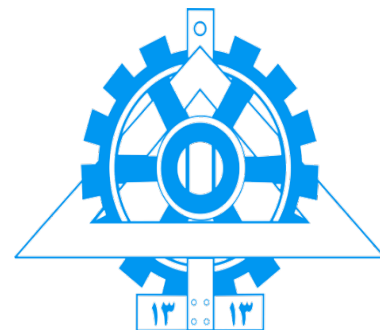


به نام خدا



دانشگاه تهران
دانشکده مهندسی برق و کامپیوتر



تمرین کامپیوتری ۳

برنامه نویسی موازی

دکتر صفری

اعضا گروه:

محمد تقی زاده گیوری ۸۱۰۱۹۸۳۷۳

نیلوفر محمدی ۸۱۰۱۹۶۶۸۷

نیمسال اول ۱۴۰۱-۰۲

سوال ۱: پیاده سازی سریال

ابتدا بزرگ ترین عنصر را برابر با اولین عنصر آرایه قرار می دهیم.

برای پیدا کردن بزرگترین عنصر آرایه و اندیس آن، در یک حلقه کل اعضا آرایه را پیمایش می کنیم:

به هر عنصر که می‌رسیم، بررسی می‌کنیم که آیا این عنصر از بزرگ ترین عنصری که تا حالا پیدا شده

(که در ابتدا اولین عنصر آرایه است)، بزرگ تر هست یا نه. اگر بزرگ تر بود پس بزرگ ترین عنصر (max_serial)

و اندیس آن (max_index_serial)، را آپدیت می‌کنیم.

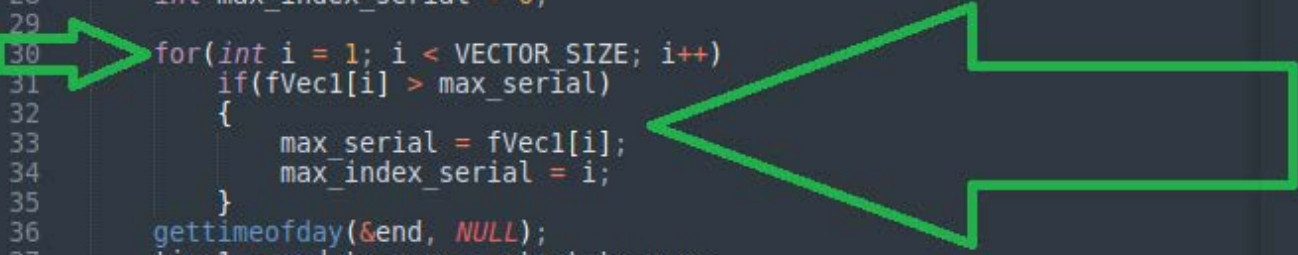
درنتیجه پس از پیمایش کل آرایه، بزرگ ترین عنصر و اندیس آن به ترتیب در متغیر های (max_serial)

و (max_index_serial)، ذخیره می‌شوند.

```

13
14     float *fVec1;
15     fVec1 = new float [VECTOR_SIZE];
16
17     if (!fVec1) {
18         printf ("Memory allocation error!!\n");
19         return 1;
20     }
21     // Initialize vectors with random numbers
22     for (long i = 0; i < VECTOR_SIZE; i++)
23         fVec1[i] = static_cast<float> (rand()) / (static_cast<float> (RA
24
25     // Serial implementation
26     gettimeofday(&start, NULL);
27     float max_serial = fVec1[0];
28     int max_index_serial = 0;
29
30     for(int i = 1; i < VECTOR_SIZE; i++)
31         if(fVec1[i] > max_serial)
32         {
33             max_serial = fVec1[i];
34             max_index_serial = i;
35         }
36     gettimeofday(&end, NULL);
37     time1 = end.tv_usec - start.tv_usec;
38

```



سوال ۱: پیاده سازی موازی

پیاده سازی موازی مشابه پیاده سازی سریال است، با این تفاوت که حلقه ای که کل آرایه را پیمایش می کند، بین

۴ ریسمان (thread)، توزیع شده و هر ریسمان (thread)، یک چهارم تکرار های حلقه را انجام می دهد.

به این ترتیب ابتدا ۴ thread، می سازیم.

