

# 并行计算报告

学院： 中电十五所

姓名： 王 俊 丰

学号： P17206006

## 目录

一、	稀疏矩阵求解 (cg)	3
1.	题目及要求	3
2.	实验结果及截图	3
	2.1 实验结果	3
	2.2 实验截图	4
3.	实验结果分析	4
二、	分形算法 Mandelbrot 并行化	5
1.	题目及要求	5
2.	实验结果及截图	5
	2.1 实验结果	5
	2.2 实验截图	5
3.	实验结果分析	7
三、	Reduction 并行程序转化	7
1.	题目及要求	7
2.	实验结果截图	7
	2.1 实验结果	7
	2.1 实验截图	7
四、	并行 I/O 编写	7
1.	题目及要求	7
2.	实验结果及截图	8
	2.1 实验结果	8
	2.2 实验截图	8
五、	实验感想	8
1.	并行计算	8
2.	自我认知	8

## 一、稀疏矩阵求解（cg）

### 1. 题目及要求

实现一个“大规模稀疏矩阵的Conjugate Gradient求解器”，即，求解 $Ax=b$ 中的 $x$ ，其中 $A$ 为一个大型、稀疏矩阵。

算法如下：

假定我们选取  $x_0$  作为  $Ax=b$  的解的初始猜测值，与之对应的残差为  $r_0=b-Ax_0$ 。初始  $p_0=r_0$ ，循环变量  $k=0$ 。然后开始如下循环：

Repeat

$$\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$$

$$x_{k+1} = x_k + \alpha_k p_k$$

$$r_{k+1} = r_k - \alpha_k A p_k$$

如果残差已经足够小，则退出循环，否则继续

$$\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$$

$$p_{k+1} = r_{k+1} + \beta_k p_k$$

$$k = k + 1$$

End repeat

要求：任选 MPI 或 OpenMP 实现这个算法。首先给出串行算法，并比较串行，1，2，4，8CPU（核）下运行的加速性和扩展性，加以分析。

### 2. 实验结果及截图

#### 2.1 实验结果

	串行	1 核	2 核	4 核	8 核
迭代次数	755	755	779	737	755
求解 x 用时 单位：毫秒	300.00	300.00	470.00	660.00	1570.00
程序总用时 单位：毫秒	155.285	156.660	199.999	83.817	99.430

【备注】在记录求解矩阵  $x$  时的用时函数是用的 `clock（）` 函数。而在计算程序总用时，用的是 `get_wtime（）` 函数。

## 2.2 实验截图

```
x[0] = 0.240192
x[1] = 0.461421
x[2] = 0.202172
x[3] = 0.129867
x[4] = 0.084243
x[5] = 0.075611
x[6] = 0.083878
x[7] = 0.096768
x[8] = 0.106582
x[9] = 0.109722
Total iteration times: 755
The execute time is 300.000000 ms.
all_use_time 155.285107 ms.
```

【串行】

```
x[0] = 0.240192
x[1] = 0.461421
x[2] = 0.202172
x[3] = 0.129867
x[4] = 0.084243
x[5] = 0.075611
x[6] = 0.083878
x[7] = 0.096768
x[8] = 0.106582
x[9] = 0.109722
Total iteration times: 755
The execute time is 300.000000 ms.
all_use_time 156.660963 ms.
```

【1 核】

```
x[0] = 0.165213
x[1] = 0.322166
x[2] = 0.211512
x[3] = 0.166607
x[4] = 0.130710
x[5] = 0.110294
x[6] = 0.099438
x[7] = 0.093822
x[8] = 0.090260
x[9] = 0.086800
Total iteration times: 779
The execute time is 470.000000 ms.
all_use_time 119.999205 ms.
```

【2 核】

```
x[0] = 0.157278
x[1] = 0.307233
x[2] = 0.214952
x[3] = 0.171967
x[4] = 0.136043
x[5] = 0.113231
x[6] = 0.099595
x[7] = 0.091792
x[8] = 0.087080
x[9] = 0.083536
Total iteration times: 737
The execute time is 660.000000 ms.
all_use_time 83.817385 ms.
```

【4 核】

```
x[0] = 0.155257
x[1] = 0.303514
x[2] = 0.214486
x[3] = 0.172482
x[4] = 0.137121
x[5] = 0.114336
x[6] = 0.100434
x[7] = 0.092236
x[8] = 0.087120
x[9] = 0.083240
Total iteration times: 755
The execute time is 1570.000000 ms.
all_use_time 99.430682 ms.
```

