# 并行大作业-使用文档及个人报告

张星辰2000011003

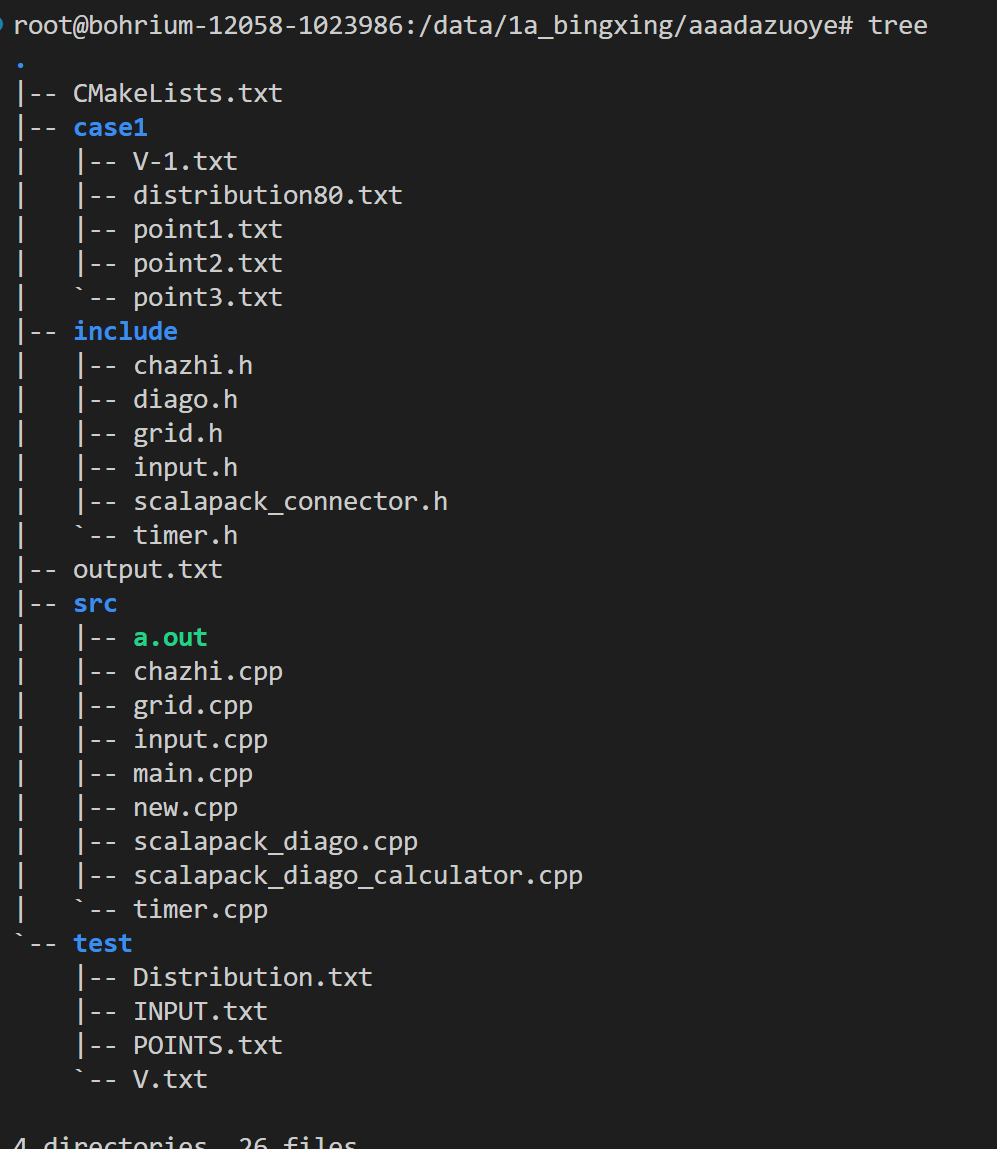
## 我用到的并行技术：

最终版本积分步骤用的是OpenMP，Scalapack对角化部分用MPI.

