



高性能计算

学生姓名： 杨杰

班 级： 111172

学 号： 20171002157

指导教师： 万林

中国地质大学地理与信息工程学院

2019 年 10 月

目录

1	引言	3
1.1	项目背景	3
1.1.1	高性能计算	3
1.1.2	Fortran	3
2	任务概述	4
2.1	总体任务概述	4
2.2	具体任务	4
2.2.1	稳态热传导	4
3	方法概述	5
3.1	迭代法 (Iterative Method)	5
3.1.1	雅可比法 (Jacobi Method)	5
3.1.2	高斯-赛德尔迭代 (Gauss - Seidel method)	6
3.1.3	SOR 逐次超松弛法 (Successive Over Relaxation)	7
3.2	方法比较	7
4	实现过程	8
4.1	实验环境等配置	8
4.1.1	虚拟机配置	8
4.1.2	系统环境配置	9
4.2	实验过程	9
4.2.1	环境、库等安装或更新	9
4.2.2	实现过程	10
4.2.2.1	雅可比法 (Jacobi Method)	10
4.2.2.2	高斯-赛德尔迭代 (Gauss - Seidel method)	10
4.2.2.3	SOR 逐次超松弛迭代法 (Successive Over Relaxation)	11
4.2.3	实验结果	12
4.2.3.1	雅可比法 (Jacobi Method)	12
4.2.3.2	高斯-赛德尔迭代 (Gauss - Seidel method)	13
4.2.3.3	SOR 逐次超松弛迭代法 (Successive Over Relaxation)	13
5	总结	14

1 引言

1.1 项目背景

1.1.1 高性能计算

高性能计算(High performance computing, 缩写 HPC)指通常使用很多处理器(作为单个机器的一部分)或者某一集群中组织的几台计算机(作为单个计算资源操作)的计算系统和环境。有许多类型的 HPC 系统,其范围从标准计算机的大型集群,到高度专用的硬件。大多数基于集群的 HPC 系统使用高性能网络互连,比如那些来自 InfiniBand 或 Myrinet 的网络互连。基本的网络拓扑和组织可以使用一个简单的总线拓扑,在性能很高的环境中,网状网络系统在主机之间提供较短的潜伏期,所以可改善总体网络性能和传输速率。

1.1.2 Fortran

Fortran 源自于“公式翻译”(英语: FormulaTranslation)的缩写,是一种编程语言。它是世界上最早出现的计算机高级程序设计语言,广泛应用于科学和工程计算领域。FORTRAN 的特性主要包括:

- Fortran 语言的最大特性是接近数学公式的自然描述,在计算机里具有很高的执行效率。
- 易学,语法严谨。
- 可以直接对矩阵和复数进行运算,这点 Matlab 有继承。
- 自诞生以来广泛地应用于数值计算领域,积累了大量高效而可靠的源程序。
- 很多专用的大型数值运算计算机针对 Fortran 做了优化。
- 广泛地应用于并行计算和高性能计算领域。
- Fortran 90, Fortran 95, Fortran 2003 的相继推出使 Fortran 语言具备了现代高级编程语言的一些特性。
- 其矩阵元素在记忆空间存储顺序是采用列优先(Column major),Matlab 也承袭这点,当前最多使用的 C 语言则采用行优先 (Row major)。

2 任务概述

2.1 总体任务概述

线性系统的并行迭代方法：使用并行的“Jacobi, Gauss-Seidel 和 SOR 方法”解决“稳态热传导”问题。

2.2 具体任务

2.2.1 稳态热传导

传热的方式主要有三种：导热、对流、辐射。

由于物体内部分子、原子和电子等微观粒子的热运动，而组成物体的物质并不发生宏观的位移，将热量从高温区传到低温区的过程称为导热。

在导热过程中，如果温度不随时间发生变化，则认为是稳态导热，否则为非稳态导热。^[1] 在稳态导热过程中，对于每一个物质元，流入和流出的热量均相等，所以又叫做热平衡。工程上许多的导热现象，可以归结为温度仅沿一个方向变化，而且与时间无关的一维稳态导热现象。例如，通过房屋墙壁和长热力管道管壁的导热等。

