

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	6-Pthreads 并行构造	专业（方向）	计算机科学与技术
学号	21307174	姓名	刘俊杰
Email	liujj255@mail2.sysu.edu.cn	完成日期	2024/5/3

1. 实验目的

parallel_for 并行应用

使用此前构造的 parallel_for 并行结构，将 heated_plate_openmp 改造为基于 Pthreads 的并行应用。

heated plate 问题描述：规则网格上的热传导模拟，其具体过程为每次循环中通过对邻域内热量平均模拟热传导过程，即：

$$w_{i,j}^{t+1} = \frac{1}{4}(w_{i-1,j-1}^t + w_{i-1,j+1}^t + w_{i+1,j-1}^t + w_{i+1,j+1}^t),$$

其 OpenMP 实现见课程资料中的 heated_plate_openmp.c。

要求：使用此前构造的 parallel_for 并行结构，将 heated_plate_openmp 实现改造为基于 Pthreads 的并行应用。测试不同线程、调度方式下的程序并行性能，并与原始 heated_plate_openmp.c 实现对比。

2. 实验过程和核心代码

2.1 实验思路

①首先使用实验 5 实现的 parallel_for 来模拟实现 openmp 的加速功能。

②将原代码中的代码分为多个部分:，每个部分构造一个函数俩来实现，方便传入到 parallel_for 中多线程执行。

③按步骤将不同的函数传入到 parallel_for 中进行线程加速。

④输出迭代过程中的 diff 值和最终收敛需要的迭代次数和运行时间。

2.2 parallel.c

实验 5 实现的 parallel_for.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4
5  // 定义结构体来传递参数给函数
6  struct parallel_args {
7      void *functor_args; // 执行函数所需参数
8      int start, end, inc; // 开始位置 结束位置 增量
9      pthread_mutex_t *mutex; // 互斥锁
10 };
11
12 // 打包成动态链接库的入口函数
13 void parallel_for(int start, int end, int inc,
14                  void *(*functor)(void*), void *arg, int num_threads, pthread_mutex_t *mutex) {
15
16     pthread_t threads[num_threads];
17     struct parallel_args args_data[num_threads];
18
19     int total_task = end - start;
20     int per_task = total_task / num_threads;
21
22     // 创建线程并执行并行for循环
23     for (int i = 0; i < num_threads; ++i) {
24         args_data[i].functor_args = arg;
25         args_data[i].start = start + i * per_task;
26         args_data[i].end = start + (i + 1) * per_task;
27         args_data[i].inc = inc;
28         args_data[i].mutex = mutex;
29         pthread_create(&threads[i], NULL, functor, (void *)&args_data[i]);
30     }
31
32     // 等待所有线程结束
33     for (int i = 0; i < num_threads; ++i) {
34         pthread_join(threads[i], NULL);
35     }
36 }
37
```

2.3 pthread 版本:heat_plate_pthread.c

①用到的传参数的结构体

```
7 //函数参数的结构体
8 struct functor_args{
9     int m; // 维度M
10    int n; // 维度N
11    double *w; // 指向矩阵w
12    double *mean; //指向平均值mean
13    double *diff; // 指向判断收敛的diff
14    double *u; //指向辅助矩阵u
15 };
16 // parallel_for参数结构体
17 struct parallel_args {
18     void *functor_args;
19     int start, end, inc;
20     pthread_mutex_t *mutex; // 锁
21 };
```

②初始化矩阵 w , 使用多线程, 每个线程负责相应的位置, 给其最左侧、最右侧、最上侧和最底部进行赋值来实现加速。

```

22 // 初始化w矩阵
23 void *set_w(void *args){
24     struct parallel_args *para_args = (struct parallel_args *)args;
25     int start = para_args->start;
26     int end = para_args->end;
27     int inc = para_args->inc;
28     pthread_mutex_t *mutex = para_args->mutex;
29
30     struct functor_args* func_args = (struct functor_args *)para_args->functor_args;
31     int m = func_args->m;
32     int n = func_args->n;
33     double (*w)[n] = (double (*)[n])func_args->w;
34     // 给w矩阵赋值
35     for ( int i = start; i < end; i++ )
36     {
37         w[i][0] = 100.0;
38     }
39
40     for ( int i = start; i < end; i++ )
41     {
42         w[i][n-1] = 100.0;
43     }
44
45     for (int j = start; j < end; j++ )
46     {
47         w[m-1][j] = 100.0;
48     }
49
50     for (int j = start; j < end; j++ )
51     {
52         w[0][j] = 0.0;
53     }
54
55     return NULL;
56 }

```

③计算初始化后矩阵 w 中的元素和(方便后续计算出 mean)，结果存储到*mean 中(mean 指针指向主函数中的 mean)，计算过程中需要对互斥区实现加锁解锁操作。

