

Parallel Programming

Assignment III

Omid Ahmadi
Mohammad Mahdi Islami

Part 1



در این سوال هدف محاسبه‌ی تفاصل دو عکس جهت تشخیص حرکت در تصویر و نمایش خروجی است. برای این محاسبات از کتابخانه OpenCV بر روی C++ استفاده کردیم تا بتوانیم محتوای تصاویر را بخوانیم، سطرهای سطون های آن را جدا کنیم و نهایتاً خروجی را بر روی یک فایل بنویسیم. برای مثال از قطعه کد زیر برای خواندن عکس ها و جداسازی گرفتن سطر و ستون های آن استفاده میکنیم:

```
firstImg = imread("CA03_Q1_Image_01.png", IMREAD_GRAYSCALE);
secondImg = imread("CA03_Q1_Image_02.png", IMREAD_GRAYSCALE);
rowN = firstImg.rows;
colN = firstImg.cols;
```

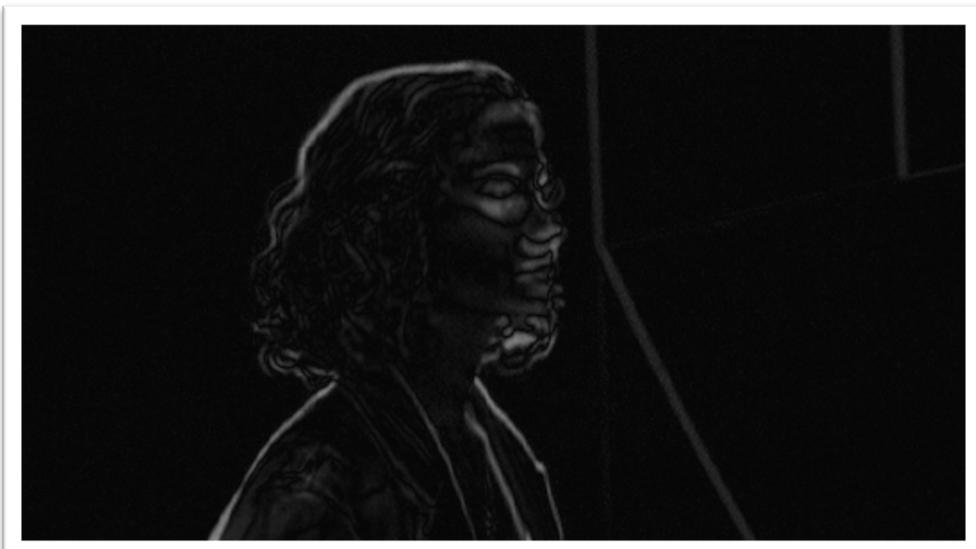
Serial

برای به دست آوردن قدر مطلق تفاضل دو تصویر به صورت سریال از کد زیر استفاده می‌کنیم:

```
/*-----** Serial -----*/
start = ippGetCpuClocks();
for(int i = 0; i < rowN; i++) {

    for(int j = 0; j < colN; j++) {
        index = i * colN + j;
        resultImgPixel[index] = abs(secondImgPixel[index] - firstImgPixel[index]);
    }
}
end = ippGetCpuClocks();
serialTime = (Ipp32u)(end - start);
cout << "Serial Runtime: " << serialTime << endl;
imwrite("serialResult.png", result);
```

با استفاده از دستورات OpenCV محتوای عکس‌ها را می‌خوانیم و درون متغیرهای مربوطه ذخیره می‌کنیم. سپس سطراها و ستون‌های هر تصویر را با استفاده از دستورات موجود در OpenCV به ترتیب درون متغیرهای `rowN` و `colN` ذخیره می‌کنیم و با حرکت روی پیکسل‌ها، متناظراً آن‌ها را از یکدیگر کم می‌کنیم و با دستور `abs` قدر مطلق تفاضل حال را محاسبه می‌کنیم. سپس خروجی را درون فایل `serialResult.png` که در پوشه اصلی پروژه قرار داده شده است ذخیره می‌کنیم.