## 编译安装运行指南：

使用了APT安装的scalapack和lapack，用cmake编译，相应的CMakeLists.txt实现了链接库操作，应该直接链接就可以

程序的结构如下：



每个头文件的用处和结构：

* Input.h:存放用于读入输入文件的输入类
* Timer.h：存放计时类
* Interpolator.h：存放插值类
* Grid.h：存放网格类
* Diago.h：存放对角化用到的lapack\_diago和scalapack\_diago函数
* Scalapack\_connector.h：存放用到的scalapack的函数

关于输入文件：在输入文件进行修改就可以，注意diago\_lib为scalapack时，主程序会讲需要对角化之前的结果输出到output.txt,需要增加一步对new.cpp的编译运行。这里有不止一个scalapack的求解器……使用new.cpp可能需要手动修改一下new.cpp中的矩阵维度和分块矩阵维数。

程序本身的编译说明：使用如下命令构建本项目：

**# 生成构建目录**

**cmake -B build**

**# 执行构建 生成可执行程序**

**cmake --build build**

**# 运行程序**

**./build/myapp**

**#如果采用scalapack进行对角化操作，需要再编译运行一个程序**

**cd /src**

**mpicxx new.cpp -lscalapack-openmpi**

**mpirun --allow-run-as-root -np 4 a.out**

## 数据结构的设计

程序包含的类有Input类、Timer类、Grid类、Interpolator类

1.Input类用于所有读入工作，该输入类的数据结构存储了程序运行所需的所有参数和数据。下面简要介绍其中一些关键信息：

isHexahedral：标志着输入的网格类型是否为六面体。

lx, ly, lz, thetaxy, thetayz, thetaxz：分别表示网格的长度和三个方向上的倾斜角度。

support\_SH：标志着是否支持球谐分析计算。

diago\_lib：用于球谐分析计算的库文件名。

support\_Periodic\_Boundary：标志着是否支持周期性边界条件。

multi\_parallel\_strategies：标志着使用的并行策略。

points\_path：存储了点坐标文件所在的路径。

venergy\_path：存储了V文件所在的路径。

distribution\_path：存储了径向分布函数文件所在的路径。

point1\_, point2\_：用于指定一个空间子域。

count：点坐标的个数。

nx, ny, nz：网格的 X、Y、Z 方向上的节点数。

cutoff：距离截断半径。

dr：网格步长。

mesh：网格类型。

l：球谐多项式阶数。

f：一维径向分布函数

V：Venergy

在做附加题1需要读入多个点时，我把点坐标的形式更改为了结构体，定义一个以结构体为单元的向量用于读入点的坐标

struct Point {

double x;

double y;

double z;

};

int count;//点坐标的个数

std::vector<Point> readpointsFile(const std::string& filename, int limit);

