# 《并行计算》实验指导书

# 实验3：基于华为云环境的MPI并行编程

## 实验1.1基于华为云的MPI并行集群环境

### 实验目的

1．了解华为云环境的使用过程；

2．掌握利用华为云环境搭建小型集群环境的过程；

### 实验内容

实验步骤：

推荐按照下列步骤使用一键部署环境脚本，或可参考实验1指导书手动部署环境。

1. **下载脚本并赋予可执行权限**（注意wget与后面网址之间的空格）

以下步骤应在第一台机器上完成

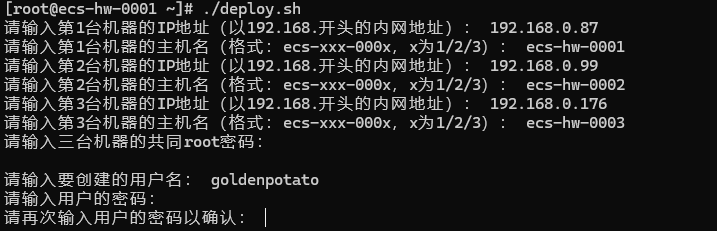
|  |
| --- |
| wget [gist.gitmirror.com/GoldenPotato137/4ae8ccb7cc1152d5e902a3b07aafda46/raw/b5efb7d759d21247bc068eff652aba9d26bb55c5/deploy.sh](https://gist.gitmirror.com/GoldenPotato137/4ae8ccb7cc1152d5e902a3b07aafda46/raw/b5efb7d759d21247bc068eff652aba9d26bb55c5/deploy.sh)  chmod +x deploy.sh |

1. **运行脚本，按照脚本提示填入相关内容**

注意检查ip与服务器名的对应关系

密码是不显示到屏幕上的，正常输入即可。若输入错误可多次按动Backspace键彻底清除之前的输入后再重新输入。

|  |
| --- |
| ./deploy.sh |



等待脚本自动部署所有环境即可。

切换到自己的用户执行后续实验

|  |
| --- |
| su - changtian |

## 实验1.2 N体问题MPI并行程序剖析与性能分析

### 实验目的

1. 掌握MPI程序设计的基本编写、编译与运行方法;

2．了解集群环境下N体问题的并行程序设计方法；

3．掌握利用加速比、运行时间、效率等测度分析并行程序性能

### 实验内容

**实验原理：**

N体问题是研究物体之间相互作用力产生的效果的问题（例如天体间通过引力相互吸引）。这个问题的目标是确定太空中通过引力相互作用的天体的位置和运动，涉及到的物理知识是牛顿定律。质量为ma和mb的两个物体相互间的引力由下式给出：



其中G是引力常数，r是物体间的距离。每个物体都会依据这个定律受其它物体的影响，所有力都会被累加（要考虑每个力的方向）。在受力的情况下，一个物体将根据牛顿第二定律加速：



其中m是物体质量，F是物体所受的力，a是得到的加速度。因此所有物体都会由于这些力的作用移动到新的位置并获得新的速度。要得到精确的数值描述，需要使用微分方程。然而，对于超过三个物体的系统的N体问题还没有确切的解析解。

使用计算机仿真时，我们使用特定时间的值，t0，t1，t2等等，时间间隔取得尽可能短以获得最精确的解。不妨设时间间隔为△t，则对于一个质量为m的特定物体，其所受力为：



（解释：冲量等于动量的变化量）

新的速度为：



其中vt+1为物体在时刻t+1的速度，vt为物体在时刻t的速度。如果物体以速度v移动了△t时间，则位置变化为：



其中xt为物体在时刻t的位置。一旦物体移动到新的位置，所受的力就会发生变化，计算过程也要重复。

在一个时间间隔△t内的速度实际上并不是一个准确的常数，因此得到的仅是一个近似解。要注意以上所涉及的力F，位置x和速度v都是矢量。在进行计算时必须被分解为三个方向。

**N体问题的串行算法研究：**

总的引力N体算法可以由以下算法伪代码描述：

|  |
| --- |
| for (t = 0; t < tmax; t++) { /\*for each time period\*/  for (i = 0; i < n; i++) { /\*for each body\*/  F = Force\_routine(i); /\*compute force on ith body\*/  v[i]new = v[i] + F \*dt / m; /\*compute new velocity and\*/  x[i]new = x[i] + v[i]new \* dt; /\*new position \*/  }  for (i = 0; i < n; i++) { /\*for each body\*/  x[i] = x[i]new; /\*update velocity and position\*/  v[i] = v[i]new;  }  } |

外层的for循环计算每一个时间步，在每个时间步内n个物体更新了速度和位置。内层的两个for循环计算每一个物体所受到的力及其新的位置和速度。内层第一个for循环内的函数Force\_routine(i)计算其它所有物体作用于第i个物体的引力之和，它的时间复杂性是O(n)。因此在这个算法中n个物体更新一次时间复杂性是O(n2)。

**N体问题的并行实现：**

这个算法使用SPMD（单程序多数据流）计算模型，每个进程将执行相同的代码。假设有m个计算资源，每个资源上启动一个进程，表示进程的标号是0、1、2、……m-1，每个进程分配的物体数是n0、n1、n2、……n(m-1)，其中n0 + n1 + n2 + ……+ n(m-1) = n。每个进程主要有数组localparticles、allparticles、sendbuf、recvbuf，其中localparticles用于保存分配给本进程的物体的信息；allparticles用于保存应用程序中所有物体的信息；sendbuf用于保存发送到下一个进程的物体；recvbuf用于保存从前一个进程接受的物体。

并行算法描述如下：

|  |
| --- |
| 获取分配给本进程的物体的初始信息localparticles；  获取应用程序中所有物体的信息allparticles；  for (每一个时间步)  {  计算所有物体对分配给本进程的物体的作用力并据此更新localparticles的本进程的物体的信息；  将本进程信息localparticles保存到发送缓冲区sendbuf，同时更新allparticles中的部分信息；  for ( i = 0; i < m - 1; i++ ) //对每个进程  {  send sendbuf to 本进程的下一个进程；  recv recvbuf from 本进程的前一个进程；  用recvbuf中的信息更新allparticles中的部分信息；  }  使所有进程在此处同步；  } |

算法中外层for循环计算每一个时间步，在每个时间步内所有参与的进程并行执行，所有进程都结束时所有n个物体速度和位置都被更新了。在外层for的循环体内，首先计算所有物体对本地物体的作用力，据此更新关于本地物体的信息；内层for循环使得每个进程都获得了所有物体的更新后的信息，实现了进程之间的合作，为下一次迭代做好准备；最后通过显示设置栅栏使所有进程同步。

**性能评估参数：**

加速比和效率是最传统的并行算法评价标准，它们体现了在并行机上运用并行算法求解实际问题所能获得的好处。

并行算法的加速比定义为:

Sp=T1/Tp

T1表示最优串行算法在单处理器上的运行时间，Tp表示p个处理器的计算时间。

并行算法的效率定义为:

Ep=Sp/P

P为处理器数量。

如果按照某种条件，如保持每个处理器的计算规模，并行算法加速比与处理器的数量成正比，则称该并行算法在该条件下，在该并行机上具有线性加速比。

实验步骤：

步骤一：配置MPI环境，编译nbody程序

|  |
| --- |
| mkdir nbody  cd nbody  mpicc nbody.c -o nbody -lm |

将nbody目录发送到其他的节点的相同目录，具体命令参见实验1。

# 单机配置文件

**echo "ecs-hw-0001:1" > /home/changtian/nbody/config\_simple\_1**

**echo "ecs-hw-0001:2" > /home/changtian/nbody/config\_simple\_2**

**echo "ecs-hw-0001:3" > /home/changtian/nbody/config\_simple\_3**

**echo "ecs-hw-0001:4" > /home/changtian/nbody/config\_simple\_4**

# 双机配置文件

**echo -e "ecs-hw-0001:1\necs-hw-0002:1" > /home/changtian/nbody/config\_double\_1**

**echo -e "ecs-hw-0001:2\necs-hw-0002:2" > /home/changtian/nbody/config\_double\_2**

**echo -e "ecs-hw-0001:3\necs-hw-0002:3" > /home/changtian/nbody/config\_double\_3**

**echo -e "ecs-hw-0001:4\necs-hw-0002:4" > /home/changtian/nbody/config\_double\_4**

# 三机配置文件

**echo -e "ecs-hw-0001:1\necs-hw-0002:1\necs-hw-0003:1" > /home/changtian/nbody/config\_triple\_1**

**echo -e "ecs-hw-0001:2\necs-hw-0002:2\necs-hw-0003:2" > /home/changtian/nbody/config\_triple\_2**

**echo -e "ecs-hw-0001:3\necs-hw-0002:3\necs-hw-0003:3" > /home/changtian/nbody/config\_triple\_3**

**echo -e "ecs-hw-0001:4\necs-hw-0002:4\necs-hw-0003:4" > /home/changtian/nbody/config\_triple\_4**

**scp -r /home/changtian/nbody changtian@ecs-hw-0002:/home/changtian/**

**scp -r /home/changtian/nbody changtian@ecs-hw-0003:/home/changtian/**

步骤二：进行如下实验并记录数据（实验报告中给出数据并绘图）

Nbody程序运行命令示例：

|  |
| --- |
| mpirun -np 6 -f /home/changtian/nbody/config /home/changtian/nbody/nbody 6000 1000 |

说明：倒数第二个参数（6000）是body的总数量，也就是数据规模，最后一个参数（1000）是每个进程处理的数据量，这里三个节点一共启动了6个进程。

实验一：单机上，数据规模为6000时，随每机进程数变化的运行时间；

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 进程数 | 1 | 2 | 3 | 4 |
| 时间 | 7.507479 | 3.765938 | 2.511349 | 1.884503 |

**mpirun -np 1 -f /home/changtian/nbody/config\_simple\_1 /home/changtian/nbody/nbody 6000 6000**

**mpirun -np 2 -f /home/changtian/nbody/config\_simple\_2 /home/changtian/nbody/nbody 6000 3000**

**mpirun -np 3 -f /home/changtian/nbody/config\_simple\_3 /home/changtian/nbody/nbody 6000 2000**

**mpirun -np 4 -f /home/changtian/nbody/config\_simple\_4 /home/changtian/nbody/nbody 6000 1500**

图片包含 文本

描述已自动生成

实验二：相同数据规模为6000，随每机进程数变化的运行时间

|  |  |  |  |
| --- | --- | --- | --- |
| 每机进程数 | 单机 | 双机 | 三机 |
| 1 | 7.507479 | 3.776565 | 2.522918 |
| 2 | 3.765938 | 1.889411 | 1.278232 |
| 3 | 2.511349 | 1.272890 | 0.861549 |
| 4 | 1.884503 | 0.952066 | 0.639768 |

**mpirun -np 2 -f /home/changtian/nbody/config\_double\_1 /home/changtian/nbody/nbody 6000 3000**

**mpirun -np 4 -f /home/changtian/nbody/config\_double\_2 /home/changtian/nbody/nbody 6000 1500**

**mpirun -np 6 -f /home/changtian/nbody/config\_double\_3 /home/changtian/nbody/nbody 6000 1000**

**mpirun -np 8 -f /home/changtian/nbody/config\_double\_4 /home/changtian/nbody/nbody 6000 750**

图形用户界面

低可信度描述已自动生成

图形用户界面, 文本

描述已自动生成

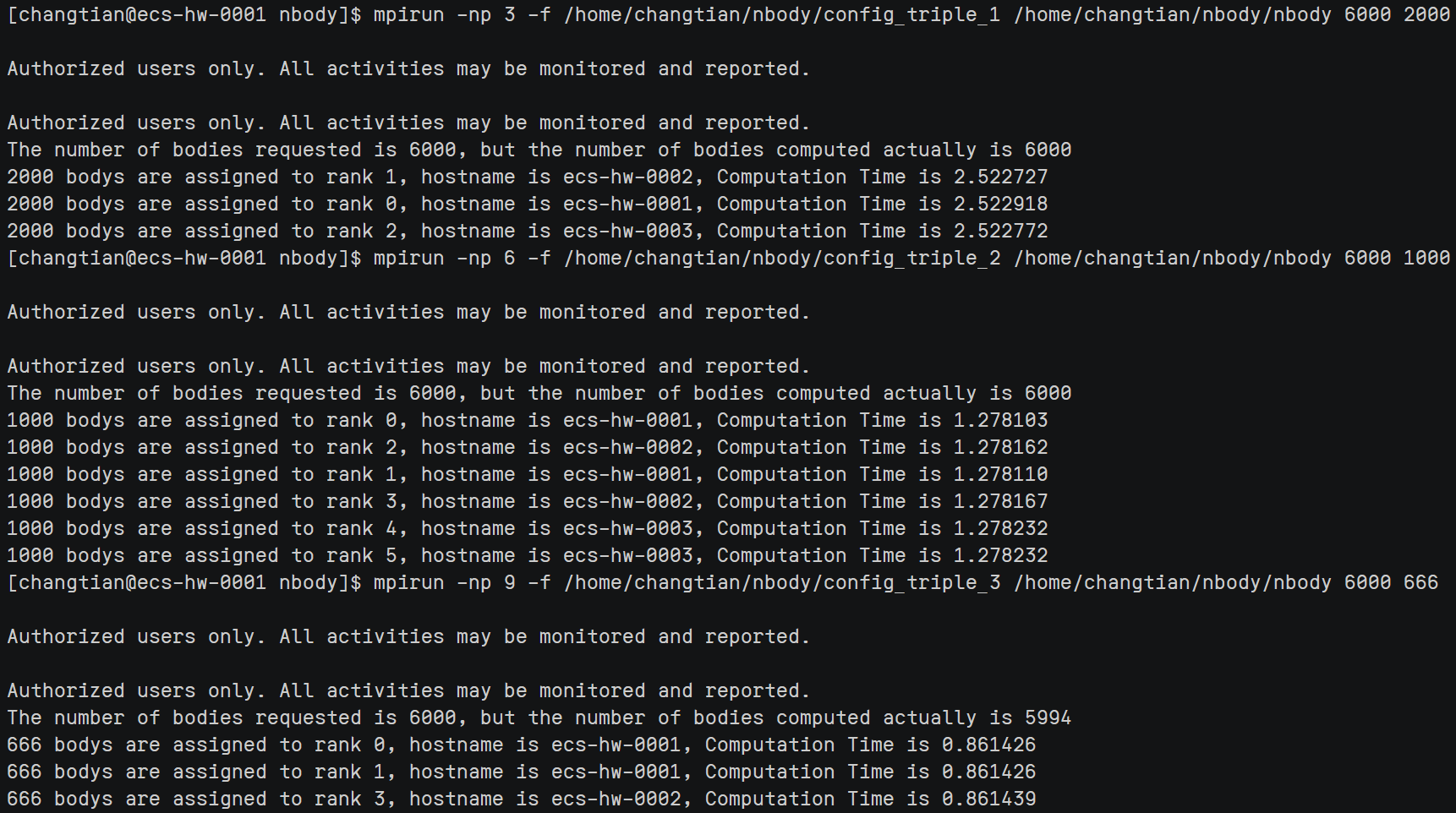
**mpirun -np 3 -f /home/changtian/nbody/config\_triple\_1 /home/changtian/nbody/nbody 6000 2000**

**mpirun -np 6 -f /home/changtian/nbody/config\_triple\_2 /home/changtian/nbody/nbody 6000 1000**

**mpirun -np 9 -f /home/changtian/nbody/config\_triple\_3 /home/changtian/nbody/nbody 6000 666**

文本

描述已自动生成**mpirun -np 12 -f /home/changtian/nbody/config\_triple\_4 /home/changtian/nbody/nbody 6000 500**