热传导方程（或称热方程）是一个重要的偏微分方程，它描述一个区域内的温度如何随时间变化。

热传导在三维的各向同性介质里的传播可用以下方程表达：

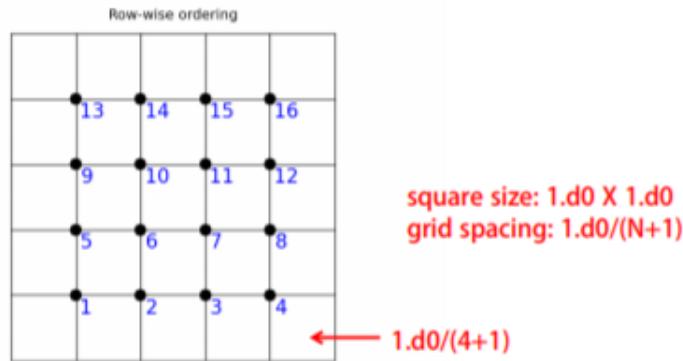
$$\frac{\partial u}{\partial t} = \text{div}(Uu) = k \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right) = k(u_{xx} + u_{yy} + u_{zz})$$

其中：

- $u=u(t, x, y, z)$ 表温度，它是时间变数 t 与空间变数 (x, y, z) 的函数。
- $\alpha u / \alpha t$ 是空间中一点的温度对时间的变化率。
- u_{xx}, u_{yy} 与 u_{zz} 与温度对三个空间坐标轴的二次导数。
- k 是热扩散率，决定于材料的热传导率、密度与热容。

热方程是傅里叶冷却律的一个推论（详见条目热传导）。如果考虑的介质不是整个空间，则为了得到方程的唯一解，必须指定 u 的边界条件。如果介质是整个空间，为了得到唯一性，必须假定解的增长速度有个指数型的上界，此假定吻合实验结果。

热方程的解具有将初始温度平滑化的特质，这代表热从高温处向低温处传播。一般而言，许多不同的初始状态会趋向同一个稳态（热平衡）。因此我们很难从现存的热分布反解初始状态，即使对极短的时间间隔也一样。



假设边界上的每个点的温度都是固定的（和已知的）在内部点，稳态值（近似）是 4 个相邻值的平均值。

3 方法概述

3.1 迭代法（Iterative Method）

迭代法（Iterative Method），在计算数学中，迭代是通过从一个初始估计出发寻找一系列近似解来解决问题（一般是解方程或者方程组）的数学过程，为实现这一过程所使用的方法统称。

跟迭代法相对应的是直接法（或者称为一次解法），即一次性解决问题，例如通过开方解方程 $x^2=4$ 。一般如果可能，直接解法总是优先考虑的。但当遇到复杂问题时，特别是在未知量很多，方程为非线性时，我们无法找到直接解法（例如五次以及更高次的代数方程没有解析解，参见阿贝尔定理），这时候或许可以通过迭代法寻求方程（组）的近似解。

3.1.1 雅可比法（Jacobi Method）

在数值线性代数中，雅可比法（Jacobi Method）是一种解对角元素几乎都是各行和各列的绝对值最大的值的线性方程组的算法。求解出每个对角元素并插入近似值。不断迭代直至收敛^[1]。这个算法是雅可比矩阵的精简版。方法的名字来源于德国数学家卡尔·雅可比。

可以用来求解协方差矩阵的特征值和特征向量。雅可比方法求全积分的一种方法，把拉格朗日查皮特方法推广到求 n 个自变量一阶非线性方程的全积分的方法称为雅可比方法。雅可比迭代法的计算公式简单，每迭代一次只需计算一次矩阵和向量的乘法，且计算过程中原始矩阵 A 始终不变，比较容易并行计算。考虑线性方程组 $Ax=b$ 时，一般当 A 为低阶稠密矩阵时，用主元消去法解此方程组是有效方法。但是，对于由工程技术中产生的大型稀疏矩阵方程组（ A 的阶数很高，但零元素较多，例如求某些偏微分方程数值解所产生的线性方程组），利用迭代法求解此方程组就是合适的，在计算机内存和运算两方面，迭代法通常都可利用 A 中有大量零元素的特点。雅可比迭代法就是众多迭代法中比较早且较简单的一种，其命名也是为纪念普鲁士著名数学家雅可比。

