

# Parallel Programming

## Assignment IV

Omid Ahmadi  
Mohammad Mahdi Islami

# Part 1

در ابتدا شماره دانشجویی اعضای گروه را پرینت می کنیم. سپس متغیر های اولیه مانند متغیرهای زمان را تعریف می کنیم. سپس 2 به توان 20 عدد ممیز شناور تصادفی را تعریف می کنیم.

## Serial

ابتدا متغیر max را تعریف می کنیم که درون آن عنصر اول ارایه است. سپس درون حلقه for بررسی می کنیم که آیا عنصر بعدی ارایه از عنصر فعلی بزرگتر است یا نه. اگر بزرگتر بود آن را جایگزین می کنیم. همچنین اندیس آن را درون متغیر index نگه می داریم. بدین ترتیب بزرگترین عنصر ارایه به همراه اندیس آن پیدا می شود.

## Parallel

ابتدا متغیر max را تعریف می کنیم که درون آن عنصر اول ارایه است. در اینجا با استفاده از pragma omp parallel ترد های مورد نیاز را می سازیم که با استفاده از num\_threads تعداد آن را 2 ترد تعیین میکنیم چرا که بهترین حالت برای استفاده از تردها است. همچنین از عبارت simd استفاده می کنیم تا از خاصیت موازی سازی ان بهره ببریم و به افزایش سرعت ما کمک می کند. سپس درون حلقه for بررسی می کنیم که آیا عنصر بعدی ارایه از عنصر فعلی بزرگتر است یا نه. اگر بزرگتر بود آن را جایگزین می کنیم. همچنین اندیس آن را درون متغیر index نگه می داریم. بدین ترتیب بزرگترین عنصر ارایه به همراه اندیس آن پیدا می شود. حال به مقایسه جواب های حاصل، مدت زمان اجرا و نسبت آن در این دو حالت می پردازیم: همانطور که مشاهده می شود جواب های یکسان است و speedup حدود یک و نیم است.

## Result

```
omid7636@DESKTOP-ITBQI5J:/mnt/e/University/Parallel/CA4$ make
mid7636@DESKTOP-ITBQI5J:/mnt/e/University/Parallel/CA4$ ./main
IDs : 810195548 , 810195346

Serial Result: 99.9998
Index: 245298
Parallel Result: 99.9998
Index: 245298

Serial Run time = 0.002165 seconds
OpenMP Run time = 0.001965 seconds

speedup(OpenMP vs Serial) = 1.10
omid7636@DESKTOP-ITBQI5J:/mnt/e/University/Parallel/CA4$
```

## Part 2

در این قسمت می‌خواهیم مرتبسازی را در برنامه‌ی سریال و موازی مقایسه کنیم.

### Serial

هسته‌ی الگوریتم Quick Sort تابع Partition است. این تابع عنصر اول بازه‌ای را که قرار است رویش عمل کند را در نظر می‌گیرد و بعد این تابع عنصرها رو به ۲ دسته تقسیم می‌کند آن‌هایی که بزرگ‌تر از عنصر اول هستند و آن‌هایی که کوچک‌تر از عنصر اول هستند و عنصر اول را بین این دو دسته قرار می‌دهد و بعد مکانی را که عنصر ما بین این دو دسته قرار دارد را بر می‌گرداند.

```
int partition(int low, int high, float *array){
    int i, j, temp, key;
    key = array[low];
    i= low + 1;
    j= high;
    while(1){
        while(i < high && key >= array[i])
            i++;
        while(key < array[j])
            j--;
        if(i < j){
            temp = array[i];
            array[i] = array[j];
            array[j] = temp;
        } else {
            temp= array[low];
            array[low] = array[j];
            array[j]= temp;
            return(j);
        }
    }
}
```

یک روش مرتب سازی بازگشتی است، بنابراین هر بازه را به بازه های کوچکتر تقسیم می کند و دوباره خودش را فراخوانی می کند:

### Serial Quick Sort Implementation:

```
void serial_quick_sort (int p, int r, float *array){  
    if (p < r) {  
        int q = partition (p, r, array);  
        serial_quick_sort (p, q - 1, array);  
        serial_quick_sort (q + 1, r, array);  
    }  
}
```

### Parallel

از میان الگوریتم های مرتب سازی که قابل موازی سازی هستند، الگوریتم Quick Sort را انتخاب می کنیم که از نظر هزینه به نسبت مطلوب است:  
 $O(n \lg n)$

سپس به کمک OpenMp اقدام به موازی سازی این الگوریتم می کنیم. برای این منظور:

- از Parallel Task استفاده می کنیم. پس از آنکه به کمک تابع Partition که در بالا ذکر شد، آرایه را تقسیم کردیم، هر کدام از بخش های تقسیم شده را به هر یک از task های openMP می دهیم تا آن را اجرا کند.
- متغیرهایی که به task ها پاس داده می شود firstprivate هستند چرا که مقداری که در ترد اصلی initialize شده اند اهمیت دارد و باید استفاده شود.
- سپس با یک تابع موازی ساز، تابعی که در بالا ساخته شده است را اجرا می کنیم.
- در شکل های زیر قطعه کدهای نوشته شده قابل مشاهده هستند:

### Parallel Quick Sort Task Implementation:

```
void quick_sort (int p, int r, float *array, int threshold) {
    if (p < r) {
        if ((r - p) < threshold) {
            serial_quick_sort(p, r, array);
        }
        else {
            int q = partition (p, r, array);
            #pragma omp task firstprivate(array, threshold, r, q)
                quick_sort (p, q - 1, array, threshold);
            #pragma omp task firstprivate(array, threshold, r, q)
                quick_sort (q + 1, r, array, threshold);
        }
    }
}
```

Parallel Quick Sort Implementation:

```
void parallel_quick_sort (int n, float *array, int threshold) {
    #pragma omp parallel
    {
        #pragma omp single nowait
            quick_sort (0, n, array, threshold);
    }
}
```

Result:

```
SIDs : 810195548 , 810195346
```

```
-----  
Parallel: 0.0869
```

```
Serial: 0.1805
```

```
Speed-up 2.076683
```

```
mahdi@MacBook-Pro-2 Part2 %
```

# Part 3

در هر یک از موارد زیر تابع سریال نیز اجرا شده و مدت زمان و میزان تسریع نسبت به برنامه موازی متناظر گزارش شده است. میانگین هر ۶ بار اجرای موازی نیز در نظر گرفته شده است.

## Schedule( static )

```
mahdi@MacBook-Pro-2 Original % ./main
OpenMP Serial Timing for 100000 Iterations
Time Elapsed=      28155 mSecs Total=32.617277 CheckSum=100000

OpenMP Parallel Timings for 100000 Iterations
Time Elapsed=14123 mSecs Total=32.617277 CheckSum=100000
Time Elapsed=14281 mSecs Total=32.617277 CheckSum=100000
Time Elapsed=14233 mSecs Total=32.617277 CheckSum=100000
Time Elapsed=14933 mSecs Total=32.617277 CheckSum=100000
Time Elapsed=14739 mSecs Total=32.617277 CheckSum=100000
Time Elapsed=13923 mSecs Total=32.617277 CheckSum=100000
Average Time Elapsed      14372
Speed-up 1.959007
mahdi@MacBook-Pro-2 Original %
```

## schedule( dynamic, 1000 )

```
mahdi@MacBook-Pro-2 Original % ./main
OpenMP Serial Timing for 100000 Iterations
Time Elapsed=      29062 mSecs Total=32.617277 CheckSum=100000

OpenMP Parallel Timings for 100000 Iterations
Time Elapsed=9774 mSecs Total=32.617277 CheckSum=100000
Time Elapsed=9919 mSecs Total=32.617277 CheckSum=100000
Time Elapsed=10965 mSecs Total=32.617277 CheckSum=100000
Time Elapsed=10620 mSecs Total=32.617277 CheckSum=100000
Time Elapsed=10700 mSecs Total=32.617277 CheckSum=100000
Time Elapsed=9898 mSecs Total=32.617277 CheckSum=100000
Average Time Elapsed      10313
Speed-up 2.817949
mahdi@MacBook-Pro-2 Original %
```

schedule( dynamic, 2000 )

```
mahdi@MacBook-Pro-2 Original % ./main
OpenMP Serial Timing for 100000 Iterations
Time Elapsed=      27902 mSecs Total=32.617277 CheckSum=100000

OpenMP Parallel Timings for 100000 Iterations
Time Elapsed=9960 mSecs Total=32.617277 CheckSum=100000
Time Elapsed=12315 mSecs Total=32.617277 CheckSum=100000
Time Elapsed=10081 mSecs Total=32.617277 CheckSum=100000
Time Elapsed=10187 mSecs Total=32.617277 CheckSum=100000
Time Elapsed=10068 mSecs Total=32.617277 CheckSum=100000
Time Elapsed=9530 mSecs Total=32.617277 CheckSum=100000
Average Time Elapsed      10357
Speed-up 2.694013
mahdi@MacBook-Pro-2 Original %
```

## توازن بار برای ترد:

```
mahdi@MacBook-Pro-2 Modified % ./main
SID: 810195548 810195346
OpenMP Parallel Timings for 100000 iterations

-----
Time Elapsed for Thread 3 =      9238
Time Elapsed for Thread 2 =      9579
Time Elapsed for Thread 1 =      9749
Time Elapsed for Thread 0 =      9914

-----
Time Elapsed for Thread 2 =      11682
Time Elapsed for Thread 3 =      11976
Time Elapsed for Thread 1 =      12131
Time Elapsed for Thread 0 =      12143

-----
Time Elapsed for Thread 3 =      10740
Time Elapsed for Thread 0 =      10779
Time Elapsed for Thread 2 =      10815
Time Elapsed for Thread 1 =      11230

-----
Time Elapsed for Thread 0 =      9287
Time Elapsed for Thread 1 =      9292
Time Elapsed for Thread 2 =      9390
Time Elapsed for Thread 3 =      9592

-----
Time Elapsed for Thread 3 =      9179
Time Elapsed for Thread 2 =      9284
Time Elapsed for Thread 1 =      9347
Time Elapsed for Thread 0 =      9583

-----
Time Elapsed for Thread 2 =      10206
Time Elapsed for Thread 0 =      10383
Time Elapsed for Thread 3 =      10526
Time Elapsed for Thread 1 =      10771

-----
mahdi@MacBook-Pro-2 Modified %
```