实验三：每机1个进程，随数据规模变化的n-body并行程序运行时间。

|  |  |  |  |
| --- | --- | --- | --- |
| 粒子数n | 单机 | 双机 | 三机 |
| 150 | 0.004830 | 0.016591 | 0.018397 |
| 300 | 0.018955 | 0.019619 | 0.011379 |
| 600 | 0.075471 | 0.053722 | 0.043577 |
| 1200 | 0.301185 | 0.153330 | 0.107714 |
| 2400 | 1.209091 | 0.605584 | 0.409054 |
| 4800 | 4.818647 | 2.414285 | 1.617593 |
| 9600 | 19.285291 | 9.653732 | 6.449484 |

**mpirun -np 1 -f /home/changtian/nbody/config\_simple\_1 /home/changtian/nbody/nbody 150 150**

**mpirun -np 1 -f /home/changtian/nbody/config\_simple\_1 /home/changtian/nbody/nbody 300 300**

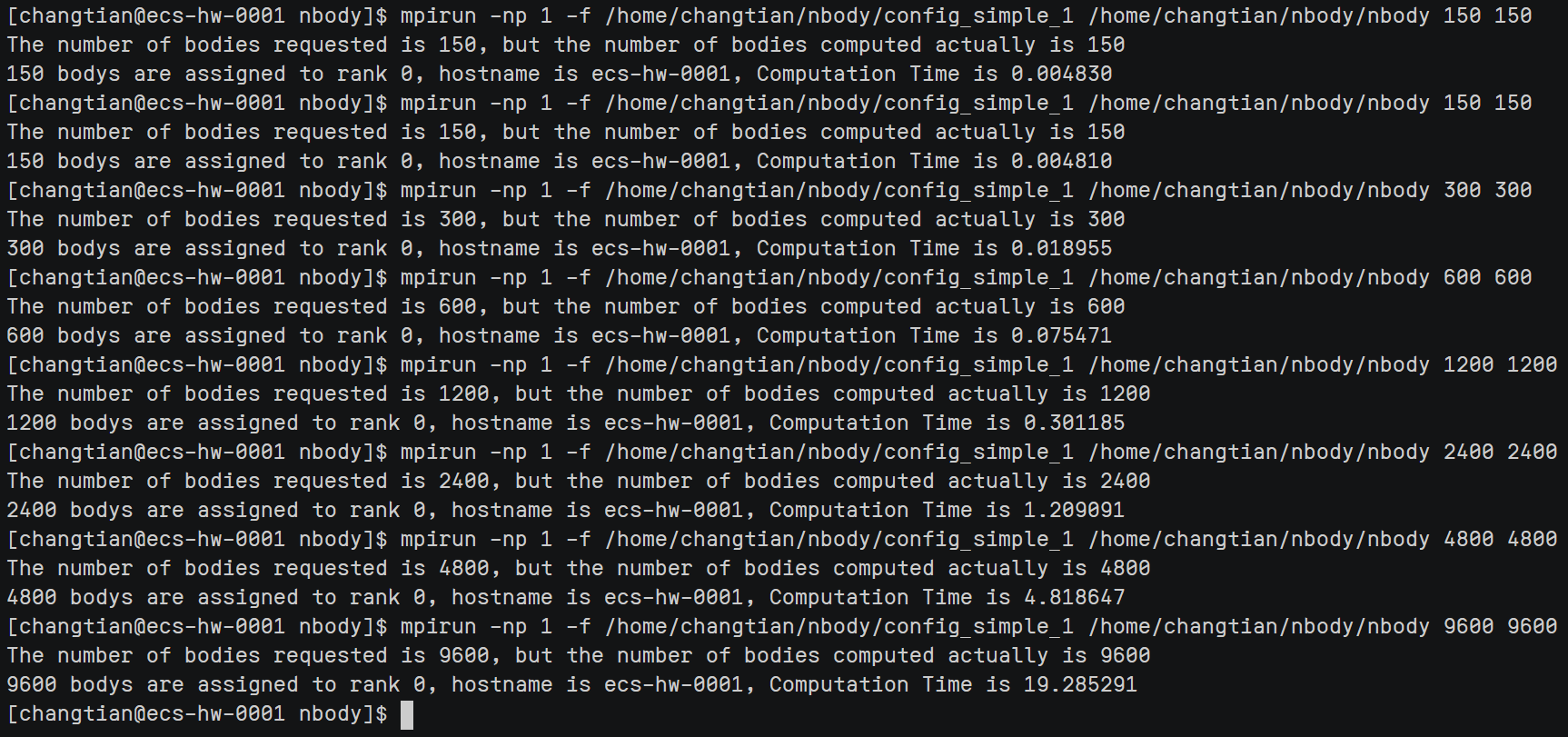
**mpirun -np 1 -f /home/changtian/nbody/config\_simple\_1 /home/changtian/nbody/nbody 600 600**

**mpirun -np 1 -f /home/changtian/nbody/config\_simple\_1 /home/changtian/nbody/nbody 1200 1200**

**mpirun -np 1 -f /home/changtian/nbody/config\_simple\_1 /home/changtian/nbody/nbody 2400 2400**

**mpirun -np 1 -f /home/changtian/nbody/config\_simple\_1 /home/changtian/nbody/nbody 4800 4800**

**mpirun -np 1 -f /home/changtian/nbody/config\_simple\_1 /home/changtian/nbody/nbody 9600 9600**



**mpirun -np 2 -f /home/changtian/nbody/config\_double\_1 /home/changtian/nbody/nbody 150 75**

**mpirun -np 2 -f /home/changtian/nbody/config\_double\_1 /home/changtian/nbody/nbody 300 150**

**mpirun -np 2 -f /home/changtian/nbody/config\_double\_1 /home/changtian/nbody/nbody 600 300**

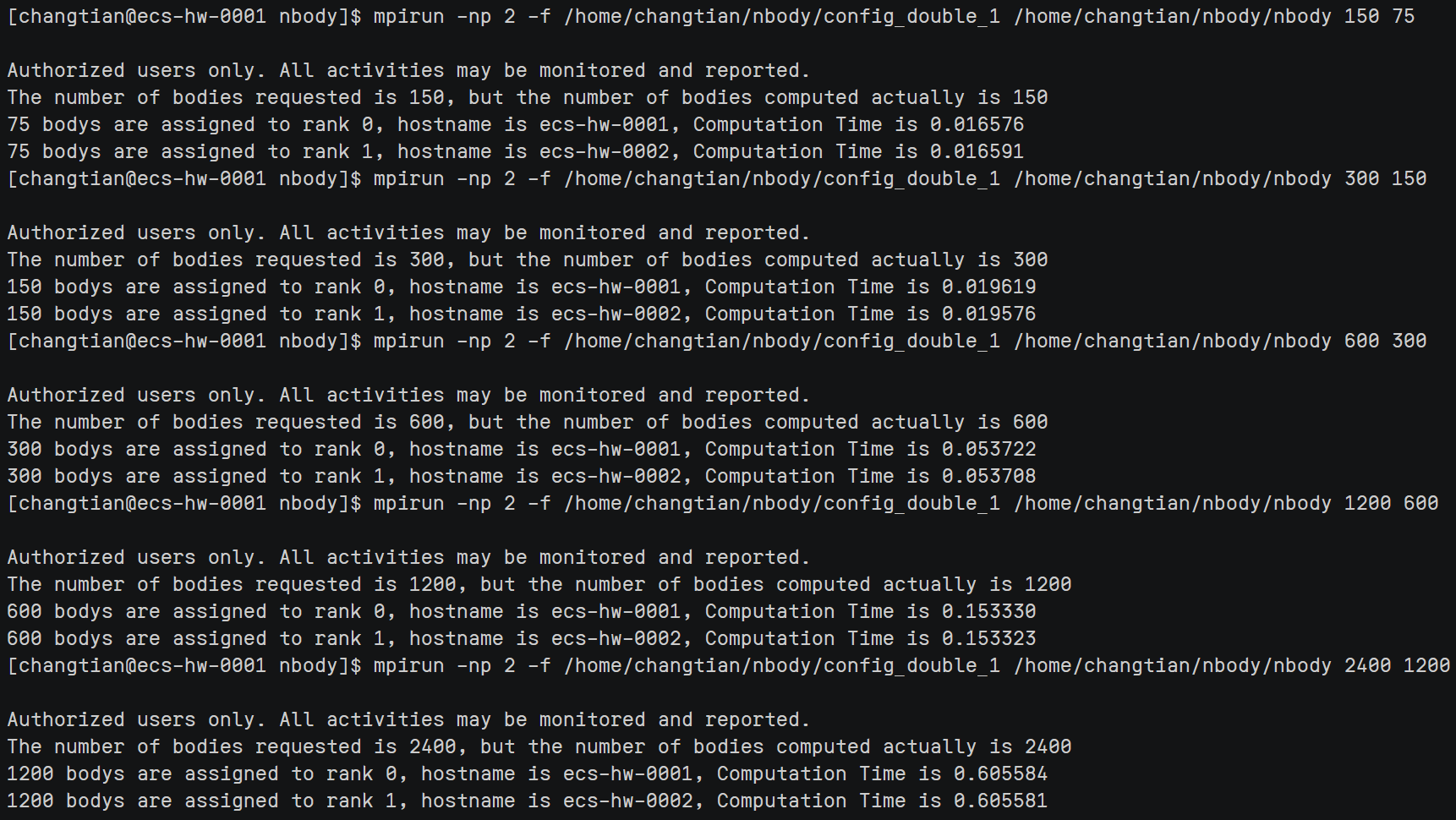
**mpirun -np 2 -f /home/changtian/nbody/config\_double\_1 /home/changtian/nbody/nbody 1200 600**