2.Grid类的数据结构存储了一个网格的信息，包括了网格的边长及每个方向的节点数等。下面简要介绍其中的变量和函数：

m\_lx, m\_ly, m\_lz：分别表示网格的长度。

m\_nx, m\_ny, m\_nz：分别表示网格的 X、Y、Z 方向上的节点数。

Grid()：构造函数，用于初始化网格对象，输入网格边长 lx, ly, lz 和每个方向点数 nx, ny, nz。

generate\_uniform\_grid()：生成均匀网格，并返回每个节点的坐标信息。

cell\_volume()：计算每个网格单元的体积。

除以上列出的信息外，还有一些其他的函数和变量，用于实现网格操作和提供方便的接口来访问存储在该类中的数据。

3.Interpolator类：

该插值类的数据结构存储了一组距离和函数值，用于实现插值操作。下面简要介绍其中的变量和函数：

r\_：存储了一组距离。

f\_：存储了一组函数值。

Interpolator()：构造函数，用于初始化插值对象，输入距离和函数值两个 vector 对象。

distance()：计算点 (x,y,z) 到点 (cx,cy,cz) 的距离。

interpolate()：根据给定距离 r 进行插值操作，返回对应的函数值。

除以上列出的信息外，还有一些其他的函数和变量，用于实现插值操作和提供方便的接口来访问存储在该类中的数据。这个插值类采用的是**三次样条插值方法**。

4.Timer类：存放计时器，就是本学期之前写的计时器，方便查看效率。

**数据结构上的其他考虑：**

除了读入都采用向量读入以外，为了避免频繁访问类的成员对象降低效率，我把访问次数>1的对象（为数组则调用动态数组）存在本地。

## 程序优化工作（写的比较零散）

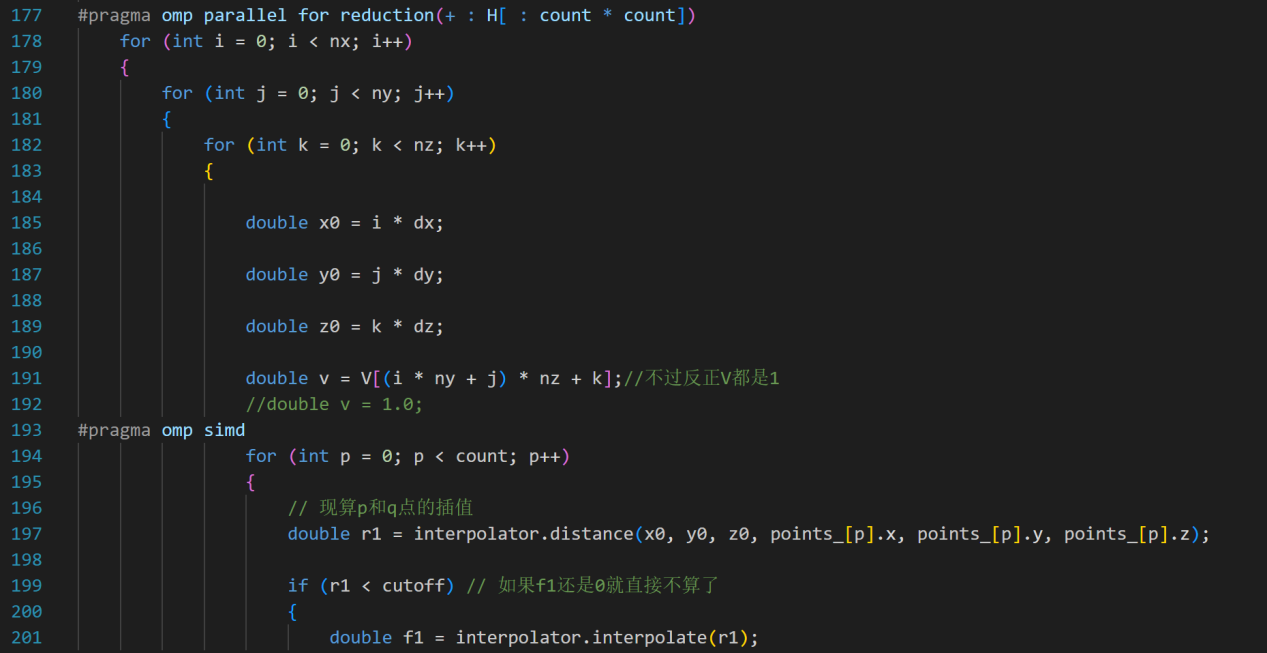
这个大作业我写到6月第一周快结束的时候中间不小心被我删了，后来重新写的，但是我觉得插值处理的没有原来的好，结果有些奇怪。