【8 核】

## 3. 实验结果分析

从实验结果不难看出，串行计算的程序总用时是最多的，而并行计算用时比串行少。并且，随着参与计算核数的增加，程序总用时就会减少，到 8 核时，程序总用时在 5 个对比中达到了最小。而在表中可以看出，随着核数增加，在求解  $x$  时候用时，在数值上呈现递增趋势，而且时间在数值上一直都大于程序总用时。

造成这一现象是由于计时函数不同导致的，首先，介绍一下两个记录时间函数：

(1) clock()函数：

clock()函数返回的是处理器执行的时间，也就是说，只要内核中有程序在 cpu 中运行，时间就会增加，使用多核并行化技术并不能并行地计算使用的时间，仍然是进行叠加。clock（）函数返回值是返回时钟周期，要转化成时间需要除以常量宏 CLOCKS\_PER\_SEC，这样就能得到一个相对时间。

(2) get\_wtime()函数:

返回的是一个观察点的时间值,这个时间将一直在程序运行时持续,也就是说,在程序运行前使用和程序运行结束使用,这个时间差是整个程序运行的时间,而不是 clock 得到的 cpu 运行的时间。返回值返回一个绝对时间值,单位是秒,项目中通过简单转化输出的是毫秒。

由上述对两个函数的介绍可以看出,在同一个程序中,求解 x 时候的用时在显示上,都大于程总用时,原因就在于此,求解 x 时候,因为用了多核处理,因此用 clock 函数计算时间时加上了各个 cpu 的处理时间,所以就显得计算用时比程序总用时多。

由本次实验就可以看出,多核处理在解决这类的计算问题时,能够大大提升计算的时间,而且,在一定范围内,核数越多,计算时间越快。

## 二、分形算法 Mandelbrot 并行化

### 1. 题目及要求

将给出分形算法用 OpenMP 并行化,并比较和分析 1, 2, 4, 8 CPU (核)的扩展性和加速性。(代码略)

### 2. 实验结果及截图

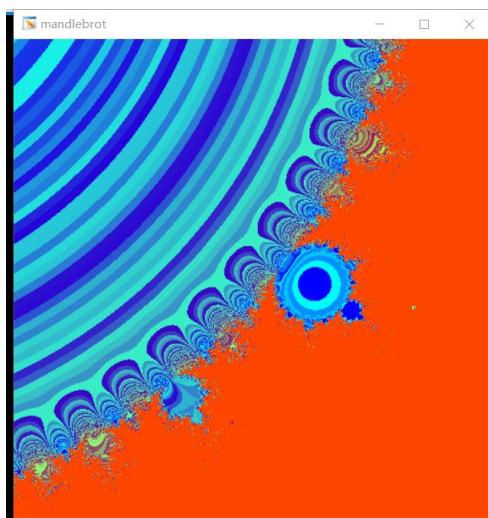
#### 2.1 实验结果

	1 核	2 核	3 核	4 核
用时 (ms)	1893.88	1804.23	1631.41	1585.28

#### 2.2 实验截图

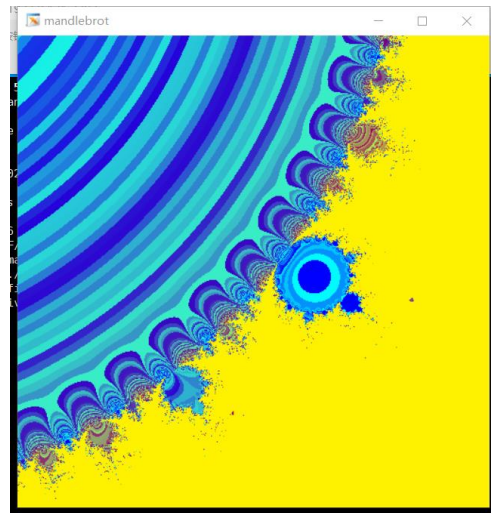
1 核:

```
libGL error: failed to
use_time: 1893.88 ms
```



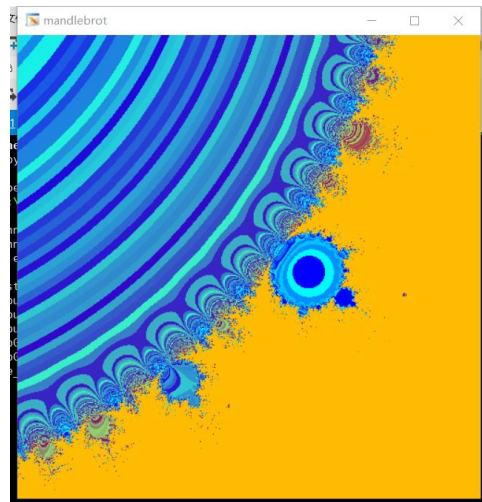
2 核:

```
libGL error: failed to
use_time: 1804.23 ms
```