**mpirun -np 2 -f /home/changtian/nbody/config\_double\_1 /home/changtian/nbody/nbody 2400 1200**

**mpirun -np 2 -f /home/changtian/nbody/config\_double\_1 /home/changtian/nbody/nbody 4800 2400**

图片包含 图形用户界面

描述已自动生成**mpirun -np 2 -f /home/changtian/nbody/config\_double\_1 /home/changtian/nbody/nbody 9600 4800**



**mpirun -np 3 -f /home/changtian/nbody/config\_triple\_1 /home/changtian/nbody/nbody 150 50**

**mpirun -np 3 -f /home/changtian/nbody/config\_triple\_1 /home/changtian/nbody/nbody 300 100**

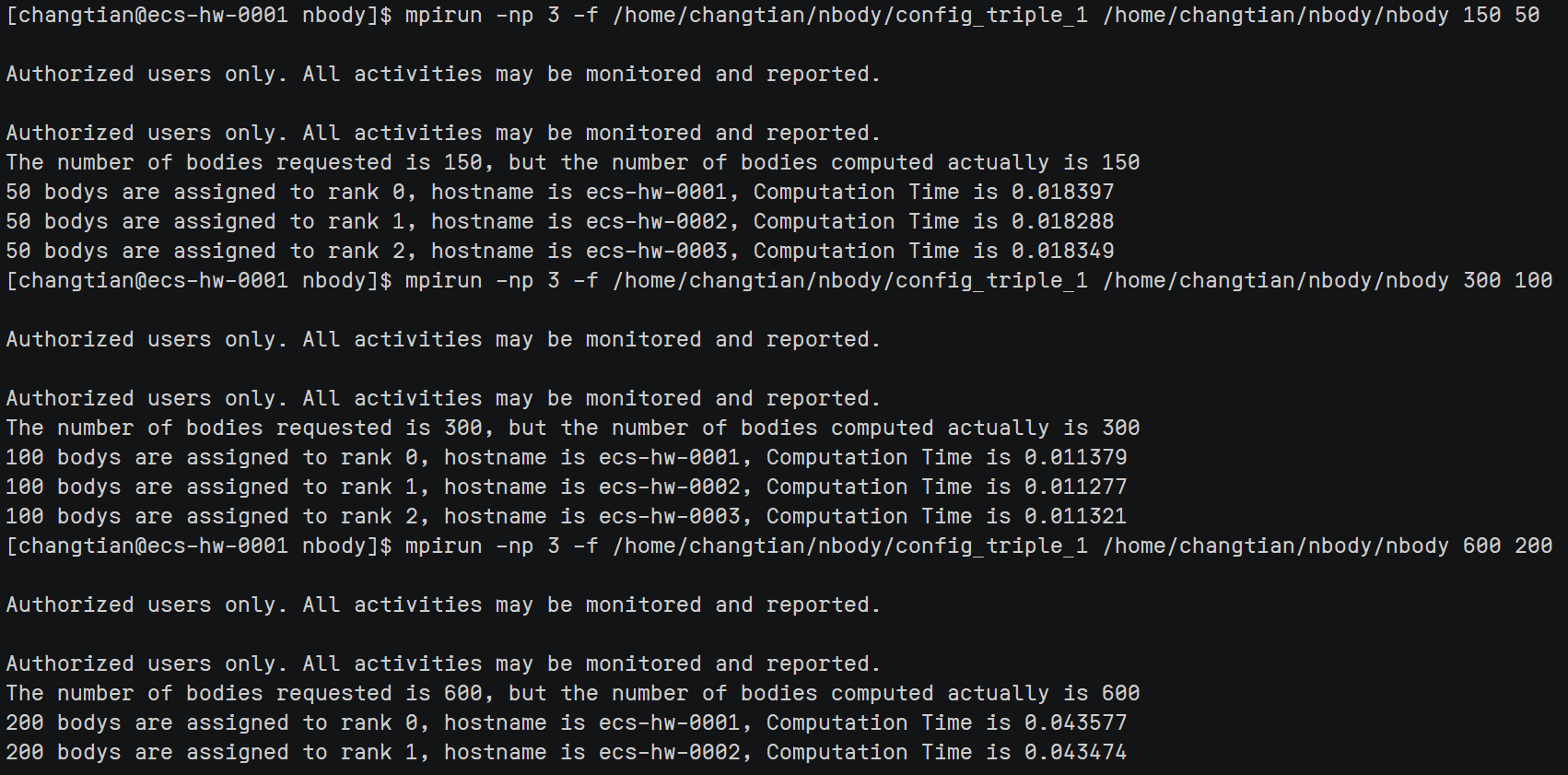
**mpirun -np 3 -f /home/changtian/nbody/config\_triple\_1 /home/changtian/nbody/nbody 600 200**

**mpirun -np 3 -f /home/changtian/nbody/config\_triple\_1 /home/changtian/nbody/nbody 1200 400**

**mpirun -np 3 -f /home/changtian/nbody/config\_triple\_1 /home/changtian/nbody/nbody 2400 800**

**mpirun -np 3 -f /home/changtian/nbody/config\_triple\_1 /home/changtian/nbody/nbody 4800 1600**

**mpirun -np 3 -f /home/changtian/nbody/config\_triple\_1 /home/changtian/nbody/nbody 9600 3200**

图片包含 图形用户界面

描述已自动生成文本

描述已自动生成

步骤三：根据记录的数据计算加速比与效率（给出数据并绘图）

实验一：单机上，粒子数为6000，随进程数变化加速比（Sp）统计。

加速比（Sp）统计表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 进程数 | 1 | 2 | 3 | 4 |
| 加速比 | 1.00 | 1.99 | 2.99 | 3.98 |

效率（Ep）统计表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 进程数 | 1 | 2 | 3 | 4 |
| 加速比 | 1.000 | 0.995 | 0.997 | 0.995 |

实验二：粒子数为6000，随每机进程数变化的加速比。

加速比（Sp）统计表

|  |  |  |  |
| --- | --- | --- | --- |
| 每机进程数 | 单机 | 双机 | 三机 |
| 1 | 1.00 | 1.99 | 2.98 |
| 2 | 1.99 | 3.97 | 5.87 |
| 3 | 2.99 | 5.89 | 8.71 |
| 4 | 3.98 | 7.88 | 11.73 |

效率（Ep）统计表

|  |  |  |  |
| --- | --- | --- | --- |
| 每机进程数 | 单机 | 双机 | 三机 |
| 1 | 1.000 | 0.995 | 0.993 |
| 2 | 0.995 | 0.993 | 0.978 |
| 3 | 0.997 | 0.982 | 0.968 |
| 4 | 0.995 | 0.985 | 0.978 |