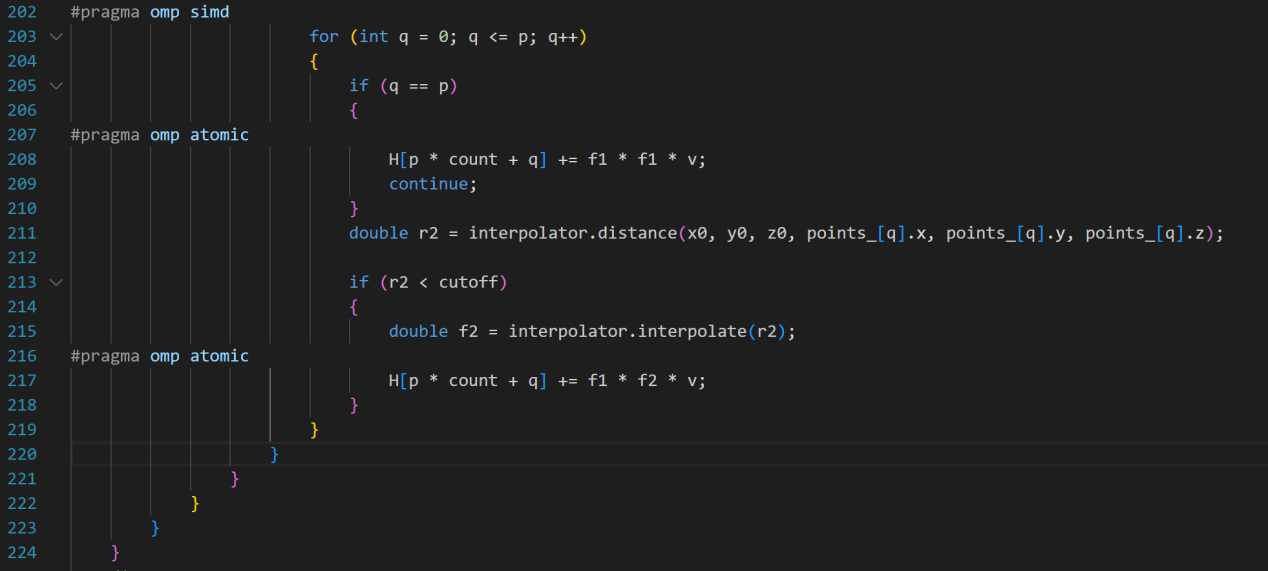
依次读入Input.txt、点坐标、径向分布函数、V函数。为避免插值积分过程中多次访问Input类的成员对象造成效率下降，把需要多次访问的量（如V、nx、ny、nz、mesh）等存到本地，插值预处理阶段还需生成一个距离数组d[mesh]。

调用Grid类生成网格（事实上这里网格没有什么实际的意义，原来我还写了一个函数叫generate\_uniform\_grid()，后来发现V1024总是被KILLED就是因为这里把网格点的坐标都给存下来了，会占用很多内存，所以这一步没有了，后续计算插值的时候直接计算网格点的坐标），后调用cell\_volume（）函数返回每个网格单元的体积。

插值阶段和积分阶段重合，定义插值类的对象interpolator（这里生成插值类需要距离数组d和一维径向分布函数数组f）。插值在计算该格点上积分的时候现行计算，因此可以节约空间成本。这里存在一个五层的嵌套循环，优化时我们知道外面三层应该是动不了的，所以我就只是把内层r>rcut的部分取消了计算，节约了一些时间成本。这一部分用OpenMP加速。其中H矩阵积分点的更新是原子操作。内层的两个循环用#pragma omp simd向量化了。同时注意了H的规约。（注：这一部分是优化的重点内容，发现最后计算量节约下来都是因为这里尽量减少了不必要的运算）

我发现只是注意处理p=q的情况，就能使程序从计算V1024P50花1300s加速到280s左右。类似的注意点还有只计算上三角部分。

计算完上三角部分后我们把下三角部分对称的赋值。之后所有格点积分乘以cell\_vol，这样可以减少在关键部分作乘法的数量。之后进行对角化操作得到最终结果。  




代码关键部分我是用的OpenMP，但是把scalapack整合后影响了原来的效率。我整合了MPI进去的V1024P50，比原来慢了好多，推测是由于线程冲突，越并行越慢……  
所以我后来尝试**把scalapack的功能单独写了一个文件**，避免线程冲突影响原来的效率，一开始放在了scalapack\_diago\_caculator.cpp里，但是这个程序我用到了一个函数，声明如下：

void pdgemr2d\_(int \*m, int \*n, double \*a, int \*ia, int \*ja, int \*desca,double \*b, int \*ib, int \*jb, int \*descb,int \*context, int \*info);

