# 《并行计算》实验指导书

# 实验2：基于共享内存的并行程序设计

## 实验2.1华为云实验环境说明

### 实验目的

1. 了解华为云环境的使用过程；

### 实验内容

实验步骤：

步骤一：登录华为云。

打开浏览器，输入华为云的域名：<https://www.huaweicloud.com>，点击右上角登录按钮，输入用户名和密码。



步骤二：购买弹性云服务器

选择产品——>计算——>弹性云服务器ECS——>购买



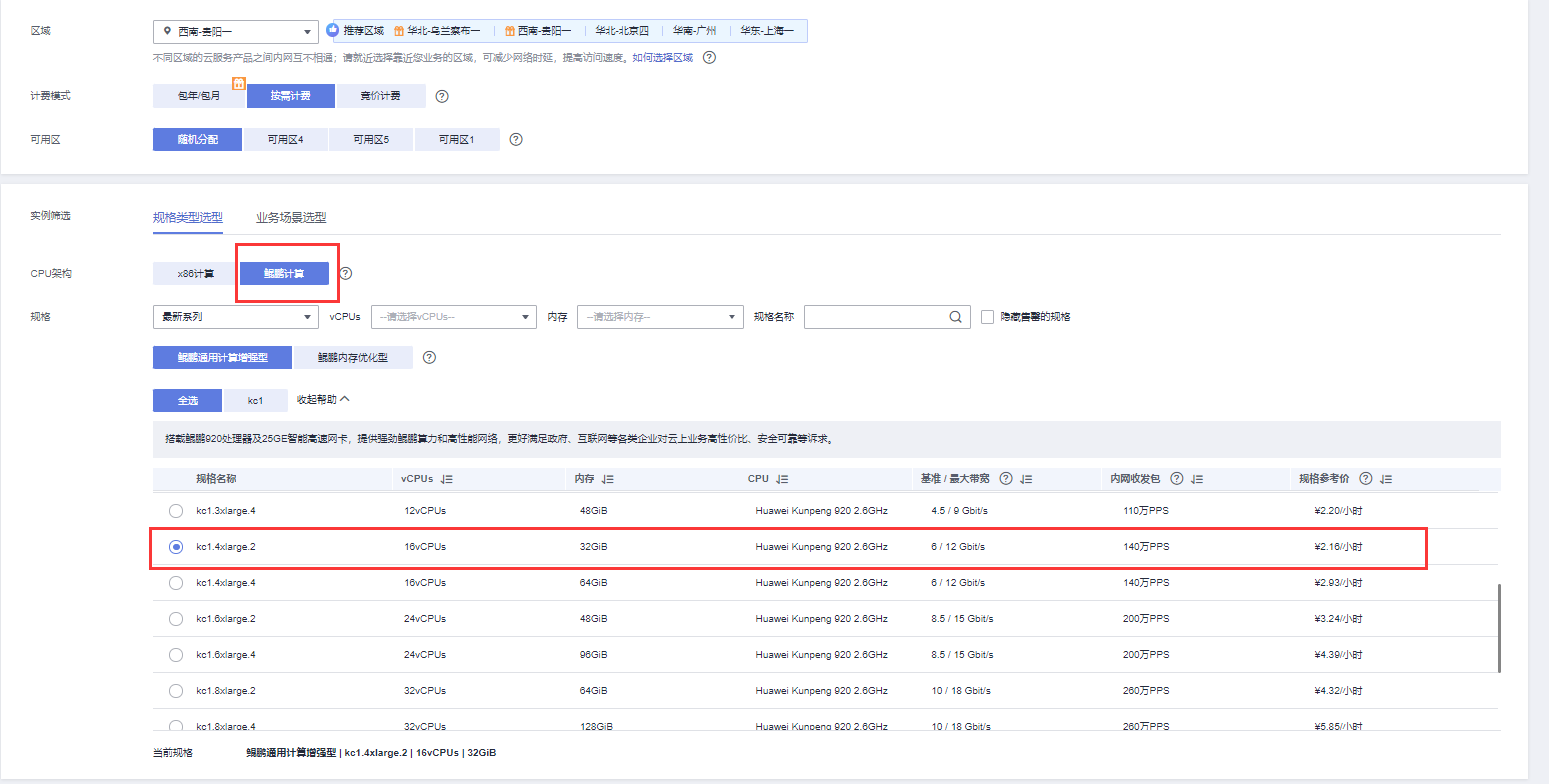


步骤三：基础配置

按照下表进行购买，购买**1台**。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 计费模式 | 区域 | CPU架构 | 规格 | 镜像 | 系统盘 |
| 按需计费 | 西南-贵阳一 | 鲲鹏计算 | kc1.4xlarge.2 | 公共镜像：openEuler 20.03 | 至少40GB |

