به نام خدا

گزارش کار تمرین کامپیوتری اوّل درس برنامه سازی موازی

محمّد مریدی 810195551

محمّد حسین عزیزیان 810195431

کدهای مربوط به این پروژه در کنار گزارش کار ضمیمه شده است. همچنین این کدها از طریق لینک گیت هاب زیر نیز قابل دسترس می باشد:

https://github.com/Moridi/Parallel-Programming-CA-1

سوال اول)

به منظور پیدا کردن کمینه اعداد در مجموعه ای از اعداد ممیز شناور ساده، لازم است که تمام درایه های آرایه مورد بررسی قرار گیرد و کمینه اعداد محاسبه گردد. قطعه کد زیر مربوط به محاسبه این کمینه به صورت سریال می باشد:

```
Ipp64u find_min_serial()
{
    Ipp64u start, end, serial_duration;
    float min_value = FLT_MAX;

    start = ippGetCpuClocks();

    for (long i = 0; i < ARRAY_SIZE; i++)
    if (vector[i] < min_value)
        min_value = vector[i];

    end = ippGetCpuClocks();
    serial_duration = end - start;

    printf("Serial Run time = %d , Min Value = %f\n", (Ipp32s)serial_duration, min_value);
    return serial_duration;
}
```

به منظور موازی سازی این عملیات از دستورات SIMD مربوط به پردازنده Intel استفاده می کنیم. بدین منظور از تابع _mm_min_ps استفاده شده است که بدین صورت عمل می کند که با داده ی 128 بیتی بصورت 4 عدد ممیز شناور برخورد می کند و حاصل کمینه هر زوج اعداد ورودی را در جایگاه متناظر مقصد نوشته و در نهایت یک اعداد 128 بیتی که که معرف چهار عدد کمینه از مقایسه 4 زوج عدد می باشد را برمی گرداند. با تعیین مقدار اولیه یک داده 128 بیتی به 4 عدد ممیز شناور بزرگ

(بیشتر از بیشترین عددی که ممکن است در آرایه وجود داشته باشد)، در هر مرحله 4 زوج کمینه را در این متغیر ذخیره کرده و در نهایت برای محاسبه کمینه نهایی بین 4 نامزد کمینه بودن، کوچکترین را محاسبه میکنیم.

```
Ipp64u find_min_parallel()
{
    Ipp64u start, end, parallel_duration;
    start = ippGetCpuClocks();

    __m128 min_values = _mm_set1_ps(MAX_FLOAT);

    for (long i = 0; i < ARRAY_SIZE; i += 4)
        min_values = _mm_min_ps(min_values, _mm_loadu_ps(&vector[i]));

    float min_value = MAX_FLOAT;

    for (int i = 0; i < 4; i++)
        if (min_values.m128_f32[i] < min_value)
            min_value = min_values.m128_f32[i];

    end = ippGetCpuClocks();
    parallel_duration = end - start;
    printf("Parallel Run time = %d , Min Value = %f\n", (Ipp32s)parallel_duration, min_value);
    return parallel_duration;
}
```

متوسط میزان تسریعی که موازی سازی برنامه به ما ارائه میدهد برابر **1.593** محاسبه گردید.

سوال دوم)

برای محاسبه مقدار خواسته شده، لازم است که تمام درایه های مربوط به بردار مورد بررسی قرار گیرد و متناظر با هر اندیس، مقدار معادل از بردار را برای محاسبه مجموع قدرمطلق فاصله های درایه ای بردار استفاده کنیم. قطعه کد زیر مربوط به محاسبه سریال این مقدار می باشد:

```
for (long i = 0; i < ARRAY_SIZE; i++)
{
    sub = vector1[i] - vector2[i];
    sub *= sub;
    result += sub;
}

result = sqrt(result);
```

برای اجرای موازی این برنامه لازم است که در ابتدا متغیر جهت ذخیره سازی مجموع مربع تفاضلات تعریف می کنیم. از آنجایی که از داده های ممیز شناور ساده استفاده میکنیم، یک متغیر با استفاده از تابع _mm_set1_ps مقدار اولیه مربوط هر 4 عدد موجود در این 128 بیت را برابر با 0 مقداردهی میکنیم.

```
__m128 sum = _mm_set1_ps(0.0f);
```

