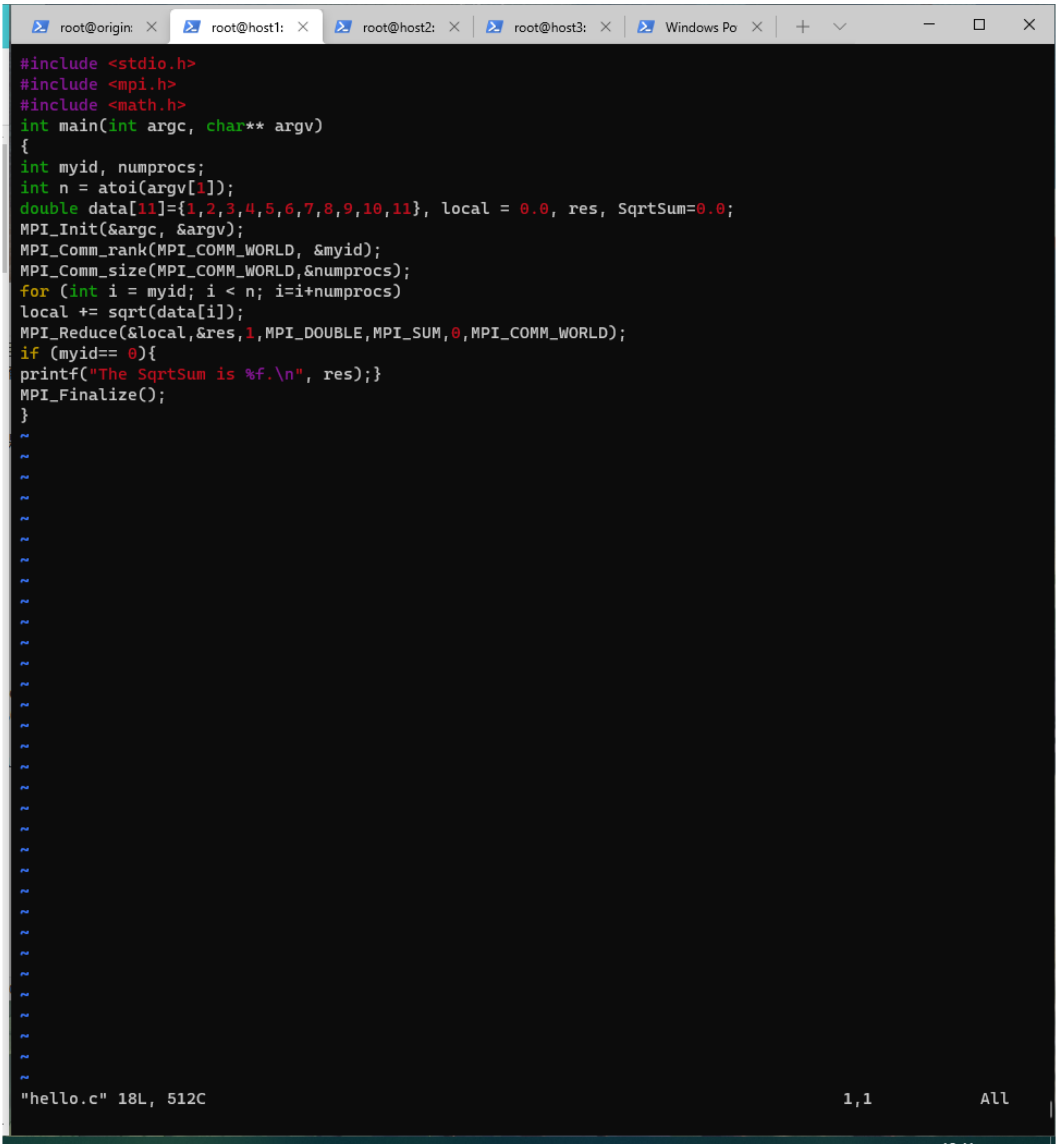