实验三：每机1个进程，随数据规模变化的n-body并行程序加速比和效率。

加速比（Sp）统计表

|  |  |  |  |
| --- | --- | --- | --- |
| 粒子数n | 单机 | 双机 | 三机 |
| 150 | 1.00 | 0.29 | 0.26 |
| 300 | 1.00 | 0.97 | 1.67 |
| 600 | 1.00 | 1.40 | 1.73 |
| 1200 | 1.00 | 1.96 | 2.80 |
| 2400 | 1.00 | 2.00 | 2.96 |
| 4800 | 1.00 | 2.00 | 2.98 |
| 9600 | 1.00 | 2.00 | 2.99 |

7

效率（Ep）统计表

|  |  |  |  |
| --- | --- | --- | --- |
| 粒子数n | 单机 | 双机 | 三机 |
| 150 | 1.00 | 0.15 | 0.09 |
| 300 | 1.00 | 0.49 | 0.56 |
| 600 | 1.00 | 0.70 | 0.58 |
| 1200 | 1.00 | 0.98 | 0.93 |
| 2400 | 1.00 | 1.00 | 0.99 |
| 4800 | 1.00 | 1.00 | 0.99 |
| 9600 | 1.00 | 1.00 | 1.00 |

## 实验1.3 利用MPI对素数计算程序进行并行优化

### 实验目的

1. 掌握素数计算程序的基本原理；
2. 掌握串行素数计算程序的MPI并行优化方法。

实验内容

下面是素数计算的串行代码，统计2-N之间素数的个数。

|  |
| --- |
| # include <stdlib.h>  # include <stdio.h>  # include <time.h>  int main ( int argc, char \*argv[] );  int prime\_part ( int id, int p, int n );  int main ( int argc, char \*argv[] )  {  int id;  int n = 100000;  int p;  int total;  int total\_part;  p = 4;  total = 0;  for ( id = 0; id < p; id++ )  {  total\_part = prime\_part ( id, p, n );  total = total + total\_part;  }  printf ( "\n" );  printf ( " Between 2 and %d, there are %d primes\n", n, total );  return 0;  }  int prime\_part ( int id, int p, int n )  {  int i;  int j;  int prime;  int total\_part;  total\_part = 0;  for ( i = 2 + id; i <= n; i = i + p )  {  prime = 1;  for ( j = 2; j < i; j++ )  {  if ( i % j == 0 )  {  prime = 0;  break;  }  }  if ( prime )  {  total\_part = total\_part + 1;  }  }  return total\_part;  } |

**实现原理：**

该程序可以串行执行，但它的编写方式表明了可以如何进行并行化：

（1）程序需要检查从2到N的整数，为此首先将整数分解为P个子列表。例如，假设P为4，N为15：

Part 0: 2, 6, 10, 14

Part 1: 3, 7, 11, 15

Part 2: 4, 8, 12

Part 3: 5, 9, 13

（2）这样如果有P个进程，那么只需给每个进程一个子列表进行计算，在执行结束时，每个进程统计所负责子列表的素数个数并将其传输到进程0，进程0可以计算总数并进行输出。

**代码修改提示：**

1. 调用MPI头文件；
2. 在并行处理之前调用MPI\_Init()，MPI\_Comm\_size()，MPI\_Comm\_rank() 函数；
3. 在并行处理之后调用MPI\_Finalize()；
4. 调用MPI\_Reduce()收集各进程的计算结果；
5. 进程0打印最终的计算结果；
6. 如果N=100000，则素数的数量应为9592。可以申请1个进程来运行程序，检查计算结果是否正确；
7. 如果程序计算结果无误，就需要评估其性能。利用MPI\_Wtime()来统计时间，并利用进程0来输出程序的执行时间；
8. 可以使用命令行输入参数 N，支持设置不同的进程数来运行程序，避免每次修改 N 后重新编译与拷贝操作。

int n = atoi(argv[1]);

1. 编译与运行命令仿照 N 体问题中的进行操作。

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

#include <mpi.h>

int prime\_part(int id, int p, int n);

int main(int argc, char \*argv[])

{

int id, p, n, total, total\_part;

double start\_time, end\_time;

// 初始化MPI环境

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &p);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &id);

// 从命令行读取N值

if (argc < 2)

{

if (id == 0) printf("Usage: %s <N>\n", argv[0]);

MPI\_Finalize();

return 1;

}

n = atoi(argv[1]);

start\_time = MPI\_Wtime(); // 开始计时

// 每个进程计算自己分配的部分素数

total\_part = prime\_part(id, p, n);

// 使用MPI\_Reduce收集所有进程的结果到进程0

MPI\_Reduce(&total\_part, &total, 1, MPI\_INT, MPI\_SUM, 0, MPI\_COMM\_WORLD);

end\_time = MPI\_Wtime(); // 结束计时

// 仅进程0输出最终结果和计算时间

if (id == 0)

{

printf("\nBetween 2 and %d, there are %d primes\n", n, total);

printf("Execution Time: %f seconds\n", end\_time - start\_time);

}

// 结束MPI环境

MPI\_Finalize();

return 0;

}

int prime\_part(int id, int p, int n)

{

int i, j, prime, total\_part = 0;

for (i = 2 + id; i <= n; i += p)

{

prime = 1;

for (j = 2; j \* j <= i; j++)

{

if (i % j == 0)

{

prime = 0;

break;

}

}

if (prime)

{

total\_part++;

}

}

return total\_part;

}

**echo "ecs-hw-0001:1" > /home/changtian/prime/config\_1**

**echo -e "ecs-hw-0001:1\necs-hw-0002:1" > /home/changtian/prime/config\_2**

**echo -e "ecs-hw-0001:2\necs-hw-0002:1\necs-hw-0003:1" > /home/changtian/prime/config\_4**

**echo -e "ecs-hw-0001:2\necs-hw-0002:2\necs-hw-0003:2" > /home/changtian/prime/config\_6**

**scp -r /home/changtian/prime ecs-hw-0002:/home/changtian/**

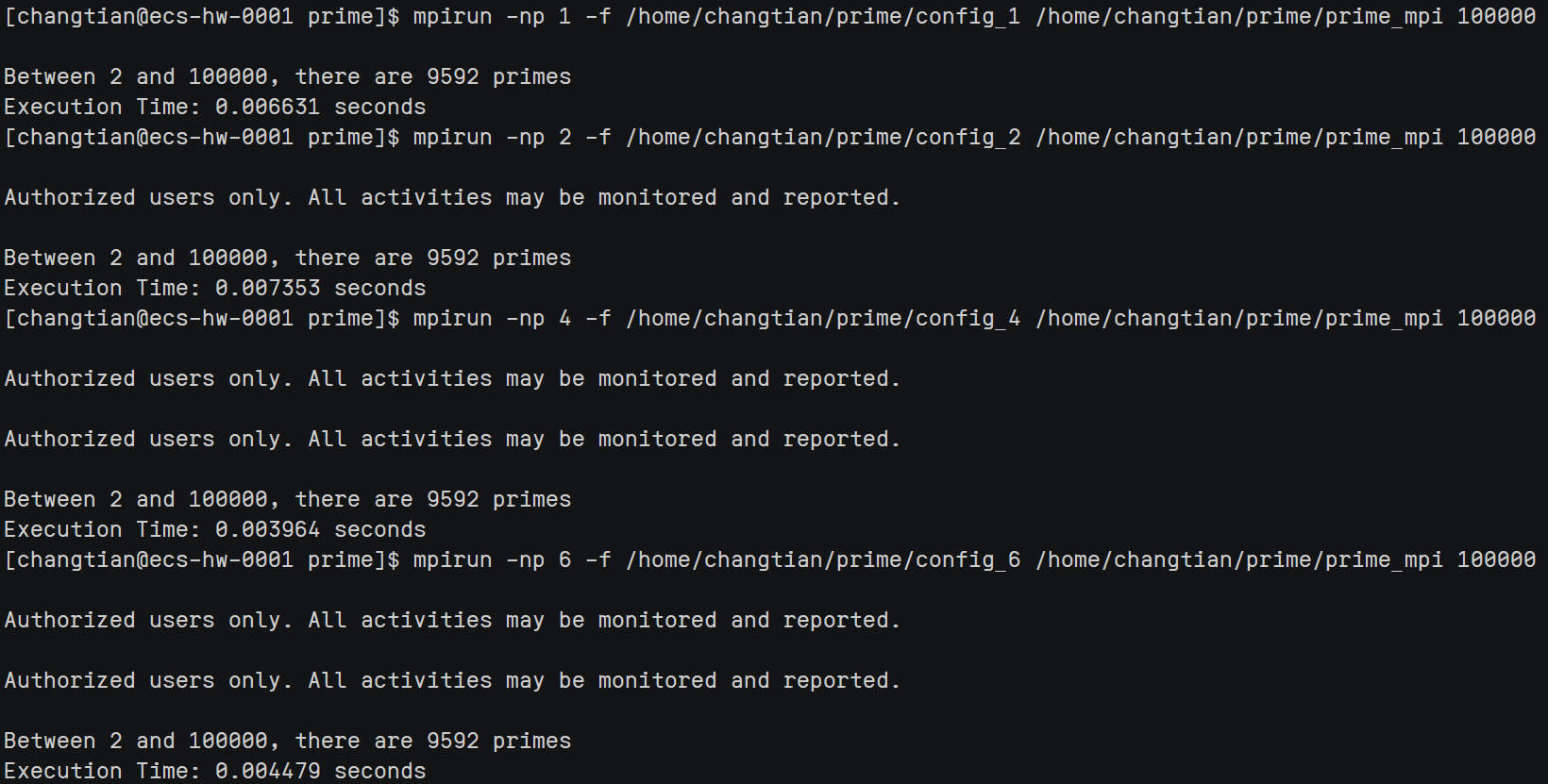
**scp -r /home/changtian/prime ecs-hw-0003:/home/changtian/**

