浙江大学实验报告

专业：计算机科学与技术
姓名：吴同
学号：3170104848
日期：2020 年 3 月 9 日

课程名称：并行计算与多核编程　指导老师：楼学庆　电子邮件：wutongcs@zju.edu.cn
实验名称：线程编程　实验类型：综合型　联系电话：18888922355

一、 实验目的

- 实现多线程 PSRS 算法
- 实现多线程高斯八皇后算法
- 分析算法的并行性和效率

二、 实验原理

1. PSRS 排序

算法的整体思想为，设待排序的数列长为 n，有 p 个线程。将 n 个元素划分为 p 段，每段有 n/p 个元素，将这 p 段分配给 p 个线程。

算法的具体流程为：首先让各线程并行调用串行排序方法对每段进行局部的排序；从每个有序段中选取 p 个样本，共 p^2 个样本；对样本元素排序；从 p 个样本中选取 $p-1$ 个主元；将 p 个线程处理的数据段根据 $p-1$ 个主元划分为 p 段；各个线程将每一段按段号交换到相应的线程；各线程使用串行排序方法再次进行局部排序。

PSRS 算法在 $n^3 > p$ 时的时间复杂度为 $O(\frac{n}{p}logn)$。

2. 高斯八皇后

高斯八皇后问题中，每个线程中采用回溯法，逐行推进，每一行逐个位置尝试，直到遍历了所有的情况。

在多线程的优化中，根据第一行中皇后的位置，将整个问题划分为若干个子问题，每个线程只需要求解一部分。

设有 n 个皇后，有 p 个线程，时间复杂度为 $O(n^n/p)$。

三、 实验过程

1. PSRS 排序

(1) 划分序列

将数据分为 p 段，第 i 段处理范围在 $[i*n/p, (i+1)*n/p)$ 的数据。

```
for (int i = 0; i < num_thread; ++i)
{
   PSRSes.push_back(std::make_shared<psrsThread>(std::vector<int>
      (dataset.begin() + i * (dataset.size() / num_thread),
      dataset.begin() + (i + 1) * (dataset.size() / num_thread))));
}
```

(2) 局部排序

每个段由一个线程进行排序，并在每个进程中选取 p 个样本。
样本的序号为 $i * n/p, i = 0, 1, ..., p-1$。

```
void sortThread(std::shared_ptr<psrsThread> t)
{
   std::sort(t->data.begin(), t->data.end());
   for (int i = 0; i < currentT; ++i)
   {
      t->sample.push_back(t->data[i * t->data.size() / currentT]);
   }
}
```

(3) 选取主元

在一个线程中对 p 个进程选出的 p^2 个样本进行排序，选取 $p-1$ 个主元。
主元的序号为 $i * p, i = 1, , ..., p-1$。

```
std::vector<int> samples;
for (const auto& it: PSRSes)
{
   samples.insert(samples.end(), it->sample.begin(), it->sample.end());
}
std::sort(samples.begin(), samples.end());
std::vector<int> mainSamples;
for (int i = 1; i < num_thread; ++i)
{
   mainSamples.push_back(i * num_thread);
}
```

(4) 划分子段

对 p 个线程中处理的数据段再进行划分，每个数据段根据主元划分为 p 个子段。

```
void divideThread(std::shared_ptr<psrsThread> t, const
    std::vector<int>& mainSamples)
{
   int index1 = 0, index2 = 0;;
```

```
    for (int i = 0; i < currentT; ++i)
    {
        if (i == currentT - 1)
        {
            t->divided.push_back(std::vector<int>(t->data.begin() +
                index1, t->data.end()));
            break;
        }
        std::vector<int> tmp;
        while (t->data[index2] <= mainSamples[i])
        {
            ++index2;
        }
        if (index1 < index2)
        {
            tmp.insert(tmp.end(), t->data.begin() + index1,
                t->data.begin() + index2);
        }
        t->divided.push_back(tmp);
        index1 = index2;
    }
}
```