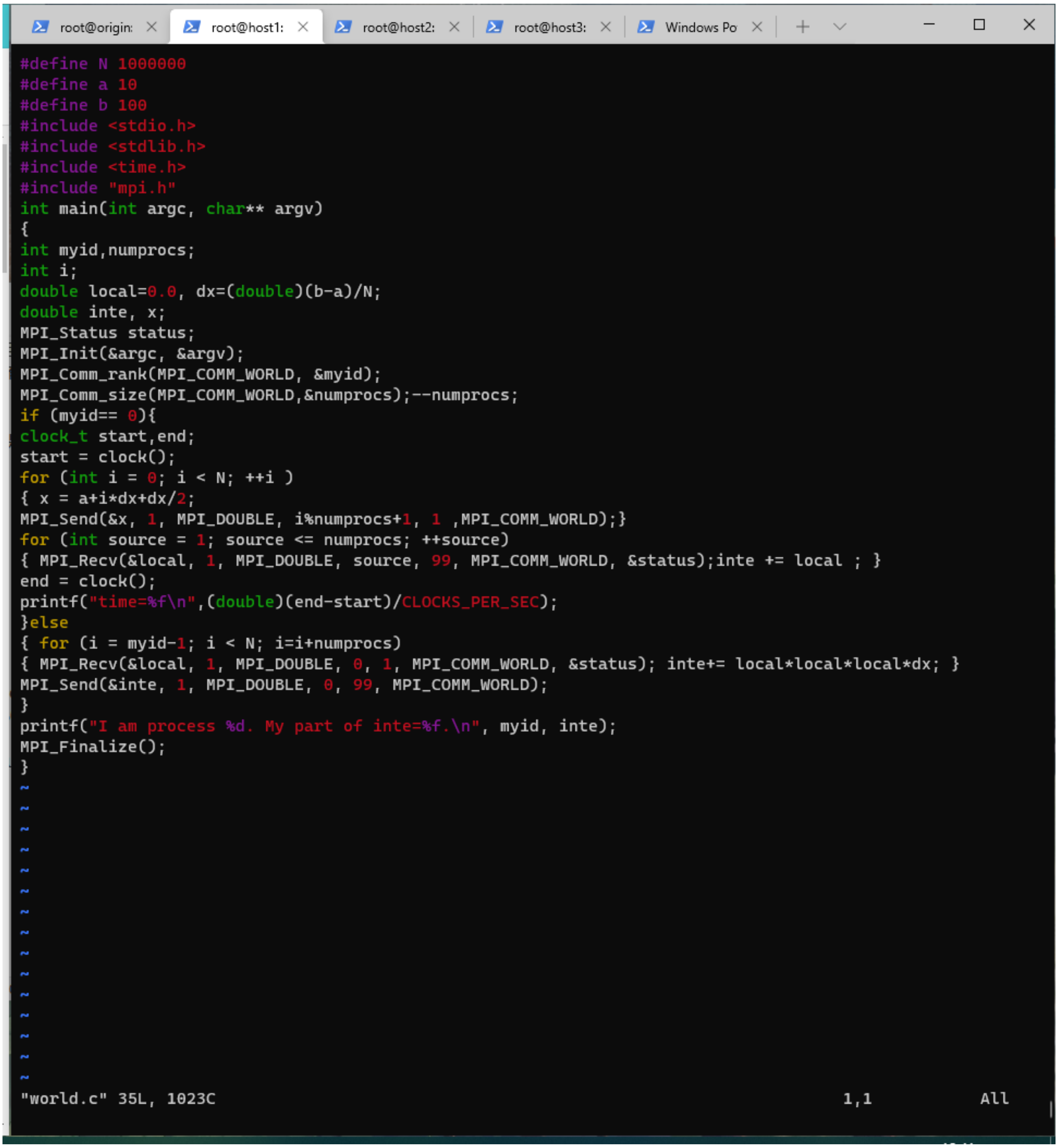
1　代码

