برنامه‌سازی موازی

گزارش پروژه‌ی سوم

سید پارسا حسینی‌نژاد 810196604

کیمیا خبیری 810196606

سوال اول

در بخش اول قصد داشتیم وجود شی متحرک در دو تصویر که فریم‌های متوالی از یک ویدیو بودند را به دو روش سریال و موازی تشخیص دهیم.

راه حل پیدا شی متحرک فریم‌ها، آن است که اندازه‌ی اختلاف رنگ پیکسل‌های نظیر به نظیر در هر دو فریم را محاسبه کنیم. در تصویری جدیدی که پیکسل‌هایش ازین روش محاسبه شده باشد قسمت‌های ثابت مشکی، و قسمت‌های متحرک بسته به میزان تحرک طیف مختلفی از رنگ‌های خاکستری خواهند بود.

ما نیز در این پروژه این محاسبات را به دو طریق سریال و موازی انجام داده‌ایم.

قطعه کدی که در زیر آمده مربوط به محاسبات بخش سریال است. به این منظور روی پیکسل‌های تصویر حرکت کرده‌ایم و اختلاف مقدار پیکسل‌ها در دو فریم را در تصویر خروجی جدید ذخیره کرده‌ایم.

```
for (int row = 0; row < NROWS; row++)
    for (int col = 0; col < NCOLS; col++) {
        *(serialOutImagePtr + row * NCOLS + col) = abs(*(image1Ptr + row * NCOLS + col) - *(image2Ptr + row * NCOLS + col));
    }
```

قطعه کد زیر مربوط به محاسبات موازی است. به این منظور روی مختصات تصویر حرکت کردیم و به ازای پیکسلی از دو فریم که در آن مختصات باشد به این ترتیب عمل کرده‌ایم:

ابتدا مقدار هر دو پیکسل را با استفاده از آدرسی که به آن‌ها اشاره می کند پیدا کرده‌ایم. سپس لازم است تا |a-b| محاسبه شود. از آنجایی که تابع مستقیمی برای محاسبه‌ی مقدار قدرمطلق در SSE3 موجود نیست، این عملیات را با استفاده از توابع کمکی دیگر شبیه‌سازی کردیم. به این صورت که دو عملیات تفریق به‌صورت saturation انجام داده‌ایم. یکی a - b و دیگری b - a . از آن‌جایی که تفریق به صورت saturation انجام شده، بنابراین همواره یکی از این دو مقدار صفر خواهد بود (همواره یا a بزرگتر مساوی b است یا b بزرگتر مساوی a. در هر حالت یکی از جواب ها کوچکتر مساوی صفر میشود که saturation آن را با عدد صفر گرد میکند.). پس کافی‌ست که عملیات or انجام دهیم و آن مقداری که صفر نیست و مثبت است را به عنوان جواب نهایی انتخاب کنیم. در نهایت نیز این مقدار در جایگاه مناسبش در تصویر خروجی ذخیره شده است.

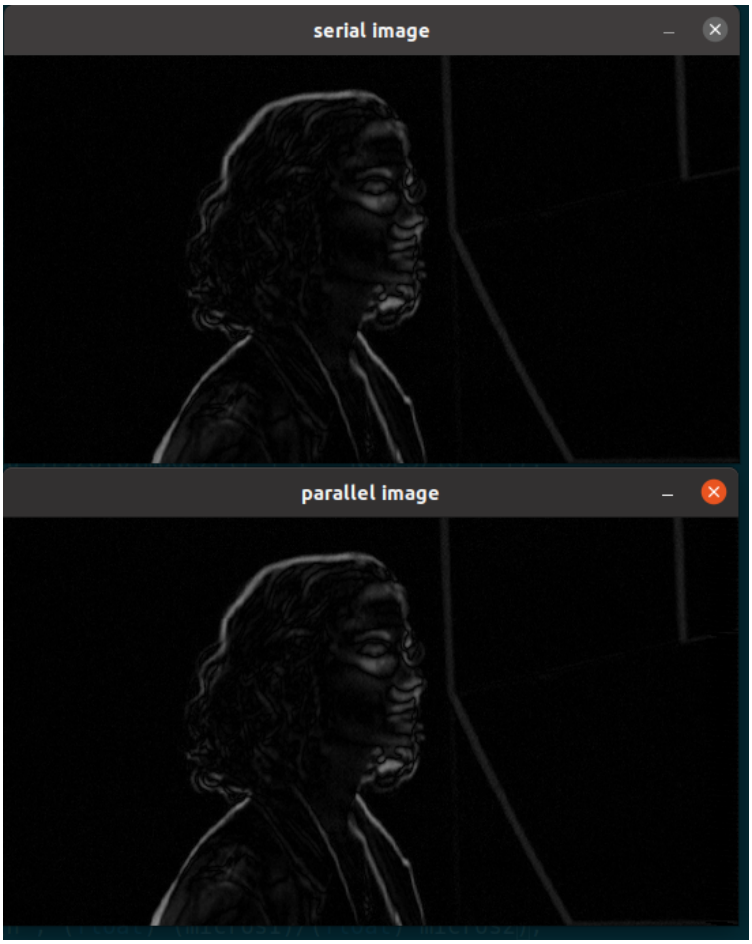
```
for (int i = 0; i < NROWS; i++)
    for (int j = 0; j < NCOLS / 16; j++)
    {
        m1 = _mm_loadu_si128(pImage1Ptr + i * NCOLS/16 + j);
        m2 = _mm_loadu_si128(pImage2Ptr + i * NCOLS/16 + j);
        m3 = _mm_subs_epu8(m1, m2);
        m4 = _mm_subs_epu8(m2, m1);
        m5 = _mm_or_si128(m3, m4);
        _mm_storeu_si128(parallelOutImagePtr + i * NCOLS/16 + j, m5);
    }
```

