سوال ۱)

در این تمرین، تفاضل دو فریم متوالی برای تشخیص حرکت را محاسبه نمودیم. ابتدا با تابع imread از مجموعه دستورات پردازش تصویر کتابخانه opencv دو تصویر را خوانده و به ترتیب در دو متغیر از جنس ساختمان داده Mat (از جنس struct) با نام های img_۱ و img_۲ ذخیره می نماییم.

حال می خواهیم به ازای تک تک پیکسل های متناظر در دو تصویر، قدر مطلق تفاضلشان را یافته و در ماتریس مربوط به تصویر خروجی ذخیره کنیم. پیاده سازی توضیح داده شده را به دو صورت سریال و موازی (با دستورات SIMD) بررسی کرده و زمان اجرا، خروجی هر یک و در نهایت نسبت زمان اجرای برنامه سریال به برنامه موازی را مشاهده می کنیم.

سریال :

Mat خروجی با نام out_img_serial و با همان ابعاد تصویر اول و دوم را می سازیم.

دو اشاره گر به ماتریس های دو تصویر ورودی (با نام های in_image_۱ و in_image_۲) و یک اشاره گر برای ماتریس تصویر خروجی با نام p و از جنس *unsigned char در نظر میگیریم.

در ادامه، روی تک تک پیکسل ها (هر پیکسل از جنس unsigned char و در محدوده ۰ تا ۲۵۵) ی متناظر دو تصویر حرکت کرده و abs تفاضلشان را در خانه متناظر در ماتریس خروجی می نویسیم.

** برای مثال برای مقدار دهی اندیس سوم آرایه پیکسل های تصویر خروجی داریم:

(p + ۲) = abs((in_image_۱ + ۲) - *(in_image_۲ +۲));

موازی :

در برنامه موازی می توانیم ۱۶ pack ۸ بیتی (سایز هر پیکسل) را با هم خوانده و در یک متغیر از جنس m۱۲۸i_ ذخیره نماییم. برای پردازش ماتریس داده ها به صورت پک های ۱۲۸ بیتی، اشاره گر به آن را نیز از جنس *m۱۲۸i_ در نظر گرفته و بدین ترتیب مانند بخش سریال، دو اشاره گر از جنس *m۱۲۸i_ به ماتریس داده های تصویر اول و دوم و به ترتیب با نام های in_image_۱_p و in_image_۲_p تعریف می کنیم. برای خروجی نیز هم چنین یک اشاره گر *m۱۲۸i_ با نام p۲ تعریف می کنیم. تعداد ستون ها در اینجا، به تعداد ستون های بخش سریال (img_۱.rows و img_۲.rows) تقسیم بر ۱۶ است. چرا که در حالت موازی، اشاره گر از نوع *m۱۲۸i_ بوده و با هر دسترسی، ۱۶ بایت یا پیکسل را باهم لود کرده و در یک متغیر از جنس m۱۲۸i_ ذخیره می نماییم (برای لود کردن از دستور mm_loadu_si۱۲۸_ استفاده می کنیم)

حال مانند قسمت قبل، روی تک تک ۱۲۸ بیتی ها ی متناظر دو تصویر حرکت کرده و قدر مطلق تفاضل حساب می کنیم (بایت بایت تفاضل قدرمطلق حساب می شود).

برای محاسبه قدرمطلق تفاضل پک ها ، تابعی تعریف کردیم که دو متغیر m۱۲۸i_که قصد داریم به صورت پک های بایتی عمل قدرمطلق تفاضل را روی بایت های آن انجام دهیم را دریافت کرده و مراحل زیر را طی می کنیم:

۱) ابتدا ۱۲۸ بیتی اول (a) منهای ۱۲۸ بیتی دوم (b) را به صورت saturation و unsigned محاسبه می نماییم (اگر در یکی از بایت ها، a از b کوچک تر باشد، تفاضل منفی شده و از آنجا که عمل تفاضل به صورت saturation است، مقدار برابر با ۰ می شود. اگر هم a ازb بزرگتر باشد که مقدار برابر با همان a-b می شود)

