南京理工大学

NANJING UNIVERSITY OF SCIENCE & TECHNOLOGY

高性能计算引论
实验报告

班　　级 9191062301

学生姓名 孙傲歆

学　　号 919106840333

题　　目 Wordcount、蒙特卡罗算法、快排的实现

任课教师 李翔宇

2022 年 6 月

目　录

第一章 概述

1.1 实验概述

本实验是基于华为鲲鹏计算云服务进行的并行编程实验。实验要求基于 MPI 或者 OpenMp 从矩阵运算、Wordcount、蒙特卡罗算法、快排、PageRank 中选取 3 个进行实现。

这里我选择了：**基于 OpenMp 实现的蒙特卡罗算法计算 PI 的值、基于 OpenMp 实现的快速排序算法、基于 MPI 实现的 Wordcount 算法。**

对于每一个算法，报告将从：实验目的、实验环境说明、实验操作步骤以及运行结果四个方面进行详细的介绍。

最后还将列出我在实验中遇到的一些问题和解决方案以及经过本次实验我的一些心得体会。

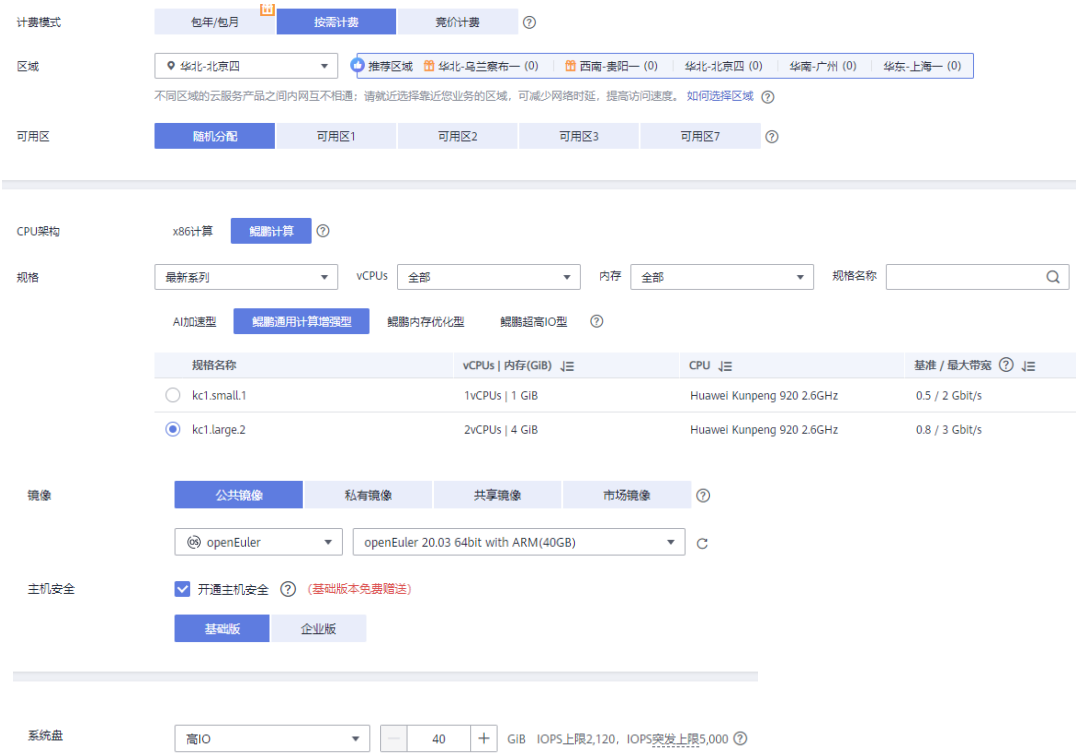
1.2 环境准备

1. 华为云的购买与配置

首先是对华为云进行基础配置，这里严格遵照了实验手册所给的配置。具体的配置如下表所示：

计费模式	区域	CPU架构	规格	镜像	系统盘
按需计费	华北-北京四	鲲鹏计算	kc1.large.2	公共镜像：openEuler 20.03	高I/O 40GB

如下是具体截图：



接着便是进行网络配置，这里为了防止余额不足，没有购买 10M 的带宽，仅购买了 5M 带宽。若后续出现问题，可以手动改变带宽。具体配置如下表所示：

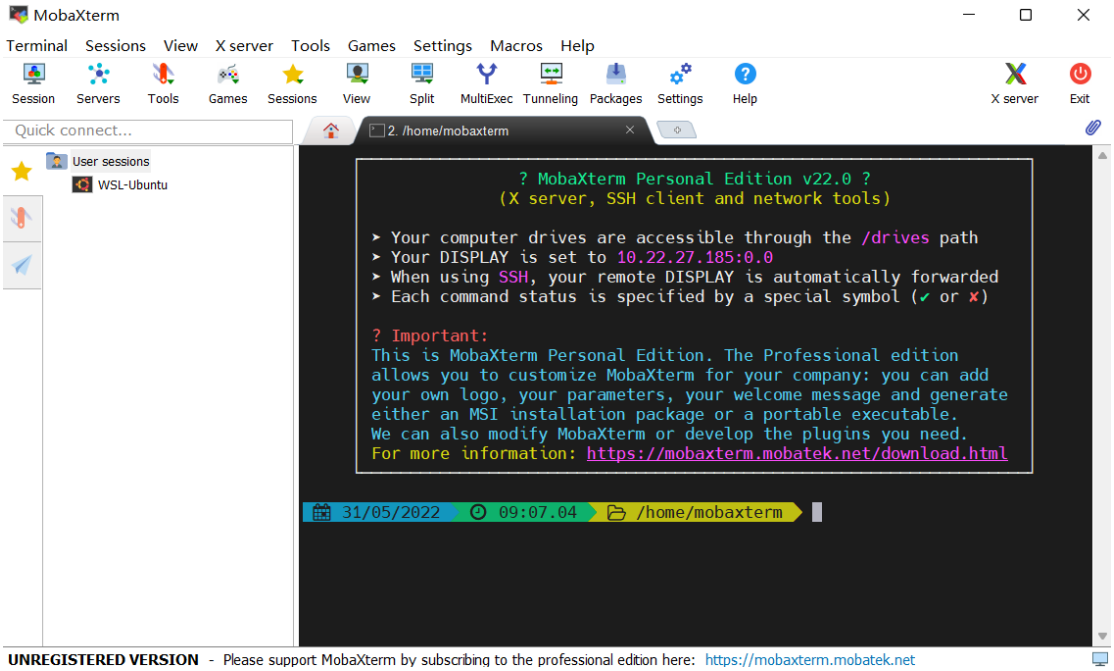
网络	安全组	弹性公网IP
默认的VPC	Sys-default	现在购买，带宽大小选择5Mbits/s

