《并行计算》上机报告

姓名：	魏钊	学号：	PB18111699	日期：	2021/5/13
上机题目：	OpenMP 并行编程实验				

实验环境：
CPU：i7-8750H ； 内存:16GB ;操作系统：WIN10 ;软件平台：Visual Studio 2017;

一、算法设计与分析：

题目一:
用 4 种不同并行方式的 OpenMP 实现 π 值的计算
①使用并行域并行化：设置两个线程，在并行域每个线程都会执行该代码，其中线程 0 进行迭代步 0,2,4,...线程 1 进行迭代步 1,3,5,...
②使用共享任务结构并行化程序：迭代平均分配给个线程，连续划分
③使用 private 子句和 critical 部分并行化的程序：共 2 个线程参加计算，其中线程 0 进行迭代步 0,2,4,...线程 1 进行迭代步 1,3,5,....当被指定为 critical 的代码段正在被 0 线程执行时，1 线程的执行也到达该代码段，则它将被阻塞知道 0 线程退出临界区
④使用并行规约的并行程序：每个线程保留一份私有拷贝 sum，x 为线程私有，最后对线程中所以 sum 进行+规约，并更新 sum 的全局值

题目二：
用 OpenMP 实现 PSRS 排序

begin
(1)均匀划分：将 n 个元素 A[1..n]均匀划分成 p 段，每个 pi 处理 A[(i-1)n/p+1..in/p]
(2)局部排序：pi 调用串行排序算法对 A[(i-1)n/p+1..in/p]排序
(3)选取样本：pi 从其有序子序列 A[(i-1)n/p+1..in/p]中选取 p 个样本元素
(4)样本排序：用一台处理器对 p
2 个样本元素进行串行排序
(5)选择主元：用一台处理器从排好序的样本序列中选取 p-1 个主元，并播送给其他 pi
(6)主元划分：pi 按主元将有序段 A[(i-1)n/p+1..in/p]划分成 p 段
(7)全局交换：各处理器将其有序段按段号交换到对应的处理器中
(8)归并排序：各处理器对接收到的元素进行归并排序
end.

二、核心代码：

题目一：

```c
#include <stdio.h>
#include <omp.h>
 static long num_steps = 100000;
 double step;
 #define NUM_THREADS 2
void main()
 {
    int i;
    double x, pi, sum[NUM_THREADS];
    step = 1.0 / (double)num_steps;
    omp_set_num_threads(NUM_THREADS); //设置2线程
#pragma omp parallel private(i) //并行域开始，每个线程(0和1)都会执行该代码
    {
        double x;
        int id;
        id = omp_get_thread_num();
        for (i = id, sum[id] = 0.0; i < num_steps; i = i + NUM_THREADS) {
            x = (i + 0.5)*step;
            sum[id] += 4.0 / (1.0 + x * x);
        }
    }
    for (i = 0, pi = 0.0; i < NUM_THREADS; i++) pi += sum[i] * step;
    printf("%lf\n", pi);
    getchar();
}
```

```c
#include <stdio.h>
#include <omp.h>
 static long num_steps = 100000;
 double step;
 #define NUM_THREADS 2
void main()
 {
    int i;
    double x, pi, sum[NUM_THREADS];
    step = 1.0 / (double)num_steps;
    omp_set_num_threads(NUM_THREADS); //设置2线程
#pragma omp parallel //并行域开始，每个线程(0和1)都会执行该代码
    {
        double x;
        int id;
        id = omp_get_thread_num();
        sum[id] = 0;
#pragma omp for //未指定chunk，迭代平均分配给各线程（0和1），连续划分
        for (i = 0; i < num_steps; i++) {
            x = (i + 0.5)*step;
            sum[id] += 4.0 / (1.0 + x * x);
        }
    }
    for (i = 0, pi = 0.0; i < NUM_THREADS; i++) pi += sum[i] * step;
    printf("%lf\n", pi);
    getchar();
}//共2个线程参加计算，其中线程0进行迭代步0~49999，线程1进行迭代步50000~99999*/
```

```c
#include <stdio.h>
#include <omp.h>
static long num_steps = 100000;
double step;
#define NUM_THREADS 2
void main()
{
    int i;
    double pi = 0.0;
    double sum = 0.0;
    double x = 0.0;
    step = 1.0 / (double)num_steps;
    omp_set_num_threads(NUM_THREADS); //设置2线程
#pragma omp parallel private(i,sum) //该子句表示x,sum变量对于每个线程是私有的
    {
        int id;
        id = omp_get_thread_num();
        for (i = id, sum = 0.0; i < num_steps; i = i + NUM_THREADS) {
            x = (i + 0.5)*step;
            sum += 4.0 / (1.0 + x * x);
        }
#pragma omp critical //指定代码段在同一时刻只能由一个线程进行执行
        pi += sum * step;
    }
    printf("%lf\n", pi);
    getchar();
} //共2个线程参加计算，其中线程0进行迭代步0,2,4,...线程1进行迭代步1,3,5,....
```

```c
#include <stdio.h>
#include <omp.h>
static long num_steps = 100000;
double step;
#define NUM_THREADS 2
void main()
{
    int i;
    double pi = 0.0;
    double sum = 0.0;
    double x = 0.0;
    step = 1.0 / (double)num_steps;
    omp_set_num_threads(NUM_THREADS); //设置2线程
#pragma omp parallel for reduction(+:sum) private(x) //每个线程保留一份私有拷贝s
        for (i = 1; i <= num_steps; i++) {
            x = (i - 0.5)*step;
            sum += 4.0 / (1.0 + x * x);
        }
    pi = sum * step;
    printf("%lf\n", pi);
    getchar();
} //共2个线程参加计算，其中线程0进行迭代步0~49999，线程1进行迭代步50000~99999
```

