**深 圳 大 学 实 验 报 告**

|  |  |
| --- | --- |
| **课程名称** | **：并行计算** |
| **实验项目名称** | **：简单排序的OpenMP并行程序** |
| **学院** | **：计算机与软件学院** |
| **专业** | **：计算机科学与技术** |
| **指导教师** | **：陆克中** |
| **报告人** | **：郑雨婷** |
| **学号** | **：2021150122** |
| **实验时间** | **：2024年04月10日** |
| **实验报告提交时间** | **：2024年04月14日** |

**教务部制**

## 一、实验目的

1. 实现简单排序的OpenMP并行程序；

2. 掌握for编译制导语句和数据划分方法；

3. 对并行程序进行简单的性能分析。

## 二、实验环境

1. 硬件环境：64核CPU、256GB内存的共享内存并行计算平台；

2. 软件环境：Ubuntu Linux、gcc、g++（g++ -O3 -fopenmp -o a.out a.cc）；

3. 远程登录：本地PowerShell中执行ssh bxjs@hpc.szu.edu.cn；

4. 传输文件：本地PowerShell中执行scp c:\a.cpp [bxjs@hpc.szu.edu.cn:/home/bxjs/](mailto:bxjs@hpc.szu.edu.cn:/home/bxjs/)或<ftp://hpc.szu.edu.cn>。

## 三、实验内容

1. 用OpenMP语言编写并行程序，排序长度为*n*的数组*a*：先将数组*a*划分成*p*个子数组，每个线程排序1个子数组，然后将这*p*个子数组归并到数组*a*或另一个数组*b*中，其中归并步骤可以串行。为了验证结果正确性，将并行计算结果和串行计算结果相比较。

2. 测试并行程序在不同线程数下的执行时间和加速比（与线程数=1时的执行时间相比）。其中，*n*固定为100000000，线程数分别取1、2、4、8、16、32、64时，为减少误差，每项实验进行5次，取平均值作为实验结果。

## 四、代码描述

#include <iostream>

#include <algorithm>

using namespace std;

int main()

{

int i, n = 1e8, p = 8; // 定义变量 n 为数组大小，p 为线程数

double\* a = new double[n]; //数组a存放随机数

double\* b = new double[n]; //数组b用于验证结果

//初始化数组a、数组b

srand(0);

for (i = 0; i < n; i++){

a[i] = rand() / (double)RAND\_MAX; //初始化数组a

b[i]=a[i]; // 数组 b 与数组 a 相同

}

//设置线程数为8

omp\_set\_num\_threads(p);

double t0 = omp\_get\_wtime(); //记录并行开始时间

int block\_size=n/p; // 计算每个线程处理的子数组大小

//并行排序代码，将一整个数组分为p个小段分别进行排序

#pragma omp parallel shared(a)

{

int tid=omp\_get\_thread\_num(); // 获取线程 ID

int start=tid\*block\_size; // 根据线程ID计算处理的**起始**位置

int end=start+block\_size; // 根据线程ID计算处理的**结束**位置

if(tid==(p-1)) end=n; // 如果是最后一个线程，结束位置为数组末尾

sort(a+start,a+end); // 使用 sort 函数对子数组进行排序

}

/\* 以下进行合并，在合并时是两两合并，所以限制了程序的线程数必须是2的幂次

本次实验中线程为1、2、4、8、16、32、64

sort\_num是当前有序段的数量，sorted\_num是当前有序段的大小\*/

int sort\_num=p,sorted\_size=block\_size;

//合并到只剩一个有序段时跳出

while(sort\_num>1){

//并行归并代码

//一次归并两段，所以一轮需要归并sort\_num/2次

#pragma omp parallel for

for (i = 0; i < sort\_num/2; ++i) {

int start=i\*2\*sorted\_size; // 计算归并起始位置

int middle=sorted\_size+start; // 计算归并中间位置

int end=middle+sorted\_size; // 计算归并结束位置

if(i==( sort\_num/2)) end=n; // 如果是最后一次归并，结束位置为数组末尾

inplace\_merge(a+start, a+middle, a+end); // inplace\_merge 函数进行归并

}

sort\_num/=2; //更新当前有序段的数量

sorted\_size\*=2; //更新当前有序段的大小

}

double t1 = omp\_get\_wtime(); //记录并行结束时间

cout << "time is " << t1 - t0 << " seconds" << endl;