به هر thread، یک چهارم تکرار های حلقه یعنی $VECTOR_SIZE / 4$ را می دهیم.

```

82 pthread_t t_h3;
83 pthread_t t_h4;
84
85 Thread_Params thread_1_params;
86 Thread_Params thread_2_params;
87 Thread_Params thread_3_params;
88 Thread_Params thread_4_params;
89
90 thread_1_params.array = fVec1;
91 thread_1_params.start_index = 0;
92 thread_1_params.end_index = (VECTOR_SIZE / 4) - 1;
93
94 thread_2_params.array = fVec1;
95 thread_2_params.start_index = VECTOR_SIZE / 4;
96 thread_2_params.end_index = (VECTOR_SIZE / 2) - 1;
97
98 thread_3_params.array = fVec1;
99 thread_3_params.start_index = VECTOR_SIZE / 2;
100 thread_3_params.end_index = ((3 * VECTOR_SIZE) / 4) - 1;
101
102 thread_4_params.array = fVec1;
103 thread_4_params.start_index = (3 * VECTOR_SIZE) / 4;
104 thread_4_params.end_index = VECTOR_SIZE - 1;
105
106
107 pthread_create(&t_h1, NULL, find_max, (void *)&thread_1_params);
108 pthread_create(&t_h2, NULL, find_max, (void *)&thread_2_params);
109 pthread_create(&t_h3, NULL, find_max, (void *)&thread_3_params);
110 pthread_create(&t_h4, NULL, find_max, (void *)&thread_4_params);
111

```

تابعی به نام `find_max`، تعریف می‌کنیم که از `start_index` تا `end_index` آرایه را پیمایش کند و بزرگ‌ترین عنصر و اندیس آن بین `start_index` و `end_index` را محاسبه کند.

```

20 void *find_max(void *arg)
21 {
22     Thread_Params *input_params = (Thread_Params *)arg;
23     float *array = input_params->array;
24     int start_index = input_params->start_index;
25     int end_index = input_params->end_index;
26
27     float max_parallel = array[start_index];
28     float max_index_parallel = start_index;
29
30     for (int i = start_index + 1; i < end_index; i++)
31     {
32         if(array[i] > max_parallel)
33         {
34             max_parallel = array[i];
35             max_index_parallel = i;
36         }
37
38     }
39
40     Thread_Results* result = new Thread_Results;
41     result->maximum = max_parallel;
42     result->maximum_index = max_index_parallel;
43     pthread_exit(result);
44 }

```

سپس با استفاده از `pthread_join()` صبر می کنیم تا کار `thread` ها تمام می شود. و خروجی `thread` ها را در یک اشاره گر به `Thread_Results` (که حاوی یک `float` برای بزرگ ترین عنصر و یک `int` برای اندیس بزرگ ترین عنصر می باشد) ذخیره می کنیم.

```

1  #include    "stdio.h"
2  #include    "sys/time.h"
3  #include    "unistd.h"
4  #include    "stdlib.h"
5  #include    "pthread.h"
6
7  #define      VECTOR_SIZE      1048576 // 2^20
8
9  typedef struct {
10     float *array;
11     int start_index;
12     int end_index;
13 }Thread_Params;
14
15 typedef struct {
16     float maximum;
17     int maximum_index;
18 }Thread_Results;
19

```

```

111
112     void *t_h1_result;
113     void *t_h2_result;
114     void *t_h3_result;
115     void *t_h4_result;
116
117     pthread_join(t_h1, &t_h1_result);
118     pthread_join(t_h2, &t_h2_result);
119     pthread_join(t_h3, &t_h3_result);
120     pthread_join(t_h4, &t_h4_result);
121
122     Thread_Results *result_th[4];
123
124     result_th[0] = (Thread_Results *) t_h1_result;
125     result_th[1] = (Thread_Results *) t_h2_result;
126     result_th[2] = (Thread_Results *) t_h3_result;
127     result_th[3] = (Thread_Results *) t_h4_result;
128

```

در آخر با یک حلقه به تعداد ۴، بین بزرگ ترین عناصر که هر thread پیدا کرده، بزرگ ترین شان را پیدا می کنیم.
در نتیجه بزرگ ترین عنصر و اندیس آن، محاسبه می شود.

```
128  
129     max_parallel = result_th[0]->maximum;  
130     max_index_parallel = result_th[0]->maximum_index;  
131     for(int i = 0; i < 3; i++)  
132         if(result_th[i]->maximum > max_parallel)  
133         {  
134             max_parallel = result_th[i]->maximum;  
135             max_index_parallel = result_th[i]->maximum_index;  
136         }  
137
```