(5) 数据交换

将每个数据段的第 i 个子段交换到第 i 个线程。每个线程将新接收到的各个数据段合并并排序。

```
void finalThread(std::shared_ptr<psrsThread> t, int num, const
    std::vector<std::shared_ptr<psrsThread>>& PSRSes)
{
    for (int i = 0; i < PSRSes.size(); ++i)
    {
        t->result.insert(t->result.end(),
            PSRSes[i]->divided[num].begin(),
            PSRSes[i]->divided[num].end());
    }
    std::sort(t->result.begin(), t->result.end());
}
```

(6) 生成测试数据

将有序的序列打乱，生成测试数据。

```
std::vector<int>& generateTest(const int& n)
{
    static std::vector<int> testData;
    testData.resize(n);
    for (int i = 0; i < n; ++i)
```

```
    {
        testData[i] = i;
    }
    srand(19260817);
    for (int i = n - 1; i > 0; --i)
    {
        std::swap(testData[i], testData[rand() % i]);
    }
    return testData;
}
```

(7) 正确性测试

测试排序结果是否满足后项大于前项。

```
bool testPSRS(const std::vector<int>& dataset, int num_thread)
{
    std::vector<int> data = dataset;
    PSRS_real(data, num_thread);
    for (int i = 0; i < data.size() - 1; ++i)
    {
        if (data[i] > data[i + 1])
        {
            return false;
        }
    }
    return true;
}
```

(8) 时间测试

测试多线程下的运行时间。

```
int N = it;
std::chrono::duration<double> t_span;
std::chrono::steady_clock::time_point t_start, t_end;
t_start = std::chrono::steady_clock::now();
while (N--)
{
    std::cerr<<n<<'\t'<<i<<'\t'<<it - N<<'\n';
    PSRS(testData, i);
}
t_end = std::chrono::steady_clock::now();
t_span =
    std::chrono::duration_cast<std::chrono::duration<double>>(t_end -
    t_start);
std::cout<<"Time: "<<t_span.count()<<'\n';
```

2. 高斯八皇后

(1) 划分线程

将整个问题划分成多个子问题，每个线程解决一部分。在每个子问题中，第一行所选的皇后位置已定，直接从第二行开始求解。

```
void queenThread(std::shared_ptr<queenT> t)
{
   for (int i = t->i; i < t->N; i += t->n)
   {
      matrix mat(t->N);
      mat.mat[0][i] = 1;
      int num = DFS(1, mat);
      t->result += num;
   }
}
```

(2) 深度搜索

递归实现回溯，逐层深入，通过穷举的方式求得解的个数。

```
int DFS(int row, matrix& mat)
{
   if (row == mat.n)
   {
      return 1;
   }
   int count = 0;
   for (int col = 0; col < mat.n; ++col)
   {
      int flag = 0;
      for (int i = 0; i < row; ++i)
      {
         if (mat.mat[i][col] ||
            (col + std::abs(row - i) < mat.n && mat.mat[i][col +
               std::abs(row - i)]) ||
            (col - std::abs(row - i) >= 0 && mat.mat[i][col -
               std::abs(row - i)]))
         {
            flag = 1;
            break;
         }
      }
      if (flag)
      {
         continue;
      }
      mat.mat[row][col] = 1;
      count += DFS(row + 1, mat);
```

```
            mat.mat[row][col] = 0;
        }
        return count;
    }
```

(3) 正确性测试

将多线程的运算结果与普通的单线程的结果对比。

```
bool testQueens(int n, int num_thread)
{
    matrix mat_right(n);
    int right = DFS(0, mat_right);
    int result = getQueens(n, num_thread);
    if (right == result)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

(4) 时间测试

重复 1000 次，测试多线程下的运行时间。