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n \end{cases} \quad \begin{aligned} Ax &= b \\ \Leftrightarrow x &= Mx + g \end{aligned}$$

(每一个 x 的下标 i 表示第 i 个 x 值, 上标 $k+1$ 表示迭代到了 $k+1$ 次)

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^n a_{ij} x_j^{(k)} \right) \quad (i=1, 2, \dots, n)$$

高斯-赛德尔迭代 (Gauss-Seidel method) 是数值线性代数中的一个迭代法, 可用来求出线性方程组解的近似值。该方法以卡尔·弗里德里希·高斯和路德维希·赛德尔命名。同雅可比法一样, 高斯-赛德尔迭代是基于矩阵分解原理。

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n = b_i$$

$$(i = 1, 2, \dots, n),$$
$$x_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^n a_{ij} x_j^{(k)} \right) / a_{ii}$$

$$(i = 1, 2, \dots, n; k = 0, 1, 2, \dots)$$

($u(i, j)$ 为内部点的温度, $[k]$ 为第 k 次迭代的值)

$$u_{i,j}^{[k+1]} = \frac{1}{4} \left(u_{i-1,j}^{[k+1]} + u_{i+1,j}^{[k+1]} + u_{i,j-1}^{[k+1]} + u_{i,j+1}^{[k+1]} \right)$$

3.1.3 SOR 逐次超松弛法 (Successive Over Relaxation)

D. M. Young 于 20 世纪 70 提出逐次超松弛 (Successive Over Relaxation) 迭代法, 简称 SOR 方法, 是一种经典的迭代算法。它是为了解决大规模系统的线性等式提出来的, 在 GS 法基础上为提高收敛速度, 采用加权平均而得到的新算法。由于超松弛迭代法公式简单, 编制程序容易, 很多工程学、计算数学中都会应用超松弛迭代方法。使用超松弛迭代法的关键在于选取合适的松弛因子, 如果松弛因子选取合适, 则会大大缩短计算时间。

一种经典的迭代算法, 是为了解决大规模系统的线性等式提出来的, 在 GS 法基础上为提高收敛速度, 采用加权平均而得到的新算法。由于超松弛迭代法公式简单, 编制程序容易, 很多工程学、计算数学中都会应用超松弛迭代方法。使用超松弛迭代法的关键在于选取合适的松弛因子, 如果松弛因子选取合适, 则会大大缩短计算时间。

结合稳态热传导的问题, 定义为:

($u(i, j)$) 为内部点的温度, [k] 为第 k 次迭代的值)

$$u_{i,j}^{[k+1]} = \frac{1}{4} \left(u_{i-1,j}^{[k+1]} + u_{i+1,j}^{[k+1]} + u_{i,j-1}^{[k+1]} + u_{i,j+1}^{[k+1]} \right)$$

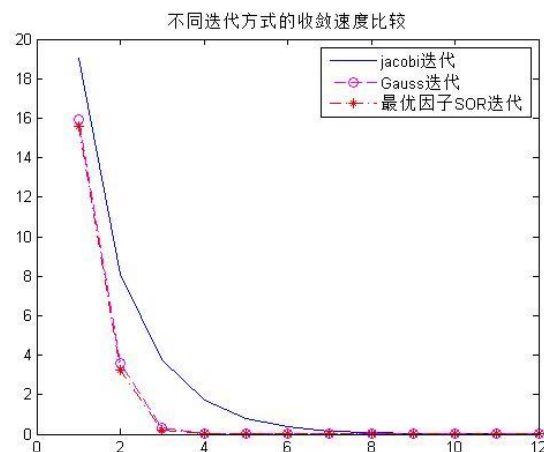
$$u_{i,j}^{[k+1]} = (1-\omega) u_{i,j}^{[k]} + \omega u_{i,j}^{[k+1]}$$