خروجی حاصل از
محاسبات سریال

Parallel

برای به دست آوردن قدر مطلق تفاضل دو تصویر به صورت موازی از کد زیر استفاده می‌کنیم:

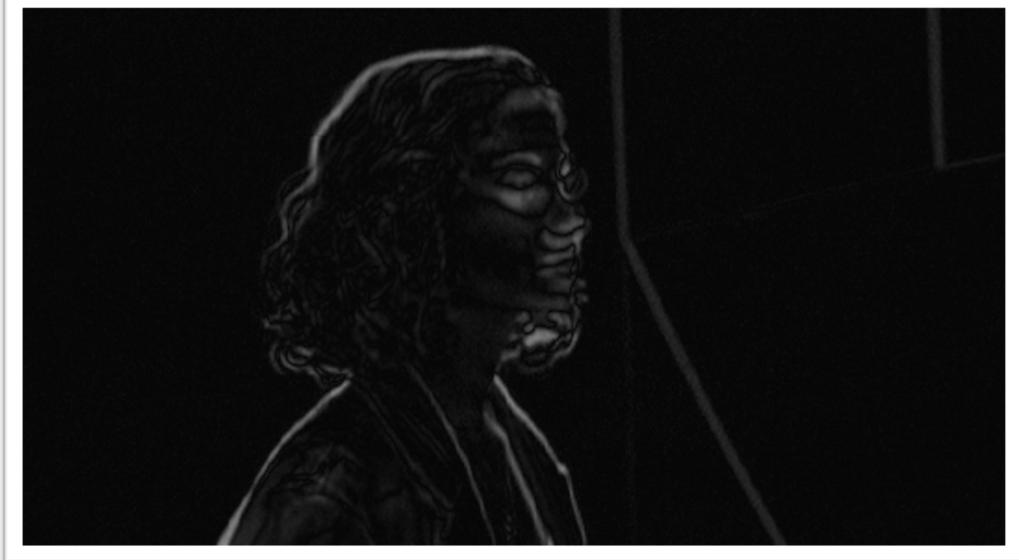
```
/*-----** Parallel -----*/
start = ippGetCpuClocks();
for(int i = 0; i < rowN; i++) {

    for(int j = 0; j < colN/16; j++) {

        index = (i * colN/16 + j);
        firstImgVec = _mm_loadu_si128(&firstImgParallelPixel[index]);
        secondImgVec = _mm_loadu_si128(&secondImgParallelPixel[index]);
        sub_0 = _mm_sub8_epi8(firstImgVec, secondImgVec);
        sub_1 = _mm_sub8_epi8(secondImgVec, firstImgVec);
        sub_0 = _mm_or_si128(sub_0, sub_1);
        _mm_store_si128(&resultImgParallelPixel[index], sub_0);
    }
}
end = ippGetCpuClocks();
parallelTime = (Ipp32u)(end - start);
imwrite("parallelResult.png", result);
```

برای به دست آوردن قدر مطلق تفاضل در روش موازی که دستوری مانند `abs` در اختیار نداریم:

- برای پیمودن محتوای عکس ها ستون هارا با توجه به نوع پیکسل ها برای لود آن ها درون رجیستر مناسب، ستون ها را به بخش های ۱۶ تایی تقسیم می‌کنیم.
- یک بار محتوای پیکسلی تصویر اول را از محتوای پیکسلی تصویر دوم کم می‌کنیم.
- و بار بعدی محتوای پیکسلی تصویر دوم را از محتوای پیکسلی تصویر اول کم می‌کنیم.
- در نهایت حاصل این دو عملیات را با یکدیگر `or` منطقی می‌کنیم.
- بنابراین چون عملیات را به صورت **Saturation** انجام می‌دهیم، نتیجه اگر منفی باشد صفر می‌شود و نهایتاً با `or` خروجی مثبت به دست می‌آید.



خروجی حاصل از
محاسبات موازی

Result

نتیجه‌ی محاسبات سریال و موازی و مقایسه آن‌ها به صورت زیر است:

```
SIDs : 810195548 , 810195346
-----
Serial Runtime: 1081862
Parallel Runtime: 88432
Speed-up : 91.8259
Program ended with exit code: 0
```

تصاویر این بخش از آدرس‌های زیر در پوشه‌های اصلی قابل مشاهده هستند:

[serialResult.png](#)
[parallelResult.png](#)

Part 2

در ابتدا عکس ها را `load` می کنیم. سپس دیتا آن ها را که شامل تعداد پیکسل های طولی و عرضی است را جدا می کنیم.