如下是具体截图：

最后便是高级配置，这里没有什么需要说明的，按照实验手册的示例，配置云服务名为“esc-hw”，密码为“Parallel2020”即可。

2. 下载 ssh 工具

这里我使用的 ssh 工具是：MobaXterm。该软件有免费版本，直接在官网下载即可。如下是该软件的初始界面：



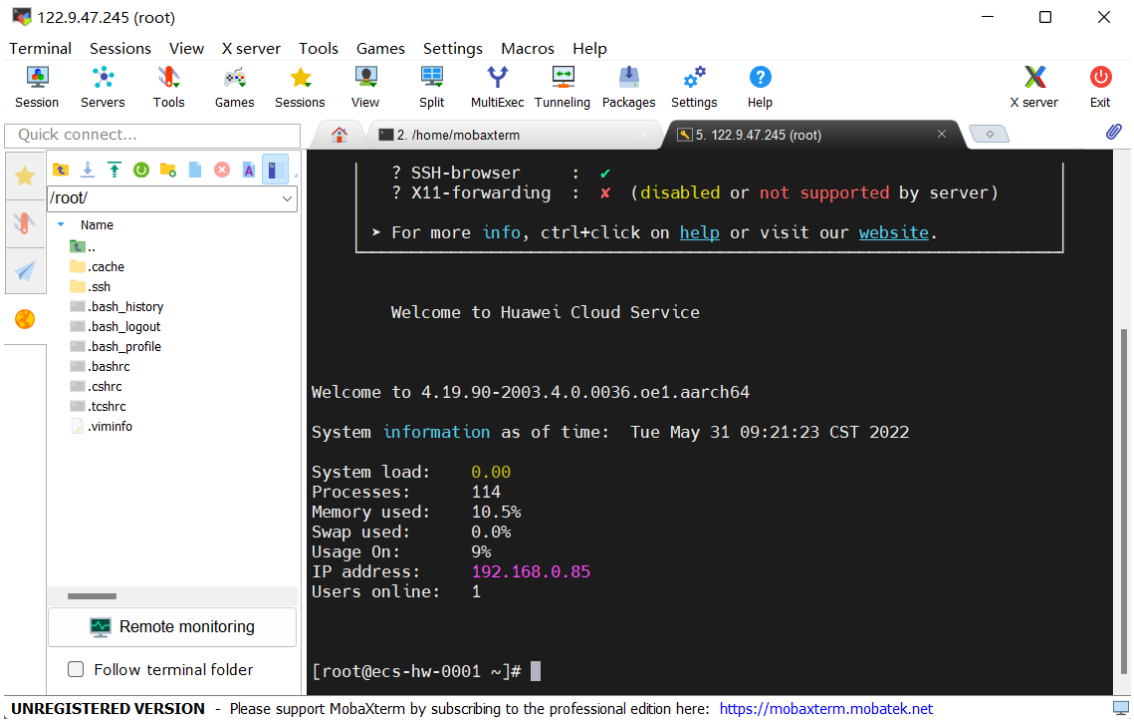
3. 登录 ESC

创建好 ECS 后，可在控制台 -> 弹性云服务器中看到弹性公网 IP。使用 SSH 工具，输入公网 IP、用户名和密码，或 ssh usr@IP 即可登陆。云服务器 IP 信息如下表所示：

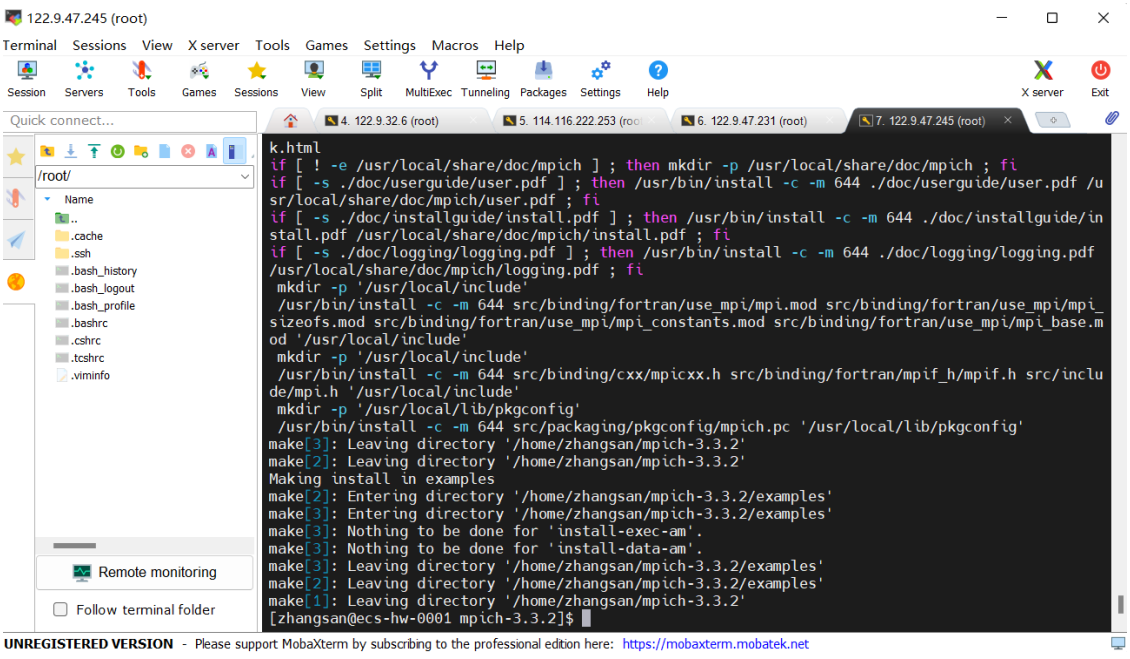
主机名	私网ip	公网IP
ecs-hw-0001	192.168.0.85	122.9.47.245
ecs-hw-0002	192.168.0.146	114.116.222.253
ecs-hw-0003	192.168.0.213	122.9.47.231
ecs-hw-0004	192.168.0.209	122.9.32.6

4. 配置环境

首先使用 MobaXterm 远程连接到华为云服务器。连接成功截图如下所示（这里以 ecs-hw-0001 为例）：



接着便是按照实验手册的步骤一步一步往下做即可，每个服务器都需重复配置一遍，所以具体操作在此不再赘述。最终完成环境配置结果截图如下所示（这里以 ecs-hw-0001 为例）：



第二章 快排

2.1 实验目的

本实验在华为鲲鹏云服务器上，基于 OpenMp 编译运行快排算法程序。实现多台主机快排算法的编译运行。旨在掌握多台主机快排算法的编写，多线程 OpenMp 并行计算的配置及程序编写以及加深对并行计算的了解。

2.2 实验环境说明

- 华为鲲鹏云服务器
- openEuler 20.03 操作系统（远程主机）
- windows-11 操作系统（本地主机）
- MobaXtrem Home Edition v22.0

2.3 实验操作步骤

步骤 1:创建源代码（四台主机都要执行）

在 home/zhangsan 目录下创建 quicksort 目录存放该程序的所有文件，并进入 quicksort 目录，使用 vim 指令创建并编辑快排算法源码 quick_sort.cpp。

具体操作如下图所示：

```
[zhangsan@ecs-hw-0001 ~]$ mkdir /home/zhangsan/quicksort
[zhangsan@ecs-hw-0001 ~]$ cd /home/zhangsan/quicksort
[zhangsan@ecs-hw-0001 quicksort]$ vim quick_sort.cpp
```

