به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس برنامه‌نویسی موازی

گزارش تمرین چهارم

نام و نام خانوادگی	شماره دانشجویی
سجاد علیزاده	۸۱۰۱۹۷۵۴۷
سارینا همدانی	۸۱۰۱۹۷۶۰۶

به منظور اجرای هر یک از برنامه‌ها کافیست وارد پوشهٔ مورد نظر شده، دو دستور زیر را به ترتیب وارد کنید تا برنامه کامپایل شود. کتابخانه opencv4 باید از پیش روی سیستم نصب شده باشد.

```
alias opencvflags="pkg-config --cflags --libs opencv4"
g++ -std=c++11 -o main.out ` opencvflags` main.cpp
```

بخش ۱

به منظور کاربردهای چندرسانه‌ای از کتابخانهٔ open-source ای که شرکت Intel تحت عنوان OpenCV ارائه داده‌است، استفاده می‌کنیم.

در این تمرین می‌بایست تفاضل دو فریم داده شده را یک بار بدون موازی‌سازی و بار دیگر با استفاده از مجموعه دستورات SIMD پردازندهٔ اینتل محاسبه کرده و در تصویر خروجی ذخیره نماییم. قدر مطلق تفاضل دو فریم را بصورت زیر محاسبه می‌کنیم:

$$D = |A - B|$$

ساختار کلی برنامه به شکل زیر نوشته شده‌است:

```
> void print_student_info() {…
> int __diff_serial(Mat A, Mat B) {…
> int __diff_parallel(Mat A, Mat B) {…
> int main() {…
```

کتابخانه‌های موردنیاز برنامه در زیر آمده است:

```
#include <opencv4/opencv2/highgui/highgui.hpp>
#include <opencv4/opencv2/imgproc/imgproc.hpp>
#include <opencv4/opencv2/core.hpp>
#include <stdio.h>
#include <x86intrin.h>
#include <sys/time.h>
#include <stdlib.h>

using namespace std;
using namespace cv;
```

ابتدا در بدنهٔ تابع main برنامه تصویر قبل و بعد را توسط تابع imread و به صورت grayscale خوانده و در متغیرهای A و B که از نوع Mat تعریف شده‌اند، ذخیره می‌کنیم. همان طور که در قطعه کد زیر نیز می‌بینیم، در صورت خالی بودن تصویر خوانده شده پیام مناسب را در خروجی چاپ کرده و برنامه را خاتمه می‌دهیم. در غیر اینصورت دو متغیر را به تابع محاسبهٔ تفاضل پاس می‌دهیم.

```
Mat A = imread(FIRST_IMAGE, IMREAD_GRAYSCALE);
Mat B = imread(SECOND_IMAGE, IMREAD_GRAYSCALE);
if (A.empty() || B.empty()) {
    printf("Error reading images.");
    return -1;
}
```

حال به بررسی بدنه تابع محاسبهٔ تفاضل نسخهٔ سریال و موازی می‌پردازیم:

نسخهٔ سریال

ابعاد تصاویر را در دو متغیر NROWS و NCOLS ذخیره می‌کنیم. ماتریس تصویر خروجی را با این ابعاد ساخته و در ابتدا رنگ متناظر هر پیکسل را برابر سیاه قرار می‌دهیم. پوینتر به داده‌های تصویر اول و دوم را در متغیر از نوع uint8_t به نام‌های AD و BD و پوینتر به داده‌های تصویر خروجی را در DD ذخیره می‌کنیم.

```
Mat D;
struct timeval start, end;
int NROWS = A.rows;
int NCOLS = A.cols;

D.create(NROWS, NCOLS, CV_8UC1);

uint8_t* AD = (uint8_t*)A.data;
uint8_t* BD = (uint8_t*)B.data;
uint8_t* DD = (uint8_t*)D.data;
```