** عمل تفاضل unsigned و saturation روی پک های بایتی : mm_subs_epu۸_

۲) ۱۲۸ بیتی دوم (b) منهای ۱۲۸ بیتی اول (a) را به صورت saturation و unsigned محاسبه می نماییم (توضیحات تفاضل مانند مرحله قبل)

۳) در این قسمت، نتایج تفاضل های دو مرحله قبل را or می کنیم. در هر دو بایت متناظر، یکی ۰ و دیگری یک مقدار مثبت است (زیرا a-b و b-a را به ترتیب حساب کرده بودیم) که با توجه به عملگر or مقدار مثبت در خروجی قرار می گیرد. جواب نهایی را به عنوان خروجی تابع بر میگردانیم (با دستور ۱۲۸si_or_mm_)

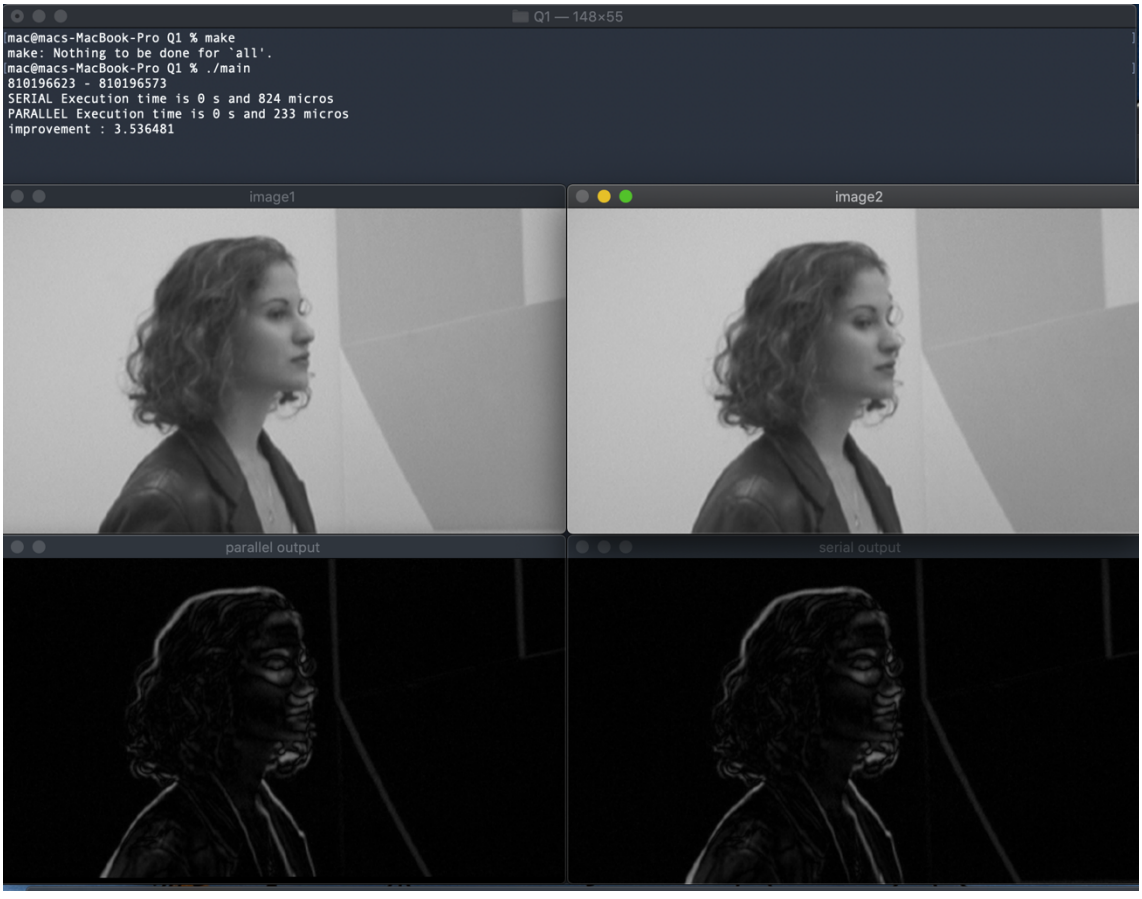
بعد از یافتن قدرمطلق تفاضل هر دو ۱۲۸ بیتی متناظر (به صورت پک های بایتی)، خروجی ۱۲۸ بیتی متناظر را در محل متناظر در حافظه (ماتریس داده ها در Mat مربوط به خروجی با نام out_img_parallel) می نویسیم (با دستور ۱۲۸si_storeu_mm_)

