(TM) i5-4200H CPU @ 2.80GHz |
| 内存型号，大小及主频 | FPM EDO DIMM SDRAM  1024 |

1. **综合题目 1 和题目 2，在提交的报告中需要给出至少四个如下表 格记录计算时间和加速比：（也可以自己设计图表描述不同线程数 时程序的运行时间及加速比）**

3000X3000（使用常数 d）

|  |  |  |
| --- | --- | --- |
| 线程数目 | 运行时间（秒） | 加速比 |
| 1 | 334.08 | 461.95/334.08=1.382 |
| 2 | 269.34 | 461.95/269.34=1.715 |
| 4 | 223.69 | 461.95/223.69=2.065 |
| 8 | 309.13 | 461.95/309.13=1.494 |
| 16 | 652.20 | 461.95/652.20=0.708 |
| …… | …… | …… |

3000X3000（使用（i-k）2和（j-k）2）

|  |  |  |
| --- | --- | --- |
| 线程数目 | 运行时间（秒） | 加速比 |
| 1 | 303.99 | 360.81/303.99=1.187 |
| 2 | 289.15 | 360.81/289.15=1.248 |
| 4 | 268.84 | 360.81/268.84=1.342 |
| 8 | 380.04 | 360.81/380.04=0.949 |
| 16 | 778.66 | 360.81/778.66=0.463 |
| …… | …… | …… |

5000X5000（使用常数 d）

|  |  |  |
| --- | --- | --- |
| 线程数目 | 运行时间（秒） | 加速比 |
| 1 | 1818.32 | 1947.51/1818.32=1.071 |
| 2 | 3685.13 | 1947.51/3685.13=0.529 |
| 4 | 3291.24 | 1947.51/3291.24=0.592 |
| 8 | 7441.92 | 1947.51/7441.92=0.262 |
| 16 | 时间太长，死机 | 根据规律，低于0.262 |
| …… | …… | …… |

5000X5000（使用（i-k）2和（j-k）2）

|  |  |  |
| --- | --- | --- |
| 线程数目 | 运行时间（秒） | 加速比 |
| 1 | 1778.46 | 4991.53/1778.46=2.807 |
| 2 | 3791.12 | 4991.53/3791.12=1.317 |
| 4 | 3323.69 | 4991.53/3323.69=1.502 |
| 8 | 8241.85 | 4991.53/8241.85=0.606 |
| 16 | 时间太长 | 根据规律，低于0.606 |
| …… | …… | …… |

1. **并行算法设计和优化思路**
2. **算法设计过程**
3. **首先写出串行实现的代码**

基本是十分简单的逐行计算

int one(int i,int j,int \*\*M,int S[26][26],char \*X,char \*Y);

计算M(i-1,j-1)+S(Xi,Yj)

int two(int i,int j,int \*\*M,int d);

计算M(k,j)+d

int three(int i,int j,int \*\*M,int d);

计算M(i,k)+d

int max(int i,int j,int \*\*M,int S[26][26],int d,char \*X,char \*Y);

计算 M(i,j)

随后逐行逐列计算M(i,j)

1. **改成并行**

**算法思路：**由于矩阵中每一个数都依赖于它左、右和斜上方的值，斜向行列之间的关联却不强，由此可以设计出一个算法。

将矩阵M分成适当的小块，当第（0，0）块计算完成时，第（1，0），（0，1）块就变得可以计算，这两块结束后，（2，0），（1，1），（0，2）变得可以计算。

将这个思路延续下去，我们利用多开的核来计算斜方向上互不相干的数据块，再利用MPI的消息传递汇拢至0号进程，循环往复，最终实现程序的并行。

1. **Block类**

用c语言写的代码不能创建class 将文件改为c++语言即可使用class类

同时发现mpicc不能编译c++ 则将makefile中编译指令改为mpicxx

Block类中有参数int row\_begin，row\_end，col\_begin，col\_end分别代表所分成小块的行数起止和列数起止，int\* block代表用一维数组存放的数据块，int\* calculate（）方法用来计算数据。

1. **计算过程**

a.算第一小块(0,0),每个进程都具有的第一步

b.计算左上分块，即斜对角线以上的所有块。利用循环，每次调用几个进程计算一条斜线列，将结果返回到0进程，并由0号进程将数据安放至M内

c.计算右下分块，即斜对角线以下的所有块，循环同b

例：block\_x=block\_line-rank;

block\_y=rank;

local\_block=blocks[block\_x][block\_y];