参考截图：





步骤四：网络配置，勾选随实例释放，不然删除实例后依然计费



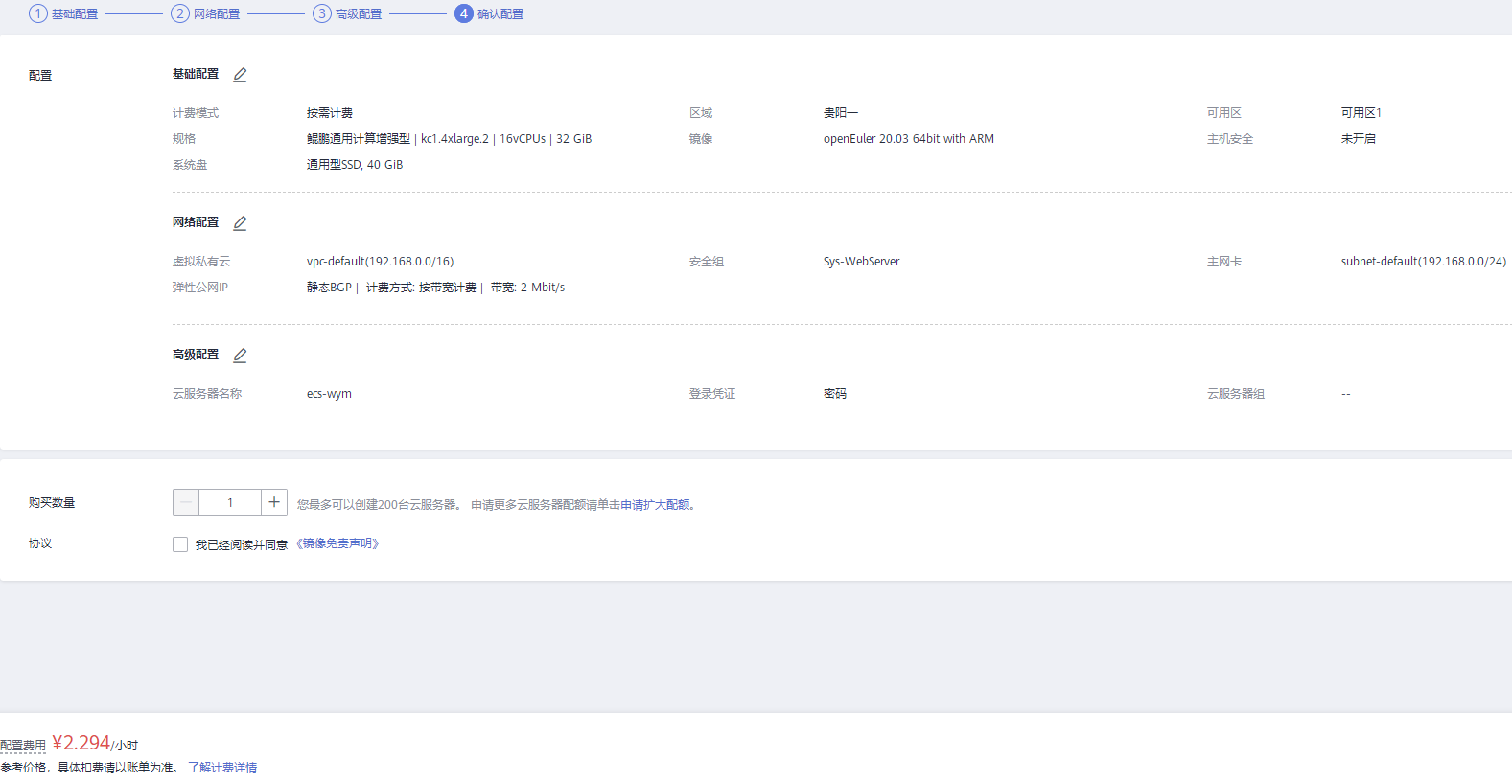
步骤五：高级配置

设置云服务器名称（eg：ecs-hw）、密码（Parallel2022），云备份选择暂不购买。



步骤六：确认配置

勾选我已阅读并同意《华为镜像免责声明》，点击确认配置



步骤七：虚拟机创建



步骤八：远程登录ECS

点击远程登录，选择CloudShell登录。或者系统中安装SSH工具，如XShell，输入公网IP、用户名和密码，即可登录。





步骤九：环境配置说明

为了防⽌⼤家的⽂件混乱，建议⼤家建⽴个⼈账户，不建议统⼀使⽤root账户，下⾯以⽤户名zhangsan 为例，

步骤十：创建用户

在root用户下建立个人账户并切换到个人账户，命令如下：

|  |
| --- |
| adduser zhangsan  passwd zhangsan // 这一步之后需要输入新密码  usermod -aG wheel zhangsan  su – zhangsan // 切换到个人账户，[zhangsan@ecs-hw ~] |

说明：zhangsan密码也设置为Parallel2022

## 实验2.2 并行环境下Pthread程序的编译和运行

### 实验目的

1. 熟练掌握C++语言；
2. 掌握PThread 开发多线程程序；
3. 掌握PThread 中互斥机制的使用方式

### 实验内容

#### 实验一：hello-world(示例)

步骤一：创建程序源码

用户为zhangsan时，执行以下命令，创建Pthread目录存放所有Pthread程序,并进入Pthread目录。

|  |
| --- |
| mkdir /home/zhangsan/Pthread  cd /home/zhangsan/Pthread |

执行以下命令，创建示例程序源码 pthread\_hello\_world.c

|  |
| --- |
| vim pthread\_hello\_world.c |

代码内容如下：