这个函数我在使用的时候后来一直报错，查看了手册等很多资料也没有解决。后来重新换用了new.cpp作为求解器，但是时间有限，这个求解器也有点瑕疵，需要手动修改矩阵的维度和分块矩阵维度。

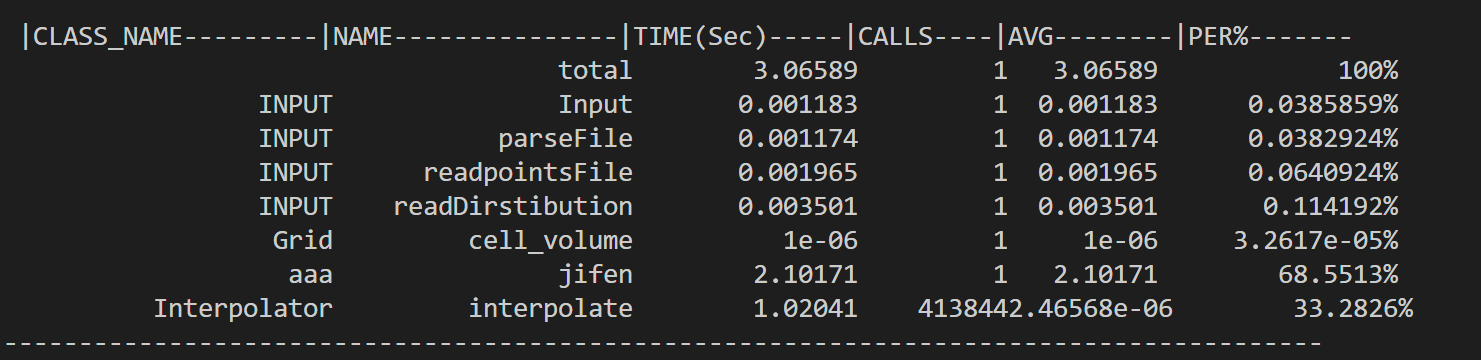
## 4.测试报告

### 正确性：

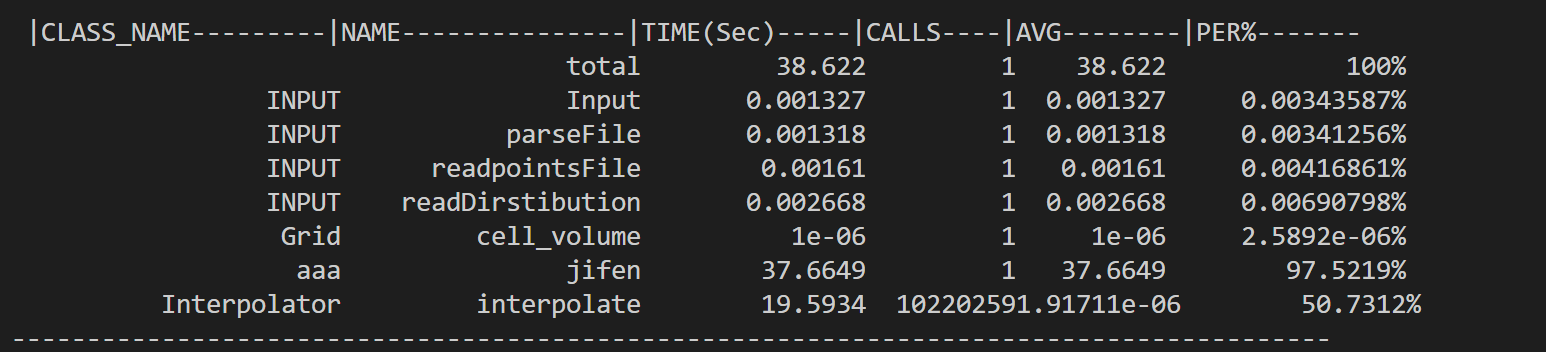
按照正确性测试的结果来看是正确的。