代码如下图所示：

```
    struct timeval start, stop;
    gettimeofday(&start, NULL);

#pragma omp parallel default(none) shared(array, n)
    {
#pragma omp single nowait
        { QuickSort(array, n); }
    }
    gettimeofday(&stop, NULL);

    double elapse = (stop.tv_sec - start.tv_sec) * 1000 +
                    (stop.tv_usec - start.tv_usec) / 1000;
    cout <<"running time:"<<elapse <<";number of data:"<< n << endl;
        int flag=0;
    for (int i = 0; i < n - 1; i++) {
        if (array[i] > array[i + 1]) {
            flag=1;
            break;
        }
    }
    if(flag==0)
        cout<<"quick sort success! \n";
    else
        cerr << "quick sort fails! \n";

    return 0;
}
                                                          78,1          Bot
```

步骤 2：创建 Makefile 文件（四台主机都要执行）

使用 vim 指令创建并编写 Makefile 文件。

具体操作如下图所示：

```
[zhangsan@ecs-hw-0001 quicksort]$ vim Makefile
```

代码如下图所示：

```
CC = g++
CCFLAGS = -I . -O2 -fopenmp
LDFLAGS = # -lopenblas

all:  quicksort

quicksort: quick_sort.cpp
        ${CC} ${CCFLAGS} quick_sort.cpp -o quicksort ${LDFLAGS}

clean:
        rm  quicksort
```

步骤 3：进行编译（四台主机都要执行）

输入 make 指令，对源码进行编译。然后使用 ll 命令查看，发现当前目录下出现了一个名为“quicksort”的文件，说明编译成功。

具体操作如下图所示：

```
[zhangsan@ecs-hw-0001 quicksort]$ make
g++ -I . -O2 -fopenmp quick_sort.cpp -o quicksort
[zhangsan@ecs-hw-0001 quicksort]$ ll
total 28K
-rw------- 1 zhangsan zhangsan  184 May 31 14:51 Makefile
-rwx------ 1 zhangsan zhangsan  72K May 31 14:52 quicksort
-rw------- 1 zhangsan zhangsan 2.1K May 31 14:48 quick_sort.cpp
```

步骤 4：建立主机配置文件（四台主机都要执行）

使用 vim 指令，创建并编写 hostfile 指令。该文件中的内容将指定不同主

机上分配的核心数。

具体操作如下图所示：

```
[zhangsan@ecs-hw-0001 quicksort]$ vim /home/zhangsan/quicksort/hostfile
```

代码如下图所示：

```
ecs-hw-0001:2
ecs-hw-0002:2
ecs-hw-0003:2
ecs-hw-0004:2
```

步骤 5：编写 run.sh 脚本（只需要在 ecs-hw-0001 上执行）

使用 vim 指令，创建并编写 run.sh 脚本文件。值得注意的是，此步骤无需在四台主机上进行，只需在 ecs-hw-0001 上进行即可。

具体操作如下图所示：

```
[zhangsan@ecs-hw-0001 quicksort]$ vim /home/zhangsan/quicksort/run.sh
```

代码如下图所示：

```
app=${1}

if [ ${app} = "quicksort" ]; then
        ./quicksort ${2} 8000000
fi
```

至此，代码运行所需的所有文件及环境配置完成。在主机 ecs-hw-0001 输入相应的执行命令，代码就可运行。下面将给出实验代码的运行结果。

2.4 运行结果

在 ecs-hw-0001 主机上输入“bash run.sh quicksort x”（其中 x 代表线程数）得到以下结果：

```
[zhangsan@ecs-hw-0001 quicksort]$ bash run.sh quicksort 1
running time:1272;number of data:8000000
quick sort success!
[zhangsan@ecs-hw-0001 quicksort]$ bash run.sh quicksort 2
running time:631;number of data:8000000
quick sort success!
[zhangsan@ecs-hw-0001 quicksort]$ bash run.sh quicksort 3
running time:627;number of data:8000000
quick sort success!
[zhangsan@ecs-hw-0001 quicksort]$ bash run.sh quicksort 4
running time:613;number of data:8000000
quick sort success!
```

```
[zhangsan@ecs-hw-0001 quicksort]$ bash run.sh quicksort 5
running time:633;number of data:8000000
quick sort success!
[zhangsan@ecs-hw-0001 quicksort]$ bash run.sh quicksort 6
running time:629;number of data:8000000
quick sort success!
[zhangsan@ecs-hw-0001 quicksort]$ bash run.sh quicksort 7
running time:652;number of data:8000000
quick sort success!
[zhangsan@ecs-hw-0001 quicksort]$ bash run.sh quicksort 8
running time:650;number of data:8000000
```

可以看到快排算法程序已经在集群中并行运行起来。分析以上结果我们不难得出:每次运行快速排序的都是成功的,即最终得到的序列都是从小到大排列的。说明并行程序编写无误，可以得到正确的结果。

而随着线程数的增加，程序的运行耗时越来越少。从开始的 1272 减少到 630 左右。但随着后续线程数的继续增加，程序的运行时间已经不再减少。这有可能是线程的数目已经超过了算法的并行程度所导致。

第三章 蒙特卡罗算法

3.1 实验目的

本实验在华为鲲鹏云服务器上，基于 OpenMp 编译运行蒙特卡罗算法程序，使用蒙特卡罗算法计算圆周率 PI 的值。实现多台主机蒙特卡罗算法的编译运行。旨在掌握多台主机蒙特卡罗算法的编写，多线程 OpenMp 并行计算的配置及程序编写以及加深对并行计算的了解。

3.2 实验环境说明

- 华为鲲鹏云服务器
- openEuler 20.03 操作系统（远程主机）
- windows-11 操作系统（本地主机）
- MobaXtrem Home Edition v22.0

3.3 实验操作步骤

步骤 1:创建源代码（四台主机都要执行）

在 home/zhangsan 目录下创建 MonteCarlo 目录存放该程序的所有文件，并进入 MonteCarlo 目录，使用 vim 指令创建并编辑蒙特卡罗算法源码 MonteCarlo.cpp。

具体操作如下图所示：

```
[zhangsan@ecs-hw-0001 ~]$ mkdir /home/zhangsan/MonteCarlo
[zhangsan@ecs-hw-0001 ~]$ cd /home/zhangsan/MonteCarlo
[zhangsan@ecs-hw-0001 MonteCarlo]$ vim MonteCarlo.cpp
```

代码如下图所示：

```
srand(time(NULL));
int t=atoi(argv[1]);
int n=atoi(argv[2]);
double count=0.0;
omp_set_num_threads(t);
struct timeval start, stop;
gettimeofday(&start, NULL);
//MC::PI::calPiRets *rets;
//rets = MC::RunMC<MC::PI::calPiArgs, MC::PI::calPiRets>(n);
int sum=0;
double x,y;
#pragma omp parallel for private(x,y) reduction(+:sum)
for(int i=0;i<n;i++)
{
    x=random(0,1);
    y=random(0,1);
    if(x*x+y*y<=1)
    sum++;
}
gettimeofday(&stop, NULL);
double pi=double(sum*4.0/n);
printf("PI = %.6f",pi);
double time = (stop.tv_sec - start.tv_sec) * 1000 +
              (stop.tv_usec - start.tv_usec) / 1000;
cout<<endl;
cout<<"running time:"<<time<<";number of data:"<<n<<endl;
return 0;
```