در یک حلقهٔ تو در تو روی سطرها و ستون‌های تصاویر ورودی پیمایش می‌کنیم. می‌دانیم در آرایه‌های دوبعدی عناصر سطر اول در یک ستون زیر یکدیگر قرار دارند و عناصر سطر بعدی در ستون کناری آن‌ها و به همین ترتیب. پس اگر بخواهیم به پوینتر به خانهٔ سطر i و ستون j ام آرایه دسترسی داشته باشیم می‌بایست حاصلضرب شماره سطر در NCOLS را با شماره ستون جمع کنیم یعنی

$$p = i * NCOLS + j$$

در ادامه به ازای هر پیکسل در دو تصویر ورودی، قدر مطلق تفاضل آن دو را به دست آورده و برابر پیکسل متناظر در تصویر خروجی قرار می‌دهیم.

```
gettimeofday(&start, NULL);
for(int i = 0; i < NROWS; i++) {
    for(int j = 0; j < NCOLS; j++) {
        int p = i * NCOLS + j;
        DD[p] = abs(AD[p] - BD[p]);
    }
}
gettimeofday(&end, NULL);
```

مدت زمان اجرای این عملیات را با استفاده از اختلاف زمان اندازه‌گیری شده در ابتدا و انتهای حلقه که توسط تابع gettimeofday به دست می‌آید، محاسبه کرده و نمایش می‌دهیم. در نهایت خروجی را در یک فایل با فرمت png و نام Q1 Serial ذخیره می‌کنیم.

```
long seconds = (end.tv_sec - start.tv_sec);
long int execution_time = ((seconds * 1000000) + end.tv_usec) - (start.tv_usec);
printf("Serial Method:\n");
printf("\tExecution time in microseconds: %ld\n\n", execution_time);
imwrite("Q1 Serial.png", D);
return execution_time;
```

نسخهٔ موازی‌سازی شده

در روش موازی‌سازی شده، داده‌های تصویر را در متغیرهای __m128i ذخیره می‌کنیم. با هر بار لود کردن ۱۲۸ بیت از حافظه خوانده می‌شود که معادل ۱۶ خانهٔ ۸ بیتی (۱ بایتی) است. به همین علت ستون‌ها در بسته‌های ۱۶ تایی خوانده شده و تعداد پیمایش ستون‌ها ۱/۱۶ تعداد ستون‌هاست.

```
__m128i* AD = (__m128i*) A.data;
__m128i* BD = (__m128i*) B.data;
__m128i* DD = (__m128i*) D.data;

__m128i a, b, sub0, sub1, abs;
```

همانند توضیحات قسمت قبل، برای بدست آوردن موقعیت هر پیکسل نسبت به آدرس شروع آرایه در اینجا بعلت استفاده از رجیسترهای ۱۲۸ بیتی به صورت ۱۶ تایی خوانده می‌شود، باید شماره سطر را در تعداد ستون‌ها تقسیم بر ۱۶ ضرب و با شماره ستون جمع نمود. یعنی