```
58 // 计算矩阵元素和,用mean记录
59 void *calculate_sum(void *args){
60     struct parallel_args *para_args = (struct parallel_args *)args;
61     int start = para_args->start;
62     int end = para_args->end;
63     int inc = para_args->inc;
64     pthread_mutex_t *mutex = para_args->mutex;
65
66     struct functor_args* func_args = (struct functor_args *)para_args->functor_args;
67     int m = func_args->m;
68     int n = func_args->n;
69     double (*w)[n] = (double (*)[n])func_args->w;
70     double*mean = func_args->mean;
71
72     for (int i = start ; i < end; i++ )
73     {
74         pthread_mutex_lock(mutex); // 上锁
75         *mean = *mean + w[i][0] + w[i][n-1];
76         pthread_mutex_unlock(mutex); // 解锁
77     }
78
79     for (int j = start; j < end; j++ )
80     {
81         pthread_mutex_lock(mutex); // 上锁
82         *mean = *mean + w[m-1][j] + w[0][j];
83         pthread_mutex_unlock(mutex); // 解锁
84     }
85
86     return NULL;
87 }
```

④计算出 mean 值后, 使用多个线程对网格中间元素赋值 mean

```

88 // 给矩阵w中间元素赋值平均值mean
89 void *set_mean(void *args){
90     struct parallel_args *para_args = (struct parallel_args *)args;
91     int start = para_args->start;
92     int end = para_args->end;
93     int inc = para_args->inc;
94     pthread_mutex_t *mutex = para_args->mutex;
95
96     struct functor_args* func_args = (struct functor_args *)para_args->functor_args;
97     int m = func_args->m;
98     int n = func_args->n;
99     double (*w)[n] = (double (*)(n))func_args->w;
100     double*mean = func_args->mean;
101
102     for (int i = start; i < end; i++ )
103     {
104         if(i == m - 1 || i==0)continue;
105         for (int j = 1; j < n - 1; j++ )
106         {
107             w[i][j] = *mean;
108         }
109     }
110
111     return NULL;
112 }

```

⑤ 将 w 的值赋值给辅助矩阵 u

```

113 // 将矩阵w的值赋值给u
114 void *w_to_u(void *args){
115     struct parallel_args *para_args = (struct parallel_args *)args;
116     int start = para_args->start;
117     int end = para_args->end;
118     int inc = para_args->inc;
119     pthread_mutex_t *mutex = para_args->mutex;
120
121     struct functor_args* func_args = (struct functor_args *)para_args->functor_args;
122     int m = func_args->m;
123     int n = func_args->n;
124     double (*w)[n] = (double (*)(n))func_args->w;
125     double*mean = func_args->mean;
126
127     double (*u)[n] = (double (*)(n))func_args->u;
128
129     for (int i = start; i < end; i++ )
130     {
131         for (int j = 0; j < n ; j++ )
132         {
133             u[i][j] = w[i][j];
134         }
135     }
136
137     return NULL;
138 }

```


⑥ 多线程模拟网格热传导

```
139 // 模拟热传导
140 void *heat_transfer(void *args){
141     struct parallel_args *para_args = (struct parallel_args *)args;
142     int start = para_args->start;
143     int end = para_args->end;
144     int inc = para_args->inc;
145     pthread_mutex_t *mutex = para_args->mutex;
146
147     struct functor_args* func_args = (struct functor_args *)para_args->functor_args;
148     int m = func_args->m;
149     int n = func_args->n;
150     double (*w)[n] = (double (*)(n))func_args->w;
151     double*mean = func_args->mean;
152
153     double (*u)[n] = (double (*)(n))func_args->u;
154
155     for (int i = start; i < end; i++ )
156     {
157         if(i == m - 1 || i==0)continue;
158         for (int j = 1; j < n - 1; j++ )
159         {
160             w[i][j] = ( u[i-1][j] + u[i+1][j] + u[i][j-1] + u[i][j+1] ) / 4.0;
161         }
162     }
163
164     return NULL;
165 }
```

⑦多线程分区域计算 diff 值, 使用锁来实现对 diff 的互斥访问