紧接着再使用 vim 指令创建并编辑其头文件 MonteCarlo.h。

具体操作如下图所示：

```
[zhangsan@ecs-hw-0001 MonteCarlo]$ vim MonteCarlo.h
```

代码如下图所示：

```
#include <sys/time.h>
#include <time.h>
#include <iostream>
#include "omp.h"
using namespace std;
namespace MC {
template <typename Args, typename Rets>
void sampling(Args &arg, Rets &ret);
template <typename Args>
void yieldSamples(Args &arg);
template <typename Args>
void delSamples(Args &arg);
template <typename Args, typename Rets>
Rets *RunMC(int sampleSize) {
    Args *args = new Args[sampleSize];
    Rets *rets = new Rets[sampleSize];
#pragma omp parallel for
    for (int i = 0; i < sampleSize; i++) {
        yieldSamples(args[i]);
        sampling(args[i], rets[i]);
        delSamples(args[i]);
    }
    delete args;
    return rets;
}
}
#endif
```

步骤 2：创建 Makefile 文件（四台主机都要执行）

使用 vim 指令创建并编写 Makefile 文件。

具体操作如下图所示：

```
[zhangsan@ecs-hw-0001 MonteCarlo]$ vim Makefile
```

代码如下图所示：

```
CC = g++
CCFLAGS = -I . -O2 -fopenmp
LDFLAGS = # -lopenblas

all: montecarlo

montecarlo: MonteCarlo.cpp
        ${CC} ${CCFLAGS} MonteCarlo.cpp -o montecarlo ${LDFLAGS}
clean:
        rm montecarlo
```

步骤 3：进行编译（四台主机都要执行）

输入 make 指令，对源码进行编译。然后使用 ll 命令查看，发现当前目录下出现了一个名为“montecarlo”的文件，说明编译成功。

具体操作如下图所示：

```
[zhangsan@ecs-hw-0001 MonteCarlo]$ make
g++ -I . -O2 -fopenmp MonteCarlo.cpp -o montecarlo
[zhangsan@ecs-hw-0001 MonteCarlo]$ ll
total 32K
-rw------- 1 zhangsan zhangsan  188 May 31 15:41 Makefile
-rwx------ 1 zhangsan zhangsan  72K May 31 15:42 montecarlo
-rw------- 1 zhangsan zhangsan 2.7K May 31 15:37 MonteCarlo.cpp
-rw------- 1 zhangsan zhangsan  778 May 31 15:40 MonteCarlo.h
```

步骤 4：建立主机配置文件（四台主机都要执行）

使用 vim 指令，创建并编写 hostfile 指令。该文件中的内容将指定不同主机上分配的核心数。

具体操作如下图所示：

```
[zhangsan@ecs-hw-0001 MonteCarlo]$ vim /home/zhangsan/MonteCarlo/hostfile
```

代码如下图所示：

```
ecs-hw-0001:2
ecs-hw-0002:2
ecs-hw-0003:2
ecs-hw-0004:2
```

步骤 5：编写 run.sh 脚本（只需要在 ecs-hw-0001 上执行）

使用 vim 指令，创建并编写 run.sh 脚本文件。值得注意的是，此步骤无需在四台主机上进行，只需在 ecs-hw-0001 上进行即可。

具体操作如下图所示：

```
[zhangsan@ecs-hw-0001 MonteCarlo]$ vim run.sh
```

代码如下图所示：

```
app=${1}

if [ ${app} = "montecarlo" ]; then
        ./montecarlo ${2} 8000000
fi
```

至此，代码运行所需的所有文件及环境配置完成。在主机 ecs-hw-0001 输入相应的执行命令，代码就可运行。下面将给出实验代码的运行结果。

3.4 运行结果

在 ecs-hw-0001 主机上输入“bash run.sh montecarlo x”（其中 x 代表线程数）得到以下结果：

```
[zhangsan@ecs-hw-0001 MonteCarlo]$ bash run.sh montecarlo 1
PI = 3.141291
running time:121;number of data:8000000
[zhangsan@ecs-hw-0001 MonteCarlo]$ bash run.sh montecarlo 2
PI = 3.141269
running time:61;number of data:8000000
[zhangsan@ecs-hw-0001 MonteCarlo]$ bash run.sh montecarlo 3
PI = 3.141342
running time:81;number of data:8000000
[zhangsan@ecs-hw-0001 MonteCarlo]$ bash run.sh montecarlo 4
PI = 3.141365
running time:61;number of data:8000000
[zhangsan@ecs-hw-0001 MonteCarlo]$ bash run.sh montecarlo 5
PI = 3.141391
running time:73;number of data:8000000
[zhangsan@ecs-hw-0001 MonteCarlo]$ bash run.sh montecarlo 6
PI = 3.141415
running time:61;number of data:8000000
[zhangsan@ecs-hw-0001 MonteCarlo]$ bash run.sh montecarlo 7
PI = 3.141321
running time:70;number of data:8000000
[zhangsan@ecs-hw-0001 MonteCarlo]$ bash run.sh montecarlo 8
PI = 3.141346
running time:61;number of data:8000000
```

可以看到蒙特卡罗算法程序已经在集群中并行运行起来。分析以上结果我们不难得出：每次程序运行都是成功的，每次的 PI 值均在 3.14 左右，与真实的 PI 值相差不大。说明程序编写是无误的。

而随着线程数的增加，程序的运行耗时越来越少。从开始的 121 减少到 60 左右。但和前面的快排算法一样，随着后续线程数的继续增加，程序的运行时间已经不再减少。

第四章 WordCount

4.1 实验目的

本实验在华为鲲鹏云服务器上，基于 MPI 编译运行 WordCount 算法程序，使用该算法统计所给的测试文件中出现的词汇和其频数。实现多台主机 WordCount 算法的编译运行。旨在掌握多台主机 WordCount 算法的编写，多主机 MPI 并行计算的配置、程序编写以及加深对并行计算的了解。

4.2 实验环境说明

- 华为鲲鹏云服务器
- openEuler 20.03 操作系统（远程主机）
- windows-11 操作系统（本地主机）
- MobaXtrem Home Edition v22.0
- mpich-3.3.2

- OpenBLAS-0.3.8

4.3 实验操作步骤

步骤 1:创建源代码（四台主机都要执行）

在 home/zhangsan 目录下创建 wordcount 目录存放该程序的所有文件，并进入 wordcount 目录，使用 vim 指令创建并编辑快排算法源码 quick_sort.cpp。

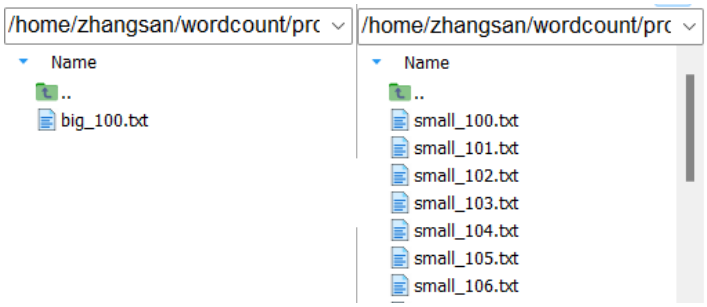
具体操作如下图所示：

```
[zhangsan@ecs-hw-0001 ~]$ mkdir /home/zhangsan/wordcount
[zhangsan@ecs-hw-0001 ~]$ cd /home/zhangsan/wordcount
```

