**中山大学计算机院本科生实验报告**

**（2024学年春季学期）**

**课程名称：并行程序设计 批改人：**

|  |  |  |  |
| --- | --- | --- | --- |
| 实验 | 3-Pthreads并行矩阵乘法与数组求和 | 专业（方向） | **计算机科学与技术** |
| 学号 | **21307174** | 姓名 | **刘俊杰** |
| Email | **liujj255@mail2.edu.cn** | 完成日期 | **2024/4/10** |

# 实验目的

### 1.1 并行矩阵乘法

使用Pthreads实现并行矩阵乘法，并通过实验分析其性能。

**输入：**三个整数，每个整数的取值范围均为[128, 2048]

**问题描述：**随机生成的矩阵及的矩阵，并对这两个矩阵进行矩阵乘法运算，得到矩阵.

**输出**：三个矩阵，及矩阵计算所消耗的时间。

**要求：**1. 使用Pthread创建多线程实现并行矩阵乘法，调整线程数量（1-16）及矩阵规模（128-2048），根据结果分析其并行性能（包括但不限于，时间、效率、可扩展性）。2. 选做：可分析不同数据及任务划分方式的影响。

### 1.2 并行数组求和

使用Pthreads实现并行数组求和，并通过实验分析其性能。

**输入：**整数，取值范围为[1M, 128M]

**问题描述：**随机生成长度为的整型数组，计算其元素和。

**输出**：数组，元素和，及求和计算所消耗的时间。

**要求：**1. 使用Pthreads实现并行数组求和，调整线程数量（1-16）及数组规模（1M, 128M），根据结果分析其并行性能（包括但不限于，时间、效率、可扩展性）。2. 选做：可分析不同聚合方式的影响。

# 实验过程和核心代码

### 2.1 并行矩阵乘法

#### 2.1.1 实验思路

①将矩阵C每一行的计算平均分给每一个线程。

②矩阵A、B、C作为共享的全局变量方便每一个线程访问和修改。

③由于每个线程负责修改的是矩阵C不同的行，所以不会发生线程竞争的问题，所以不需要使用锁、信号量等实现互斥。

#### 2.1.2 实验过程

①首先接收运行的线程数目和矩阵的维度，为矩阵开创空间并使用随机数初始化矩阵。

(线程数目、矩阵维度、矩阵A、B、C均为全局共享变量)

②pthread\_t threads[num\_threads]，用来存储多个线程的标识符。

③pthread\_create()创建新进程,每个线程执行multiply\_matrices()函数，每个线程的标识符作为参数传递。

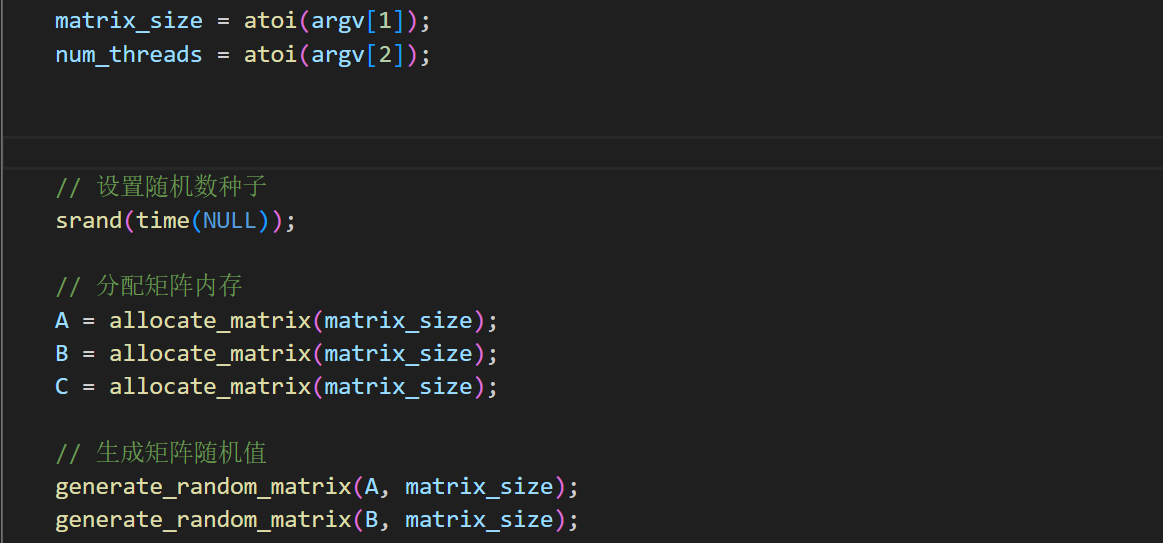
④每个线程根据线程数目和矩阵维度计算出平均每个线程需要负责计算的行数,根据传入参数的线程标识符计算出每个线程负责的对应的行，每个线程再对对应的行实现计算。