```

166 // 计算diff
167 void *calculate_diff(void *args){
168     struct parallel_args *para_args = (struct parallel_args *)args;
169     int start = para_args->start;
170     int end = para_args->end;
171     int inc = para_args->inc;
172     pthread_mutex_t *mutex = para_args->mutex;
173
174     struct functor_args* func_args = (struct functor_args *)para_args->functor_args;
175     int m = func_args->m;
176     int n = func_args->n;
177     double (*w)[n] = (double (*)[n])func_args->w;
178     double (*u)[n] = (double (*)[n])func_args->u;
179     double*mean = func_args->mean;
180
181     double *diff = func_args->diff;
182
183
184     for (int i = start; i < end; i++ )
185     {
186         if(i == m - 1 || i==0)continue;
187         for (int j = 1; j < n - 1; j++ )
188         {
189             if(*diff<fabs(w[i][j] - u[i][j])){
190                 pthread_mutex_lock(mutex);
191                 *diff = fabs(w[i][j] - u[i][j]);
192                 pthread_mutex_unlock(mutex);
193             }
194         }
195     }
196     return NULL;
197 }

```

3. 实验结果

3.1 head_plate_openmp.c

openmp 版本的实验结果：

针对 500X500 的网格, 用 4 个线程的加速结果：


```

ljj@ljj-virtual-machine:~/parallel_programming/Lab6$ gcc -fopenmp heated_plate_openmp.c -o run
ljj@ljj-virtual-machine:~/parallel_programming/Lab6$ ./run

HEATED_PLATE_OPENMP
C/OpenMP version
A program to solve for the steady state temperature distribution
over a rectangular plate.

Spatial grid of 500 by 500 points.
The iteration will be repeated until the change is <= 1.000000e-03
Number of processors available = 4
Number of threads = 4

MEAN = 74.949900

Iteration  Change
      1  18.737475
      2   9.368737
      4   4.098823
      8   2.289577
     16   1.136604
     32   0.568201
     64   0.282805
    128   0.141777
    256   0.070808
    512   0.035427
   1024   0.017707
   2048   0.008856
   4096   0.004428
   8192   0.002210
  16384   0.001043

 16955   0.001000

Error tolerance achieved.
Wallclock time = 20.713869

HEATED_PLATE_OPENMP:
Normal end of execution.
ljj@ljj-virtual-machine:~/parallel_programming/Lab6$

```

3.2 parallel_for 实验结果(针对 500X500 的网格)

①1 个线程加速:

```

ljj@ljj-virtual-machine:~/parallel_programming/Lab6$ ./run 1

HEATED_PLATE_PTHREAD
C/PTHREAD version
A program to solve for the steady state temperature distribution
over a rectangular plate.

Spatial grid of 500 by 500 points.
The iteration will be repeated until the change is <= 1.000000e-03
Number of threads = 1

MEAN = 75.050100

Iteration  Change
      1  18.762525
      2   9.381263
      4   4.104302
      8   2.292638
     16   1.138124
     32   0.568961
     64   0.283183
    128   0.141967
    256   0.070903
    512   0.035474
   1024   0.017731
   2048   0.008868
   4096   0.004434
   8192   0.002213
  16384   0.001044

 16978   0.001000

Error tolerance achieved.
Wallclock time = 59.589935

HEATED_PLATE_PTHREAD:
Normal end of execution.

```

② 2 个线程加速:

```
ljj@ljj-virtual-machine:~/parallel_programming/Lab6$ ./run 2
HEATED_PLATE_PTHREAD
C/PTHREAD version
A program to solve for the steady state temperature distribution
over a rectangular plate.

