Parallel Programming
CA 05

PREPARED BY

Mahdi Hadiloo - 810196587

Mohsen Fayyaz - 810196650

1 ماکسیمم

ابتدا یک استراکت برای دادن به عنوان آرگومان به هر ترد میسازیم که در آن آرایه مورد نظر و اندیس شروع و پایان قسمتی که آن ترد باید بررسی کند و شماره آن ترد داده می‌شود.
همچنین پس از اجراهای مختلف دیده شد که 5 ترد تقریبا بازدهی بهتری در این مقیاس دارد.
یک متغیر گلوبال هم ساخته می‌شود که هر ترد اندیس عدد ماکسیمم بخش خودش را در آن قرار می‌دهد.

```
#define VECTOR_SIZE 1048576
#define NUM_THREADS 5

struct Find_max_struct {
    float *v;  // Find max in vector v
    long start_index;  // From where
    long end_index;  // To where
    int thread_index;
};

long max_index[NUM_THREADS];
```

هر ترد پس از گرفتن آرگومان که مشخص می‌کند باید در کدام قسمت بگردد این کار را انجام می‌دهد و اندیس مورد نظر را در متغیر گلوبال گفته شده می‌گذارد. (چون فقط همین ترد روی آن خانه خاص می‌نویسد مشکلی از نظر ریس وجود ندارد)

```
void *parallel_posix_find_max_thread(void *arg){
    Find_max_struct *fms = (Find_max_struct*) arg;
    float local_max = (fms->v)[fms->start_index];
    long local_max_index = fms->start_index;
    // printf("%ld-%ld->%d\n", fms->start_index, fms->end_index, fms->thread_index);
    for(long i = fms->start_index; i <= fms->end_index; i++){
        if((fms->v)[i] >= local_max){
            local_max_index = i;
            local_max = (fms->v)[i];
        }
    }
    // Global Var
    max_index[fms->thread_index] = local_max_index;
    pthread_exit(NULL);
}
```

و در تابع اصلی، ترد ساخته می‌شوند و آرگومان خاص آنها ساخته می‌شود. اگر آخرین ترد باشد برای اینکه اگر تعداد ترد ها به اندازه آرایه نمی‌خورد یک شرط گذاشته شده تا هر چه باقیمانده را به ترد آخر بدهد و مشکلی پیش نیاید.
پس از آن ترد ها را تک تک جوین می‌کنیم و نتیجه آنها را ماکسیمم می‌گیریم و نتیجه آماده می‌شود.

```
void parallel_posix(float *v1)
{
    pthread_t th[NUM_THREADS];
    long each_thread_len = VECTOR_SIZE / NUM_THREADS;
    for (int i = 0; i < NUM_THREADS; i++){
        Find_max_struct *fms = new Find_max_struct;
        if(i != NUM_THREADS - 1){
            *fms = {v1, i * each_thread_len, (i + 1) * each_thread_len - 1, i};
        }else{
            *fms = {v1, i * each_thread_len, VECTOR_SIZE - 1, i};
        }
        pthread_create(&th[i], NULL, parallel_posix_find_max_thread, fms);
    }

    float max = v1[0];
    long max_i;
    for (int i = 0; i < NUM_THREADS; i++){
        pthread_join(th[i], NULL);
        if(v1[max_index[i]] >= max){
            max_i = max_index[i];
            max = v1[max_index[i]];
        }
    }

    printf("max = %f , index = %ld \n", max, max_i);
}
```

نتیجه در پایین دیده می‌شود که مشخص است حدودا مشابه کد omp است که همین انتظار هم می‌رفت.

```
mohsen@ubuntu:~/Desktop/Parallel_Programming_Course/CA_05/CA_05_Q1$ ./main
Mohsen Fayyaz - Mahdi Hadiloo
  810196650   -   810196587
max = 99.999832 , index = 245298
max = 99.999832 , index = 245298
max = 99.999832 , index = 245298
Serial Run time = 2152
Parallel_OMP Run time = 1096
Parallel_POSIX Run time = 904
Speedup(Serial/POSIX) = 2.380531
```

