并行计算 上机报告

> 上机题目：
>
> 1. 用四种不同并行方式的OpenMP实现π值的计算
> 2. 用OpenMP实现PSRS排序
>
> 姓名：张劲暾
>
> 学号：PB16111485
>
> 日期：2019年4月27日
>
> 实验环境：
>
> CPU：Intel® Core™ i7-6500U CPU @ 2.50GHz × 4
>
> 内存：7.7 GiB
>
> 操作系统：Ubuntu 18.10 64bit
>
> 软件平台：gcc (Ubuntu 8.2.0-7ubuntu1) 8.2.0

算法设计与分析

题目一

用四种不同并行方式的OpenMP实现π值的计算：

设计：

求π的积分方法：使用公式arctan(1)=π/4以及(arctan(x))'=1/(1+x^2). 在求解arctan(1)时使用矩形法求解： 求解arctan(1)是取a=0, b=1.

$$\begin{gathered}\int_b^a f(x)dx = y_0\Delta x + y_1\Delta x + \cdots + y_{n-1}\Delta x \\ \Delta x = (b-a)/n \\ y = f(x) \\ y_i = f(a + i*(b-a)/n) \qquad i = 0, 1, 2, \ldots, n\end{gathered} \tag{1}$$

使用private子句和critical部分并行化

```c++
#include <stdio.h>
#include <omp.h>

static long num_steps = 1e5;    // 积分区间数
double step;                    // 积分步长

#define NUM_THREADS 2   // 并行线程数

void main()
{
    int i;
    double pi = 0.0, sum = 0.0, x = 0.0;
```

```
    step = 1.0 / (double)num_steps;

    omp_set_num_threads(NUM_THREADS);   // 设置线程数量
    //========================================================================
    #pragma omp parallel private(i,x,sum) // private子句表示x,sum变量对于每个线程是私有的
    {
        int id = omp_get_thread_num();
        for(i = id, sum = 0.0; i < num_steps; i = i + NUM_THREADS)
        {   // NUM_THREADS 个线程参加计算，
            // 其中线程 NUM_THREADS-1 迭代 NUM_THREADS-1, 2*NUM_THREADS-1, ... 步
            x = (i + 0.5) * step;
            sum += 4.0 / (1.0 + x * x);
        }
        //************************************************************
        #pragma omp critical            // critical代码段在同一时刻只能由一个线程执行
        {                               // 当某线程在这里执行时，其他到达该段代码的线程
            pi += sum * step;           // 被阻塞直到正在执行的线程退出临界区

        }
        //************************************************************
    }
    //========================================================================
    printf("%lf\n",pi);
}
```

结果：结果正确

```shell
$ gcc privateAndCritical.c -fopenmp -o privateAndCritical
$ ./privateAndCritical
3.141593
$
```

使用并行域并行化

```c++
// 使用并行域并行化
#include <stdio.h>
#include <omp.h>

static long num_steps = 1e5;    // 积分区间数
double step;                    // 积分步长

#define NUM_THREADS 2   // 并行线程数

void main()
{
    int i;
    double pi, sum[NUM_THREADS];
    step = 1.0 / (double)num_steps;

    omp_set_num_threads(NUM_THREADS);   // 设置线程数量
    //=============================================================================================
    #pragma omp parallel private(i)     // 并行域开始，每个线程各自执行这段代码
    {
        double x;
        int id = omp_get_thread_num();
```

```
        for(i = id, sum[id] = 0.0; i < num_steps; i = i + NUM_THREADS)
        {   // NUM_THREADS 个线程参加计算，
            // 其中线程 NUM_THREADS-1 迭代 NUM_THREADS-1, 2*NUM_THREADS-1, ... 步
            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);
}
```

结果：结果正确

```shell
$ gcc parallelRegion.c  -fopenmp -o parallelRegion
$ ./parallelRegion
3.141593
$
```