حال لازم است که در هر مرحله عملیات تعیین شده در صورت سوال را بر روی درایه های متناظر در دو بردار اعمال کنیم و با مقادیر محاسبه شده پیشین جمع کنیم. تمامی توابعی که برای این منظور محاسبه شده اند با داده های 128 بیتی بصورت 4 عدد ممیز شناور ساده برخورد میکنند.

```
for (long i = 0; i < ARRAY_SIZE; i += 4)
    sum = _mm_add_ps(sum, _mm_mul_ps(
            _mm_sub_ps(_mm_loadu_ps(&vector1[i]),
            _mm_loadu_ps(&vector2[i])),
            _mm_sub_ps(_mm_loadu_ps(&vector1[i]),
            _mm_loadu_ps(&vector2[i]))));
```

در انتها نیز برای بدست آوردن مجموع نهایی لازم است که هر 4 عدد موجود در این 128 بیت با یکدیگر جمع شوند که بدین منظور می توان دو مرتبه عمل Horizontal Add را انجام داد که بدین صورت هر 4 خانه مجموع هر 4 عدد را شامل می شود و برای بدست آوردن مقدار نهایی یکی از این اعداد را استخراج میکنیم.

```
sum = _mm_hadd_ps(sum, sum);
sum = _mm_hadd_ps(sum, sum);
float result = _mm_cvtss_f32(sum);

result = sqrt(result);
```

متوسط میزان تسریعی که موازی سازی برنامه به ما ارائه میدهد برابر **2.040** محاسبه گردید.

سوال سوم)

در این سوال برای کار کردن با داده های تصویر از کتابخانه OpenCV استفاده گردیده است که امکانات زیادی برای کار کردن با داده های چندرسانه ای در اختیار ما قرار میدهد.

```
void load_image()
{
    in_img = cvLoadImage("./lena.png", CV_LOAD_IMAGE_GRAYSCALE);

    if (!in_img)
    {
        cout << "could not open or find the image" << std::endl;
        exit(EXIT_FAILURE);
    }

    img_height = in_img->height;
    img_width = in_img->width;

    in_img_char = (unsigned char *)in_img->imageData;
}
```

به منظور Smooth کردن یک تصویر، به 9 خانه نزدیک یک خانه مراجعه کردیم و متوسط مقدار موجود در این پیکسل ها را بعنوان خروجی مربوط آن پیکسل تعیین کردیم. به منظور عدم تخطی از بازه آرایه ها طول و عرض مربوط به عکس تا دو شماره کمتر پیمایش می شوند و در نهایت این دو سطر و ستون با مقدار اولیه تصویر جایگزین می شوند.

```
for (int i = 0; i < img_height - 2; ++i)
    for (int j = 0; j < img_width - 2; ++j)
    {
        int avg = 0;
        for (int u = 0; u < 3; u++)
            for (int v = 0; v < 3; v++)
                avg += in_img_char[(i + u) * img_width + (j + v)];
        smooth_img_char[i * img_width + j] = avg / 9;
    }

for (int i = img_height - 2; i < img_height; i++)
    for (int j = img_width - 2; j < img_width; j++)
        smooth_img_char[i * img_width + j] = in_img_char[i * img_width + j];
```

حال برای موازی سازی این برنامه از ایده مربوط به سوالات قبل استفاده میکنیم، با این تفاوت که با 128 بیت داده به صورت 16 بایت برخورد میکنیم. همچنین برای این که بتوانیم تمام اعمالی که در یک پیمایش انجام میدهیم را بصورت موازی انجام دهیم و از حلقه جدایی جهت تعیین مقدار استفاده نکنیم، پس از محاسبه میانگین 8 خانه کنار یک پیکسل لازم است که یک میانگین وزن دار با مقدار موجود در پیکسلی که قصد تعیین مقدار آن را داریم صورت گیرد. اگر بخواهیم میانگین وزن دار صحیح را محاسبه کنیم باید وزن مربوط به میانگین 8 خانه مجاور برابر با 8 لحاظ شود وزن مربوط به پیکسل مورد بررسی یک. ولی بدین منظور که تمام اعمال بصورت موازی صورت گیرد، 3 مرتبه میانگین گیری اعمال شده است که عملا وزن 7 به میانگین 8 خانه مجاور و وزن 1 به خود خانه