见 hello.c 和 world.c，分别是改写的大数组开方求和与积分的程序。



```
#include <stdio.h>
#include <mpi.h>
#include <math.h>
int main(int argc, char** argv)
{
int myid, numprocs;
int n = atoi(argv[1]);
double data[11]={1,2,3,4,5,6,7,8,9,10,11}, local = 0.0, res, SqrtSum=0.0;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myid);
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
for (int i = myid; i < n; i=i+numprocs)
local += sqrt(data[i]);
MPI_Reduce(&local,&res,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
if (myid== 0){
printf("The SqrtSum is %f.\n", res);}
MPI_Finalize();
}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"hello.c" 18L, 512C                                        1,1           All
```

图 1: hello.c



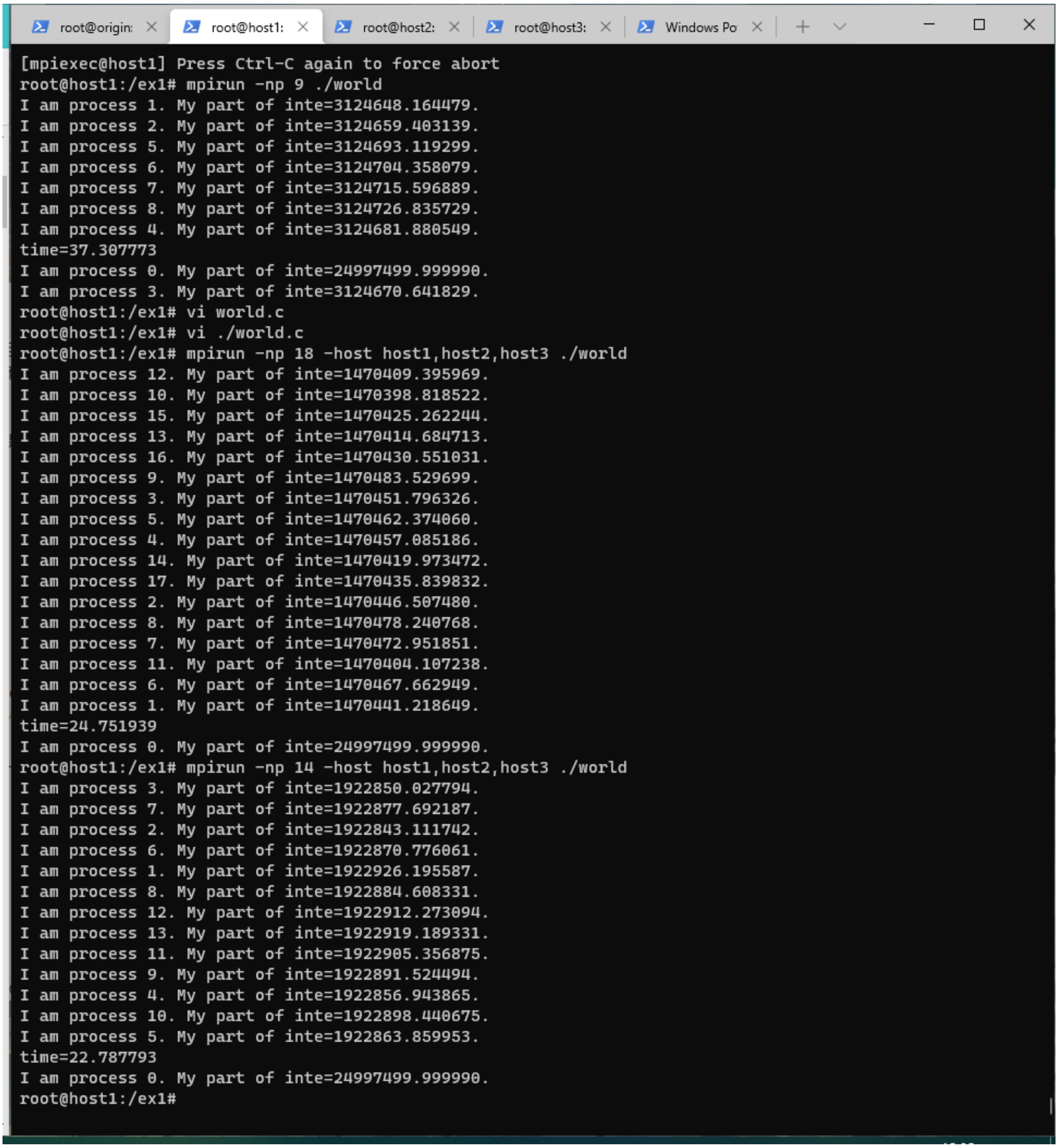
```
#define N 1000000
#define a 10
#define b 100
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "mpi.h"
int main(int argc, char** argv)
{
int myid,numprocs;
int i;
double local=0.0, dx=(double)(b-a)/N;
double inte, x;
MPI_Status status;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myid);
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);--numprocs;
if (myid== 0){
clock_t start,end;
start = clock();
for (int i = 0; i < N; ++i )
{ x = a+i*dx+dx/2;
MPI_Send(&x, 1, MPI_DOUBLE, i%numprocs+1, 1 ,MPI_COMM_WORLD);}
for (int source = 1; source <= numprocs; ++source)
{ MPI_Recv(&local, 1, MPI_DOUBLE, source, 99, MPI_COMM_WORLD, &status);inte += local ; }
end = clock();
printf("time=%f\n",(double)(end-start)/CLOCKS_PER_SEC);
}else
{ for (i = myid-1; i < N; i=i+numprocs)
{ MPI_Recv(&local, 1, MPI_DOUBLE, 0, 1, MPI_COMM_WORLD, &status); inte+= local*local*local*dx; }
MPI_Send(&inte, 1, MPI_DOUBLE, 0, 99, MPI_COMM_WORLD);
}
printf("I am process %d. My part of inte=%f.\n", myid, inte);
MPI_Finalize();
}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"world.c" 35L, 1023C                                              1,1           All
```