⑤使用pthread\_join()等待每个线程执行完毕，主线程最后输出结果并释放空间。

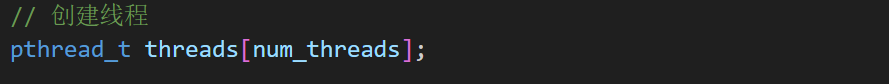
#### 2.1.3 核心代码

①首先接收运行的线程数目和矩阵的维度，为矩阵开创空间并使用随机数初始化矩阵。

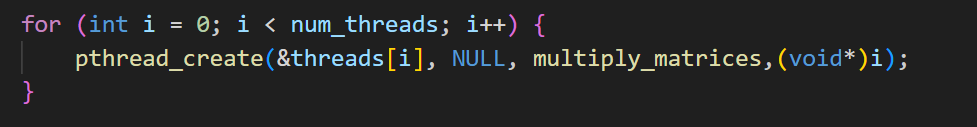
(线程数目、矩阵维度、矩阵A、B、C均为全局共享变量)



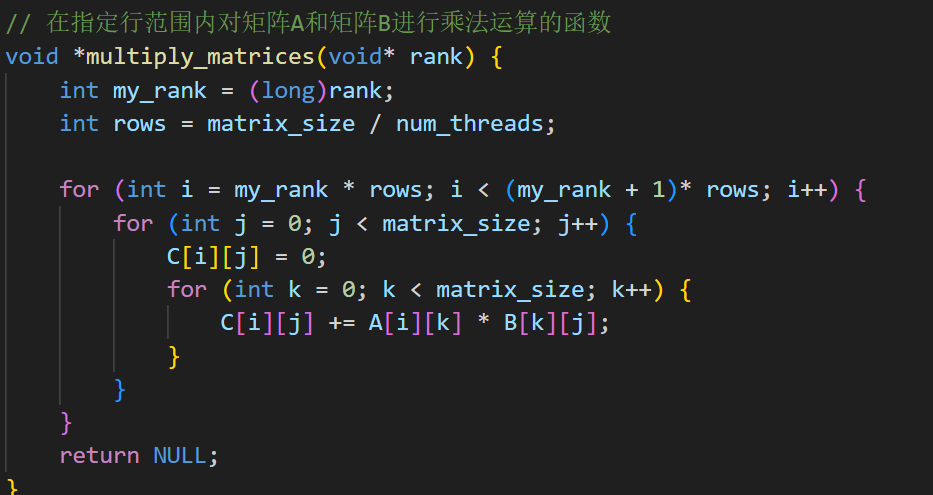
②pthread\_t threads[num\_threads]，用来存储多个线程的标识符。



③pthread\_create()创建新进程,每个线程执行multiply\_matrices()函数，每个线程的标识符作为参数传递。



④每个线程根据线程数目和矩阵维度计算出平均每个线程需要负责计算的行数,根据传入参数的线程标识符计算出每个线程负责的对应的行，每个线程再对对应的行实现计算。



⑤使用pthread\_join()等待每个线程执行完毕，主线程最后输出结果并释放空间。



### 2.2 并行数组求和

#### 2.2.1 实验思路

①将数组设为共享全局变量方便每个线程访问。

②将数组的每个元素平均地分给每个线程计算出局部和。

③每个线程将计算的局部和加到全局的数组的和total\_sum,由于此时可能会发生线程竞争访问total\_sum，所以这里使用锁来实现互斥。

#### 2.2.2 实验过程

①首先接收运行的线程数目和数组的大小，为数组开创空间并使用随机数初始化数组。

(线程数目、数组大小、结果total\_sum、数组均为全局共享变量)