$$p = i * NCOLS/16 + j$$

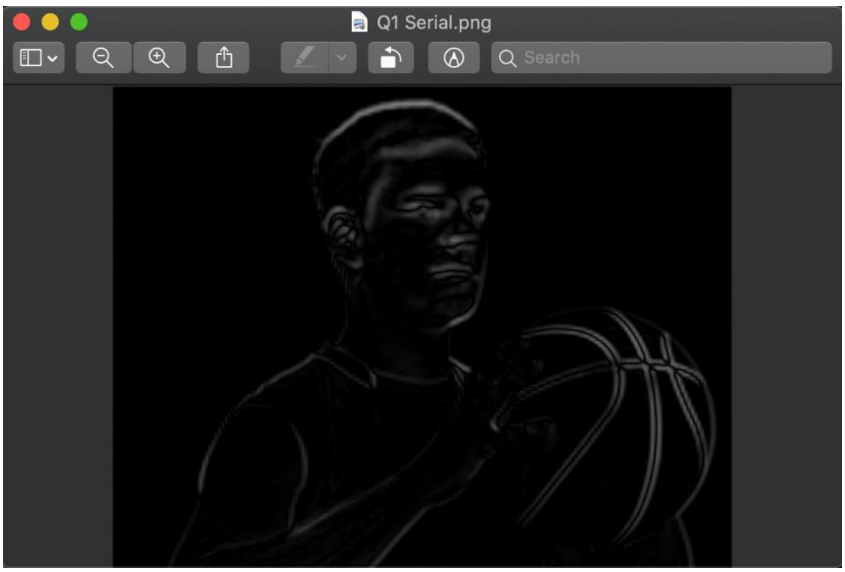
با استفاده از دستور _mm_loadu_si128 ۱۶ مقدار ۸ بیتی را که پوینتر آن به خانهٔ p+(AD|BD) اشاره دارد که AD و BD به ترتیب به آدرس شروع آرایهٔ مربوط به داده‌های تصویر اول و دوم اشاره دارد و خروجی را در متغیرهای a و b از نوع __mm128i ذخیره می‌کنیم. با استفاده از دستور _mm_subs_epu8 یکبار مقادیر a را از b و بار دوم مقادیر b را از a کم کرده و چون تفاضل از نوع saturation است با or کردن این دو مقدار حاصل تفاضل مثبت (یا همان قدر مطلق تفاضل) به دست می‌آید. در نهایت این حاصل را در پوینتر مربوط به پیکسل متناظر در تصویر خروجی به کمک دستور _mm_store_si128 ذخیره می‌کنیم.

```
gettimeofday(&start, NULL);
for(int i = 0; i < NROWS; i++) {
    for(int j = 0; j < NCOLS/16; j++) {
        int p = i * (NCOLS/16) + j;
        a = _mm_loadu_si128((__m128i*)(AD + p));
        b = _mm_loadu_si128((__m128i*)(BD + p));
        sub0 = _mm_subs_epu8(a, b);
        sub1 = _mm_subs_epu8(b, a);
        abs = _mm_or_si128(sub0, sub1);
        _mm_store_si128((__m128i*)(DD + p), abs);
    }
}
gettimeofday(&end, NULL);
```

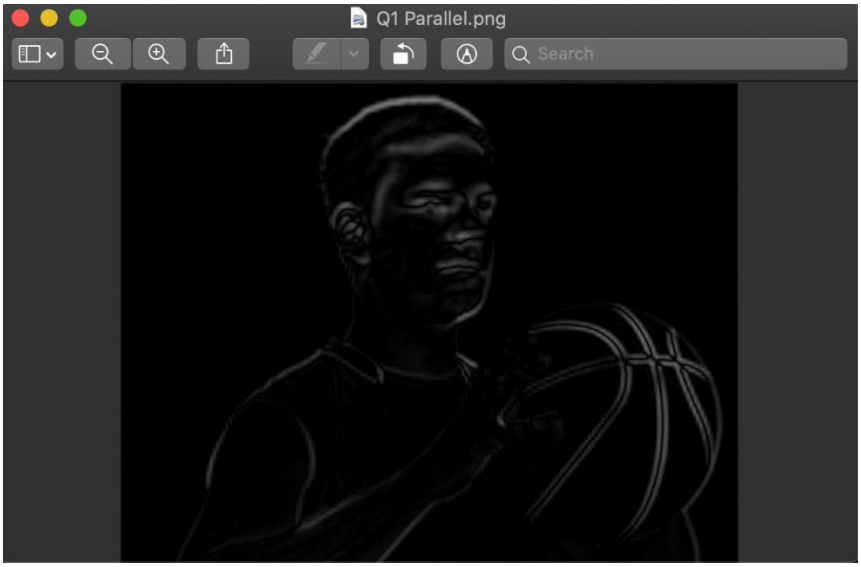
تصویر نهایی با استفاده از تابع imwrite در فایل Q1 Parallel.png ذخیره می‌شود.

```
long seconds = (end.tv_sec - start.tv_sec);
long int execution_time = ((seconds * 1000000) + end.tv_usec) - (start.tv_usec);
printf("Parallel Method:\n");
printf("\tExecution time in microseconds: %ld\n\n", execution_time);
imwrite("Q1 Parallel.png", D);
return execution_time;
```