图 2: world.c

2 运行截图

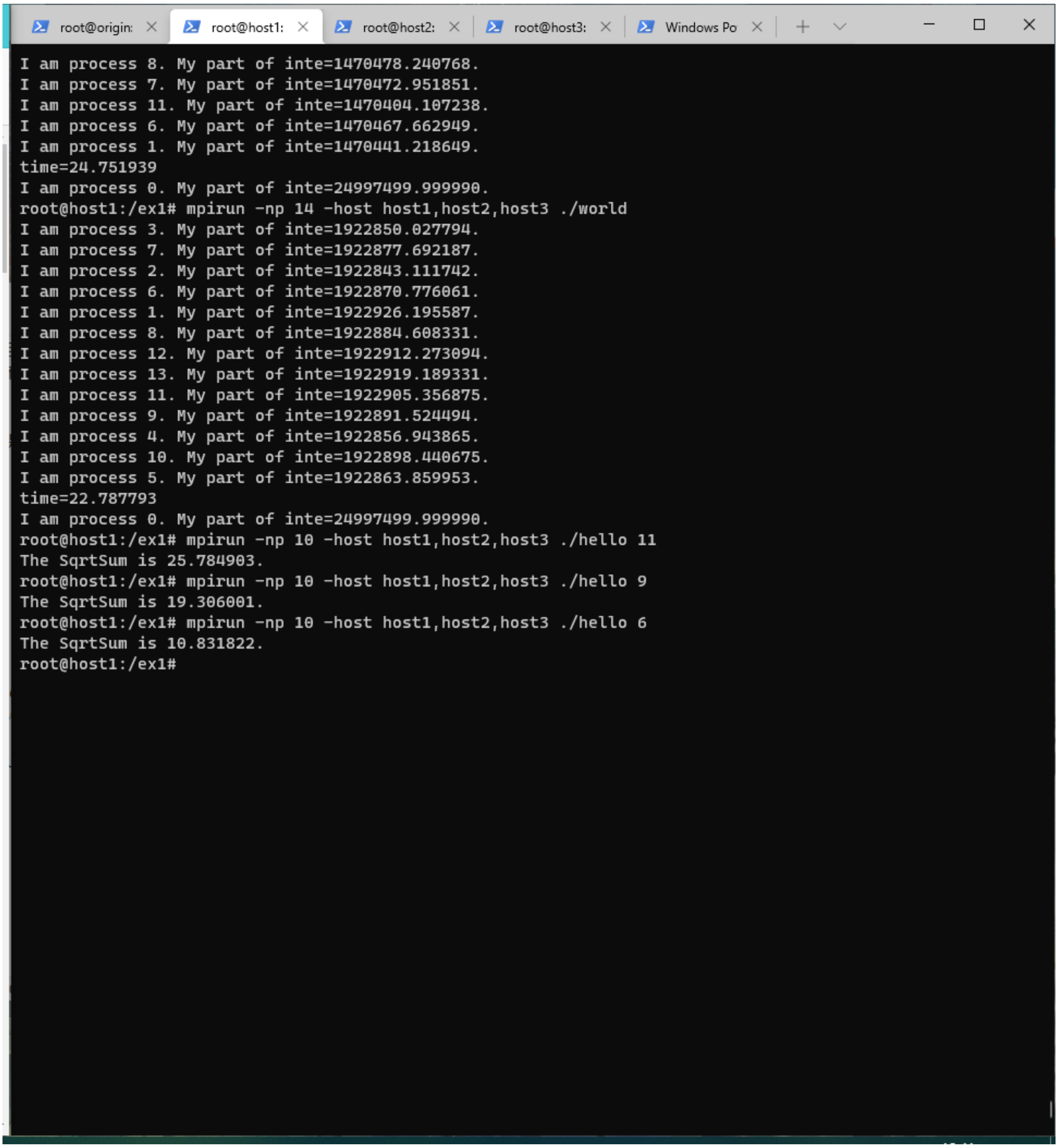
事先编译：

```
mpicc hello.c -o hello -std=c99 -lm
mpicc world.c -o world -std=c99
```



```
[mpiexec@host1] Press Ctrl-C again to force abort
root@host1:/ex1# mpirun -np 9 ./world
I am process 1. My part of inte=3124648.164479.
I am process 2. My part of inte=3124659.403139.
I am process 5. My part of inte=3124693.119299.
I am process 6. My part of inte=3124704.358079.
I am process 7. My part of inte=3124715.596889.
I am process 8. My part of inte=3124726.835729.
I am process 4. My part of inte=3124681.880549.
time=37.307773
I am process 0. My part of inte=24997499.999990.
I am process 3. My part of inte=3124670.641829.
root@host1:/ex1# vi world.c
root@host1:/ex1# vi ./world.c
root@host1:/ex1# mpirun -np 18 -host host1,host2,host3 ./world
I am process 12. My part of inte=1470409.395969.
I am process 10. My part of inte=1470398.818522.
I am process 15. My part of inte=1470425.262244.
I am process 13. My part of inte=1470414.684713.
I am process 16. My part of inte=1470430.551031.
I am process 9. My part of inte=1470483.529699.
I am process 3. My part of inte=1470451.796326.
I am process 5. My part of inte=1470462.374060.
I am process 4. My part of inte=1470457.085186.