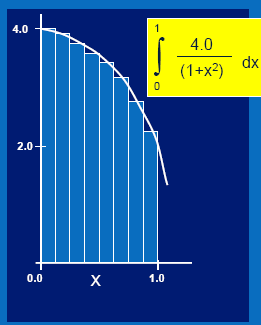
|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <pthread.h>  const int NumThreads = 16;  static void\* HelloFunc(void\* pArg)  {  printf("Hello Thread %d !\n", \*((int\*)pArg)); // type cast and dereference  }  int main()  {  int Num[NumThreads];  pthread\_t ThreadIDs[NumThreads];  pthread\_attr\_t attr[NumThreads];  for (int i = 0; i < NumThreads; i++) {  Num[i] = i;  pthread\_attr\_init(&attr[i]);  pthread\_attr\_setdetachstate(&attr[i], PTHREAD\_CREATE\_JOINABLE);  }  for (int i = 0; i < NumThreads; i++) {  int err = pthread\_create(&ThreadIDs[i], &attr[i], HelloFunc, (void\*)&Num[i]);  if(err != 0) {  printf("ERROR: pthread\_create() return code: %d\n", err);  }  }  for (int i = 0; i < NumThreads; i++) {  pthread\_join(ThreadIDs[0], NULL);  printf("Thread %d end !\n", i);  }  return 0;  } |

步骤二：编译和执行

|  |
| --- |
| gcc pthread\_hello\_world.c -lpthread -o pthread\_hello\_world.bin  ./pthread\_hello\_world.bin |