实验提供两个文件夹，第一个文件夹包含 100 个小文件，第二文件夹包含一个大文件。执行 mkdir 命令，创建存放测试数据的目录。然后直接在 MobaXterm 的“SSH browser”里找到新创建的文件夹，直接将测试数据复制粘贴到相应文件夹即可，十分方便。

具体操作如下图示：

```
[zhangsan@ecs-hw-0001 wordcount]$ mkdir -p project_file/big_file
[zhangsan@ecs-hw-0001 wordcount]$ mkdir -p project_file/small_file
```



接着使用 vim 指令创建并编辑 wordcount 算法源码 wordcount.cpp。

具体操作如下图所示：

```
[zhangsan@ecs-hw-0001 wordcount]$ vim wordcount.cpp
```

代码如下图所示：

```
    std::vector<std::string> wdlst;
    for (int i = 0; i < small_files.size(); i++) {
        auto content = readFile(small_files[i]);
        wdlst.push_back(content);
    }
    auto counts = countWords(wdlst);

    treeMerge(rank, worldSize, counts);
    gettimeofday(&stop, NULL);
    if (rank == 0) {
        cout << "SmallFiles"<<endl;
        cout << "running time:"
             << (stop.tv_sec - start.tv_sec) * 1000.0 +
                 (stop.tv_usec - start.tv_usec) / 1000.0
             << " ms"<< endl;
        int sum=counts.size();
                cout<<"number of different words:"<<sum<<endl;
    }

    // if (rank == 0) {
    //     for (auto it = counts.begin(); it != counts.end(); it++) {
    //         std::cout << it->first << " : " << it->second << endl;
    //     }
    // }
}
    MPI_Finalize();
    return 0;
}
"wordcount.cpp" 282L, 8315C                                   282,1         Bot
```

步骤 2：创建 Makefile 文件（四台主机都要执行）

使用 vim 指令创建并编写 Makefile 文件。

具体操作如下图所示：

```
[zhangsan@ecs-hw-0001 wordcount]$ vim Makefile
```

代码如下图所示：

```
CC = mpic++
CCFLAGS = -O2 -fopenmp
LDFLAGS = -lopenblas

all:  wordcount

wordcount: wordcount.cpp
        ${CC} ${CCFLAGS} wordcount.cpp -o wordcount ${LDFLAGS}

clean:
        rm  wordcount
```

步骤 3：进行编译（四台主机都要执行）

由于本实验是基于 MPI 编写的，其编译步骤和前两个实验相比较为复杂。首先要获取并安装包“OpenBLAS-0.3.8”，其需要执行指令以完成包的下载、解压、编译和安装，具体过程十分繁琐这里便不再展示。以下是需要执行的指令：

wget https://github.com/xianyi/OpenBLAS/archive/v0.3.8.tar.gz

tar -zxvf v0.3.8.tar.gz && cd OpenBLAS-0.3.8

make -j2

sudo make PREFIX=/usr/local/openblas install

sudo chmod –R 777 /usr/local/openblas/

sudo ln –s /usr/local/openblas/lib/libopenblas.so /usr/lib/libopenblas.so

执行完成上述指令后，OpenBLAS-0.3.8 包安装完成。接下来需要执行 vim 指令进行 OpenBLAS 环境的配置。

具体操作如下图所示：

```
[zhangsan@ecs-hw-0001 wordcount]$ vim ~/.bashrc
```

代码如下图所示：

```
# Source default setting
[ -f /etc/bashrc ] && . /etc/bashrc

# User environment PATH
PATH="$HOME/.local/bin:$HOME/bin:$PATH"
export PATH
export CPLUS _INCLUDE_PATH=$CPLUS_INCLUDE_PATH: /usr/local/openblas/include

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/openblas/lib
```

执行 source 命令，使环境变量均生效。

具体操作如下图所示：

```
[zhangsan@ecs-hw-0001 wordcount]$ source ~/.bashrc
```

执行完以上步骤，后面的配置就和前两个实验差距不大了。输入 make 指令，对源码进行编译。然后使用 ll 命令查看，发现当前目录下出现了一个名为“wordcount”的文件，说明编译成功。

具体操作如下图所示：

```
[zhangsan@ecs-hw-0001 wordcount]$ make
mpic++ -O2 -fopenmp wordcount.cpp -o wordcount -lopenblas
[zhangsan@ecs-hw-0001 wordcount]$ ll
total 12M
-rw-------  1 zhangsan zhangsan   57 Jun  1 10:29 hostfile
-rw-------  1 zhangsan zhangsan  180 Jun  1 09:52 Makefile
drwx------ 16 zhangsan zhangsan 4.0K Jun  1 10:25 OpenBLAS-0.3.8
drwx------  4 zhangsan zhangsan 4.0K Jun  1 09:12 project_file
-rw-------  1 zhangsan zhangsan   98 Jun  1 10:28 run.sh
-rw-------  1 zhangsan zhangsan  12M Jun  1 09:53 v0.3.8.tar.gz
-rwx------  1 zhangsan zhangsan  80K Jun  1 15:17 wordcount
-rw-------  1 zhangsan zhangsan 8.2K Jun  1 15:17 wordcount.cpp
```

步骤 4：建立主机配置文件（四台主机都要执行）

使用 vim 指令，创建并编写 hostfile 指令。该文件中的内容将指定不同主机上分配的核心数。

具体操作如下图所示：

```
[zhangsan@ecs-hw-0001 wordcount]$ vim /home/zhangsan/wordcount/hostfile
```

代码如下图所示：

```
ecs-hw-0001:2
ecs-hw-0002:2
ecs-hw-0003:2
ecs-hw-0004:2
```

步骤 5：编写 run.sh 脚本（只需要在 ecs-hw-0001 上执行）

使用 vim 指令，创建并编写 run.sh 脚本文件。值得注意的是，此步骤无需在四台主机上进行，只需在 ecs-hw-0001 上进行即可。具体操作如下图所示：

```
[zhangsan@ecs-hw-0001 wordcount]$ vim run.sh
```

```
app=${1}

if [ ${app} = "wordcount" ]; then
        mpirun --hostfile hostfile -np ${2} ./wordcount
fi
```