مقایسه دو پیاده سازی:

برنامه را ۱۰ بار اجرا کرده و زمان اجرای قسمت سریال و موازی را مشاهده کردیم (با استفاده از gettimeofday)

همچنین سرعت برنامه موازی به طور میانگین ۳.۵ برابر پیاده سازی سریال است.

هم چنین برای مشاهده هر یک از تصاویر، یک namedWindow برای آن با قابلیت تغییر سایز اتومات ساختیم و با دستور imshow کتابخانه opencv ، Mat خروجی را در آن نمایش دادیم.



سوال ۲)

در این سوال دو تصویر داده شده است.هدف این است که یک تصویر را با درجه شفافیت α (در این جا ۰.۵ در نظر گرفته شده است)به یک تصویر دیگر اضافه کنیم.این کار را به دو روش سری و موازی انجام داده و زمان اجرای هر کدام را به دست می آوریم.در انتها نیز میزان تسریع برنامه محاسبه می شود.

سریال:

در ابتدا به کمک imread دو تصویر را خوانده و در img_۱ و در img_۲ ذخیره می کنیم.سپس به کمک rows. و cols. تعداد سطر و ستون های هر عکس را به دست می آوریم.دو متغیر rowdiff و coldiff نشان دهنده ی اختلاف سطر و ستون های دو تصویر است که در ادامه از آنها استفاده شده است.حال متغیری از جنس MAT ساخته تا مقادیر تصویر خروجی در آن ریخته شود(از آن جا که تصویر کوچک تر بر روی تصویر بزرگ قرار می گیرد تعداد سطر و ستون های تصویر خروجی نیز برابر با تعداد سطر و ستون های تصویر بزرگ تر است)به کمک data. سه پوینیتر به ماتریس پیکسل های تصویر به وجود می آوریم و در ادامه برای پرکردن ماتریس خروجی از این پوینتر ها استفاده می کنیم.به کمک دو for تو در تو بر روی پیکسل های تصویر بزرگ حرکت کرده و در صورتی که روی پیکسل هایی حرکت کنیم که درون عکس کوچک تر نیز هست(سطر و ستون های مورد بررسی کوچک تر از تعداد سطر و ستون های تصویرکوچک باشد)آن پیکسل را با ضریب ۰.۵ از پیکسل کوچک به علاوه پیکسل بزرگ مقدار دهی کرده و در غیر این صورت پیکسل بزرگ را به جای آن قرار می دهیم.همچنین برای این که در جمع ممکن است overflow اتفاق بیافتد در صورتی که حاصل جمع بزرگتر از ۲۵۵ بود آن را برابر ۲۵۵ و در صورتی که کوچکتر از ۰ بود آن را برابر ۰ در نظر می گیریم.

حال چگونگی محاسبه ی index ها توضیح داده می شود:

فرض کنید پیکسل کنونی در سطر row و ستون col باشد.از آن جا که آرایه ها در حافظه به صورت پشت هم قرار می گیرند.اگر آدرس شروع عکس in_image_۱ باشد، باید همه ی پیکسل ها تا پیکسل مورد نظر را رد کنیم تا به آدرس درست برسیم یعنی آدرس این پیکس برابر است با:

$$In\_image\_۱+row*NCOLS+col$$

که در آن NCOLS برابر تعداد ستون های تصویر بزرگ است.