题目二：

```
#pragma omp parallel shared(a,sample,pivot,b,c,Final,Len,seg_len,seg_len_last,len) private(id,i,j,k,L)//各段局部排序
    {
        id = omp_get_thread_num();
        if (id == NUM_THREADS - 1)
        {
            sort(a + id * seg_len, a + id * seg_len + seg_len_last);
        }
        else
        {
            sort(a + id * seg_len, a + id * seg_len + seg_len);
        }
        //正则采样
        for (i = 0; i < NUM_THREADS; i++)
            sample[id*NUM_THREADS + i] = a[id*seg_len + i * seg_len / NUM_THREADS];

#pragma omp barrier//同步
```

```
#pragma omp master//采样排序
        {
            sort(sample, sample + NUM_THREADS * NUM_THREADS);
            //debug
            /*
            for (i = 0; i < NUM_THREADS*NUM_THREADS; i++)
                cout << sample[i] << " ";
            cout << "\n";
            */
            for (i = 0; i < NUM_THREADS - 1; i++)//选择主元
            {
                pivot[i] = sample[i*NUM_THREADS + NUM_THREADS];
                //cout << pivot[i] << " ";
            }
            //cout << "\n";
        }
```

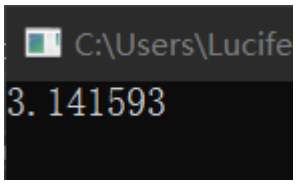
```
#pragma omp barrier//同步
        //主元划分
        if (id != NUM_THREADS - 1)//一般处理
        {
            j = 0;
            k = 0;
            for (i = 0; i < seg_len; i++)
            {
                if (a[id*seg_len + i] <= pivot[j] && j<NUM_THREADS-1)
                {
                    b[id][j][k] = a[id*seg_len + i];
                    c[id][j]++;
                    k++;
                }
                else
                {
                    if (j < NUM_THREADS - 1)
                    {
                        k = 0;
                        j++;
                        b[id][j][k] = a[id*seg_len + i];
                        c[id][j]++;
                        k++;
                    }
                    else
                    {
                        b[id][j][k] = a[id*seg_len + i];
                        c[id][j]++;
                        k++;
                    }
                }
            }
        }
```