#### 实验二：利用中值积分定理计算Pi值

程序原理介绍：





串行代码(Pi.c)如下：

|  |
| --- |
| #include <stdio.h>  #include <sys/time.h>  long long num\_steps = 1000000000;  double step;  int main(int argc, char\* argv[])  {  struct timeval TimeStampStart, TimeStampStop;  double ExeTime;  double x, pi, sum=0.0;  int i;  step = 1./(double)num\_steps;  gettimeofday(&TimeStampStart, NULL);  for (i=0; i<num\_steps; i++)  {  x = (i + .5)\*step;  sum = sum + 4.0/(1.+ x\*x);  }    pi = sum\*step;  gettimeofday(&TimeStampStop, NULL);  ExeTime = (double)(TimeStampStop.tv\_sec - TimeStampStart.tv\_sec) + (double)(TimeStampStop.tv\_usec - TimeStampStart.tv\_usec) \* 1e-6;  printf("The value of PI is %15.12f\n",pi);  printf("The time to calculate PI was %f seconds\n", (ExeTime));  return 0;  } |

编译和执行

|  |
| --- |
| gcc Pi.c -lpthread -o pi.bin  ./pi.bin  The value of PI is 3.141592653590  The time to calculate PI was 7.055304 seconds |

要多线程实现的话，主要是把for循环的计算过程分到几个线程中去，由于每次计算都要更新sum的值，就有可能发生一个线程已经更新了sum的值，而另一个线程读到的还是旧的sum值，所以在这里使用临界区，把sum放到临界区中，这样一次只能有一个线程访问和修改sum的值。

pthread\_create（） //线程创建函数

pthread\_mutex\_t mut; //互斥量

pthread\_mutex\_init(&mut,NULL); //互斥量初始化

pthread\_mutex\_lock(&mut);

pthread\_mutex\_unlock(&mut);

pthread\_mutex\_destroy(&mut); // 销毁互斥量

#### 实验三：互斥量的使用 CountWords

串行代码(Serial.c)如下：

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <pthread.h>  #include <stdbool.h>  #include <sys/time.h>  FILE \*fd;  int TotalEvenWords = 0, TotalOddWords = 0, TotalWords = 0;  int GetNextLine(FILE \*f, char \*Line)  {    if (fgets(Line, 132, f)==NULL) if (feof(f))return EOF; else return 1;  }  int GetWordAndLetterCount(char \*Line)  {  int Word\_Count = 0, Letter\_Count = 0;  for (int i=0;i<132;i++)  {  if ((Line[i]!=' ')&&(Line[i]!=0)&&(Line[i]!='\n')) Letter\_Count++;  else {  if (Letter\_Count % 2) {  TotalOddWords++;  Word\_Count++;  Letter\_Count = 0;  }  else {  TotalEvenWords++;  Word\_Count++;  Letter\_Count = 0;  }  if (Line[i]==0) break;  }  }  return (Word\_Count); // encode two return values  }  int CountWords()  {  bool bDone = false;  char inLine[132];  while (!bDone)  {  bDone = (GetNextLine(fd, inLine) == EOF);  if (!bDone){  TotalWords += GetWordAndLetterCount(inLine) ;  }  }  return 0;  }  int main()  {  fd = fopen("./InFile1.txt", "r"); // Open file for read  struct timeval TimeStampStart, TimeStampStop;  double ExeTime;  gettimeofday(&TimeStampStart, NULL);  CountWords();  gettimeofday(&TimeStampStop, NULL);  ExeTime = (double)(TimeStampStop.tv\_sec - TimeStampStart.tv\_sec) + (double)(TimeStampStop.tv\_usec - TimeStampStart.tv\_usec) \* 1e-6;  fclose(fd);  printf("Total Words = %8d\n", TotalWords);  printf("Total Even Words = %7d\nTotal Odd Words = %7d\n", TotalEvenWords, TotalOddWords);  printf("The time to read words was %f seconds\n", (ExeTime));  return 0;  } |