داده می شود. با توجه به قطعه کد زیر، داده های مربوط به 16 پیکسل به طور همزمان محاسبه می گردد. این عملیات بدین صورت می باشد که از 3 سطر، 3 ستون خوانده می شود و برای هر پیکسل میانگین 8 خانه مجاور آن محاسبه گردیده و در نهایت میانگین وزن دار با مقدار یکی از خانه ها گرفته می شود. تابع _mm_avg_epu8 بدین صورت عمل میکند که میانگین 16 بایت را دو به دو محاسبه کرده و در 128 بیت مقصد می نویسد.

```
for (int i = 0; i < img_height - 2; i++)
    for (int j = 0; j < img_width - 2; j += 16)
    {
        for (int u = 0; u < 3; u++)
            for (int v = 0; v < 3; v++)
                smoothed_data[u][v] = _mm_loadu_si128(
                        (__m128i*)(in_img_char + (i + u) * img_width + (j + v)));

        smoothed_data[0][0] = _mm_avg_epu8(smoothed_data[0][0], smoothed_data[0][1]);
        smoothed_data[2][0] = _mm_avg_epu8(smoothed_data[2][0], smoothed_data[2][1]);
        smoothed_data[0][0] = _mm_avg_epu8(smoothed_data[0][0], smoothed_data[2][0]);

        smoothed_data[0][2] = _mm_avg_epu8(smoothed_data[0][2], smoothed_data[2][2]);
        smoothed_data[1][0] = _mm_avg_epu8(smoothed_data[1][0], smoothed_data[1][2]);
        smoothed_data[0][2] = _mm_avg_epu8(smoothed_data[0][2], smoothed_data[1][0]);

        smoothed_data[0][0] = _mm_avg_epu8(smoothed_data[0][2], smoothed_data[0][0]);

        avg = _mm_avg_epu8(smoothed_data[0][0], smoothed_data[0][1]);
        avg = _mm_avg_epu8(smoothed_data[0][0], avg);
        avg = _mm_avg_epu8(smoothed_data[0][0], avg);

        _mm_storeu_si128((__m128i*)(smooth_img_char + i * img_width + j), avg);
    }
```

متوسط میزان تسریعی که موازی سازی برنامه به ما ارائه میدهد برابر **6.438** محاسبه گردید.

سوال چهارم)

در این سوال نیز همانند سوال قبل تصاویر مربوطه با استفاده از کتابخانه OpenCV خوانده می شود و عمل قدرمطلق تفاضل برای هر پیکسل آن محاسبه می گردد و در تصویر نهایی نوشته می شود. کد سریال مربوطه در ذیل آمده است:

```
for (int i = 0; i < height; ++i)
    for (int j = 0; j < width; ++j)
        diff_img_char[i * width + j] = abs(second_data[i * width + j] -
                first_data[i * width + j]);
```

به منظور موازی سازی این عملیات ها نیز از توابع _mm_sub_epi8 و _mm_abs_epi8 به منظور تفریق بایت به بایت داده های 128 بیتی از یکدیگر و در نهایت محاسبه قدرمطلق این مقادیر استفاده گردیده است:

```
for (int i = 0; i < height; ++i)
    for (int j = 0; j < width; j += 16)
    {
        __m128i m1 = _mm_loadu_si128((__m128i*)(first_data + i * width + j));
        __m128i m2 = _mm_loadu_si128((__m128i*)(second_data + i * width + j));
        __m128i diff = _mm_sub_epi8(m1, m2);
        __m128i abs = _mm_abs_epi8(diff);

        _mm_storeu_si128((__m128i*)(diff_img_char + i * width + j), abs);
    }
```

متوسط میزان تسریعی که موازی سازی برنامه به ما ارائه میدهد برابر **9.586** محاسبه گردید.