使用共享任务结构并行化

```c++
// 使用共享任务结构并行化
#include <stdio.h>
#include <omp.h>

static long num_steps = 1e5;    // 积分区间数
double step;                    // 积分步长

#define NUM_THREADS 2   // 并行线程数

void main()
{
    int i;
    double pi, sum[NUM_THREADS];
    step = 1.0 / (double)num_steps;
    omp_set_num_threads(NUM_THREADS);   // 设置线程数量
    //=========================================================================
    #pragma omp parallel                // 并行域开始，每个线程各自执行这段代码
    {
        double x;
        int id = omp_get_thread_num();
        sum[id] = 0.0;
        //*************************************************************************
        #pragma omp for                 // 未指定chunk，迭代**连续而平均地**分配给各线程
        for(i = 0; i < num_steps; i++)
        {   // 两个线程参加计算，线程0进行迭代步0~49999，线程1进行迭代步50000~99999
            x = (i + 0.5) * step;
            sum[id] += 4.0 / (1.0 + x * x);
        }
        //*************************************************************************
    }
```

```c
    //================================================================
    for(i = 0, pi = 0.0; i < NUM_THREADS; i++)
    {
        pi += sum[i] * step;
    }
    printf("%lf\n", pi);
}
```

结果：结果正确

```shell
$ gcc shareStructure.c -fopenmp -o shareStructure
$ ./shareStructure
3.141593
$
```

使用并行规约

```c
// 使用并行规约
#include <stdio.h>
#include <omp.h>

static long num_steps = 1e5;    // 积分区间数
double step;                    // 积分步长

#define NUM_THREADS 2   // 并行线程数

void main()
{
    int i;
    double pi = 0.0, sum = 0.0, x = 0.0;
    step = 1.0 / (double)num_steps;

    omp_set_num_threads(NUM_THREADS);   // 设置线程数量
    //================================================================================
    #pragma omp parallel for reduction(+:sum) private(x)
    // 每个线程保留一份私有拷贝sum，x为线程私有
    // 最后对线程中所有sum进行+规约，并更新sum的全局值
    for(i = 0; i < num_steps; i++)
    {
        x = (i + 0.5) * step;
        sum += 4.0 / (1.0 + x * x);
    }
    //================================================================================
    pi = sum * step;
    printf("%lf\n",pi);
}
```