I am process 14. My part of inte=1470419.973472.
I am process 17. My part of inte=1470435.839832.
I am process 2. My part of inte=1470446.507480.
I am process 8. My part of inte=1470478.240768.
I am process 7. My part of inte=1470472.951851.
I am process 11. My part of inte=1470404.107238.
I am process 6. My part of inte=1470467.662949.
I am process 1. My part of inte=1470441.218649.
time=24.751939
I am process 0. My part of inte=24997499.999990.
root@host1:/ex1# mpirun -np 14 -host host1,host2,host3 ./world
I am process 3. My part of inte=1922850.027794.
I am process 7. My part of inte=1922877.692187.
I am process 2. My part of inte=1922843.111742.
I am process 6. My part of inte=1922870.776061.
I am process 1. My part of inte=1922926.195587.
I am process 8. My part of inte=1922884.608331.
I am process 12. My part of inte=1922912.273094.
I am process 13. My part of inte=1922919.189331.
I am process 11. My part of inte=1922905.356875.
I am process 9. My part of inte=1922891.524494.
I am process 4. My part of inte=1922856.943865.
I am process 10. My part of inte=1922898.440675.
I am process 5. My part of inte=1922863.859953.
time=22.787793
I am process 0. My part of inte=24997499.999990.
root@host1:/ex1#
```