محاسبه ی آدرس تصویر کوچک نیز به همین صورت است اما از آن جا که تعداد ستون های تصویر کوچک متفاوت است در عبارت بالا به جای NCOLS تعداد سطر های تصویر کوچک(img۲_cols) و به جای آدرس شروع عکس بزرگ آدرس شروع عکس کوچک گذاشته می شود.بنابراین آدرس پیکسل تصویر کوچک در حافظه برابر است با:

$$In\_image\_۲+row*img۲\_cols+col$$

آدرس حافظه در پیکسل خروجی نیز مطابق توضیحات بالا به صورت زیر محاسبه می شود:

$$p+row*NCOLS+col$$

که در آن p آدرس شروع تصویر خروجی است.

موازی:

در این قسمت می توانیم هر 16 پیکسل را با هم پردازش کنیم بنابراین بر روی ستون ها 16 تایی حرکت می کنیم. مجددا همانند قسمت قبل سه پوینتر تعریف می کنیم(تصویر کوچک،تصویر بزرگ و خروجی) تفاوت در این جاست که این پوینتر ها از نوع m128i_ هستند یعنی داده 16 داده ی 8 بایتی را در خود نگه می دارند همچنین هنگام جمع نیز به صورت 16 تایی(هر کدام 8بیت) حرکت می کنند.بنابراین جهت به دست آوردن آدرس حافظه باید این موضوع در نظر گرفته شود.برای این کار تعداد ستون ها را بر 16 تقسیم کرده و طبق آن جلو میرویم در این صورت تمام آدرس های حافظه مانند قسمت قبل محاسبه می شوند.به کمک یک for تو در تو بر روی پیکسل ها حرکت می کنیم و بسته

های16 تایی را می خوانیم در صورتی که محدوده ی برداشته شده جزو پیکسل های عکس کوچک تر نیز باشد باید بسته های 16 تایی تصویر کوچک را در نیم ضرب کرده(به کمک شیفت به راست) و سپس با بسته های 16 تایی متناظر در تصویر بزرگ جمع کنیم و نتیجه را در بسته های 16 تایی خروجی بنویسیم.درغیر این صورت تنها بسته های 16 تایی از عکس بزرگ را در تصویر خروجی ذخیره می کنیم.باید توجه داشت که در حالت موازی تابعی جهت شیفت داده های 8 بیتی وجود ندارد به همین علت در ابتدا به کل 128 بیت را شیفت می دهیم اما می دانیم که در این حالت شیفت به درستی انجام نمی شود(چرا که بیت آخر از بیت سمت چپ می رسد و لزوما صفر نیست).جهت رفع این مشکل از متغیر mask استفاده می شود.این متغیر به این صورت است که در صورتی که هر کدام از داده های درون بسته بزرگتر یا مساوی 128 بود مقدار آن 0xff می شود و در غیر این صورت مقدار آن برابر 0 می شود.برای این کار ابتدا هر کدام از داده ها با عدد 128 مقایسه شده و بین آن ها ماکسیمم گرفته می شود حال اگر ماکزیمم گرفته شده برابر با داده بود یعتی آن داده بزرگتر از 128 بوده(یعنی در شیفت به جای 0 یک قرار گرفته) و در غیر این صورت داده کمتر از 128 بوده و به درستی شیفت داده شده است.حال به کمک تابع blendv شرط بر روی هر کدام از داده های بسته بررسی می شود.درصورتی که در شیفت به آن یک اضافه شده بود آن را از 128 کم کرده تا چپ ترین یک به صفر تبدیل شود و در غیر این صورت خود آن نگه داشته می شود.سپس جمع به صورت saturation(به علت overflow) برروی دو تصویر انجام شده و نتیجه در حافظه ی مربوط به تصویر خروجی ذخیره می شود.

مقایسه دو خروجی:

پس از اجرای برنامه نتایج زیر به دست می آید.همان طور که مشاهده می شود دو تصویر به دست آمده همانند یکدیگر هستند وبا به دست آوردن تصویر در حالت پارالل به تسریعی برابر با 4.5 رسیده ایم.