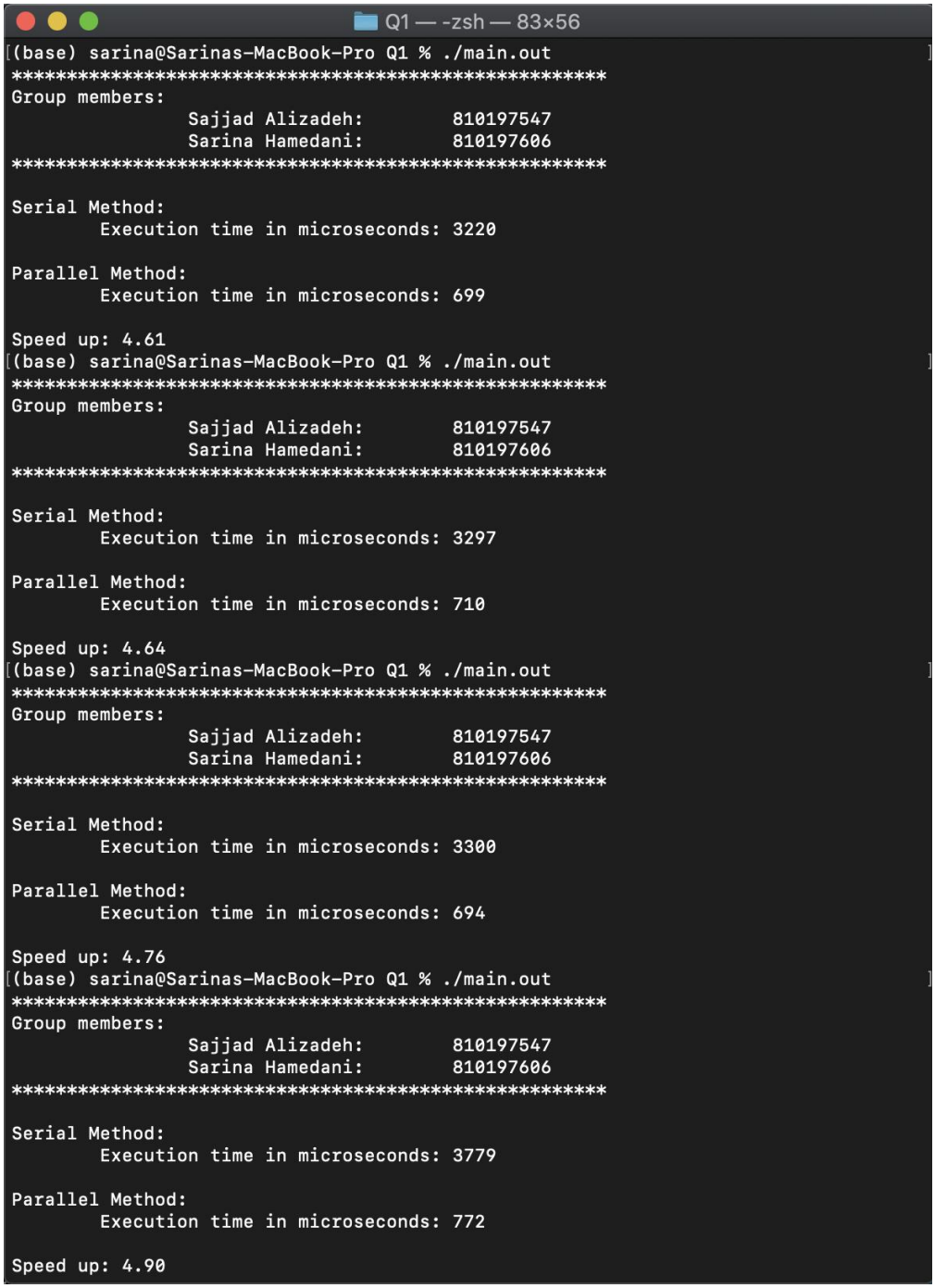
میزان تسریع در مقایسه با نسخهٔ سریال به طور میانگین برابر ۴,۷۲ است و خروجی همان طور که در تصاویر زیر قابل مشاهده است، یکسان است.