**mpirun -np 1 -f /home/changtian/prime/config\_1 /home/changtian/prime/prime\_mpi 100000**

**mpirun -np 2 -f /home/changtian/prime/config\_2 /home/changtian/prime/prime\_mpi 100000**

**mpirun -np 4 -f /home/changtian/prime/config\_4 /home/changtian/prime/prime\_mpi 100000**

**mpirun -np 6 -f /home/changtian/prime/config\_6 /home/changtian/prime/prime\_mpi 100000**

****

**mpirun -np 1 -f /home/changtian/prime/config\_1 /home/changtian/prime/prime\_mpi 200000**

**mpirun -np 2 -f /home/changtian/prime/config\_2 /home/changtian/prime/prime\_mpi 200000**

**mpirun -np 4 -f /home/changtian/prime/config\_4 /home/changtian/prime/prime\_mpi 200000**

**mpirun -np 6 -f /home/changtian/prime/config\_6 /home/changtian/prime/prime\_mpi 200000**

**文本

描述已自动生成**

**mpirun -np 1 -f /home/changtian/prime/config\_1 /home/changtian/prime/prime\_mpi 400000**

**mpirun -np 2 -f /home/changtian/prime/config\_2 /home/changtian/prime/prime\_mpi 400000**

**mpirun -np 4 -f /home/changtian/prime/config\_4 /home/changtian/prime/prime\_mpi 400000**

**mpirun -np 6 -f /home/changtian/prime/config\_6 /home/changtian/prime/prime\_mpi 400000**

**文本

描述已自动生成**

**mpirun -np 1 -f /home/changtian/prime/config\_1 /home/changtian/prime/prime\_mpi 800000**

**mpirun -np 2 -f /home/changtian/prime/config\_2 /home/changtian/prime/prime\_mpi 800000**

**mpirun -np 4 -f /home/changtian/prime/config\_4 /home/changtian/prime/prime\_mpi 800000**

**mpirun -np 6 -f /home/changtian/prime/config\_6 /home/changtian/prime/prime\_mpi 800000**

**文本

描述已自动生成**

**要求：**

（1）代码修改完成后，编译执行，进行N在不同取值下的实验并记录数据：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N  进程数 | 1 | 2 | 4 | 6 |
| 100000 | 0.006631 | 0.007353 | 0.003964 | 0.004479 |
| 200000 | 0.017357 | 0.017963 | 0.009230 | 0.009282 |
| 400000 | 0.045851 | 0.046528 | 0.023570 | 0.023413 |
| 800000 | 0.122154 | 0.122650 | 0.061738 | 0.061105 |

计算N在不同取值下的加速比如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N  进程数 | 1 | 2 | 4 | 6 |
| 100000 | 1.00 | 0.90 | 1.67 | 1.48 |
| 200000 | 1.00 | 0.97 | 1.88 | 1.87 |
| 400000 | 1.00 | 0.99 | 1.94 | 1.96 |
| 800000 | 1.00 | 1.00 | 1.98 | 2.00 |

1. 查看程序加速比（可扩展性）是否有问题，并分析原因。提示：程序的负载均衡问题。

（3）在分析您目前的程序时，加速比的可扩展性问题主要在于负载均衡。现有代码将整数范围按步长 p（进程数）分配给各个进程，这样做可以均匀分配任务，但由于每个整数判断素数的复杂度不同，因此导致每个进程实际工作量不同，特别是在素数密集的范围。

为了解决负载均衡问题，可以使用 **任务队列** 的方法，将待处理的整数以更小的粒度分配给各进程，让进程完成一个任务后立即领取下一个任务，这样可以动态平衡负载。

**修改代码：实现动态任务分配**

我们将代码修改为一个动态任务分配版本。在这个版本中，进程 0 作为任务调度者，分配整数给其他进程来判断素数，直到所有任务完成。

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

#include <mpi.h>

int is\_prime(int number);

int main(int argc, char \*argv[])

{

int id, p, n, total = 0, local\_count = 0;

double start\_time, end\_time;

MPI\_Status status;

// 初始化MPI环境

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &p);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &id);

// 从命令行读取N值

if (argc < 2)

{

if (id == 0) printf("Usage: %s <N>\n", argv[0]);

MPI\_Finalize();

return 1;

}

n = atoi(argv[1]);

start\_time = MPI\_Wtime(); // 开始计时

if (id == 0) {

// 进程0分配任务，管理任务队列

int next\_value = 2; // 起始任务从2开始

int active\_workers = p - 1;

// 向每个进程分发第一个任务

for (int i = 1; i < p; i++) {

MPI\_Send(&next\_value, 1, MPI\_INT, i, 0, MPI\_COMM\_WORLD);

next\_value++;

}

// 循环分发任务直到所有任务完成

while (active\_workers > 0) {

int result, worker\_result;

MPI\_Recv(&worker\_result, 1, MPI\_INT, MPI\_ANY\_SOURCE, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

int worker\_id = status.MPI\_SOURCE;

// 收集来自工作进程的素数数量

total += worker\_result;

if (next\_value <= n) {

// 如果还有任务，继续分配

MPI\_Send(&next\_value, 1, MPI\_INT, worker\_id, 0, MPI\_COMM\_WORLD);

next\_value++;

} else {

// 没有任务，发送终止信号

int stop\_signal = 0;

MPI\_Send(&stop\_signal, 1, MPI\_INT, worker\_id, 1, MPI\_COMM\_WORLD);

active\_workers--;

}

}

} else {

// 其他进程接收任务并判断素数

while (1) {

int number;

MPI\_Recv(&number, 1, MPI\_INT, 0, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

if (status.MPI\_TAG == 1) {

break; // 终止信号

}

if (is\_prime(number)) {

local\_count++;

}

MPI\_Send(&local\_count, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

local\_count = 0; // 重置计数

}

}

end\_time = MPI\_Wtime(); // 结束计时

// 仅进程0输出最终结果和计算时间

if (id == 0) {

printf("\nBetween 2 and %d, there are %d primes\n", n, total);

printf("Execution Time: %f seconds\n", end\_time - start\_time);

}

// 结束MPI环境

MPI\_Finalize();

return 0;