### 效率：（8核）

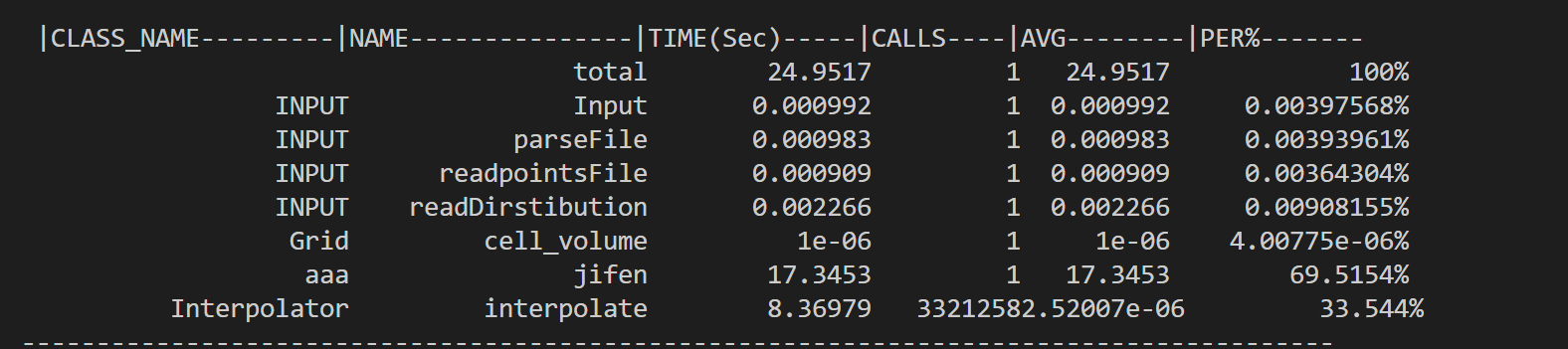
V512 Points2 运行约2s



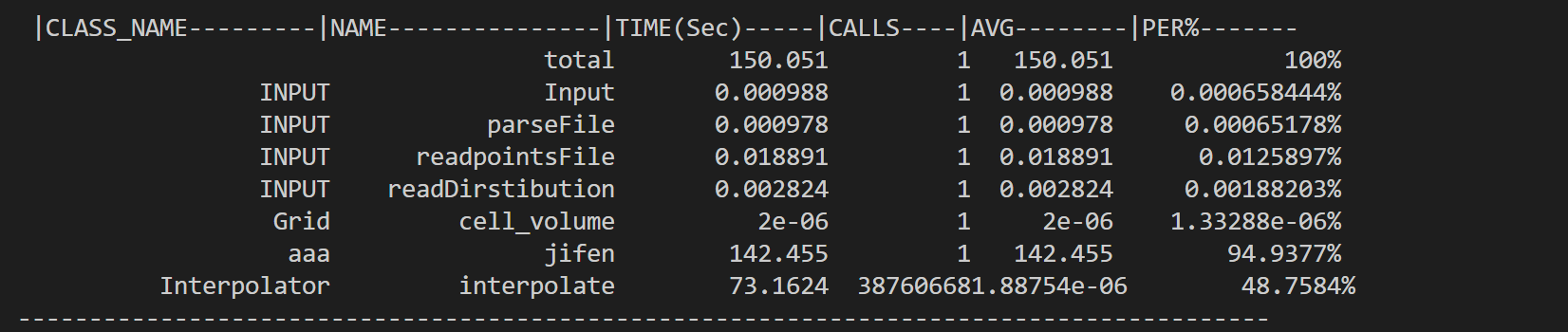
V512 Points50 运行约37s



V1024 Points2 运行约 17s



V1024 Points10运行约 142s



V1024,Points50运行约283s