输入文件：InFile1.txt 在附件中，将InFile1.txt放到相应目录下

编译和执行

|  |
| --- |
| gcc Serial.c -lpthread -o Serial.bin  ./Serial.bin  Total Words = 1148472  Total Even Words = 636517  Total Odd Words = 511955  The time to read words was 0.044190 seconds |

### 实验要求

（1）并行化代码，执行并行程序，查看结果，并截图保存。

## 实验2.3 并行环境下OpenMP程序的编译和运行

### 实验目的

1. 掌握 OpenMP 的基本功能、构成方式、句法；
2. 掌握 OpenMP 体系结构、特点与组成；
3. 掌握采用 OpenMP 进行多核架构下多线程编程的基本使用方法。

### 实验内容

#### 实验一：hello-world (示例)

步骤一：创建程序源码

执行以下命令，创建OpenMP目录存放所有OpenMP程序,并进入OpenMP目录。

|  |
| --- |
| mkdir /home/zhangsan/OpenMP  cd /home/zhangsan/OpenMP |

执行以下命令，创建示例程序源码 openmp\_hello\_world.c

|  |
| --- |
| vim openmp\_hello\_world.c |

代码内容如下：

|  |
| --- |
| #include <stdio.h>  #include <omp.h>  int main() {  int i;  #pragma omp parallel  {  printf("Hello World\n");  for(i=0; i<4; i++) {  printf("Iter:%d\n",i);  }  printf("GoodBye World\n");  }  return 0;  } |

步骤二：编译和执行

|  |
| --- |
| gcc openmp\_hello\_world.c -fopenmp -o openmp\_hello\_world.bin  ./openmp\_hello\_world.bin |

#### 实验二：利用中值积分定理计算Pi值

串行代码(pi.c)如下：

|  |
| --- |
| #include <stdio.h>  #include <sys/time.h>  long long num\_steps = 1000000000;  double step;  int main(int argc, char\* argv[])  {  struct timeval TimeStampStart, TimeStampStop;  double ExeTime;  double x, pi, sum=0.0;  int i;  step = 1./(double)num\_steps;  gettimeofday(&TimeStampStart, NULL);  for (i=0; i<num\_steps; i++)  {  x = (i + .5)\*step;  sum = sum + 4.0/(1.+ x\*x);  }    pi = sum\*step;  gettimeofday(&TimeStampStop, NULL);  ExeTime = (double)(TimeStampStop.tv\_sec - TimeStampStart.tv\_sec) + (double)(TimeStampStop.tv\_usec - TimeStampStart.tv\_usec) \* 1e-6;  printf("The value of PI is %15.12f\n",pi);  printf("The time to calculate PI was %f seconds\n", (ExeTime));  return 0;  } |

编译和执行

|  |
| --- |
| gcc pi.c -fopenmp -o pi.bin  /pi.bin  The value of PI is 3.141592653590  The time to calculate PI was 6.991294 seconds |

**并行化方法：在源程序代码的主程序体中进行omp方式优化**

① 需要并行运算的程序体：

加上#pragma omp parallel

{

}段

parallel语句后面被大括号括起来的代码是并行执行的。

② 找到for循环体引入omp并行处理方法

加上#pragma omp for

for（xxx:yyy:zzz）

{

}段

#pragma omp for语句是一个工作分担结构，它使得循环由一个线程池的线程并行执行。

③ 检查所有变量，将需要进行特别声明的变量进行omp处理：

#pragma omp parallel private(varname,vaname)

reduction(+:varname)

shared(varname)

{

}段

reduction子句为变量指定一个操作符，每个线程都会创建reduction变量的私有拷贝，在OpenMP区域结束处，将使用各个线程的私有拷贝的值通过制定的操作符进行迭代运算，并赋值给原来的变量。