}

// 判断是否为素数的函数

int is\_prime(int number) {

if (number < 2) return 0;

for (int i = 2; i \* i <= number; i++) {

if (number % i == 0) return 0;

}

return 1;

}

**代码解释**

1. **动态任务分配**：进程 0 负责任务分配，它启动时将 2 到 n 的整数分配给其他进程。
2. **终止信号**：进程 0 将 MPI\_TAG 为 1 的消息作为终止信号发送给工作进程，告知无更多任务。
3. **负载均衡**：任务分为更小的单位（每个整数作为单个任务），工作进程完成一个任务后请求下一个任务，减轻了由于任务复杂度不同导致的负载不均衡问题。
4. **单个素数判断函数**：is\_prime() 实现了简单的素数检查逻辑。由于单次任务很小，负载较均匀。

**使用方法**

编译并运行的方法与原来相同，可使用不同的 config 文件来指定多机配置。

此代码修改了任务分配方式，改善了负载均衡，尤其在数据量较大时能更好地利用多进程资源，提高扩展性。

**vim prime\_mpi\_2.c**

**mpicc -o prime\_mpi\_2 prime\_mpi\_2.c**

**scp -r /home/changtian/prime ecs-hw-0002:/home/changtian/**

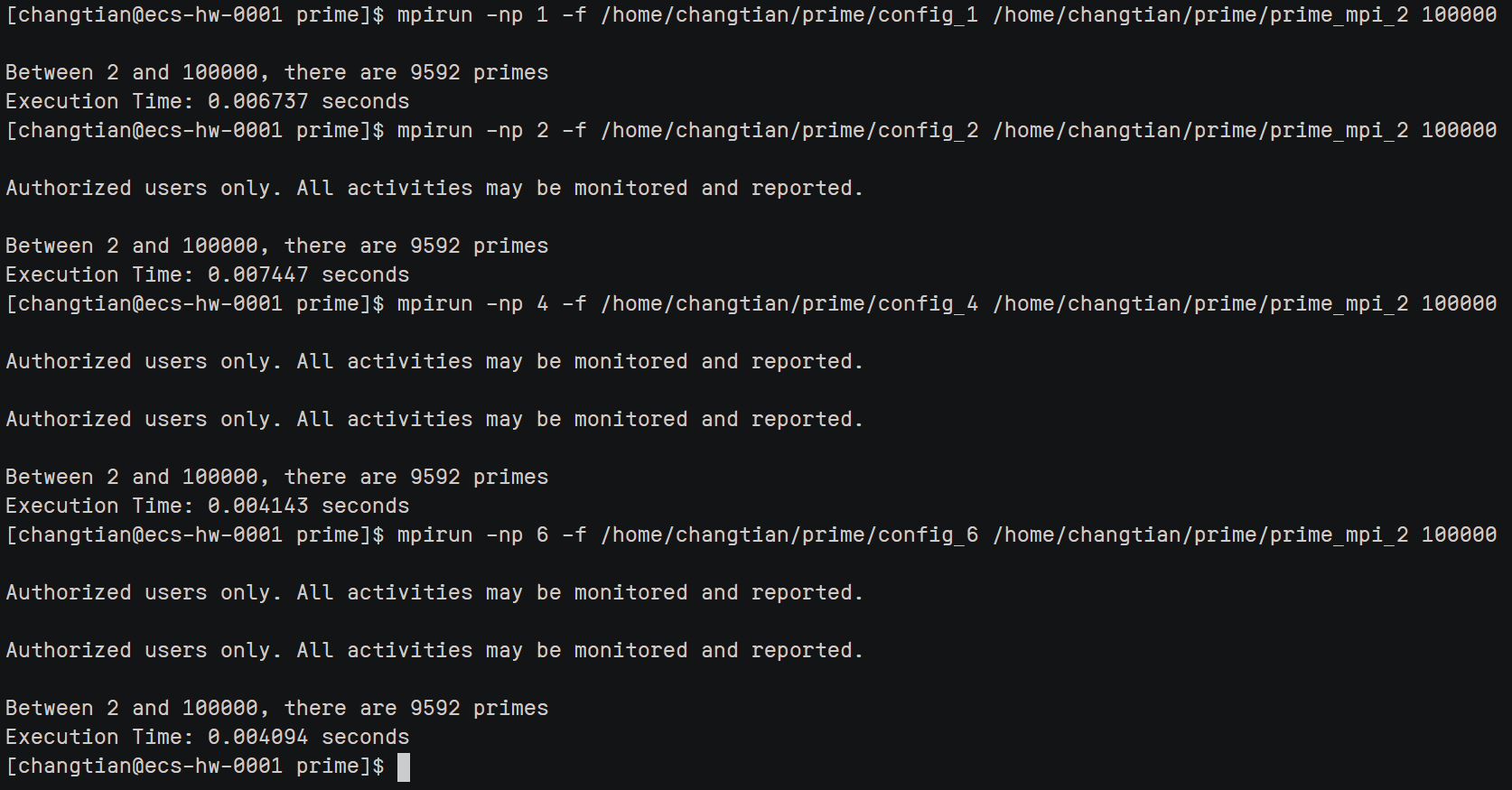
**scp -r /home/changtian/prime ecs-hw-0003:/home/changtian/**