```
int N = it;
std::chrono::duration<double> t_span;
std::chrono::steady_clock::time_point t_start, t_end;
t_start = std::chrono::steady_clock::now();
while (N--)
{
    std::cerr<<n<<'\t'<<i<<'\t'<<it - N<<'\n';
    queens(n, i);
}
t_end = std::chrono::steady_clock::now();
t_span =
    std::chrono::duration_cast<std::chrono::duration<double>>(t_end -
    t_start);
std::cout<<"Time: "<<t_span.count()<<'\n';
```

四、 实验结果

1. PSRS 排序

测试环境：

- CPU：Intel(R) Xeon(R) Platinum 8269CY CPU @ 2.50GHz (8 vCPU)
- 操作系统：CentOS 8.0 (Linux 4.18.0)

测试数据规模为 48、480、4800、48000、480000、4800000、48000000，测试的线程数为 1、2、3、4，每个数据规模每个线程数运行 PSRS 算法 1000 次，得到的运行时间如下表：

运行时间 (s) 线程数 / 数据规模	1	2	3	4
48	0.0657805	0.105937	0.140101	0.178110
480	0.076213	0.113921	0.148665	0.188683
4800	0.321856	0.380374	0.319060	0.360365
48000	3.47843	4.41326	2.34527	2.38935
480000	40.8749	52.3339	28.2308	25.2780
4800000	463.469	395.806	297.397	209.006
48000000	5499.33	4234.56	3516.74	2346.84

各个线程的运行时间随数据规模变化的折线图如下：

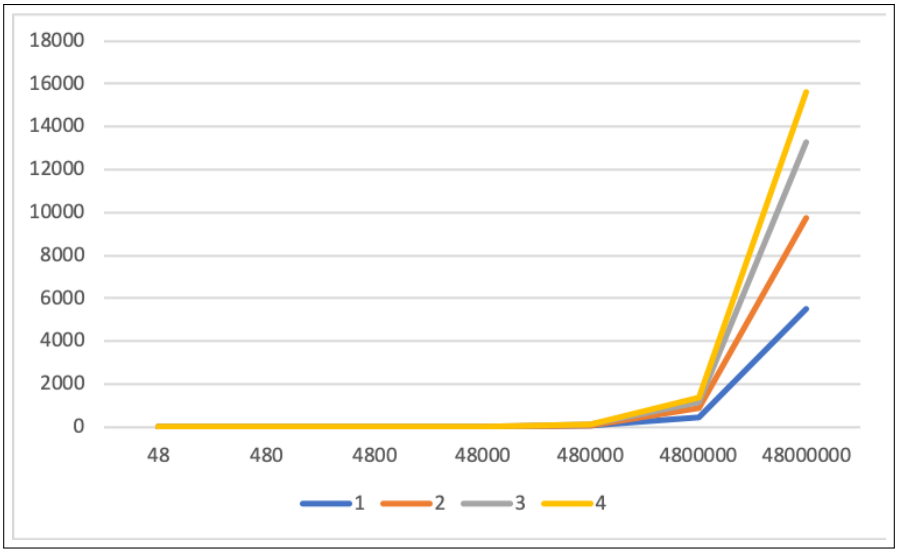