حال در خروجی مشخص است که speedup چیزی در حدود 4 است. برای محاسبه‌ی زمان اجرا هم نیز از gettimeofday

مربوط به کتابخانه‌ی `sys/time.h` استفاده می‌کنیم.

```
~/Desktop/Parallel Programming/CA3/Q1  g++ -o main main.cpp `pkg-config opencv4 --cflags --libs`
~/Desktop/Parallel Programming/CA3/Q1  ./main
Kimia Khabiri: 810196606 - Parsa Hoseininejad: 810196604
Serial Run time = 634
Parallel Run time = 168
Speedup = 3.773809
```

این تصاویری‌ست که پس از انجام این محاسبات ساخته شده است. همانطور که میبینید خروجی محاسبات به‌صورت سریال با محاسبات به‌صورت موازی یکسان است.



سوال دوم

ابتدا هر دو عکس را لود می‌کنیم و عکس خروجی را نیز می‌سازیم. سپس برای قسمت سریال، ابتدا تصویر بزرگتر را بر تصویر کوچکتر کپی میکنیم. سپس، روی تصویر کوچکتر حرکت کرده و عدد جدید را برای تصویر خروجی میذاریم که برابر عدد عکس اول به علاوه‌ی نصف عدد عکس است. دقت شود که اگر این عدد از 255 بیشتر شود، به این معناست که از ماکزیمم رد شده و باید آن را به ماکزیموم عدد که همان 255 است برسانیم. دقت شود ابتدا عکس بزرگتر را روی تصویر خروجی انداختیم تا دیگر درگیر چک کردن خارج شدن از محدوده‌ی حافظه‌ی تصویر اول نشویم.

```
for (int row = 0; row < NROWS; row++)
    for (int col = 0; col < NCOLS; col++)
        *(serialOutImagePtr + row * NCOLS + col) = *(image1Ptr + row * NCOLS + col);

for (int row = 0; row < NROWS2; row++)
    for (int col = 0; col < NCOLS2; col++) {
        temp = *(image1Ptr + row * NCOLS + col) + *(image2Ptr + row * NCOLS2 + col) / 2;
        if (temp > 255)
            temp = 255;
        *(serialOutImagePtr + row * NCOLS + col) = temp;
    }
```

برای قسمت موازی نیز ابتدا پیکسل های عکس بزرگتر را 16 تا 16 تا خوانده و روی عکس خروجی می‌اندازیم. سپس، روی عکس دوم حرکت می‌کنیم و پیکسل‌های مربوط به هر دو عکس را لود می‌کنیم. سپس، برای تقسیم کردن مقدار پیکسل‌های عکس دوم بر دو، از شیفت استفاده می‌کنیم. یعنی ابتدا عدد لود شده را 16 بیت 16 بیت شیفت می‌دهیم (چون sse3 از شیفت 8 بیتی ساپورت نمی کند)، سپس هر 8 بیت آن را با عدد 01111111 اند می‌کنیم تا اگر به MSB عدد یک وارد شده باشد، صفر شود. سپس عدد مربوط به عکس اول و نصف عکس دوم را 16 تا 16 تا جمع می‌کنیم و در عکس خروجی می‌گذاریم. از جمع با saturation نیز استفاده می‌کنیم تا اگر جمع از 255 بیشتر شد، همان 255 قرار گیرد.

```
for (int i = 0; i < NROWS; i++)
    for (int j = 0; j < NCOLS / 16; j++)
    {
        m1 = _mm_loadu_si128(pImage1Ptr + i * NCOLS/16 + j);
        _mm_storeu_si128(parallelOutImagePtr + i * NCOLS/16 + j, m1);
    }

for (int i = 0; i < NROWS2; i++)
    for (int j = 0; j < NCOLS2 / 16; j++)
    {
        m1 = _mm_loadu_si128(pImage1Ptr + i * NCOLS/16 + j);
        m2 = _mm_loadu_si128(pImage2Ptr + i * NCOLS2/16 + j);

        m4 = _mm_srli_epi16(m2, 1);
        m5 = _mm_and_si128(m2, m3);
        m6 = _mm_adds_epu8(m1, m5);

        _mm_storeu_si128(parallelOutImagePtr + i * NCOLS/16 + j, m6);
    }
```

حال در خروجی مشخص است که speedup چیزی در حدود 5 است. برای محاسبه‌ی زمان اجرا هم نیز از gettimeofday

مربوط به کتابخانه‌ی `sys/time.h` استفاده می‌کنیم. اسپید آپ نیز میزان نسبت زمان اجرای این دو است.

```
~/Desktop/Parallel Programming/CA3/Q2  g++ -o main main.cpp `pkg-config opencv4 --cflags --libs`
~/Desktop/Parallel Programming/CA3/Q2  ./main
Kimia Khabiri: 810196606 - Parsa Hoseininejad: 810196604
Serial Run time = 1389
Parallel Run time = 288
Speedup = 4.822917
```

همانطور که مشاهده می‌شود، تصویر خروجی هر دو نیز یکی است.