۱ - خروجی برنامه سریال



۲ - خروجی برنامه پارالل



۳ - اجرای برنامه اول

بخش ۲

در این بخش می‌خواهیم برنامه‌ای بنویسیم که تصویر ۲ را با درجهٔ شفافیت ۰٫۵ به تصویر ۱ اضافه کند. برای اضافه کردن یک تصویر با درجهٔ شفافیت α به یک تصویر دیگر از رابطهٔ زیر استفاده می‌کنیم:

$$Result = Img1 + Img2 \times \alpha$$

ساختار کلی برنامه به شکل زیر نوشته شده‌است:

```
> void print_student_info() {…

> int __attach_serial(Mat Img1, Mat Img2, Mat Result) {…

> int __attach_parallel(Mat Img1, Mat Img2, Mat Result) {…

> int main() {…
```

در این بخش نیز ابتدا دو تصویر Img_01 و Img_02 را با تابع imread خوانده و در دو متغیر از نوع Mat با نام‌های Img1 و Img2 ذخیره می‌کنیم. همچنین تصویر خروجی را نیز برابر تصویر اول قرار می‌دهیم. این سه متغیر را به تابع مربوط به اضافه کردن دو تصویر به یکدیگر پاس می‌دهیم. ابعاد تصویر دوم را در متغیر NROWS_2 و NCOLS_2 ذخیره می‌کنیم.

```
struct timeval start, end;

int NCOLS = Img1.cols;
int NROWS_2 = Img2.rows;
int NCOLS_2 = Img2.cols;
```

نسخهٔ سریال

در روش سریال، به شرح زیر تصویر دوم را با درجهٔ شفافیت ۰٫۵ به تصویر اول اضافه می‌کنیم.

پوینتر به داده‌های تصویر اول و دوم را در متغیر از نوع uint8_t به نام‌های D1 و D2 ذخیره می‌کنیم.

```
uint8_t* D1 = (uint8_t*)Img1.data;
uint8_t* D2 = (uint8_t*)Img2.data;
uint8_t* R = (uint8_t*)Result.data;
```

اکنون برای به دست آوردن تصویر خروجی باید تمامی پیکسل‌های تصویر دوم را مطابق فرمول داده‌شده به تصویر اول اضافه کنیم. بدین منظور روی سطر و ستون‌های تصویر دوم پیمایش می‌کنیم. مطابق توضیحات بخش قبل پوینتر به هر پیکسل برابرست با مجموع حاصلضرب شماره سطر در NCOLS و شماره ستون. یعنی اگر شماره سطر و ستون به ترتیب برابر i و j باشند، پوینتر موردنظر در تصویر اول با تعداد ستون‌های NCOLS برابرست با $p = i * NCOLS + j$ و این پوینتر در تصویر دوم برابر

است $p = i * NCOLS\_2 + j$.

اکنون دو پیکسل متناظر از تصویر دوم را طبق فرمول در ۰٫۵ ضرب می‌کنیم که معادل آنست که آن را یک واحد به راست شیفت دهیم. سپس پیکسل شفاف شده را با پیکسل متناظر آن در تصویر اول جمع می‌کنیم و در تصویر نهایی ذخیره می‌کنیم.

```
gettimeofday(&start, NULL);
for(int i = 0; i < NROWS_2; i++) {
    for(int j = 0; j < NCOLS_2; j++) {
        int p1 = i * NCOLS + j;
        int p2 = i * NCOLS_2 + j;
        R[p1] = (D1[p1] + D2[p2]) >> 1;
    }
}
gettimeofday(&end, NULL);
```