④ 对于特殊的共享变量，可以进行加锁处理

#pragma omp critical

{

}段

#### 实验三：PI值蒙特卡洛算法

串行代码(pimonte\_serial.c)如下：

|  |
| --- |
| #include <stdlib.h>  #include <sys/time.h>  #include <time.h>  #define BLOCK\_SIZE 500  int main(){  struct timeval TimeStampStart, TimeStampStop;  double ExeTime;  unsigned int iter=200000000;  int i, j;  double x, y;  double dUnderCurve=0.0;  double pi=0.0;  double r[BLOCK\_SIZE\*2]; //Careful!!!  //you need a private copy of whole array for each thread  srand((unsigned)time(NULL));  gettimeofday(&TimeStampStart, NULL);  for(j=0; j<iter/BLOCK\_SIZE; j++) {  // Create random numbers into array r  // 生成 BLOCK\_SIZE\*2 个在 0.0-1.0 内均匀分布的随机数, 作为横纵坐标  for (i=0; i<BLOCK\_SIZE\*2; i++) {  r[i] = 0.0 + 1.0 \* rand() / RAND\_MAX \* ( 1.0 - 0.0 );  }    for (i=0; i<BLOCK\_SIZE; i++) {  x=r[i]; //X Coordinate  y=r[i+BLOCK\_SIZE]; //Y Coordinate  if (x\*x + y\*y <= 1.0) { //is distance from Origin under Curve  dUnderCurve++;  }  }  }  pi = dUnderCurve / (double) iter \* 4;  gettimeofday(&TimeStampStop, NULL);  ExeTime = (double)(TimeStampStop.tv\_sec - TimeStampStart.tv\_sec) + (double)(TimeStampStop.tv\_usec - TimeStampStart.tv\_usec) \* 1e-6;  printf ("pi = %10.9f\n", pi);  printf("The time to calculate PI was %f seconds\n", (ExeTime));    return 0;  } |

|  |
| --- |
| gcc pimonte\_serial.c -fopenmp -o pimonte\_serial.bin  ./pimonte\_serial.bin  pi = 3.141659380  run time = 9.567676000 s |

编译和执行

srand函数为rand函数指定种子，如果没指定，rand函数默认的种子为1，那么每次产生的随机序列是相同的。srand以时间作为随机数的种子，使得rand每次取到的随机序列都不一样。但由于每调用一次rand会改变它的状态值(如next值)，而对于同一进程的多个线程共享这些状态值，所以rand函数是非线程安全的，多线程应该明确指定seed值的rand\_r。

rand函数的使用：

srand((unsigned)time(NULL));

rand()%MAX;

rand\_r函数的使用：

Static unsigned int seed = 123;

rand\_r(&seed);

注意：当rand\_r指定的固定值为种子时，每次产生的随机序列是同一个。

**并行化方法：在源程序代码的主程序体中进行omp方式优化**

① 需要并行运算的程序体：

加上#pragma omp parallel

{

}段

② 找到for循环体引入omp并行处理方法

加上#pragma omp for

for（xxx:yyy:zzz）

{

}段

③ 检查所有变量，将需要进行特别声明的变量进行omp处理：

#pragma omp parallel private(varname,vaname)\

reduction(+:varname,varname)\

shared(varname,varname)

{

}段

④ 对于特殊的共享变量，进行加锁处理

#pragma omp critical

{

}段

### 实验要求

（1）并行化代码，执行并行程序，查看结果，并截图保存。

### 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：串行程序采用PThread进行线程化基本步骤？

思考题2：PThread有几种共享数据互斥机制可以采用，如何应用？

思考题3：OpenMP的主要功能，基本构成体有哪些？

思考题4：试分析如何使用OpenMP实现多线程并行运算，提高系统运算效能，其引入环节应如何选取？

思考题5：对本次实验最开始时的pthread\_hello\_world.c修改后编译并多次运行程序，记录线程执行顺序，分析线程程序执行顺序是否不可预见及其产生原因。

思考题6：根据你的实验，OMP\_NUM\_THREADS是不是设置得越大越好？说明理由。