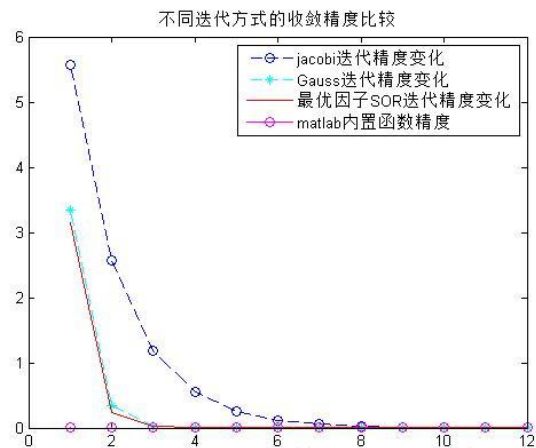
3.2 方法比较

Jacobi 迭代比起 Gauss-Seidel 和 SOR 方法没有更多优势, 特别是在速度方面。但是良好的并行性质使得它在解方程组当中有了一席之地。

Gauss-Seidel 迭代其实是 Jacobi 迭代的一种改进, 在计算 x 的每一分量时, 有新值不用旧值, 使得迭代的速度有了一个质的飞跃。

SOR 逐次超松弛迭代方法, 和 Gauss-Seidel 迭代相比, 在有新值的时候不用旧值, 但也不用新值, 取二者的加权平均, 当权重取得比较合适时, 在三者当中是最优的。