همانند قسمت قبل، مدت زمان اجرای این عملیات را با استفاده از اختلاف زمان اندازه‌گیری شده در ابتدا و انتهای حلقه محاسبه کرده و نمایش می‌دهیم. تصویر خروجی را نیز در فایل Q2 Serial با فرمت png. ذخیره می‌شود.

```
long seconds = (end.tv_sec - start.tv_sec);
long int execution_time = ((seconds * 1000000) + end.tv_usec) - (start.tv_usec);
printf("Serial Method:\n");
printf("\tExecution time in microseconds: %ld\n\n", execution_time);
imwrite("Q2 Serial.png", Result);
return execution_time;
```

نسخهٔ موازی‌سازی شده

در روش پارالل داده‌های تصاویر ورودی و خروجی در پوینترهایی از نوع __m128i ذخیره می‌کنیم.

```
__m128i* D1 = (__m128i*) Img1.data;
__m128i* D2 = (__m128i*) Img2.data;
__m128i* R = (__m128i*) Result.data;
```

همانطور که در بخش قبلی نیز توضیح داده شد، با هر بار لود کردن ۱۲۸ بیت از حافظه خوانده می‌شود که معادل ۱۶ خانهٔ ۸ بیتی (۱ بایتی) است. به همین علت ستون‌ها در بسته‌های ۱۶ تایی خوانده شده و تعداد پیمایش ستون‌ها ۱/۱۶ تعداد ستون‌هاست. برای بدست آوردن موقعیت هر پیکسل نسبت

به آدرس شروع آرایه در اینجا بعلت استفاده از رجیسترهای ۱۲۸ بیتی به صورت ۱۶ تایی خوانده می‌شود، باید شماره سطر را در تعداد ستون‌ها تقسیم بر ۱۶ ضرب و با شماره ستون جمع نمود. یعنی

$$p = i * NCOLS/16 + j$$

با استفاده از دستور _mm_loadu_si128 ۱۶ مقدار ۸ بیتی را که پوینتر آن در تصویر اول به خانهٔ D1+p1 و در تصویر دوم به D2+p2 اشاره دارد، در دو متغیر im1 و im2 ذخیره می‌کنیم. مقدار im2 را یک واحد به راست شیفت می‌دهیم تا درجهٔ شفافیت آن ۰,۵ شود. پس از این شیفت ۱ بیت به سمت چپ وارد می‌شود که باید آن را خنثی کنیم و بدین منظور باید هر بایت m2 را با 01111111 and کنیم. حاصل را با m1 با دستور _mm_adds_epu8 جمع می‌کنیم و در پیکسل متناظر تصویر خروجی ذخیره می‌نماییم.

```
__m128i N = _mm_set1_epi8(127);
__m128i im1, im2, res;
gettimeofday(&start, NULL);
for (int i = 0; i < NROWS_2; i++) {
    for (int j = 0; j < NCOLS_2 / 16; j++) {
        int p1 = i * (NCOLS / 16) + j;
        int p2 = i * (NCOLS_2 / 16) + j;

        im1 = _mm_loadu_si128(D1 + p1);
        im2 = _mm_loadu_si128(D2 + p2);
        im2 = im2 >> 1;
        im2 = _mm_and_si128(N, im2);
        res = _mm_adds_epu8(im1, im2);
        _mm_store_si128(R + p1, res);
    }
}
gettimeofday(&end, NULL);
```