Spatial grid of 500 by 500 points.
The iteration will be repeated until the change is <= 1.000000e-03
Number of threads =          2

MEAN = 75.050100

Iteration  Change
          1  18.762525
          2   9.381263
          4   4.104302
          8   2.292638
         16   1.138124
         32   0.568961
         64   0.283183
        128   0.141967
        256   0.070903
        512   0.035474
       1024   0.017731
       2048   0.008868
       4096   0.004434
       8192   0.002213
      16384   0.001044

      16978   0.001000

Error tolerance achieved.
Wallclock time = 40.103887

HEATED_PLATE_PTHREAD:
Normal end of execution.
```

③ 4 个线程加速:

```
Normal end of execution.
ljj@ljj-virtual-machine:~/parallel_programming/Lab6$ ./run 4
HEATED_PLATE_PTHREAD
C/PTHREAD version
A program to solve for the steady state temperature distribution
over a rectangular plate.

Spatial grid of 500 by 500 points.
The iteration will be repeated until the change is <= 1.000000e-03
Number of threads =          4

MEAN = 75.050100

Iteration  Change
          1  18.762525
          2   9.381263
          4   4.104302
          8   2.292638
         16   1.138124
         32   0.568961
         64   0.283183
        128   0.141967
        256   0.070903
        512   0.035474
       1024   0.017731
       2048   0.008868
       4096   0.004434
       8192   0.002213

      14838   0.000595

Error tolerance achieved.
Wallclock time = 30.738437

HEATED_PLATE_PTHREAD:
Normal end of execution.
```

④ 8 个线程加速

```

ljj@ljj-virtual-machine:~/parrallel_programming/Lab6$ ./run 8

HEATED_PLATE_PTHREAD
C/PTHREAD version
A program to solve for the steady state temperature distribution
over a rectangular plate.

Spatial grid of 500 by 500 points.
The iteration will be repeated until the change is <= 1.000000e-03
Number of threads =      8

MEAN = 74.498998

Iteration  Change
      1  18.624749
      2   9.312375
      4   4.074164
      8   2.275803
     16   1.129766
     32   0.564783
     64   0.281103
    128   0.140924
    256   0.070382
    512   0.035214
   1024   0.017601
   2048   0.008802
   4096   0.004401
   8192   0.002197

  13515  0.000361

Error tolerance achieved.
Wallclock time = 39.187043

HEATED_PLATE_PTHREAD:
Normal end of execution.

```

⑤16 个线程加速

```

ljj@ljj-virtual-machine:~/parrallel_programming/Lab6$ ./run 16

HEATED_PLATE_PTHREAD
C/PTHREAD version
A program to solve for the steady state temperature distribution
over a rectangular plate.

Spatial grid of 500 by 500 points.
The iteration will be repeated until the change is <= 1.000000e-03
Number of threads =     16

MEAN = 74.498998

Iteration  Change
      1  18.624749
      2   9.312375
      4   4.074164
      8   2.275803
     16   1.129766
     32   0.564783
     64   0.281103
    128   0.140924
    256   0.070382
    512   0.035214
   1024   0.017601
   2048   0.008802
   4096   0.004401
   8192   0.002197
  16384   0.001036