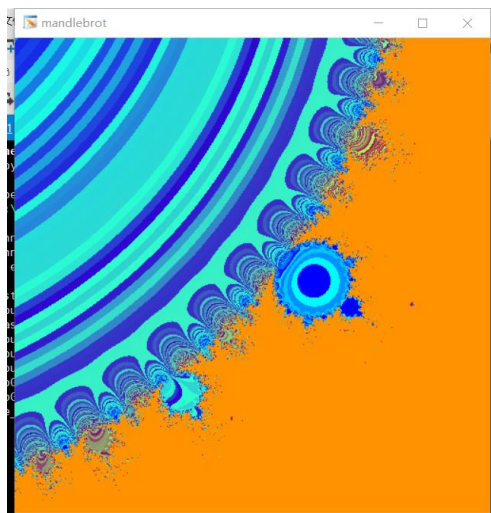
4 核:

```
libGL error: No match
libGL error: failed to
use_time: 1631.41 ms
```



8 核:

```
libGL error: No match
libGL error: failed to
use_time: 1585.28 ms
```



### 3. 实验结果分析

使用 OpenMP 指令的一个优点是将并行性和算法分离，阅读代码时候无需考虑并行化是如何实现的。

从实验结果可以看出，随着核数增加，项目运行速度也大大加快，说明在一定范围内，核数越多，并行化后计算的速度就会越快。同时，从获得图片可以看出，核数不同，绘图着色也有所不同。

最后就是关于绘图着色的问题，大部分都是通过让迭代次数来决定所绘制的颜色，而不同核数处理问题时候，计算时间不一样，同时需要迭代的次数也不一样，所绘制出来的图颜色也有区别。

## 三、Reduction 并行程序转化

### 1. 题目及要求

读懂并将串行算法 Reduction.cpp 改成 mpi 并行。

用 Random\_matrix.py 生成随机矩阵。

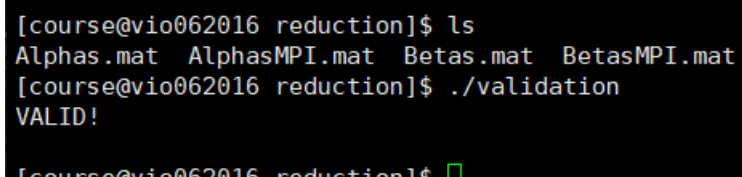
用 Validationg\_mpi.cpp 验证结果。

### 2. 实验结果截图

#### 2.1 实验结果

修改成功，输出 VALID，结果如下图。

#### 2.1 实验截图



```
[course@vio062016 reduction]$ ls
Alphas.mat  AlphasMPI.mat  Betas.mat  BetasMPI.mat
[course@vio062016 reduction]$ ./validation
VALID!
[course@vio062016 reduction]$
```

## 四、并行 I/O 编写

### 1. 题目及要求

用 MPI 编写一个并行 I/O 例子。

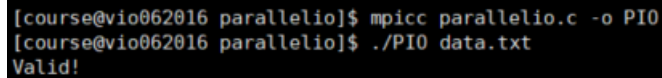
具体要求在作业附件【parallelio.docx】中有，此处略。

## 2. 实验结果及截图

### 2.1 实验结果

编写成功，输出 Valid，截图如下。

### 2.2 实验截图



```
[course@vio062016 parallelio]$ mpicc parallelio.c -o PIO
[course@vio062016 parallelio]$ ./PIO data.txt
Valid!
```

## 五、实验感想

### 1. 并行计算

通过这次实验，对并行计算的认知更深，或许在小数据中，并行计算的优点并不能很好的体现出来，但是当数据越来越多，计算越来越繁杂的时候，并行计算就能比普通的运算快很多，能够成倍的节省计算时间，为科研和生产提高了效率。

### 2. 自我认知

由于专业方向原因，我对并行计算的接触不多，可以说是通过本学期的学习和了解刚刚入门，对并行计算有了一些粗浅的了解。但是还是感觉有点迷茫，毕竟刚刚接触，在编写代码的时候感觉很不会。所以在做 4 个题目时候，都请教了其他同学，特别是在做第一题稀疏矩阵求解和第四题并行 I/O 编写时，很多都是同学一点点教的。因此代码上会有很多相似的地方，如有雷同，敬请谅解。

通过这次实验，深深知道自己的不足，希望在以后的学习中，能够学到更多知识，弥补自己的缺点，让自己的能力有更多的提升。