至此，代码运行所需的所有文件及环境配置完成。在主机 ecs-hw-0001 输入相应的执行命令，代码就可运行。下面将给出实验代码的运行结果。

4.4 运行结果

在 ecs-hw-0001 主机上输入“bash run.sh quicksort x”（其中 x 代表启动核心数，x 这里取 2,4,6,8）得到以下结果：

```
[zhangsan@ecs-hw-0001 wordcount]$ bash run.sh wordcount 2
Bigfile:
running time:176.855 ms
number of different words:11173
SmallFiles
running time:404.159 ms
number of different words:51303
```

```
[zhangsan@ecs-hw-0001 wordcount]$ bash run.sh wordcount 4

Authorized users only. All activities may be monitored and reported.
Bigfile:
running time:287.538 ms
number of different words:14029
SmallFiles
running time:409.381 ms
number of different words:51303
```

```
[zhangsan@ecs-hw-0001 wordcount]$ bash run.sh wordcount 6

Authorized users only. All activities may be monitored and reported.

Authorized users only. All activities may be monitored and reported.
Bigfile:
running time:324.205 ms
number of different words:14780
SmallFiles
running time:414.434 ms
number of different words:51303
```

```
[zhangsan@ecs-hw-0001 wordcount]$ bash run.sh wordcount 8

Authorized users only. All activities may be monitored and reported.

Authorized users only. All activities may be monitored and reported.

Authorized users only. All activities may be monitored and reported.
Bigfile:
running time:372.374 ms
number of different words:15230
SmallFiles
running time:417.65 ms
number of different words:51303
```

通过上述运行，可以看出 wordcount 算法程序已经在集群中并行运行起来。其中第一行输出代表的是大文件统计耗时，第二行输出代表的是小文件统计耗时。分析结果我们可以发现对于大文件，其统计的单词数随着主机数目的增大也在增大，而小文件的单词数目却是稳定不变的。这有可能是因为多个主机在对大文件进行交互操作时，有重复统计的可能。而小文件是分发各个主机去处理，不存在重复统计问题。

这里选择启动核心数为 2,4,6,8 就要结合者 hostfile 文件来看。当 x=2 时，此时只在 ecs-hw-0001 上运行；当 x=4 时，此时同时启动了 ecs-hw-0001 和 ecs-hw-0002……以此类推。从以上结果我们也不难看当 x 取 4,6,8 时不同的主机确实进行了信息的交互，程序在多台主机上同时运行。而且运行时间随着并行度增加没有减少，反而有上升趋势，这可能是额外的通信开销导致的。

第五章 总结

5.1 实验过程中遇到的问题及解决方案

问题一：华为云配置问题

问题描述：若按照实验手册中所给配置，每小时费用较高，只能使用差不多两天时间，实验可能无法按时完成。

解决方案：在带宽设置时，减小带宽。实验手册的推荐带宽为 10M，这里我只设置了 5M 带宽，任然不影响实验的进行，但价格却十分低廉，每小时只需不到 2.4 元。有效地解决了可能的余额不足问题，有了更加充足的时间去完成实验。

问题二：蒙特卡罗算法结果有误

问题描述：运行蒙特卡罗算法时 PI 的计算结果有误。**如下图所示**，第一行是 8000000 次计算的命中次数，第二行是 PI 的值。可以看到 8000000 次计算全部命中，导致结果 PI 一直为 4。

```
8000000
PI = 4.000000
running time:411;number of data:8000000
```

解决方案：经过排查发现是坐标值 x、y 的类型设置成了 int，**如下图所示。**导致每次随机生成的 x,y 值都为 0 0，从而得到了错误的结果。通过将 x、y 的值改为 double 成功获得了正确的结果。

```
int sum=0,x,y;
#pragma omp parallel for private(x,y) reduction(+:sum)
    for(int i=0;i<n;i++)
    {
            x=random(0,1);
            y=random(0,1);
            if(x*x+y*y<=1)
            sum++;
    }
cout<<sum<<endl;
```

问题三：环境变量设置问题

问题描述：在进行 MPI 上的 wordcount 实现时，在步骤 3 编译一步中。实现 source 命令使环境变量生效。但每次调用 source 指令都会产生“export： not a valid identifier”错误。同样的错误在程序运行时也会发生。**如下图所示：**

```
[zhangsan@ecs-hw-0001 wordcount]$ source ~/.bashrc
-bash: export: `/usr/local/openblas/include': not a valid identifier
```

```
[zhangsan@ecs-hw-0001 wordcount]$ bash run.sh wordcount 8

Authorized users only. All activities may be monitored and reported.

Authorized users only. All activities may be monitored and reported.

Authorized users only. All activities may be monitored and reported.
/home/zhangsan/.bashrc: line 7: export: `/usr/local/openblas/include': not a valid identifier
/home/zhangsan/.bashrc: line 7: export: `/usr/local/openblas/include': not a valid identifier
/home/zhangsan/.bashrc: line 7: export: `/usr/local/openblas/include': not a valid identifier
```

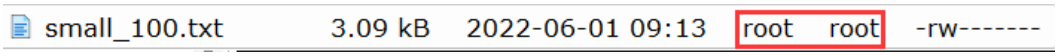
解决方案：经过排查发现是实验手册中所给的代码中‘/user’前多了一个空格导致，**错误代码如下图所示。**实验手册还有许多类似的错误，包括“-”误打成了“—”。所以说不能完全相信实验手册，有时还需要自己实践找出问题。通过删除该空格成功解决了以上问题。

```
# Source default setting
[ -f /etc/bashrc ] && . /etc/bashrc

# User environment PATH
PATH="$HOME/.local/bin:$HOME/bin:$PATH"
export PATH
export CPLUS _INCLUDE_PATH=$CPLUS_INCLUDE_PATH /usr/local/openblas/include

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/openblas/lib
```

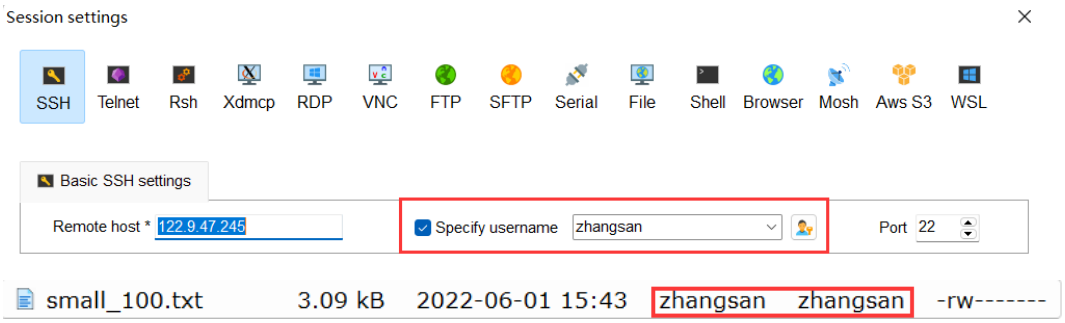
问题四：wordcount 算法结果有误

问题描述：第一次运行 wordcount 算法时，发现程序速度运行的非常快。感觉有些不对劲，便分别输出了大文件和小文件的统计信息。发现大文件和小文件的统计数都显示为 0。如果代码没有问题的话，那显然是文件的读取出现了问题。

解决方案：无意间使用 vim 指令尝试直接打开测试文件，系统提示没有打开权限。后来经过排查，发现是复制测试文件时，系统默认是 root 用户进行复制，所以 zhangsan 用户没有权限访问。这也是程序无法读入数据的原因。**如下图所示：**



后来经过直接在主界面修改 session 连接的数据，让 SSH 连接时便默认使用 zhangsan 用户进行登录。然后在 zhangsan 用户下，再进行复制操作，成功改变了文件的访问权限，**如下图所示：**