Serial

در دو حلقه با استفاده از فرمول ذکر شده ($\text{Result} = \text{Img1} + \text{Img2} * a$) حاصل به دست آمده را در عکس اول نگهداری می کنیم. سپس به دلیل سرریز نیاز به **Saturation** داریم. در نتیجه چک می کنیم اگه نتیجه از اپرند کوچکتر بود بجای آن 255 میداریم.

در نهایت عکس را نمایش می دهیم:

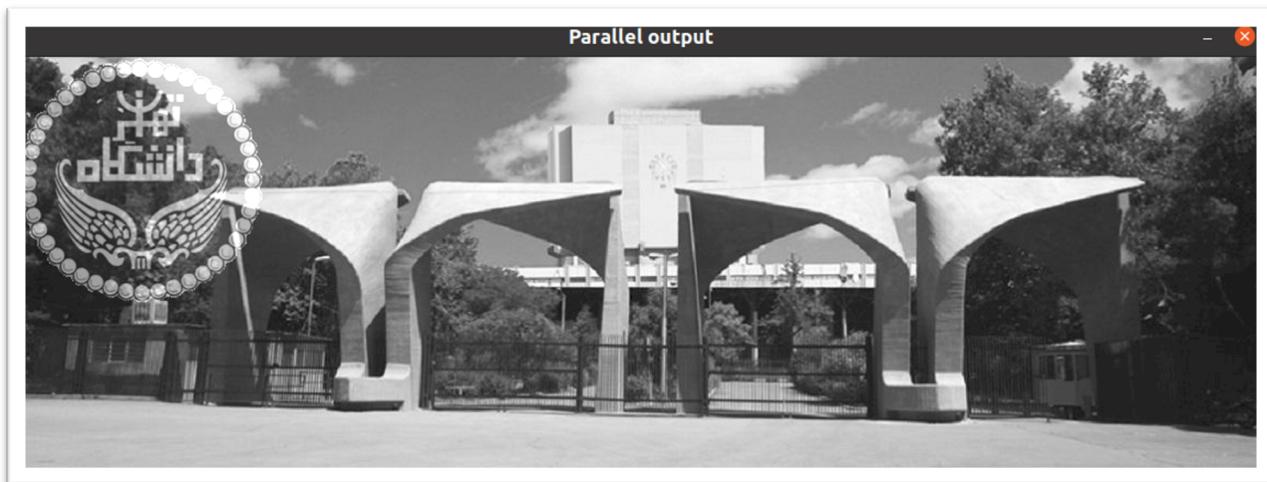


Parallel

در دو حلقه تو در تو عکس ها را لود میکنم و درون دو متغیر `m1` و `m2` میریزم. از انجا که وجود ندارد مجبور به انجام آن در دو مرحله هستیم.

```
m2 = _mm_srl_i_epi8(m2, 1);
m2 = _mm_and_si128(m2, _mm_set1_epi8(0x7F));
```

در ابتدا با عملگر 16 بیتی شیفت می دهیم و سپس 8 بیت اول را با `And` نادیده می گیریم. در نهایت با دستور `adds` دو مقدار را به هم اضافه کرده و `Saturation` را با استفاده از همین تابع انجام می دهیم و عکس را ذخیره می کنیم.



در نهایت عکس را نمایش می دهیم:

حال به مقایسه مدت زمان اجرا و نسبت آن در این دو حالت می پردازیم:

```
omid@omid-VirtualBox:~/Desktop/CA03$ Make
make: Nothing to be done for 'all'.
omid@omid-VirtualBox:~/Desktop/CA03$ ./main
Serial Run time is 0 seconds and 501 micro seconds

Parallel Run time is 0 seconds and 122 micro seconds

Speedup = 4.106557

omid@omid-VirtualBox:~/Desktop/CA03$
```