图 1: 运行时间随数据规模变化的折线图

每个数据规模下运行时间随线程数变化的折线图如下：

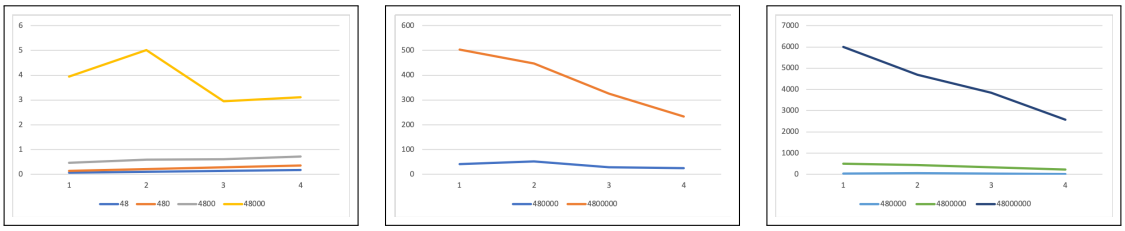

图 2: 运行时间随线程数变化的折线图

2. 高斯八皇后

测试环境:

- CPU：Intel(R) Core(TM) i5-7267U CPU @ 3.10GHz
- 操作系统：macOS Catalina 10.15.4

测试数据规模为 8，9，10，11，12，13，测试的线程数为 1、2、3、4，每个数据规模每个线程数运行 1000 次，得到的运行时间如下表:

运行时间 (s) 线程数 / 数据规模	1	2	3	4
8	0.540760	0.365119	0.348200	0.432581
9	1.92043	1.14704	1.00036	0.931476
10	8.49335	4.56479	3.72847	3.39971
11	43.5915	24.5444	17.4842	15.0890
12	238.780	130.912	98.8633	89.3587

各个线程的运行时间随数据规模变化的折线图如下:

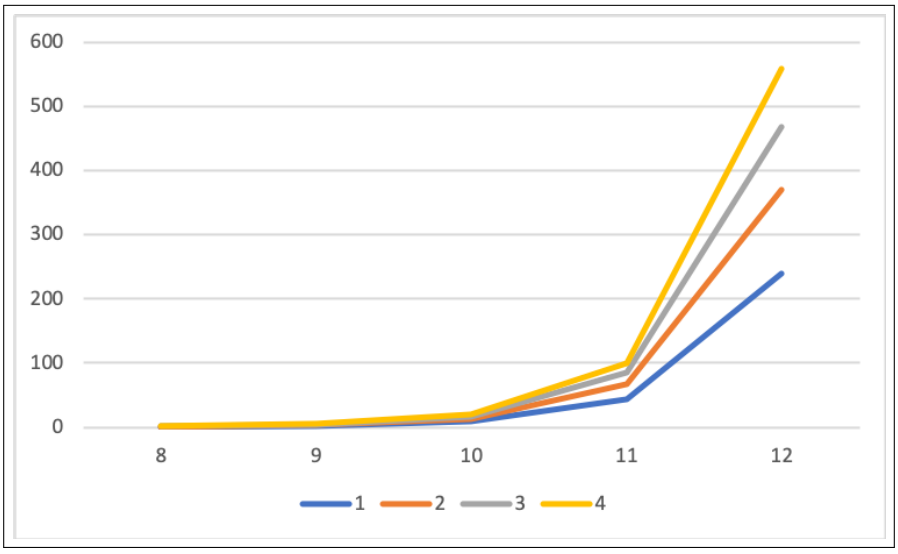

图 3: 运行时间随数据规模变化的折线图

每个数据规模下运行时间随线程数变化的折线图如下：

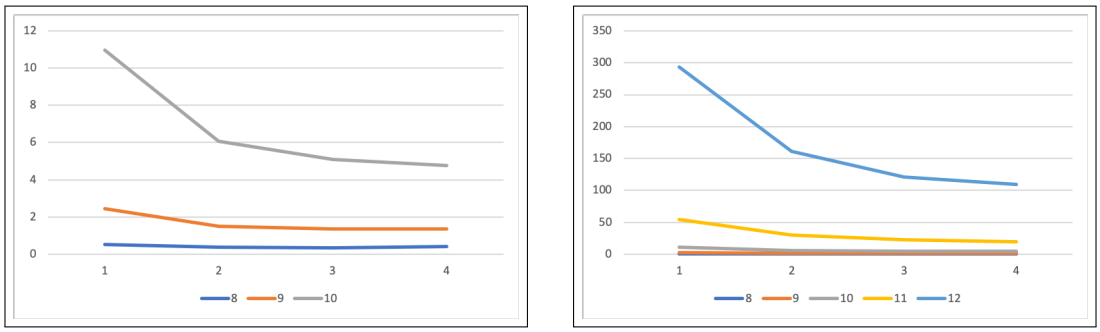

图 4: 运行时间随线程数变化的折线图

五、 分析讨论

在 PSRS 排序实验中，当数据规模较低时，多线程排序的耗时反而比单线程要高，当数据规模足够大时，才显现出多线程的优势。原因在于在创建线程等过程中，产生了较大的额外开销。

在高斯八皇后实验中，随着数据规模的增大，耗时呈指数级增长，多线程的优化效果十分明显。

在两个实验中，都出现了线程越多，增加线程时性能提升越不明显的情况。我认为其原因在于硬件性能的限制。四个线程不能同时在 CPU 上运行，所以三线程到四线程的过程中，性能提升效果甚微。

通过本次实验，我对并行算法的设计思想有了一定的理解，并对多线程编程更加熟悉。然而受限于本人电脑的 CPU 只有双核，无法真正看到多个核心同时发挥作用时的效果，这还是很遗憾的。