上述三行代码是过程b中用于确定分块位置与进程关系的，即本进程应该被分配到哪一块数据块。

d.分块为正方形，可能矩阵M中会有没有分成块的剩余行列，由进程0在最后单独计算，完成矩阵M。

**5.获取命令行参数dimension 和M 的行号 row\_number**

条件：int main (int argc, char \*\*argv)中 argv储存了命令行输入的参数，argv[0]是./main命令本身，argv[1]为dimension，argv[2]为row\_number

Atoi（）用于将字符串转为整型数

1.获取dimension

dimension=atoi(argv[1]);

dimensionx=dimension+1; //实际的矩阵比输入的行列大一圈

2.获取row\_number

row\_number=atoi(argv[2]);

1. **消息传递**

a.MPI\_Bcast(M\_storage,LX\*LY,MPI\_INT,0,MPI\_COMM\_WORLD);

用于将0号进程内的M矩阵散播给各个进程，更新它们的数据

b.MPI\_Barrier(MPI\_COMM\_WORLD);

每个进程运行到此处都会停下，直到所有进程都调用到了这个方法，用于同步进程。

c.点对点通信

if(rank!=0){

MPI\_Send(&row\_begin,1,MPI\_INT,0,1,MPI\_COMM\_WORLD);

MPI\_Send(&col\_begin,1,MPI\_INT,0,2,MPI\_COMM\_WORLD);

MPI\_Send(bloc,length\*length,MPI\_INT,0,3,MPI\_COMM\_WORLD); }

除0号之外的进程将本身计算完毕的数据块发送给0

for(int c=1;c<=block\_line;c++){

MPI\_Recv(&row\_begin,1,MPI\_INT,c,1,MPI\_COMM\_WORLD,MPI\_STATUS\_IGNORE); MPI\_Recv(&col\_begin,1,MPI\_INT,c,2,MPI\_COMM\_WORLD,MPI\_STATUS\_IGNORE); MPI\_Recv(bloc,length\*length,MPI\_INT,c,3,MPI\_COMM\_WORLD,MPI\_STATUS\_IGNORE);}

0号进程接收每一个进程的数据并放入M内

1. **测试时间**

单独创建time.cc，用于测试main.cc的运行速度0

使用MPI\_Wtime()方法记录时间，并调用system（）运行待测试程序

例：./time 3 main 200 7

./time是测试时间的程序，3是运行的核数，main为并行计算程序，200是维度数dimension，7为希望输出的行数row\_number

将会输出程序运行时间

1. **优化**
2. **代码中**

由于类Block中int\* calculate（）方法内计算数据块时已经随时更新了矩阵M中的数值，所以在进程0中就不必将自己bloc块中的数字复制到M上去了，只需要接收从其他进程传来的bloc数组，将他们的数据放进M里

b.

1. **算法上**

广播矩阵M

由于M是个二维数组，每一行的地址是不连续的，所以没办法用一个MPI\_Bcast()就广播出去。

int\* M\_storage=new int[dimensionx\*dimensionx];

int\*\* M=(int\*\*)malloc(sizeof(int\*)\*dimensionx);

for(i=0;i<dimensionx;i++)

M[i]=&M\_storage[i\*dimensionx];

如上，指定一个一维数组M\_storage,一个二维数组M，将二者的地址联系起来，如此M的行与行之间就连续了。

M\_storage专门用来广播，M用来作二维数组调用

1. **编译时**

-O2 提供更加高级的代码优化,会占用更长的编译时间

在Makefile的编译语句中加入-O2，注意O为大写

经测试，的确提升了程序运行速度

1. **说明**

并行计算

在“源程序“文件夹中有”题目1并行版本“，”题目1串行版本“，”题目2并行版本“，”题目2串行版本“等文件夹

每个文件夹中都有相应的源程序代码和 Makefile,ReadMe等，里面有单独的执行命令

执行命令在此总览如下

main为计算程序

time为测试运行时间程序

使用make命令编译

以下为命令行指令

并行

mpirun -np 核数 ./main dimension row\_number

例： mpirun -np 3 ./main 7 5

意为：用3个核进行计算，执行main程序，维度为7,输出第5行的结果

检测并行计算时间

./time 核数 main dimension row\_number

例： ./time 3 main 200 7

意为：用3个核进行计算，执行main程序，维度为200,输出第7行的结果，并输出程序运行时间

串行

./mainchuanxing dimension row\_number

例： ./mainchuanxing 200 7

意为：执行main程序，维度为200,输出第7行的结果

检测串行时间

./timechuanxing mainchuanxing dimension row\_number

例： ./timechuanxing mainchuanxing 200 7

意为：执行main程序，维度为200,输出第7行的结果，并输出程序运行时间