结果：结果正确

```shell
$ gcc reduce.c -fopenmp -o reduce
$ ./reduce
3.141593
$
```

题目二

用OpenMP实现PSRS排序

设计：bingxingku

```c++
/*****************************************************************************
 * PSRS (Parallel Sorting by Regular Sampling) 排序算法：
 * STEP1 均匀划分：将n个元素A[1,...,n]均匀划分为p段，每个pi处理A[(i-1)n/p+1,...,in/p]
 * STEP2 局部排序：pi调用串行排序算法对A[(i-1)n/p+1,...,in/p]排序
 * STEP3 正则采样：pi从有序子序列A[(i-1)n/p+1,...,in/p]中选取p个样本元素
 * STEP4 采样排序：用一台处理器对p^2个样本元素进行串行排序
 * STEP5 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，并传播给其他pi
 * STEP6 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
 * STEP7 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
 * STEP8 局部排序：各处理器对接收到的元素进行局部排序
 *****************************************************************************/
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <omp.h>

#define NUM_THREADS 3   // 并行线程数

#define RANDOM_LIMIT 50
#define TEST_SIZE 81
#define SHOW_CORRECTNESS
//#define SHOW_DISTRIBUTION

double Myrandom(void){
    int Sign = rand() % 2;
    return (rand() % RANDOM_LIMIT) / pow(-1,Sign + 2);
}
void swap(int* a, int* b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
int partition(int* array, int left, int right)
{
    int x = array[right];
    int i = left - 1;
    for(int j = left; j < right; j++)
    {
        if(array[j] <= x)
        {
            swap(&array[++i],&array[j]);
        }
```

```
    }
    swap(&array[i + 1],&array[right]);
    return i + 1;
}
void quickSort(int* array, int left, int right)
{
    if(left < right)
    {
        int q = partition(array, left, right);
        quickSort(array, left, q - 1);
        quickSort(array, q + 1, right);
    }
}
void PSRSSort(int* array, int length)
{
    int base = length / NUM_THREADS;                    // 划分段长度(这里假设能整除，不能整除补无穷大)
    int sample[NUM_THREADS * NUM_THREADS];              // 正则采样数
    int pivot[NUM_THREADS - 1];                         // 主元
    int count[NUM_THREADS][NUM_THREADS] = {0};          // 各cpu段长度
    int pivotArray[NUM_THREADS][NUM_THREADS][50] = {0}; // 各cpu段

    omp_set_num_threads(NUM_THREADS);   // 设置线程数量
    //=============================================================================
    #pragma omp parallel
    {
        int id = omp_get_thread_num();
        // 并行局部排序
        quickSort(array, id * base, (id + 1) * base - 1);
        // 正则采样
        for(int j = 0; j < NUM_THREADS; j++)
        {
            sample[id * NUM_THREADS + j] = array[id * base + (j + 1) * base / (NUM_THREADS + 1)];
        }
        #pragma omp barrier             // 设置路障，同步队列中的所有线程
        //*****************************************************************************
        #pragma omp master
        {   // 主线程采样排序
            quickSort(sample, 0, NUM_THREADS * NUM_THREADS - 1);
            // 选择主元
            for(int i = 1; i < NUM_THREADS; i++)
            {
                pivot[i - 1] = sample[i * NUM_THREADS];
            }
        }
        #pragma omp barrier             // 设置路障，同步队列中的所有线程
        //*****************************************************************************
        for(int k = 0, m = 0/*主元指针*/; k < base; k++)       // 主元划分
        {
            if(array[id * base + k] < pivot[m])
            {
                pivotArray[id][m][count[id][m]++] = array[id * base + k];
            }
            else
            {
                m != NUM_THREADS - 1 ? m++ : 0;     // 最后一段的处理
                pivotArray[id][m][count[id][m]++] = array[id * base + k];
            }
        }
        //*****************************************************************************
        #pragma omp barrier             // 设置路障，同步队列中的所有线程
```

```
106             // 全局交换
107             for(int k = 0; k < NUM_THREADS; k++)
108             {
109                 if(k != id)
110                 {
111                     memcpy(pivotArray[id][id] + count[id][id], pivotArray[k][id], sizeof(int) * count[k]
    [id]);
112                     count[id][id] += count[k][id];
113                 }
114             }
115             // 局部排序
116             quickSort(pivotArray[id][id], 0, count[id][id] - 1);
117         }
118         //============================================================================
119         // 结果输出
120         #ifdef SHOW_DISTRIBUTION
121         for(int z = 0; z < NUM_THREADS; z++)
122             printf("%d\t",count[z][z]);
123         printf("\n");
124         #endif
125         #ifdef SHOW_CORRECTNESS
126         printf("The Sorted Array is:\n");
127         for(int x = 0; x < NUM_THREADS; x++)
128         {
129             for(int y = 0; y < count[x][x]; y++)
130                 printf("%d\t",pivotArray[x][x][y]);
131             printf("\n");
132         }
133         #endif
134     }
135     int main(int argc, char* argv[])
136     {   //=====================================================================
137         srand((unsigned int)time(NULL));
138         int test[TEST_SIZE];
139         for(int i = 0;i < TEST_SIZE;i++)
140             test[i] = Myrandom();
141         //=====================================================================
142         #ifdef SHOW_CORRECTNESS
143         printf("The Original结果正确 Array is:\n");
144         for(int i = 0; i < 9; i++)
145         {
146             for(int j = 0; j < 9; j++)
147                 printf("%d\t",test[i * 9 + j]);
148             printf("\n");
149         }
150         #endif
151         //=====================================================================
152         PSRSSort(test, TEST_SIZE);
153         //==================结果正确=============================================
154         return 0;
155     }
```

结果：结果正确

```shell
1   $ g++ psrs.cpp -fopenmp -o psrs -lm
2   $ ./psrs
3   The Original Array is:
4   -39 -46 -18 -38 44  8   24  16  -31
```

```
-36 0   -32 -27 12  31  0   0   42
48  44  22  44  -28 -2  6   29  29
21  16  -37 42  -20 37  14  -47 29
-14 20  -39 7   -15 5   0   3   -13
17  -31 -1  2   -6  -27 -20 11  -39
17  47  32  29  -32 4   28  -5  -10
-31 22  -13 25  44  29  -37 31  31
-41 0   -46 21  29  -37 34  4   8
The Sorted Array is:
-47 -46 -46 -41 -39 -39 -39 -38 -37 -37 -37 -36 -32 -32 -31 -31 -31 -28 -27 -27 -20 -20 -18 -15 -14 -13
-13 -10 -6  -5  -2  -1
0   0   0   0   0   2   3   4   4   5   6   7   8   8   11  12  14  16  16
17  17  20  21  21  22  22  24  25  28  29  29  29  29  29  31  31  31  32  34  37  42  42  44  44
44  44  47  48
$
```

总结

通过算法实现锻炼了并行思维，熟悉了OpenMP并行库的使用。