②pthread\_t threads[num\_threads]，用来存储多个线程的标识符。

③pthread\_create()创建新进程,每个线程执行calculate\_partial\_sum()函数，每个线程的标识符作为参数传递。

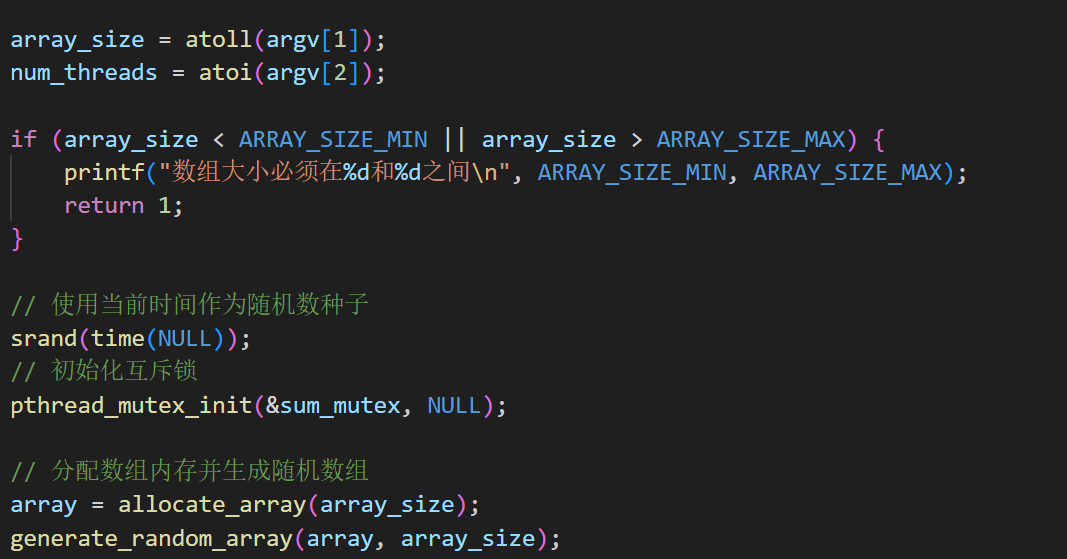
④每个线程根据线程数目和数组长度计算出平均每个线程需要负责计算的元素的数目,根据传入参数的线程标识符计算出每个线程负责的对应的起始位置和结束位置，每个线程再对对应的局部元素实现计算，并使用锁将局部变量加入到total\_sum中。

⑤使用pthread\_join()等待每个线程执行完毕，主线程最后输出结果并释放空间。

#### 2.2.3 核心代码

①首先接收运行的线程数目和数组的大小，为数组开创空间并使用随机数初始化数组。

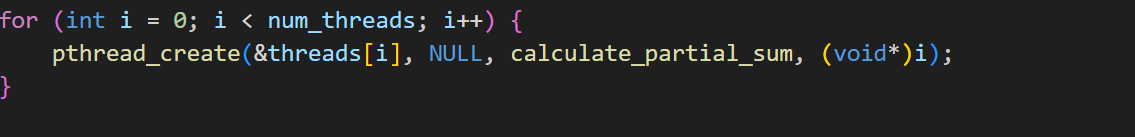
(线程数目、数组大小、结果total\_sum、数组均为全局共享变量)