سوال ۱: خروجی پیاده سازی سریال و موازی

در تصویر زیر، ابتدا خروجی های پیاده سازی سریال و موازی چاپ می شوند. سپس زمان اجرا پیاده سازی سریال و موازی چاپ شده و میزان تسریع، با تقسیم زمان سریال بر زمان موازی، محاسبه شده و چاپ می شود.

```

63
64     printf ("\nSerial Max = %f , index = %d\n", max_serial, max_index_serial);
65     printf ("Parallel Max = %f", max_parallel);
66     printf (" , index = %d\n", max_index_parallel);
67     printf ("Serial Run time = %ld u seconds\n", time1);
68     printf ("Parallel Run time = %ld u seconds\n", time2);
69     printf ("Speedup = %f\n\n", (float) (time1)/(float) time2);
70
71     return 0;

```

```

ariyataghizadeh@ariyataghizadeh-VirtualBox:~/Parallel_Programming/CA_2&3/CA_3/Q
1$ ./main

```

```

Serial Max = 99.999832 , index = 245298
Parallel Max = 99.999832, index = 245298
Serial Run time = 3864 u seconds
Parallel Run time = 3627 u seconds
Speedup = 1.065343

```

```

ariyataghizadeh@ariyataghizadeh-VirtualBox:~/Parallel_Programming/CA_2&3/CA_3/Q
1$ ./main

```

```

Serial Max = 99.999832 , index = 245298
Parallel Max = 99.999832, index = 245298
Serial Run time = 20176 u seconds
Parallel Run time = 9525 u seconds
Speedup = 2.118215

```

```
ariyataghizadeh@ariyataghizadeh-VirtualBox:~/Parallel_Programming/CA_2&3/CA_3/Q  
1$ ./main  
  
Serial Max = 99.999832 , index = 245298  
Parallel Max = 99.999832, index = 245298  
Serial Run time = 6835 u seconds  
Parallel Run time = 5882 u seconds  
Speedup = 1.162020
```


سوال ۲: پیاده سازی سریال

ابتدا یک عنصر از آرایه را به عنوان محور (pivot)، در نظر می گیریم که این عنصر را آخرین عنصر آرایه در نظر گرفتیم. سپس در طی یک حلقه، عناصری که کوچک تر از pivot هستند را به سمت چپ عنصر pivot، منتقل می کنیم و عناصری که بزرگ تر از pivot هستند، در سمت راست عنصر pivot قرار می گیرند. در نتیجه با فراخوانی بازگشتی quick_sort_serial، ابتدا زیر آرایه سمت چپ عنصر pivot، را مرتب می کنیم، سپس با فراخوانی بازگشتی quick_sort_serial، زیر آرایه سمت راست عنصر pivot، را مرتب می کنیم. در نتیجه آرایه به صورت صعودی مرتب می شود.

```
32 void quick_sort_serial(float *array, int start_index, int end_index)
33 {
34     if(start_index >= end_index)
35         return;
36
37     int pivot_index = partitioning_serial(array, start_index, end_index);
38
39     quick_sort_serial(array, start_index, pivot_index - 1);
40
41     quick_sort_serial(array, pivot_index + 1, end_index);
42 }
43
```

در تابع `partitioning_serial`، آرایه را طوری تغییر می دهیم که عناصر کوچک تر از `pivot`، سمت چپ `pivot` قرار گیرند و عناصر بزرگ تر از `pivot`، سمت راست `pivot` قرار گیرند. اندیس `pivot`، بیانگر موقعیت `pivot`، پس از تغییر آرایه است.

برای اینکه عناصر کوچک تر قبل از `pivot` قرار گیرند:

ابتدا اندیس `pivot` را برابر با اولین عنصر آرایه قرار می دهیم سپس طی یک حلقه کل آرایه را پیمایش می کنیم، در هر پیمایش، اگر عنصری که بررسی می کنیم از `pivot`، کوچک تر باشد، پس آن را با `array[pivot_index]` جابجا می کنیم. تا عنصر بررسی شده (که کوچک تر از `pivot` است)، به سمت چپ `pivot` منتقل شود. در نتیجه `pivot_index` را یک واحد اضافه می کنیم.

```

1  #include <stdio.h>
2  #include "sys/time.h"
3  #include "unistd.h"
4  #include "stdlib.h"
5  #include "omp.h"
6
7  #define VECTOR_SIZE 1048576 // 2^20
8
9  int partitioning_serial(float *array, int start_index, int end_index)
10 {
11     float pivot = array[end_index];
12     int pivot_index = start_index;
13     for(int i = start_index; i < end_index; i++)
14     {
15         if(array[i] <= pivot)
16         {
17             float temp = array[pivot_index];
18             array[pivot_index] = array[i];
19             array[i] = temp;
20             pivot_index++;
21         }
22     }
23     float temp = array[pivot_index];
24     array[pivot_index] = array[end_index];
25     array[end_index] = temp;
26     return pivot_index;
27 }

```