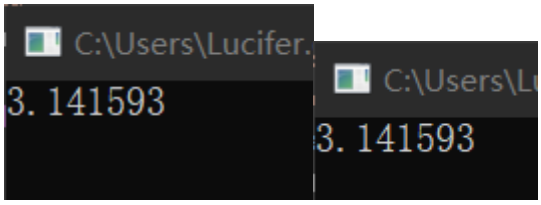
```
        else//最后一段特殊处理
        {
            j = 0;
            k = 0;
            for (i = 0; i < seg_len_last; i++)
            {
                if (a[id*seg_len + i] <= pivot[j] && j < NUM_THREADS - 1)
                {
                    b[id][j][k] = a[id*seg_len + i];
                    c[id][j]++;
                    k++;
                }
                else
                {
                    if (j < NUM_THREADS - 1)
                    {
                        k = 0;
                        j++;
                        b[id][j][k] = a[id*seg_len + i];
                        c[id][j]++;
                        k++;
                    }
                    else
                    {
                        b[id][j][k] = a[id*seg_len + i];
                        c[id][j]++;
                        k++;
                    }
                }
            }
        }
```

```
        //统计各划分总长度
        for (i = 0; i < NUM_THREADS; i++)
            Len[id] = c[i][id] + Len[id];
        //cout << id << ":" << Len[id] << "\n";
#pragma omp barrier//同步
        //全局交换
        L = 0;
        for (i = 0; i < id; i++)
            L = L + Len[i];
        //cout <<id<<" t: "<< L << "\n";
        i = 0;
        for (j = 0; j < NUM_THREADS; j++)
        {
            for (k = 0;; k++)
            {
                if (k < c[j][id])
                {
                    Final[L + i] = b[j][id][k];
                    i++;
                }
                else
                    break;
            }
        }
        sort(Final + L, Final + L + Len[id]);//局部排序
```

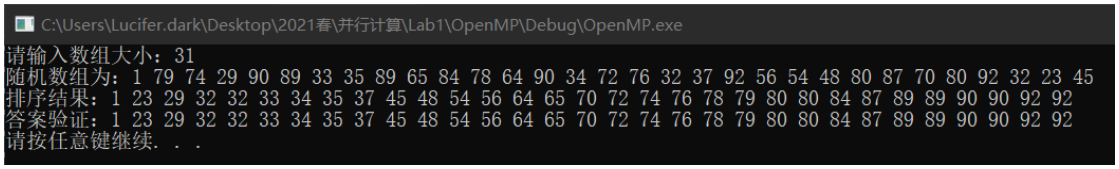
三、结果与分析：

题目一:





题目二：



四、备注（* 可选）：

有可能影响结论的因素：

PSRS 主要用于处理大数据排序，这里数组个数 n 应该大于等于(N*(N-1)+1)*N, 过小的数据量可能出错。（N 为线程数）

总结：

并行化计算可以高效利用硬件资源，将串行代码并行化，可以得到较好收益。

附录（源代码）

算法源代码（C/C++/JAVA 描述）

Pi 的源代码文档中包含，这里只给朱 PSRS 源代码。

```
#include <stdio.h>
#include <omp.h>
#include <stdlib.h>
#include <algorithm>
#include <cstdlib>
#include <ctime>
#include <iostream>
using namespace std;
#define NUM_THREADS 3


void PSRS(int *a,int len)
{
    int seg_len = len / NUM_THREADS;//每段长度
    int seg_len_last = len - (NUM_THREADS - 1)*seg_len;//特别处理最后一段
    int id;//线程号
    int i,j,k;
    int *sample=new int[NUM_THREADS*NUM_THREADS];//采样
    int *pivot = new int[NUM_THREADS - 1];//主元
    int b[NUM_THREADS][NUM_THREADS][100] = { 0 };//主元划分
    int c[NUM_THREADS][NUM_THREADS] = { 0 };//统计数目
    int *Final = new int[len];//排序结果
    int Len[NUM_THREADS] = { 0 };
    int L = 0;
    omp_set_num_threads(NUM_THREADS);
#pragma omp parallel
shared(a,sample,pivot,b,c,Final,Len,seg_len,seg_len_last,len)
private(id,i,j,k,L)//各段局部排序
    {
        id = omp_get_thread_num();
        if (id == NUM_THREADS - 1)
        {
            sort(a + id * seg_len, a + id * seg_len + seg_len_last);
        }
        else
        {
            sort(a + id * seg_len, a + id * seg_len + seg_len);
        }
        //正则采样
        for (i = 0; i < NUM_THREADS; i++)
            sample[id*NUM_THREADS + i] = a[id*seg_len + i * seg_len /
```

```
NUM_THREADS];

#pragma omp barrier//同步
        //debug
        /*
        #pragma omp critical
        {
            cout << id << ":";
            i = 0;
            while (1)
            {
                cout << a[id*seg_len + i] << " ";
                i++;
                if ((id != NUM_THREADS - 1 && i == seg_len) || (id ==
NUM_THREADS - 1 && i == seg_len_last))
                    break;
            }
            cout << "\n";
        }
        #pragma omp master
        {
            for (i = 0; i < NUM_THREADS*NUM_THREADS; i++)
                cout << sample[i] << " ";
            cout << "\n";
        }
        */
#pragma omp master//采样排序
        {
            sort(sample, sample + NUM_THREADS * NUM_THREADS);
            //debug
            /*
            for (i = 0; i < NUM_THREADS*NUM_THREADS; i++)
                cout << sample[i] << " ";
            cout << "\n";
            */
            for (i = 0; i < NUM_THREADS - 1; i++)//选择主元
            {
                pivot[i] = sample[i*NUM_THREADS + NUM_THREADS];
                //cout << pivot[i] << " ";
            }
            //cout << "\n";
        }
#pragma omp barrier//同步
        //主元划分
```

```
        if (id != NUM_THREADS - 1)//一般处理
        {
            j = 0;
            k = 0;
            for (i = 0; i < seg_len; i++)
            {
                if (a[id*seg_len + i] <= pivot[j] && j<NUM_THREADS-1)
                {
                    b[id][j][k] = a[id*seg_len + i];
                    c[id][j]++;
                    k++;
                }
                else
                {
                    if (j < NUM_THREADS - 1)
                    {
                        k = 0;
                        j++;
                        b[id][j][k] = a[id*seg_len + i];
                        c[id][j]++;
                        k++;
                    }
                    else
                    {
                        b[id][j][k] = a[id*seg_len + i];
                        c[id][j]++;
                        k++;
                    }
                }
            }
        }
        else//最后一段特殊处理
        {
            j = 0;
            k = 0;
            for (i = 0; i < seg_len_last; i++)
            {
                if (a[id*seg_len + i] <= pivot[j] && j < NUM_THREADS
- 1)
                {
                    b[id][j][k] = a[id*seg_len + i];
                    c[id][j]++;
                    k++;
                }
```

```
                else
                {
                    if (j < NUM_THREADS - 1)
                    {
                        k = 0;
                        j++;
                        b[id][j][k] = a[id*seg_len + i];
                        c[id][j]++;
                        k++;
                    }
                    else
                    {
                        b[id][j][k] = a[id*seg_len + i];
                        c[id][j]++;
                        k++;
                    }
                }
            }
        }

#pragma omp barrier//同步
//debug
/*
#pragma omp master
        {
            for (i = 0; i < NUM_THREADS; i++)
            {
                for (j = 0; j < NUM_THREADS; j++)
                {
                    for (k = 0; k < c[i][j]; k++)
                    {
                        cout << b[i][j][k] << " ";
                    }
                    cout << "\n";
                }

            }
        }
*/
        //统计各划分总长度
        for (i = 0; i < NUM_THREADS; i++)
            Len[id] = c[i][id] + Len[id];
        //cout << id << ":" << Len[id] << "\n";
#pragma omp barrier//同步
```

```
            //全局交换
            L = 0;
            for (i = 0; i < id; i++)
                L = L + Len[i];
            //cout <<id<<" t："<< L << "\n";
            i = 0;
            for (j = 0; j < NUM_THREADS; j++)
            {
                for (k = 0;; k++)
                {
                    if (k < c[j][id])
                    {
                        Final[L + i] = b[j][id][k];
                        i++;
                    }
                    else
                        break;
                }
            }
            sort(Final + L, Final + L + Len[id]);//局部排序
#pragma omp barrier//同步
#pragma omp master
            {
                cout << "排序结果：";
                for (i = 0; i < len; i++)
                    cout << Final[i] << " ";
                cout << "\n";
            }
        }
}

int main()
{
    int n,i;//数组大小
    cout << "请输入数组大小：";
    cin >> n;
    srand((int)time(0));//随机数种子
    int *a = new int[n];
    printf("随机数组为：");
    for (i = 0; i < n; i++)
    {
        a[i] = rand() % 100;
        cout << a[i] << " ";
    }
```

```
    cout << "\n";
    PSRS(a,n);
    sort(a, a + n);
    cout << "答案验证：";
    for (i = 0; i < n; i++)
        cout << a[i] << " ";
    cout << "\n";
    system("pause");
}
```