②pthread\_t threads[num\_threads]，用来存储多个线程的标识符。



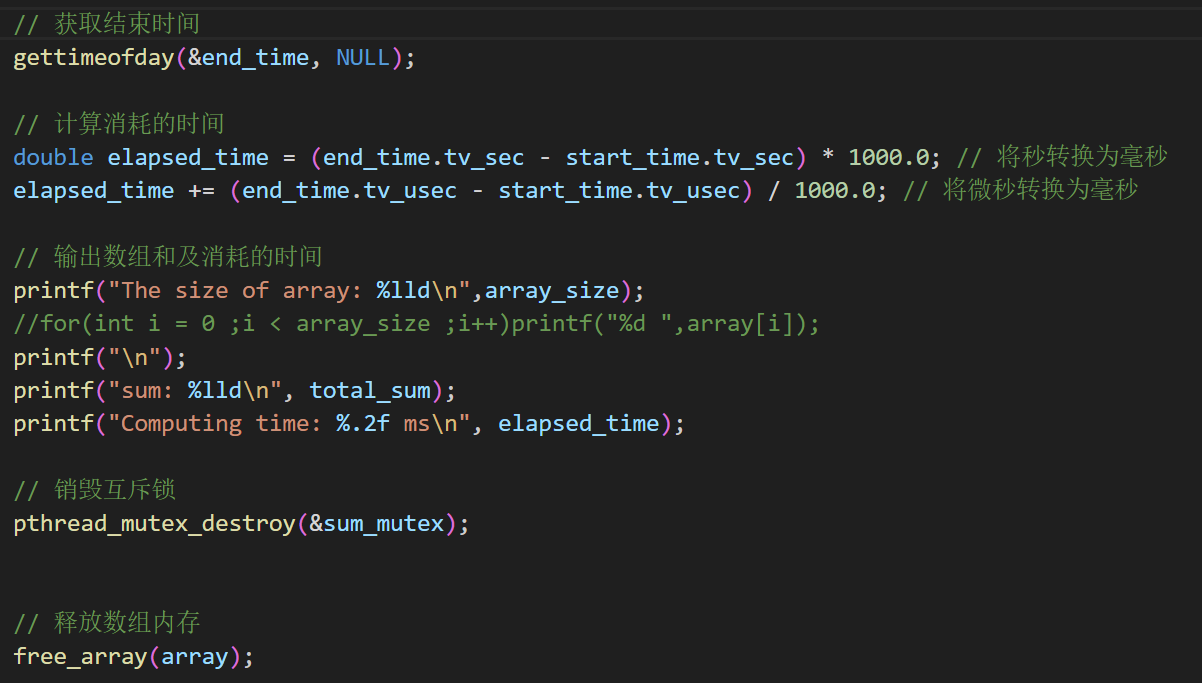
③pthread\_create()创建新进程,每个线程执行calculate\_partial\_sum()函数，每个线程的标识符作为参数传递。



④每个线程根据线程数目和数组长度计算出平均每个线程需要负责计算的元素的数目,根据传入参数的线程标识符计算出每个线程负责的对应的起始位置和结束位置，每个线程再对对应的局部元素实现计算，并使用锁将局部变量加入到total\_sum中。