图 3: 多节点不同进程数对比运行 mpi 积分



```
I am process 8. My part of inte=1470478.240768.
I am process 7. My part of inte=1470472.951851.
I am process 11. My part of inte=1470404.107238.
I am process 6. My part of inte=1470467.662949.
I am process 1. My part of inte=1470441.218649.
time=24.751939
I am process 0. My part of inte=24997499.999990.
root@host1:/ex1# mpirun -np 14 -host host1,host2,host3 ./world
I am process 3. My part of inte=1922850.027794.
I am process 7. My part of inte=1922877.692187.
I am process 2. My part of inte=1922843.111742.
I am process 6. My part of inte=1922870.776061.
I am process 1. My part of inte=1922926.195587.
I am process 8. My part of inte=1922884.608331.
I am process 12. My part of inte=1922912.273094.
I am process 13. My part of inte=1922919.189331.
I am process 11. My part of inte=1922905.356875.
I am process 9. My part of inte=1922891.524494.
I am process 4. My part of inte=1922856.943865.
I am process 10. My part of inte=1922898.440675.
I am process 5. My part of inte=1922863.859953.
time=22.787793
I am process 0. My part of inte=24997499.999990.
root@host1:/ex1# mpirun -np 10 -host host1,host2,host3 ./hello 11
The SqrtSum is 25.784903.
root@host1:/ex1# mpirun -np 10 -host host1,host2,host3 ./hello 9
The SqrtSum is 19.306001.
root@host1:/ex1# mpirun -np 10 -host host1,host2,host3 ./hello 6
The SqrtSum is 10.831822.
root@host1:/ex1#
```

图 4: 命令行传入 N 控制开方求和

使用 MPICH。为方便并没有人工生成长数组，用小数组模拟。读取命令行传参并用 atoi 函数进行类型转换；多机运行时注意配置好文件后仍需对各主机进行公钥登录。

可以发现并不是线程越多所用时间越少，反而存在最佳点。考虑到并行计算的部分, 线程同步的部分，串行部分种种影响因素，确实在线程过多时会引发各种问题。

3 问题总结及解决方案

3.1 准备工作

默认情况下 Ubuntu14.04 上也安装有 vi 编辑器但是 vim-common 版本，基本上用不了 vim，vi 用起来也会有一些 BUG，比如上移、下移、左移和右移键使用起来有些问题。所以先把这个版本的 vi 卸载掉

```
sudo apt-get remove vim-common
```

卸载成功之后接着执行

```
sudo apt-get install vim
```

安装好之后就能正常使用了。

3.2 代码修改

修补课件代码中的语法性错误，如 &data[i]、&SqrtSum，补充一些变量声明，注意 numprocs 引发的循环边界问题等等，并采用 c99 标准进行编译。

```
mpicc hello.c -o hello -std=c99 -lm
```

3.3 计时方式

直接在程序最前最后计时只会并行计算各进程的运行时间。由于我们仅考虑进程数与运行时间的粗略关系，不妨仅针对 0 进程计时，忽略其它相同工作的准备时间，单独计时主节点开始分发任务到最终获得结果来大致反映程序总体的运行情况。

4 其他思考

send-recv 点对点方法需要在初期指明主节点，并由 0 号节点亲自发送数据，指配用于传递信息的中间变量，而 reduce 节点集合通信方法每个节点直接从各自本地程序中读取，减少消息分发并且无需事先指定主节点，所有节点都参与了计算，更加高效且可靠。