5.2 心得体会

这是我大学期间第二次接触华为云的相关实验了，第一次华为云实验是软件课程设计(I)课程中的 openGauss 数据库实验。

但这次并行计算实验与前面的数据库实验完全不一样。首先数据库实验依旧

是单主机的操作，其不涉及多主机之间的交互问题，所以更为简单。而本次实验则涉及多个主机之间的交互操作，对我来说不仅是一次十分新奇的体验，更是一个不小的挑战。

本次实验要求我们从六个实验中题目中选取三个来做。分别是基于 MPI 实现的 HelloWorld 编程、矩阵运算编程、wordcount 算法编程，以及基于 OpenMp 的蒙特卡罗算法编程、快排算法编程以及 PageRank 算法编程。这里我选择了 wordcount、蒙特卡罗以及快排这三个题目。

我首先进行的是基于 OpenMp 的两个实验。因为之前选过一门选修课叫《并行计算导论》，是专门讲述 OpenMp 编程的一门课，所以我本身对 OpenMp 编程也较为熟悉。虽然熟悉但以前也都是在 Windows 环境下进行的编程，在 Linux 环境下进行 OpenMp 程序的编程还是第一次。但我跟着实验手册一步一步的走，又不会的问题在 CSDN 和百度中寻找答案，很快便完成了这两个实验。

其次便是最具挑战性的 wordcount 编程。对于 MPI 编程，除了助教的一节辅导课外，我从来没有接触过。由于疫情原因，课堂上 MPI 编程也成了选修课程。这就要求我在网络上搜索相关资料进行学习、研究。而且在实际进行实验时，发现实验手册所给的内容也有一定的错误或者描述模糊不清的地方，所以说也不能完全相信实验手册，实践才能出真知。总之，经过努力，最终也成功完成了 wordcount 代码的编写，并使之在多主机集群中运行起来。

此次华为云的编程实验让我对多主机环境下的并行编程有了更深了解，并且也提高了我对高性能计算的兴趣。我想如果有机会的话，将来读研的时候也可以继续进行高性能计算相关领域的研究。高性能计算作为一个前沿的、发展迅速的技术，国家对这方面人才的需求量也是很大的，说不定将来还能为我国的高性能计算发展做出一定贡献呢。

在最后，感谢李翔宇老师和韩露露助教在本次课程设计中的指导与所做的努力，祝两位身体健康，科研顺利！

附　录

实验一代码：quick_sort.cpp

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>
#include <iostream>
#include "omp.h"
using namespace std;
void QuickSort(int *&array, int len)
{
    if(len<=1)
        return;
    int pivot=array[len/2];
```

```
    int left_ptr=0;
    int right_ptr=len-1;
    while (left_ptr<=right_ptr)
    {
        while (array[left_ptr]<pivot) left_ptr+=1;
        while (array[right_ptr]>pivot) right_ptr-=1;
        if(left_ptr<=right_ptr)
        {
            swap(array[left_ptr],array[right_ptr]);
            left_ptr+=1;
            right_ptr-=1;
        }
    }
    int *sub_array[]={array,&(array[left_ptr])};
    int sub_len[]={right_ptr+1,len-left_ptr};
    #pragma omp task default(none) firstprivate(sub_array,sub_len)
    { QuickSort(sub_array[0],sub_len[0]);}
    #pragma omp task default(none) firstprivate(sub_array,sub_len)
    { QuickSort(sub_array[1],sub_len[1]);}
}

int main(int argc,char *argv[])
{
    srand(time(NULL));
    int t=atoi(argv[1]);
    int n=atoi(argv[2]);
    int *array=new int[n];
    omp_set_num_threads(t);
    unsigned int seed=1024;
    #pragma omp parallel for
    for(int i=0;i<n;i++)
        array[i]=rand_r(&seed);
    struct timeval start, stop;
    gettimeofday(&start, NULL);
    #pragma omp parallel default(none) shared(array,n)
    {
        #pragma omp single nowait
        { QuickSort(array,n); }
    }
    gettimeofday(&stop, NULL);
    double time= (stop.tv_sec - start.tv_sec) * 1000 +
                 (stop.tv_usec - start.tv_usec) / 1000;
    cout <<"running time:"<<time<<";number of data:"<<n<<endl;
    int flag=0;
```

```
    for (int i=0;i<n-1;i++)
    {
        if (array[i]>array[i+1])
        {
            flag=1;
            break;
        }
    }
    if(flag==0)
        cout<<"quick sort success! \n";
    else
        cerr << "quick sort fails! \n";
    return 0;
}
```

实验二代码：MonteCarlo.cpp

```
#include "MonteCarlo.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>
#include <iostream>
#include "omp.h"
#define random(a,b) a+(b-a)*(rand()/(RAND_MAX+1.0))
using namespace std;
using namespace MC;
namespace MC {
namespace PI {
struct calPiArgs {
    int x,y;
};
struct calPiRets {
    bool hit;
};
void sampling(calPiArgs &arg,calPiRets &ret)
{
    ret.hit=((arg.x*arg.x+arg.y*arg.y)<=1008*1008);
}
void yieldSamples(calPiArgs &args){
    unsigned int seed = 1024;
    args.x = rand_r(&seed) % 1009;
    args.y = rand_r(&seed) % 1009;
}
```