⑤使用pthread\_join()等待每个线程执行完毕，主线程最后输出结果并释放空间。



# 实验结果

### 3.1 并行矩阵乘法

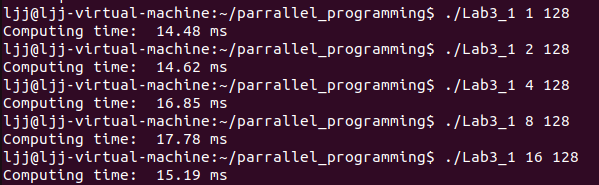
#### 3.1.1 验证实验正确性

### 

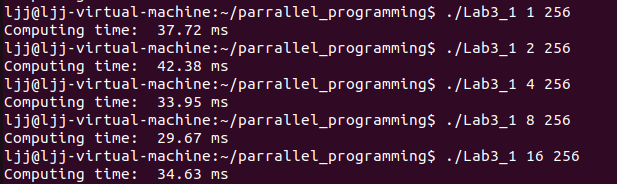
可以看到计算出来的矩阵C是正确的。

#### 3.1.2 实验结果

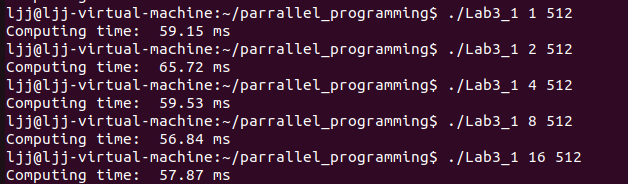
①矩阵维度为128



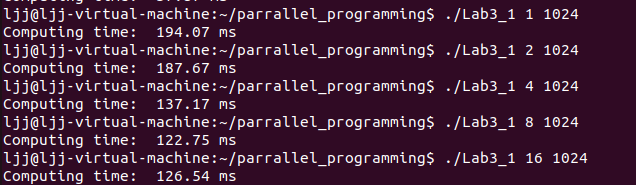
②矩阵维度为256



③矩阵维度为512



④矩阵维度为1024



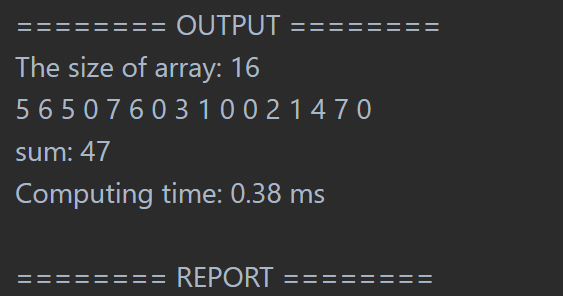
⑤矩阵维度为2048

### 

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 线程数 | 矩阵规模 | | | | |
| 128 | 256 | 512 | 1024 | 2048 |
| 1 | 14.48ms | 37.72ms | 59.15ms | 194.07ms | 400.17ms |
| 2 | 14.62.ms | 42.38ms | 65.72ms | 187.67ms | 389.47ms |
| 4 | 16.85ms | 33.95ms | 59.53ms | 137.17ms | 271.18ms |
| 8 | 17.78ms | 29.7ms | 56.84ms | 122,75ms | 238.64ms |
| 16 | 15.19ms | 34.63ms | 57.87ms | 126.54ms | 217.62ms |