//验证代码

sort(b,b+n); // 对数组 b 进行排序

int tag=0;

for (int i = 0; i < n; ++i) {

if(a[i]!=b[i]) tag=1; // 判断排序结果是否正确

}

if(tag==1) cout<<"WRONG!"<<endl;

else cout<<"RIGHT!"<<endl;

delete[] a, b; // 释放内存

return 0;

}

## 五、实验结果和分析

测试n固定为100000000，线程数分别取1、2、4、8、16、32、64时并行程序的执行时间，每项实验进行5次，测试结果如图1~图7。

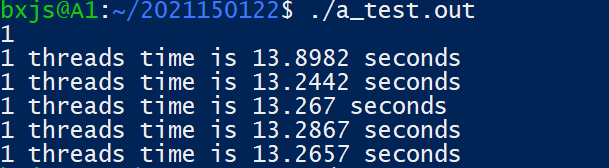
**

图1 线程数为1 时的执行时间

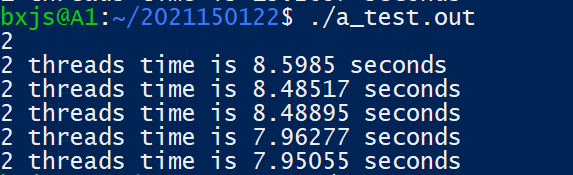
**

图2 线程数为2 时的执行时间

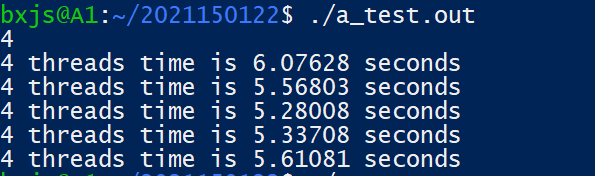
**

图3 线程数为4时的执行时间

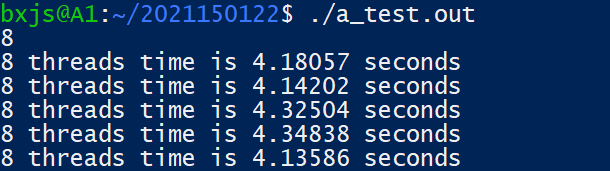
**

图4 线程数为8时的执行时间

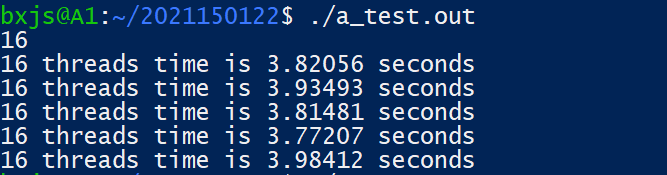
**

图5 线程数为16 时的执行时间

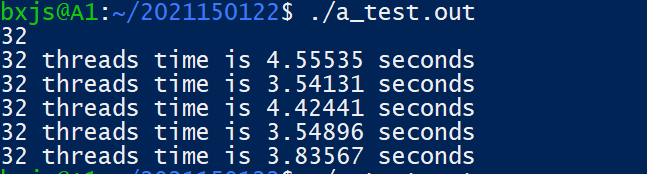
**

图6 线程数为32时的执行时间

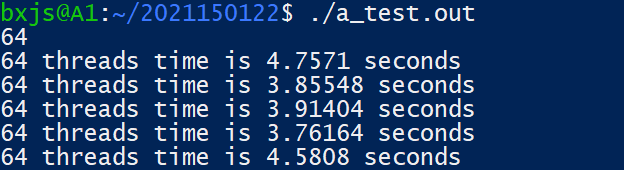
**

图7 线程数为64 时的执行时间

将程序在不同线程数下的执行时间和加速比（与线程数=1时的执行时间相比）记录在下表1 中。

表1 并行程序在不同线程数下的执行时间（秒）和加速比（*n*=100000000）

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 线程数  执行时间 | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
| 第1次 | 13.8982 | 8.5985 | 6.07628 | 4.18057 | 3.82056 | 4.55535 | 4.7571 |
| 第2次 | 13.2442 | 8.48517 | 5.56803 | 4.14202 | 3.93493 | 3.54131 | 3.85548 |