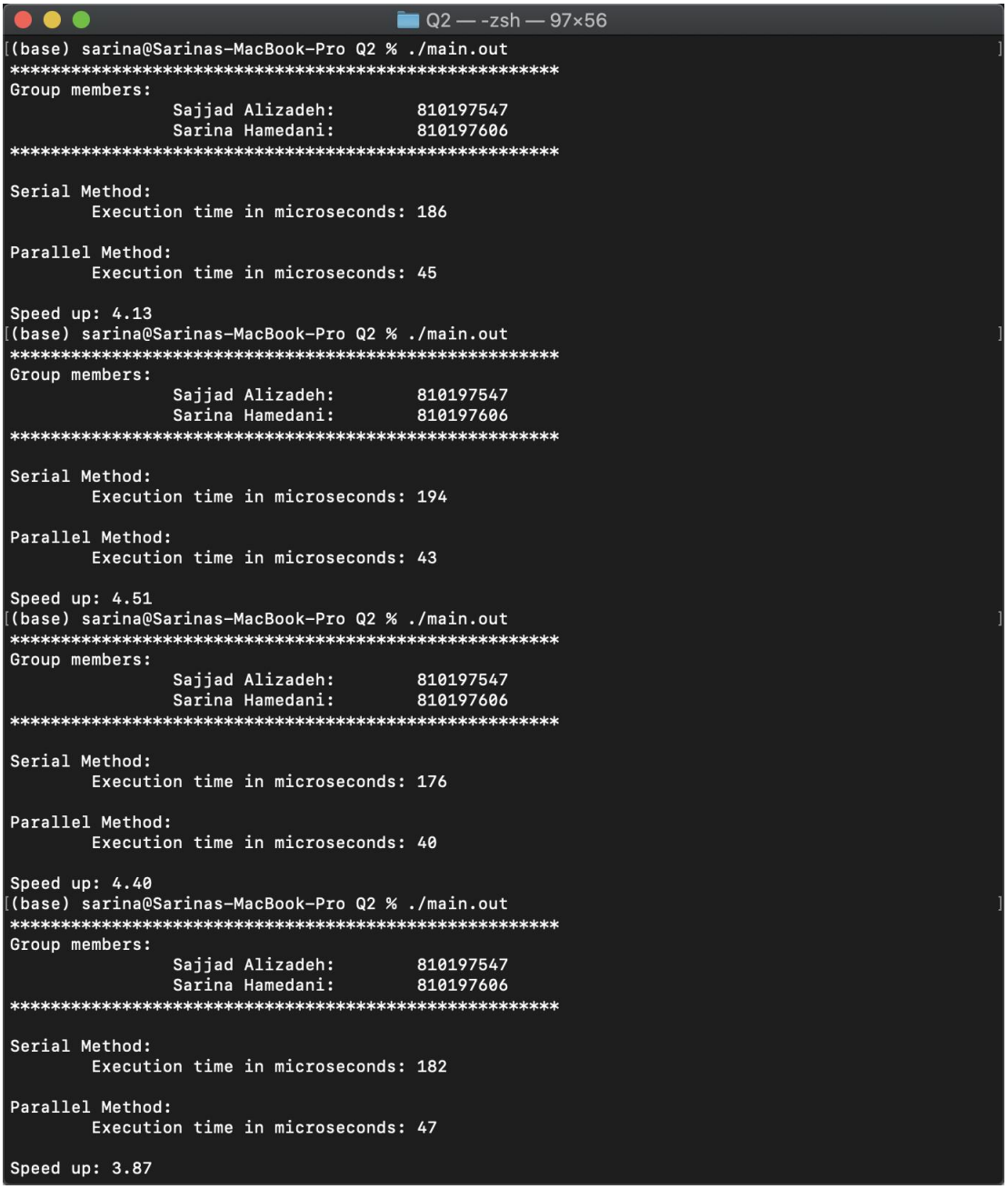
مدت زمان اجرای این عملیات را با استفاده از اختلاف زمان اندازه‌گیری شده در ابتدا و انتهای حلقه محاسبه کرده و نمایش می‌دهیم. تصویر خروجی را نیز در فایل Q2 Parallel با فرمت png. ذخیره می‌شود.

```
long seconds = (end.tv_sec - start.tv_sec);
long int execution_time = ((seconds * 1000000) + end.tv_usec) - (start.tv_usec);
printf("Parallel Method:\n");
printf("\tExecution time in microseconds: %ld\n\n", execution_time);
imwrite("Q2 Parallel.png", Result);
return execution_time;
```

میزان تسریع برنامه موازی‌سازی شده در مقایسه با نسخهٔ سریال به طور میانگین برابر ۴,۲۲ است.

۴ - خروجی برنامه پارالل

۵ - خروجی برنامه سریال



۶ - اجرای برنامه دوم