```
void delSamples(calPiArgs &args) {}
}
}
int main(int argc,char *argv[])
{
    srand(time(NULL));
    int t=atoi(argv[1]);
    int n=atoi(argv[2]);
    double count=0.0;
    omp_set_num_threads(t);
    struct timeval start, stop;
    gettimeofday(&start, NULL);
    //MC::PI::calPiRets *rets;
    //rets = MC::RunMC<MC::PI::calPiArgs, MC::PI::calPiRets>(n);
    int sum=0;
    double x,y;
    #pragma omp parallel for private(x,y) reduction(+:sum)
    for(int i=0;i<n;i++)
    {
        x=random(0,1);
        y=random(0,1);
        if(x*x+y*y<=1)
        sum++;
    }
    gettimeofday(&stop, NULL);
    double pi=double(sum*4.0/n);
    printf("PI = %.6f",pi);
    double time = (stop.tv_sec - start.tv_sec) * 1000 +
                  (stop.tv_usec - start.tv_usec) / 1000;
    cout<<endl;
    cout<<"running time:"<<time<<";number of data:"<<n<<endl;
    return 0;
}
```

实验三代码：wordcount.cpp

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <string>
#include <cstring>
#include <fstream>
#include <sstream>
#include <iterator>
```

```cpp
#include <vector>
#include <map>
#include <unordered_map>
#include <dirent.h>
#include <iostream>
#include "mpi.h"
using namespace std;
void getFiles(string path,vector<string>& filenames)
{
    DIR *pDir;
    struct dirent* ptr;
    if(!(pDir=opendir(path.c_str())))
    {
        return;
    }
    while((ptr = readdir(pDir))!=0)
    {
        if(strcmp(ptr->d_name,".")!=0&&strcmp(ptr->d_name,"..")!= 0)
        {
            filenames.push_back(path +"/"+ptr->d_name);
        }
    }
    closedir(pDir);
}

std::string readFile(std::string filename) {
    std::ifstream in(filename);
    in.seekg(0,std::ios::end);
    size_t len=in.tellg();
    in.seekg(0);
    std::string contents(len+1,'\0');
    in.read(&contents[0],len);
    return contents;
}
std::vector<std::string> split(std::string const &input)
{
    std::istringstream buffer(input);
    std::vector<std::string>
ret((std::istream_iterator<std::string>(buffer)),
                                 std::istream_iterator<std::string>());
    return ret;
}

std::vector<std::string> getWords(
```

```
    std::string &content,int rank,int worldsize)
    {
    std::vector<std::string> wordList=split(content);
    std::vector<std::string> re;
    std::string tmp;
    for (int i=0;i<wordList.size();i++)
    {
        if (i % worldsize)
        {
            tmp+=" "+wordList[i];
        }
    }
    re.push_back(tmp);
    return re;
}
std::vector<pair<std::string,int>> countWords(
    std::vector<std::string> &contentList) {
    std::vector<std::string> wordList;
    std::string concat_content;
    for (auto it=contentList.begin();it!=contentList.end();it++)
    {
        std::string content=(*it);
        concat_content+=" "+content;
    }
    wordList=split(concat_content);
    std::map<std::string,int> counts;
    for (auto it=wordList.begin();it!=wordList.end();it++)
    {
        if(counts.find(*it)!=counts.end())
        {
            counts[*it]+=1;
        }
        else
        {
            counts[*it]=1;
        }
    }
    std::vector<pair<std::string,int>> res;
    for (auto it=counts.begin();it!=counts.end();it++)
    {
        res.push_back(std::make_pair(it->first,it->second));
    }
    return res;
}
```

```
std::vector<pair<std::string,int>> mergeCounts(
    std::vector<pair<std::string,int>> &countListA,
    std::vector<pair<std::string,int>> &countListB)
    {
    std::map<std::string, int> counts;
    for (auto it = countListA.begin();it!=countListA.end();it++)
    {
        counts[it->first]=it->second;
    }
    for (auto it = countListB.begin();it!=countListB.end();it++)
    {
        if (counts.find(it->first)==counts.end())
            counts[it->first]=it->second;
        else
            counts[it->first]+=it->second;
    }
    std::vector<pair<std::string, int>> res;
    for (auto it=counts.begin();it!=counts.end();it++)
    {
        res.push_back(std::make_pair(it->first,it->second));
    }
    return res;
}

void sendLocalCounts(int from,int to,std::vector<pair<std::string,
int>> &counts)
{
    int num=counts.size();
    MPI_Send(&num,1,MPI_INT,to,from,MPI_COMM_WORLD);
    if (num)
    {
        int *counts_array=new int[num];
        int i=0;
        for(auto it=counts.begin();it!=counts.end();it++,i++)
        {
            counts_array[i]=it->second;
        }
        MPI_Send(counts_array,num,MPI_INT,to,from,MPI_COMM_WORLD);
        delete counts_array;
    }

    std::string words =" ";
    for (auto it=counts.begin();it!=counts.end();it++)
```

```
    {
        words+=it->first;
        words+=" ";
    }
    num=words.length();
    MPI_Send(&num,1,MPI_INT,to,from,MPI_COMM_WORLD);
    if(num)
    {
        char *_words=new char[num];
        words.copy(_words, num);
        MPI_Send(_words, num,MPI_CHAR,to,from,MPI_COMM_WORLD);
        delete _words;
    }
}
void recvCounts(int from,int to,std::vector<pair<std::string,int>>
&counts)
{
    MPI_Status status;
    int _num=0,num=0;
    int *counts_array;
    char *_words;
    std::string words;
    MPI_Recv(&_num,1,MPI_INT,from,from,MPI_COMM_WORLD,&status);
    if (_num)
    {
        counts_array=new int[_num];
        MPI_Recv(counts_array,
_num,MPI_INT,from,from,MPI_COMM_WORLD,&status);
    }
    MPI_Recv(&num,1,MPI_INT,from,from,MPI_COMM_WORLD,&status);
    if (num)
    {
        _words=new char[num];
        MPI_Recv(_words,num,MPI_CHAR,from,from,MPI_COMM_WORLD,&status);
        for(int _i=0; _i<num; _i++) words+=_words[_i];
        delete _words;
    }
    if(_num)
    {
        std::vector<std::string> word_vec=split(words);
        for (int i=0;i<_num;i++)
        {
            counts.push_back(std::make_pair(word_vec[i],
counts_array[i]));
```

```
        }
        delete counts_array;
    }
}

void treeMerge(int id,int worldSize,
               std::vector<pair<std::string, int>> &counts){
    int participants=worldSize;
    while (participants>1) {
        MPI_Barrier(MPI_COMM_WORLD);
        int _participants=participants/2+(participants%2?1:0);
        if (id<_participants) {
            if (id+_participants<participants)
            {
                std::vector<pair<std::string,int>> _counts;
                std::vector<pair<std::string,int>> temp;
                recvCounts(id+ _participants,id, _counts);
                temp = mergeCounts(_counts, counts);
                counts = temp;
            }
        }
        if (id>=_participants&&id<participants)
        {
            sendLocalCounts(id,id-_participants,counts);
        }
        participants=_participants;
    }
}

int main(int argc,char *argv[])
{
    int rank;
    int worldSize;
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&worldSize);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
{
    struct timeval start,stop;
    gettimeofday(&start,NULL);
    std::string big_file="./project_file/big_file/big_100.txt";
    auto content=readFile(big_file);
    auto partContent=getWords(content,rank,worldSize);
    auto counts=countWords(partContent);
    treeMerge(rank,worldSize,counts);
```

```
    gettimeofday(&stop,NULL);
    if (rank==0){
        cout<<"Bigfile:"<<endl;
        cout<<"running time:"
            <<(stop.tv_sec - start.tv_sec)*1000.0+
                (stop.tv_usec - start.tv_usec)/1000.0
            <<" ms"<<endl;
        int sum=counts.size();
        cout<<"number of different words:"<<sum<<endl;
    }
}
{
    struct timeval start,stop;
    gettimeofday(&start,NULL);
    std::string small_file_folder="./project_file/small_file/";
    std::vector<std::string> _small_files,small_files;
    getFiles(small_file_folder,_small_files);
    for(auto it=_small_files.begin();it!=_small_files.end();it++)
    {
        std::size_t _hs=std::hash<std::string>{}(*it);
        if(int(_hs%worldSize)==rank)
        {
            small_files.push_back(*it);
        }
    }
    std::vector<std::string> wdlst;
    for (int i=0;i<small_files.size();i++)
    {
        auto content=readFile(small_files[i]);
        wdlst.push_back(content);
    }
    auto counts = countWords(wdlst);
    treeMerge(rank,worldSize,counts);
    gettimeofday(&stop,NULL);
    if(rank==0){
        cout <<"SmallFiles"<<endl;
        cout <<"running time:"
            <<(stop.tv_sec - start.tv_sec)*1000.0+
                (stop.tv_usec - start.tv_usec)/1000.0
            <<" ms"<<endl;
        int sum=counts.size();
        cout<<"number of different words:"<<sum<<endl;
    }
}
```

```
    MPI_Finalize();
    return 0;
}
```