| 第3次 | 13.267 | 8.48895 | 5.28008 | 4.32504 | 3.81481 | 4.42441 | 3.91404 |
| 第4次 | 13.2867 | 7.96277 | 5.33708 | 4.34838 | 3.77207 | 3.54896 | 3.76164 |
| 第5次 | 13.2657 | 7.95055 | 5.61081 | 4.13586 | 3.98412 | 3.83567 | 4.5808 |
| 平均值 | 13.39236 | 8.297188 | 5.574456 | 4.226374 | 3.865298 | 3.98114 | 4.173812 |
| 加速比 | 1 | 1.614084 | 2.402451 | 3.168759 | 3.464768 | 3.363951 | 3.208664 |

**实验结果**：从上表1可以看出，随着线程数的增加，执行时间逐渐减少，而加速比逐渐增加。然而，随着线程数的进一步增加，加速比反而下降。在本实验中，最佳线程数是 16。同时，通过多次实验的结果，我们可以看到线程数为32和64 的时候，运行时间波动相对较大，程序的稳定性与1-16线程相比有所下降。

**分析和综合概括**：从线程数为1到线程数为16的实验结果看，随着线程数的增加，加速比逐渐增大，是符合预期的，说明并行程序在利用多核处理器的同时取得了较好的性能提升。但是在32个线程和64个线程的情况下，加速比反而下降。这是因为在归并的时候，由于每个线程的merge函数中使用的共同数组a[]，受到处理器的硬件结构和内存子系统的限制，虽然处理器具有多个核，但这些核**共享内存总线**或内存通道，因此同时访问内存的核心数量是有限制的。在线程数过大时，可能需要等待排队读写内存。

## 六、实验结论

在本次实验中，我利用了OpenMP语言的for编译制导语句，将数组a划分为p个子数组，每个线程负责排序一个子数组，之后使用归并排序将这些子数组合并。这样测试时，程序运行n=100000000且p=8的时间远大于老师给的参考时间4.6s,为了提高性能，将归并的部分也使用了for编译制导语句，运行时间明显缩短。

在使用for编译制导语的时候需要进行数据划分，我掌握到可以根据当前线程的线程ID来进行数据划分，这样就可以确保不会数据不会重叠，最终也验证了结果是正确的。

此外，我还单独测量了只对数组并行排序的时间（不包括归并部分），可以看到图8中随着线程数的增加，时间的确在减少，验证了是由于归并导致时长随线程数增加而增加，因此导致总时间长。

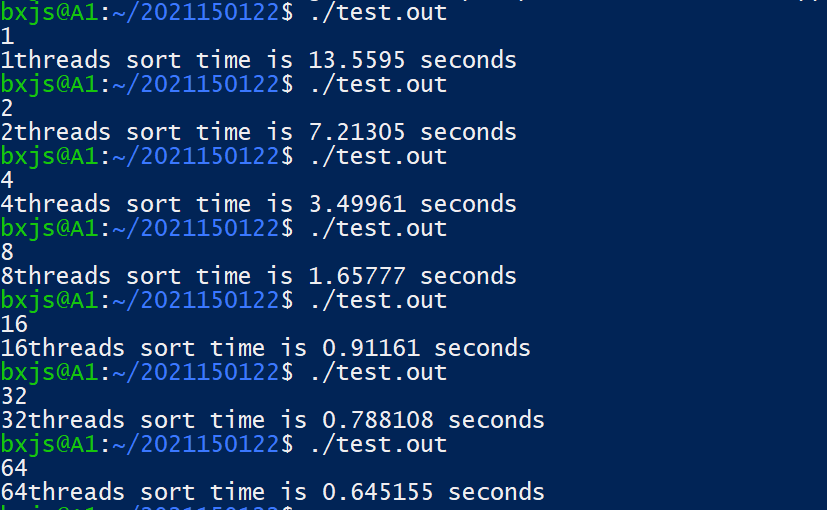


图8 测量排序部分发并行时间