برنامه‌سازی موازی

گزارش پروژه‌ی پنجم

سید پارسا حسینی‌نژاد 810196604

کیمیا خبیری 810196606

سوال اول

میخواهیم برنامه‌ای بنویسیم که به دو روش سریال و موازی بزرگترین عنصر یک آرایه‌ی ۲ به توان ۲۰ خانه‌ای به همراه ایندکس آن را پیدا کند.

در حالت سریال باید روی تمام عناصر این آرایه حلقه‌ی for بزنیم و عنصری که از تمام عناصر دیگر بزرگتر است را پیدا کنیم. پیاده‌سازی روش سریال در تصویر زیر آمده است.

```
for (i = 0; i < DATA_SIZE; i++)
    if (a[i] > maxS) {
        maxS = a[i];
        maxSIdx = i;
    }
```

در حالت موازی آرایه را به ۴ بخش تقسیم می کنیم و مسئولیت پیدا کردن ماکزیموم در هر بخش را به یک ترد می سپاریم. در نهایت بزرگترین عدد از میان ۴ عدد پیدا شده ماکسیموم آرایه است. برای پیاده‌سازی به روش موازی از ایجاد ترد از کتابخانه‌ی pthread استفاده شده که این پیاده‌سازی در تصویر زیر قابل مشاهده است.

```
// creating threads
for (int i = 0; i < NUM_OF_THREADS; i++) {
    pthread_create(&th[i], NULL, findMaximum, (void*) &in_param[i]);
}

// joining threads
for (int i = 0; i < NUM_OF_THREADS; i++) {
    pthread_join(th[i], &thread_result);
    return_value[i] = (out_param_t*) thread_result;
}

// finding maximums
float maxP = return_value[0]->localMax;
int maxPIdx = return_value[0]->localMaxIdx;

for (int i = 1; i < NUM_OF_THREADS; i++) {
    if (return_value[i]->localMax > maxP) {
        maxP = return_value[i]->localMax;
        maxPIdx = return_value[i]->localMaxIdx;
    }
}
```

پارامترهای هر ترد را نیز به شکل زیر مشخص میکنیم که شامل شماره‌ی ترد و اندیس آغازین و پایانی اختصاص داده شده به هر ترد

است.

```
in_param_t in_param[NUM_OF_THREADS] = {
    {0, 0, DATA_SIZE/4},
    {1, DATA_SIZE/4, DATA_SIZE/2},
    {2, DATA_SIZE/2, DATA_SIZE*3/4},
    {3, DATA_SIZE*3/4, DATA_SIZE}
};
```

دقت شود که هر چهار ترد یک تابع با نام findMaximum را اجرا می‌کنند که همانطور که در کلاس اشاره شد، ورودی و خروجی اینها void* است و باید از casting برای گرفتن ورودی استفاده کنیم. این تابع اندیس و خود ماکزیمم محلی را پیدا کرده و در یک struct میریزد و بر میگرداند. دقت شود این struct با malloc ساخته شده تا با تمام شدن stack مربوط به این تابع، فضای مربوطه آزاد نشود و در main آزاد می‌شود.

```
void* findMaximum(void* arg) {
    in_param_t* inp = (in_param_t*) arg;
    int localMaxIdx = inp->startIndex;
    float localMax = a[inp->startIndex];

    for (int i = inp->startIndex; i < inp->endIndex; i++) {
        if (a[i] > localMax) {
            localMax = a[i];
            localMaxIdx = i;
        }
    }

    out_param_t *out_param = (out_param_t*) malloc(sizeof(out_param_t));
    out_param->localMax = localMax;
    out_param->localMaxIdx = localMaxIdx;

    pthread_exit(out_param);
}
```

در نهایت speedup روش موازی نسبت به سریال چیزی حدود 2.8 برابر بدست آمد.

```
~/Desktop/Parallel Programming/CA5/Q1  g++ main.cpp -o main -lpthread
~/Desktop/Parallel Programming/CA5/Q1  ./main
Kimia Khabiri: 810196606 - Parsa Hoseininejad: 810196604
Parallel output: 1049086.000000 with index: 1047869, Serial output: 1049086.000000 with index: 1047869
Serial Run time = 8535
Parallel Run time = 3048
Speedup = 2.800197
```

سوال دوم

میخواهیم برنامه‌ای بنویسیم که به دو روش سریال و موازی آرایه‌ای با ۲ به توان ۲۰ عدد ممیز شناور را مرتب کند.

در روش سریال quick sort روی تمام آرایه اعمال شده است. پیاده‌سازی تابع quick sort و اعمال آن روی تمامی آرایه در کدهای ضمیمه قابل مشاهده است.

جهت بهبود بخشیدن در سرعت مرتب‌سازی آرایه، در روش موازی به جای اعمال quicksort روی تمامی آرایه، آرایه را به ۴ بخش تقسیم کردیم و به هر ترد وظیفه‌ی مرتب کردن یکی از بخش‌ها به وسیله‌ی روش quick sort را سپرده‌ایم. عملیات quick sort در هر thread به‌وسیله‌ی تابع quickSortParallel انجام میشود که این تابع صرفا ورودی void* را به ورودی مورد نیاز تابع quick sort اصلی تبدیل میکند. پارامترهای هر ترد را نیز به شکل زیر مشخص میکنیم که شامل شماره‌ی ترد و اندیس آغازین و پایانی اختصاص داده شده به هر ترد است.

```
void* quickSortParallel(void* arg) {
    in_param_t* inp = (in_param_t*) arg;

    quickSort(p, inp->startIndex, inp->endIndex);

    pthread_exit(NULL);
}
```

```
in_param_t in_param[NUM_OF_THREADS] = {
    {0, 0, DATA_SIZE/4 - 1},
    {1, DATA_SIZE/4, DATA_SIZE/2 - 1},
    {2, DATA_SIZE/2, DATA_SIZE*3/4 - 1},
    {3, DATA_SIZE*3/4, DATA_SIZE - 1}
};
```

در نهایت خروجی‌های تولید شده توسط ترد ها را به‌وسیله‌ی merge sort مرتب کردیم و به آرایه‌ی مرتب نهایی رسیدیم. مراحل گفته شده با استفاده از کتابخانه‌ی pthread پیاده‌سازی شده و در کد ضمیمه قابل مشاهده است.

```
// creating threads
for (i = 0; i < NUM_OF_THREADS; i++) {
    pthread_create(&th[i], NULL, quickSortParallel, (void*) &in_param[i]);
}

// joining threads
for (i = 0; i < NUM_OF_THREADS; i++) {
    pthread_join(th[i], NULL);
}

// merging sorted parts
mergeSortedArrays(p, 0, DATA_SIZE/4 - 1, p, DATA_SIZE/4, DATA_SIZE/4 * 2 - 1, sOut1);
mergeSortedArrays(p, DATA_SIZE/4 * 2, DATA_SIZE/4 * 3 - 1, p, DATA_SIZE/4 * 3, DATA_SIZE - 1, sOut2);
mergeSortedArrays(sOut1, 0, DATA_SIZE/2 - 1, sOut2, 0, DATA_SIZE/2 - 1, p);
```

در نهایت speedup روش موازی نسبت به سریال چیزی حدود 2.4 برابر بدست آمد.

```
~/Desktop/Parallel Programming/CA5/Q2  g++ main.cpp -o main -lpthread
~/Desktop/Parallel Programming/CA5/Q2  ./main
Kimia Khabiri: 810196606 - Parsa Hoseininejad: 810196604
Serial Run time = 199300
Parallel Run time = 81652
Speedup = 2.440846
```