2 مرتب سازی

ابتدا یک استراکت برای مشخص کردن قسمتی که هر ترد باید مرتب کند و یک عدد به عنوان عمق کنونی این ترد دادیم که در یکی از حالات کد استفاده شد.

```
struct Arg_struct {
    float* v;
    long left;
    long right;
    int depth;
};
```

در حالت اول تا یک عمق خاص که مشخص می‌شود، برای هر دو بخش اطراف پارتیشن، یک ترد مجزا می‌سازیم و اگر به عمق لازم رسیدیم دیگر به صورت سریال مرتب سازی را انجام می‌دهیم.

```
void *parallelQuickSortPOSIX(void *arg){
    Arg_struct *args = (Arg_struct*) arg;
    float *v = args->v;
    long left = args->left;
    long right = args->right;
    int depth = args->depth;

    if(left < right){
        long p = partition(v, left, right);
        if(depth > MAX_THREDED_DEPTH){  // Do it serial if it's deep enough
            quickSort(v, left, p -1);
            quickSort(v, p + 1, right);
        }else{
            // printf("%ld-%ld->%d\n", left, right, depth);
            Arg_struct left_arg = {v, left, p - 1, depth + 1};
            pthread_t left_th;
            pthread_create(&left_th, NULL, parallelQuickSortPOSIX, &left_arg);

            Arg_struct right_arg = {v, p + 1, right, depth + 1};
            pthread_t right_th;
            pthread_create(&right_th, NULL, parallelQuickSortPOSIX, &right_arg);

            // parallelQuickSortPOSIX(&right_arg);

            pthread_join(left_th, NULL);
            pthread_join(right_th, NULL);
        }
    }
    pthread_exit(NULL);
}
```

نتیجه این روش در زیر دیده می‌شود. یکی از مشکلات این روش این است که پارتیشن لزوما متقارن نیست و به همین دلیل ممکن است کار یک ترد خیلی بیشتر از دیگری شود.

```
mohsen@ubuntu:~/Desktop/Parallel_Programming_Course/CA_05/CA_05_Q2$ ./main
Mohsen Fayyaz - Mahdi Hadiloo
  810196650   -   810196587
Serial Run time = 170368
Parallel_OMP Run time = 64506
Parallel_POSIX Run time = 67280
Speedup(Serial/POSIX) = 2.532223
```

به دلیلی که بالا گفته شد این بار به جای شرط براساس عمق، بر اساس تعداد اعدادی که در بازه‌ای که به ترد داده شده است عمل می‌کنیم و اگر این تعداد کمتر از یک عدد خاص بود دیگر به صورت سریال ادامه می‌دهیم. در اینجا آن عدد خاص را طول آرایه تقسیم بر یک عدد که تقریبا نشان دهنده‌ی تعداد ترد است قرار دادیم.

```
void *parallelQuickSortPOSIX(void *arg){
    Arg_struct *args = (Arg_struct*) arg;
    float *v = args->v;
    long left = args->left;
    long right = args->right;
    int depth = args->depth;

    if(left < right){
        long p = partition(v, left, right);
        if(right - left < VECTOR_SIZE / NUM_THREADS){  // Do it serial
            quickSort(v, left, p -1);
            quickSort(v, p + 1, right);
        }else{
            // printf("%ld-%ld->%d\n", left, right, depth);
            Arg_struct left_arg = {v, left, p - 1, depth + 1};
            pthread_t left_th;
            pthread_create(&left_th, NULL, parallelQuickSortPOSIX, &left
```

و همانطور که دیده می‌شود در این حالت سرعت خیلی بیشتر شده و به بیش از 3 برابر سرعت سریال رسیدیم.

```
mohsen@ubuntu:~/Desktop/Parallel_Programming_Course/CA_05/CA_05_Q2$ ./main
Mohsen Fayyaz - Mahdi Hadiloo
  810196650   -   810196587
Serial Run time = 170170
Parallel_OMP Run time = 57824
Parallel_POSIX Run time = 52312
Speedup(Serial/POSIX) = 3.252982
```