**mpirun -np 1 -f /home/changtian/prime/config\_1 /home/changtian/prime/prime\_mpi\_2 100000**

**mpirun -np 2 -f /home/changtian/prime/config\_2 /home/changtian/prime/prime\_mpi\_2 100000**

**mpirun -np 4 -f /home/changtian/prime/config\_4 /home/changtian/prime/prime\_mpi\_2 100000**

**mpirun -np 6 -f /home/changtian/prime/config\_6 /home/changtian/prime/prime\_mpi\_2 100000**

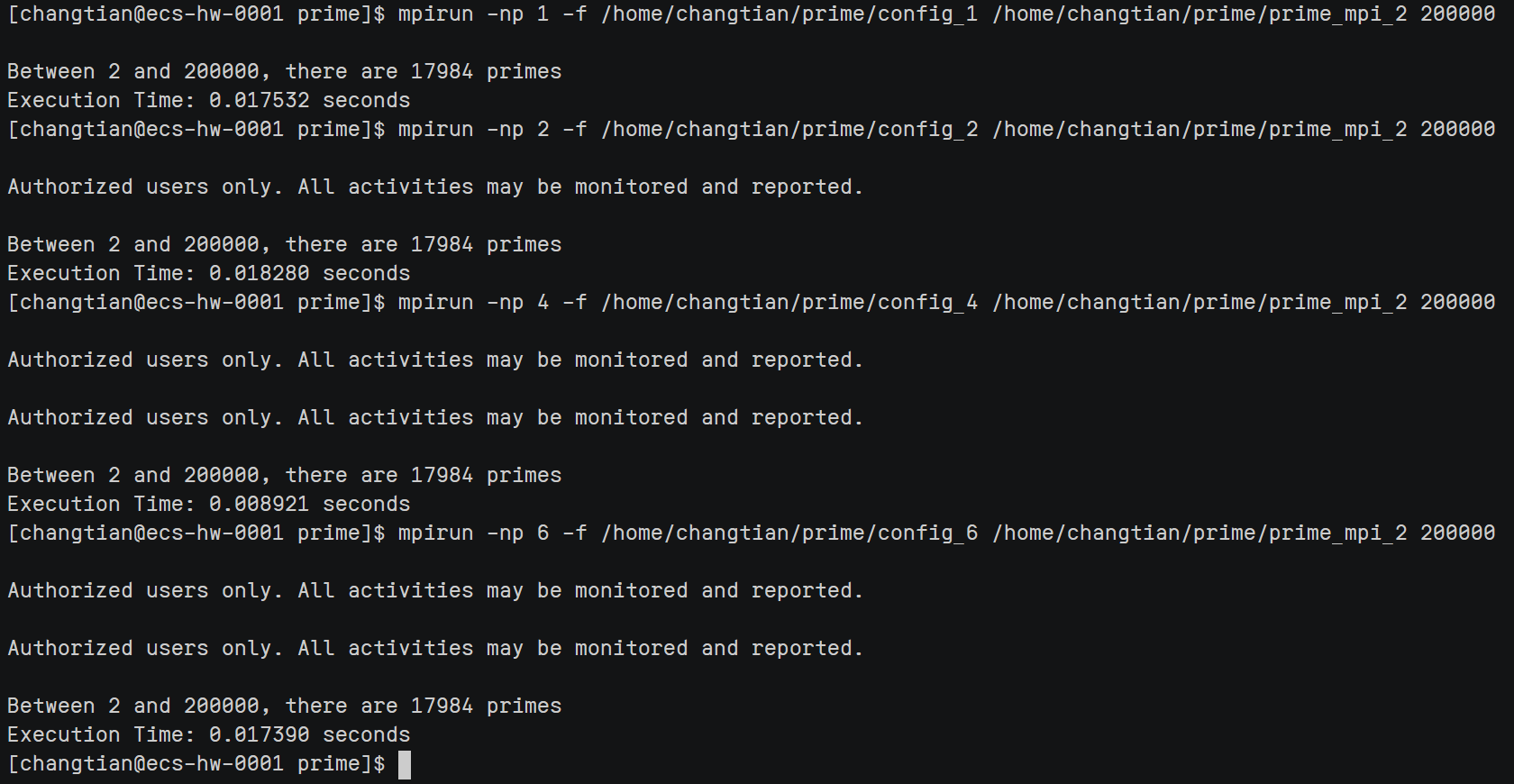
****

**mpirun -np 1 -f /home/changtian/prime/config\_1 /home/changtian/prime/prime\_mpi\_2 200000**

**mpirun -np 2 -f /home/changtian/prime/config\_2 /home/changtian/prime/prime\_mpi\_2 200000**

**mpirun -np 4 -f /home/changtian/prime/config\_4 /home/changtian/prime/prime\_mpi\_2 200000**

**mpirun -np 6 -f /home/changtian/prime/config\_6 /home/changtian/prime/prime\_mpi\_2 200000**

****

**mpirun -np 1 -f /home/changtian/prime/config\_1 /home/changtian/prime/prime\_mpi\_2 400000**

**mpirun -np 2 -f /home/changtian/prime/config\_2 /home/changtian/prime/prime\_mpi\_2 400000**

**mpirun -np 4 -f /home/changtian/prime/config\_4 /home/changtian/prime/prime\_mpi\_2 400000**

**mpirun -np 6 -f /home/changtian/prime/config\_6 /home/changtian/prime/prime\_mpi\_2 400000**

**文本

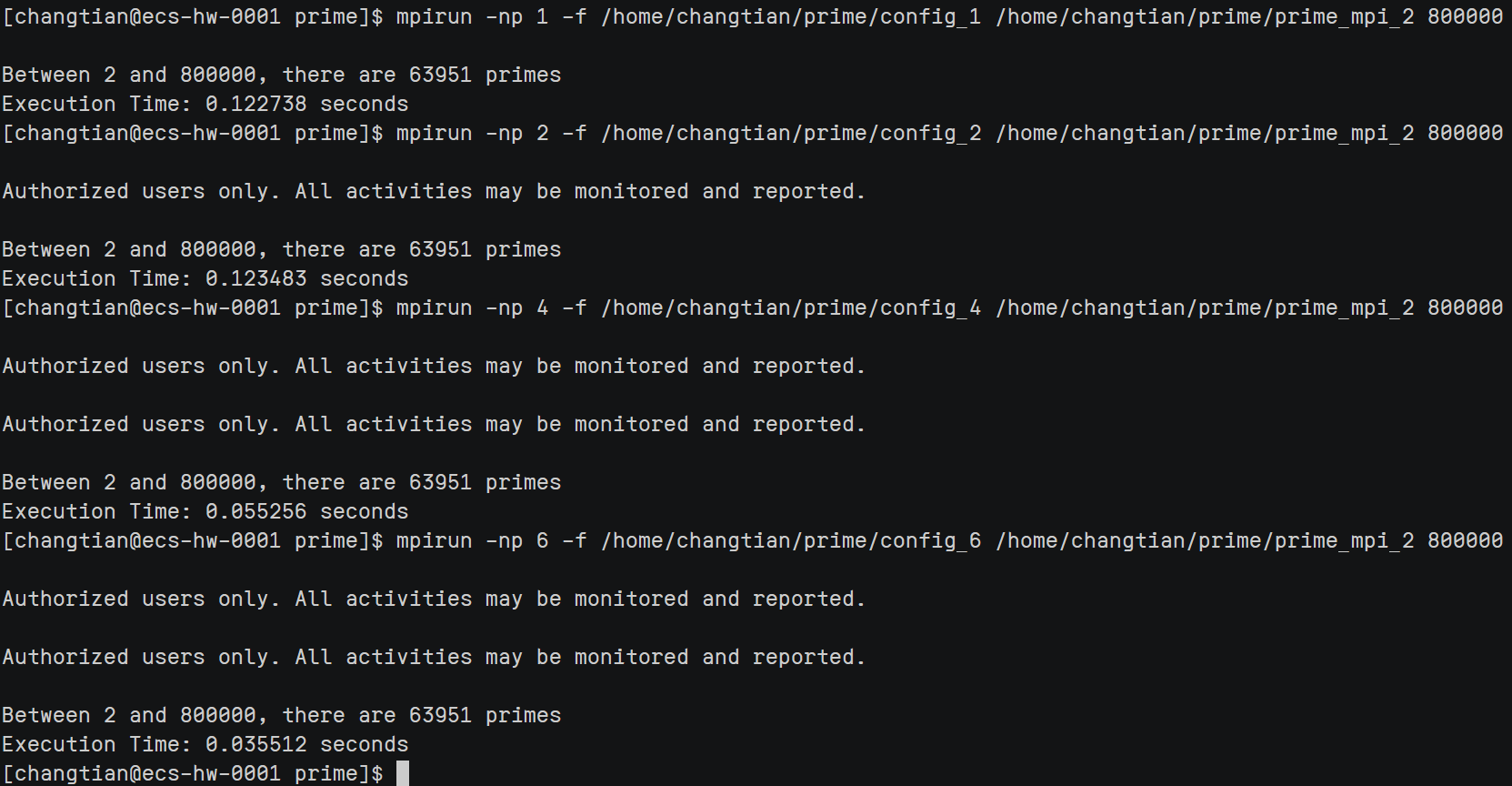
描述已自动生成**

**mpirun -np 1 -f /home/changtian/prime/config\_1 /home/changtian/prime/prime\_mpi\_2 800000**

**mpirun -np 2 -f /home/changtian/prime/config\_2 /home/changtian/prime/prime\_mpi\_2 800000**

**mpirun -np 4 -f /home/changtian/prime/config\_4 /home/changtian/prime/prime\_mpi\_2 800000**

**mpirun -np 6 -f /home/changtian/prime/config\_6 /home/changtian/prime/prime\_mpi\_2 800000**

****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N  进程数 | 1 | 2 | 4 | 6 |
| 100000 | 0.006737 | 0.007447 | 0.004405 | 0.004104 |
| 200000 | 0.017532 | 0.018280 | 0.008921 | 0.006941 |
| 400000 | 0.046499 | 0.047012 | 0.021380 | 0.014670 |
| 800000 | 0.122738 | 0.123483 | 0.055256 | 0.035512 |

进一步修改代码，进行下列实验并记录数据：

计算N在不同取值下的加速比如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N  进程数 | 1 | 2 | 4 | 6 |
| 100000 | 1.00 | 0.90 | 1.53 | 1.64 |
| 200000 | 1.00 | 0.96 | 1.97 | 2.53 |
| 400000 | 1.00 | 0.99 | 2.17 | 3.17 |
| 800000 | 1.00 | 0.99 | 2.22 | 3.46 |

### 实验评分标准

一、课堂表现（10分）

二、实验结果（50分）

三、实验报告（40分）

### 思考题

思考题1：从算法层面上分析素数计算程序负载不均衡的原因，并提出改进方法。

思考题2：根据该实验，简述并行程序编程和优化的步骤。