  16863  0.001000

Error tolerance achieved.
Wallclock time = 72.638829

HEATED_PLATE_PTHREAD:
Normal end of execution.

```

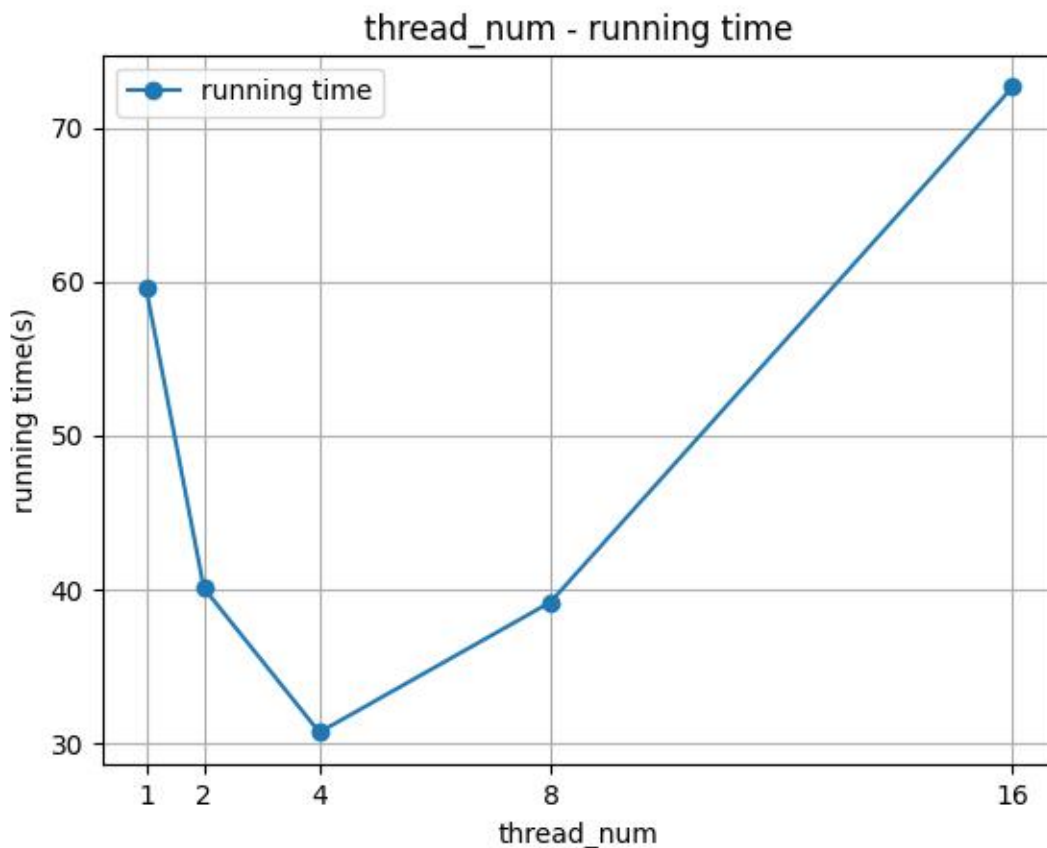
3.3 综合对比

openmp:

线程数	迭代次数	运行时间
4	16955	20.713869s

parallel_for:

线程数	迭代次数	运行时间
1	16978	59.589935s
2	16978	40.103887s
4	14838	30.738437s
8	13515	39.187043s
16	16863	72.638829s



由于虚拟机资源数有限, 不足以支持 8 线程和 16 线程的运行, 所以其运行时间反而增长了。

看从 1 个线程到 4 个线程的运行时间的减少, 可以看出 `parallel_for` 成功实现了对热传导模拟的多线程加速。

但是与 `openmp` 的 4 个线程加速的效果来看, `parallel_for` 的效果相对来说没有那么好, 应该是 `parallel_for` 只有静态调度, 没有实现其他的线程调度, 可能导致有部分线程资源浪费, 同时 `openmp` 库中可能有一些优化操作。

4. 实验感想

4.1 问题

①程序运行过程中出现段错误

原因及解决办法:

自己存储函数参数的结构体是指针, 并没有为其分配内存空间。

最后为其开辟空间后解决。

②出现线程越多, 运行时间越长的结果

原因及解决办法:

出现这种情况的原因是因为我在一开始的计时过程中使用了 `clock` 函数计时。

`clock()` 函数的功能: 这个函数返回从“开启这个程序进程”到“程序中调用 C++ `clock()` 函数”时之间的 CPU 时钟计时单元 (`clock tick`) 数当程序单线程或者单核心机器运行时, 这种时间的统计方法是正确的。但是如果要执行的代码多个线程并发执行时就会出问题, 因为最终 `end-begin` 将会是多个核心总共执行的时钟嘀嗒数, 因此造成时间偏大。

最后使用 `clock_gettime()` 来计时, 解决了这个问题。

4.2 感悟

学会了使用自己构造的 `parallel_for.c` 对特定的问题进行加速, 同时在不同问题中注意到了对共享变量的保护, 会使用锁来形成互斥去来避免竞争访问。

同时注意到了多线程程序中使用 `clock` 计时的不可靠性, 因为 `clock` 中的时钟会受到多个线程核心的影响而造成时间增加。

之后的改进可以为 `parallel_for.c` 实现可选择的不同的线程调度方式(如实现一个线程池来实现动态调度, 给每个空闲的线程分配一个任务, 避免有的线程执行完自身的任务而闲置造成资源浪费), 避免一些线程资源的浪费来进一步实现加速。