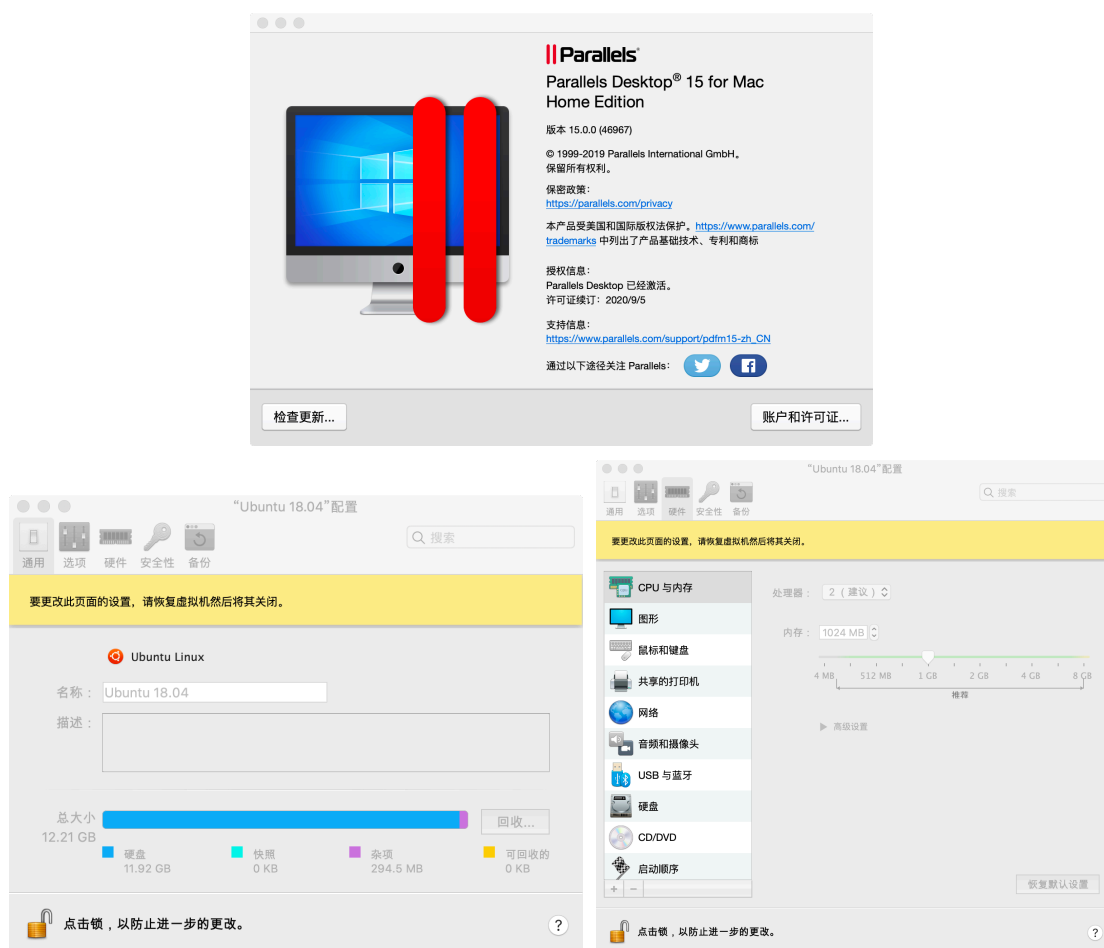




4 实现过程

4.1 实验环境等配置

4.1.1 虚拟机配置



4. 1. 2 系统环境配置

```
parallels@parallels-Parallels-Virtual-Platform: ~  
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)  
parallels@parallels-Parallels-Virtual-Platform:~$ python  
Python 2.7.15+ (default, Oct 7 2019, 17:39:04)  
[GCC 7.4.0] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
>>> exit()  
parallels@parallels-Parallels-Virtual-Platform:~$ gfortran -v  
Using built-in specs.  
COLLECT_GCC=gfortran  
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/7/lto-wrapper  
OFFLOAD_TARGET_NAMES=nvptx-none  
OFFLOAD_TARGET_DEFAULT=1  
Target: x86_64-linux-gnu  
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 7.4.0-1ubuntu1~18.04.1' --with-bugurl=file:///usr/share/doc/gcc-7/README.Bugs --enable-languages=c,ada,c++,go,brig,d,fortran,objc,obj-c++ --prefix=/usr --with-gcc-major-version-only --program-suffix=-7 --program-prefix=x86_64-linux-gnu- --enable-shared --enable-linker-build-id --libexecdir=/usr/lib --without-included-gettext --enable-threads=posix --libdir=/usr/lib --enable-nls --with-sysroot=/ --enable-clocale=gnu --enable-libstdcxx-debug --enable-libstdcxx-time=yes --with-default-libstdcxx-abi=new --enable-gnu-unique-object --disable-vtable-verify --enable-libmpx --enable-plugin --enable-default-pie --with-system-zlib --with-target-system-zlib --enable-objc-gc=auto --enable-multiarch --disable-werror --with-arch-32=i686 --with-abi=m64 --with-multilib-list=m32,m64,mx32 --enable-multilib --with-tune=generic --enable-offload-targets=nvptx-none --without-cuda-driver --enable-checking=release --build=x86_64-linux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu  
Thread model: posix  
gcc version 7.4.0 (Ubuntu 7.4.0-1ubuntu1~18.04.1)  
parallels@parallels-Parallels-Virtual-Platform:~$
```

4. 2 实验过程

4. 2. 1 环境、库等安装或更新

```
$ sudo apt update
```

```
$ sudo apt upgrade
```

```
$ sudo apt install gfortran
```

```
$ sudo apt install openmpi-bin
```

```
$ sudo apt install liblapack-dev
```

```
$ sudo apt install make
```

```
$ sudo apt install python-numpy
```

```
$ sudo apt install python-matplotlib
```

```
$ sudo apt install python-setuptools
```

4.2.2 实现过程

4.2.2.1 雅可比法 (Jacobi Method)

```
do j=1,n

    do i=1,n

        u(i,j) = 0.25d0*(uold(i-1,j) + uold(i+1,j) + &

            uold(i,j-1) + uold(i,j+1) + h**2*f(i,j))

        dumax = max(dumax, abs(u(i,j)-uold(i,j)))

    enddo

enddo
```

4.2.2.2 高斯-赛德尔迭代 (Gauss - Seidel method)

一般方法:

与雅可比法相比, 将每次迭代 “uold” 换成 “u”, 在计算 x 的每一分量时, 有新值不用旧值, 使得迭代的速度有了一个质的飞跃。

```
do j=1,n

    do i=1,n

        u(i,j) = 0.25d0*(u(i-1,j) + u(i+1,j) + &

            u(i,j-1) + u(i,j+1) + h**2*f(i,j))

        dumax = max(dumax, abs(u(i,j)-uold(i,j)))

    enddo

enddo
```

改进方法:

与一般方法相比,拆分为 $(\text{MOD}(i+j, 2) \text{ .eq. } 0)$ 和 $(\text{MOD}(i+j, 2) \text{ .eq. } 1)$,优化在对数据存取方面对处理,在一般方法上提速。

```
do j=1,n
  do i=1,n
    if(MOD(i+j,2) .eq. 0) then
      u(i, j) = 0.25d0*(u(i-1, j) + u(i+1, j) + &
        u(i, j-1) + u(i, j+1) + h**2*f(i, j))
      dumax = max(dumax, abs(u(i, j)-uold(i, j)))
    end if
  enddo
enddo

do j=1,n
  do i=1,n
    if(MOD(i+j,2) .eq. 1) then
      u(i, j) = 0.25d0*(u(i-1, j) + u(i+1, j) + &
        u(i, j-1) + u(i, j+1) + h**2*f(i, j))
      dumax = max(dumax, abs(u(i, j)-uold(i, j)))
    end if
  enddo
enddo
```

4.2.2.3 SOR 逐次超松弛迭代法(Successive Over Relaxation)

一般方法:

在有新值的时候不用旧值,不用新值,用二者的加权平均,当权重取得比较合适时,在三者当中是最优的。

```
do j=1,n
  do i=1,n
    u(i, j)=uold(i, j)+1.95d0*(0.25d0*(u(i-1, j) + u(i+1, j) + &
      u(i, j-1) + u(i, j+1) + h**2*f(i, j))-uold(i, j))
    dumax = max(dumax, abs(u(i, j)-uold(i, j)))
  enddo
enddo
```

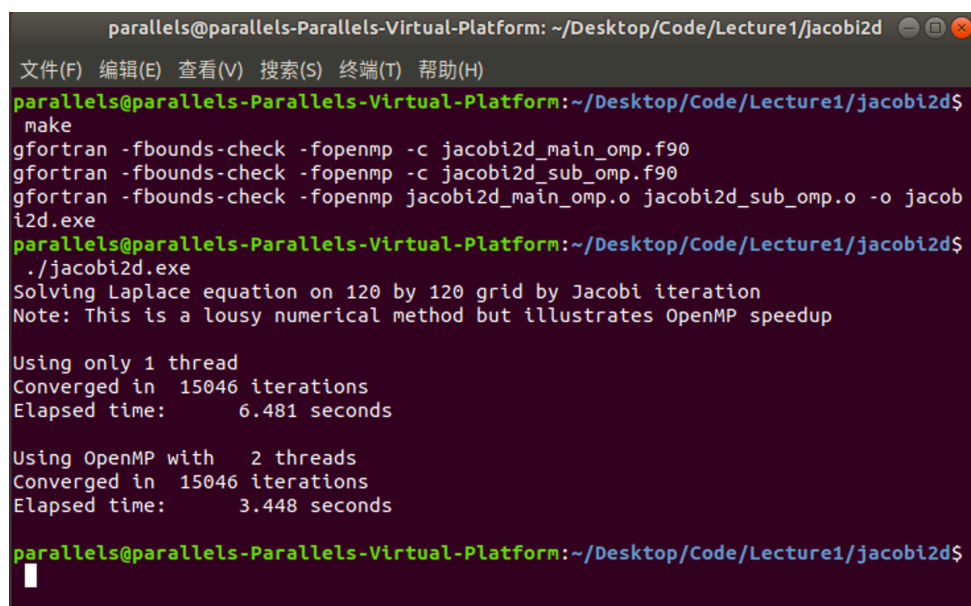
改进方法:

同理, 拆分 $(\text{MOD}(i+j, 2) .eq. 0)$ 和 $(\text{MOD}(i+j, 2) .eq. 1)$, 优化数据存取方面的因素, 再次提速。

```
do j=1,n
  do i=1,n
    if (MOD(i+j,2) .eq. 0) then
      u(i,j)=uold(i,j)+1.95d0*(0.25d0*(u(i-1,j) + u(i+1,j) + &
        u(i,j-1) + u(i,j+1) + h**2*f(i,j))-uold(i,j))
      dumax = max(dumax, abs(u(i,j)-uold(i,j)))
    end if
  enddo
enddo
do j=1,n
  do i=1,n
    if (MOD(i+j,2) .eq. 1) then
      u(i,j)=uold(i,j)+1.95d0*(0.25d0*(u(i-1,j) + u(i+1,j) + &
        u(i,j-1) + u(i,j+1) + h**2*f(i,j))-uold(i,j))
      dumax = max(dumax, abs(u(i,j)-uold(i,j)))
    end if
  enddo
enddo
```

4.2.3 实验结果

4.2.3.1 雅可比法 (Jacobi Method)



```
parallels@parallels-Parallels-Virtual-Platform: ~/Desktop/Code/Lecture1/jacobi2d
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
parallels@parallels-Parallels-Virtual-Platform:~/Desktop/Code/Lecture1/jacobi2d$
make
gfortran -fbounds-check -fopenmp -c jacobi2d_main_omp.f90
gfortran -fbounds-check -fopenmp -c jacobi2d_sub_omp.f90
gfortran -fbounds-check -fopenmp jacobi2d_main_omp.o jacobi2d_sub_omp.o -o jacobi2d.exe
parallels@parallels-Parallels-Virtual-Platform:~/Desktop/Code/Lecture1/jacobi2d$
./jacobi2d.exe
Solving Laplace equation on 120 by 120 grid by Jacobi iteration
Note: This is a lousy numerical method but illustrates OpenMP speedup

Using only 1 thread
Converged in 15046 iterations
Elapsed time: 6.481 seconds

Using OpenMP with 2 threads
Converged in 15046 iterations
Elapsed time: 3.448 seconds

parallels@parallels-Parallels-Virtual-Platform:~/Desktop/Code/Lecture1/jacobi2d$
```

4.2.3.2 高斯-赛德尔迭代 (Gauss - Seidel method)

一般方法:

```
Using only 1 thread
Converged in 8582 iterations
Elapsed time: 3.777 seconds

Using OpenMP with 2 threads
Converged in 8582 iterations
Elapsed time: 2.012 seconds
```

改进方法:

```
Using only 1 thread
Converged in 8552 iterations
Elapsed time: 3.077 seconds

Using OpenMP with 2 threads
Converged in 8552 iterations
Elapsed time: 2.233 seconds
```

4.2.3.3 SOR 逐次超松弛迭代法 (Successive Over Relaxation)

一般方法:

```
Using only 1 thread
Converged in 289 iterations
Elapsed time: 0.167 seconds

Using OpenMP with 2 threads
Converged in 291 iterations
Elapsed time: 0.106 seconds
```

改进方法:

```
Using only 1 thread
Converged in 253 iterations
Elapsed time: 0.125 seconds

Using OpenMP with 2 threads
Converged in 253 iterations
Elapsed time: 0.090 seconds
```

5 总结

本学期的高性能计算课程对我来说收获很大,万老师将课程知识及课程代码的知识的和细节讲的十分清楚,使我对高性能计算方面有了入门基础。通过学习知晓了 Fortran 语言的原理和使用等,能够读懂和简单使用这门语言,但是还需要继续学习,更熟练的使用该语言。

通过这次实习我对三种迭代方法有了深入的认识和理解。通过对老师提供的 Jacobi 迭代法解决“稳态热传导”问题示例代码的学习及其他两种迭代法对学习,使用另外两种方法求解问题。实习相对较难的部分就是将公式转化为 Fortran 代码,通过上网大量查阅资料,才得以解决。总得来说实习收获很大,但也需要继续努力学习。