### 3.2 并行数组求和

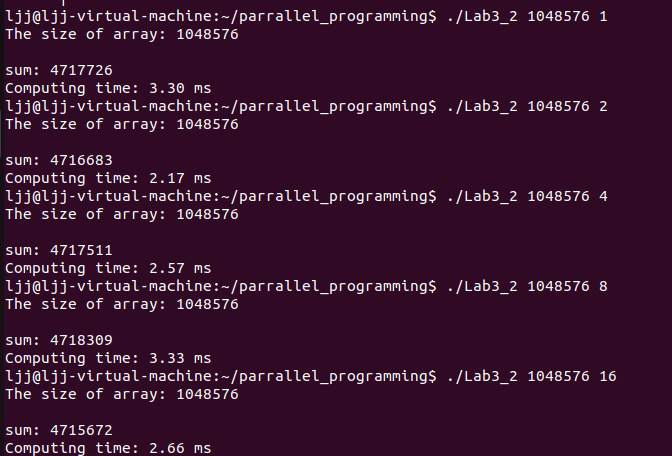
#### 3.2.1 验证实验正确性



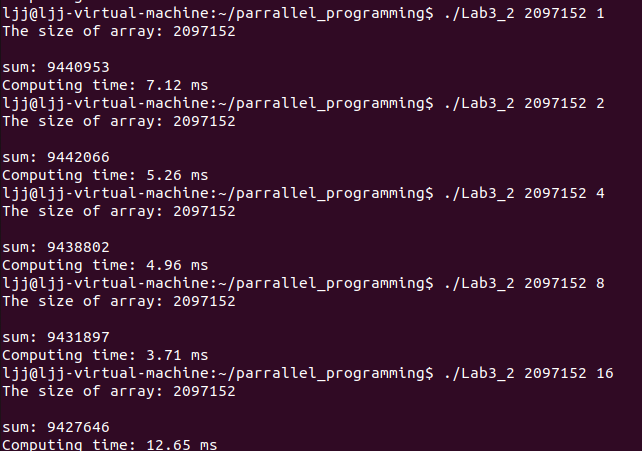
可以看到计算出来的数组和是正确的。

#### 3.2.2 实验结果

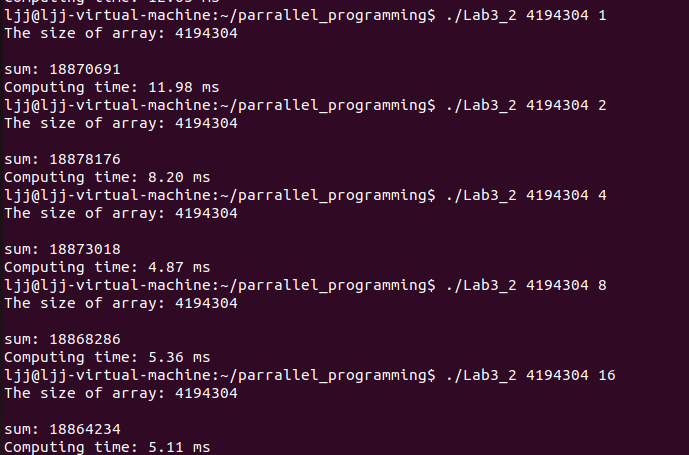
①数组长度为1M



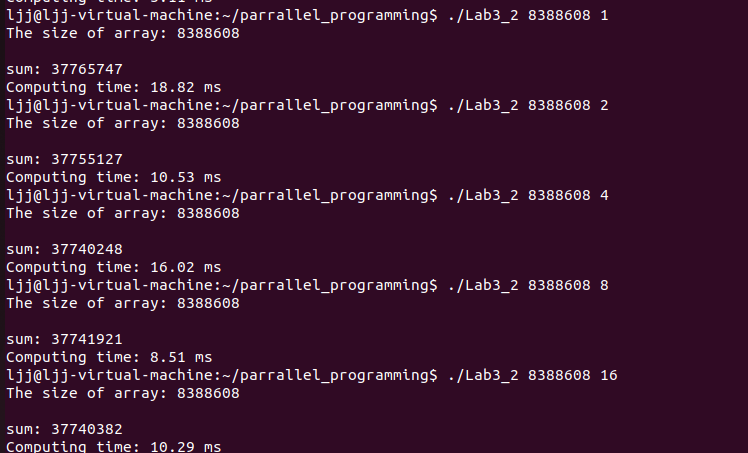
②数组长度为2M



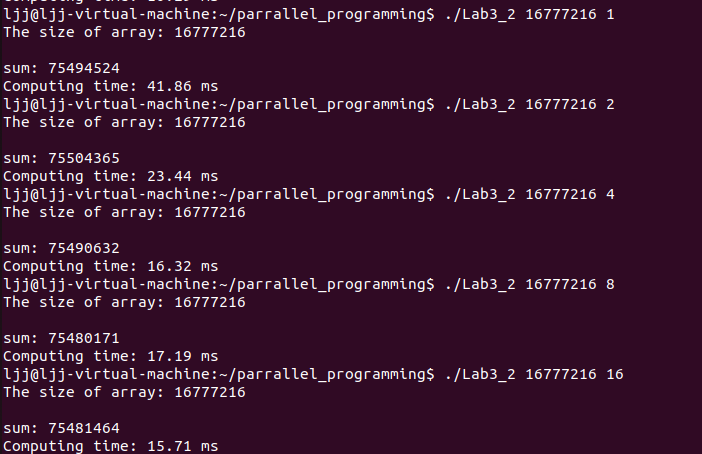
③数组长度为4M



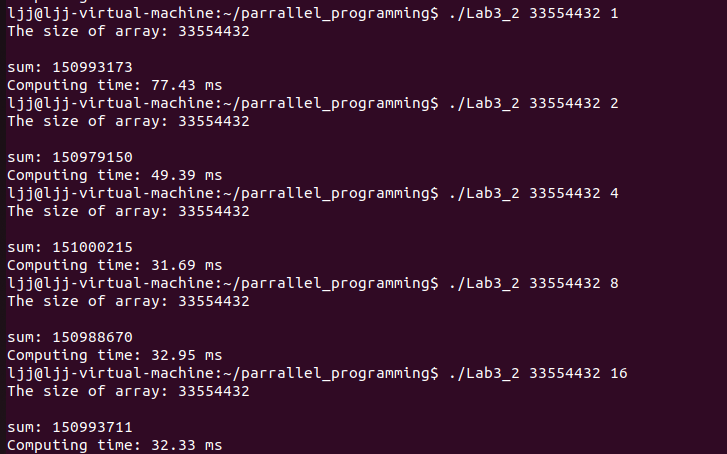
④数组长度为8M



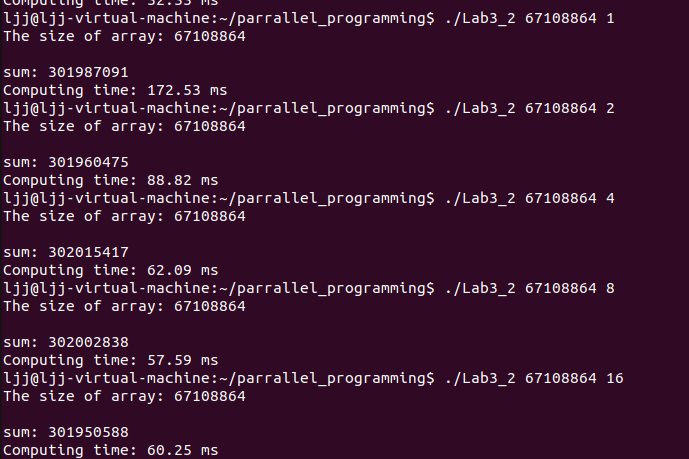
⑤数组长度为16M



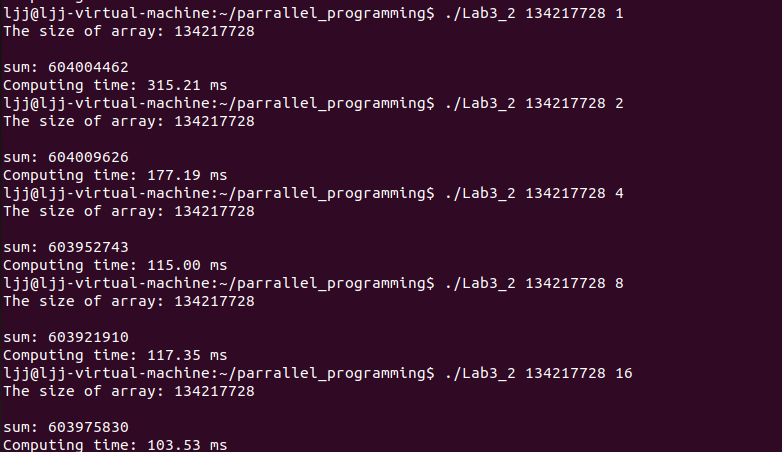
⑥数组长度为32M



⑦数组长度为64M



⑧数组长度为128M



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 线程数 | 数组长度 | | | | | | | |
| 1M | 2M | 4M | 8M | 16M | 32M | 64M | 128M |
| 1 | 3.30ms | 7.12ms | 11.98ms | 10.82ms | 41.86ms | 77.43.ms | 172.53ms | 315.21md |
| 2 | 2.17ms | 5.26ms | 8.20ms | 10.53ms | 23.44ms | 49.39.ms | 88.82ms | 177.19ms |
| 4 | 2.57ms | 4.96ms | 4.87ms | 16.02ms | 16.32ms | 31.69ms | 62.09ms | 115.00ms |
| 8 | 3.33ms | 3.71ms | 5.36ms | 8.51ms | 17.19ms | 32.95ms | 57.59ms | 117.35ms |
| 16 | 2.66ms | 12.65ms | 5.11ms | 10.29ms | 15.71ms | 32.33ms | 60.25ms | 103.53ms |

## 实验感想

并行计算是一种重要的性能优化手段，能够充分利用多核处理器的性能。

在实现并行算法时，需要考虑线程创建和销毁的开销、任务分配的平衡性、数据共享与同步等问题。

通过实验可以更好地理解并行计算的原理，并对系统性能进行优化。