CA3 - OpenCV

Parallel Programming

امید بداقی 810196423

مبینا شاه بنده 810196488

پاییز 99

دکتر صفری

مقدمه

در این پروژه دو هدف وجود دارد که اولی بدست آوردن تفاضل دو frame و دومی overlay کردن یک تصویر روی تصویر دیگر است. در این پروژه از کتابخانه ی پردازش تصویر OpenCV و دستورات برنامه نویسی موازی SIMD استفاده شده است.

بدست آوردن تفاضل دو frame

در ابتدا دو تصویر مربوط به دو frame را با استفاده از دستور `imread` خوانده و در متغیری ذخیره می کنیم (به صورت Grayscale). سپس تصویر سیاه و سفیدی با ابعاد تصویرهای خوانده شده در قسمت قبل را می سازیم. این تصویر همان نتیجه نهایی خواهد بود. فیلد دیتای آن همان مقادیر پیکسل ها است که در ادامه با آن کار خواهیم کرد.

پیاده سازی سری

حلقه ی تو در تویی برای پیمایش پیکسل های تصاویر ساخته و در هر بار اجرای آن قدر مطلق تفاضل مقدار پیکسل کنونی از تصویر یک و مقدار پیکسل کنونی از تصویر دو را محاسبه کرده و در جای مناسب در خروجی قرار می دهیم. برای به دست آوردن قدر مطلق از تابع abs کتابخانه ی cstdlib استفاده شده است.

پیاده سازی موازی

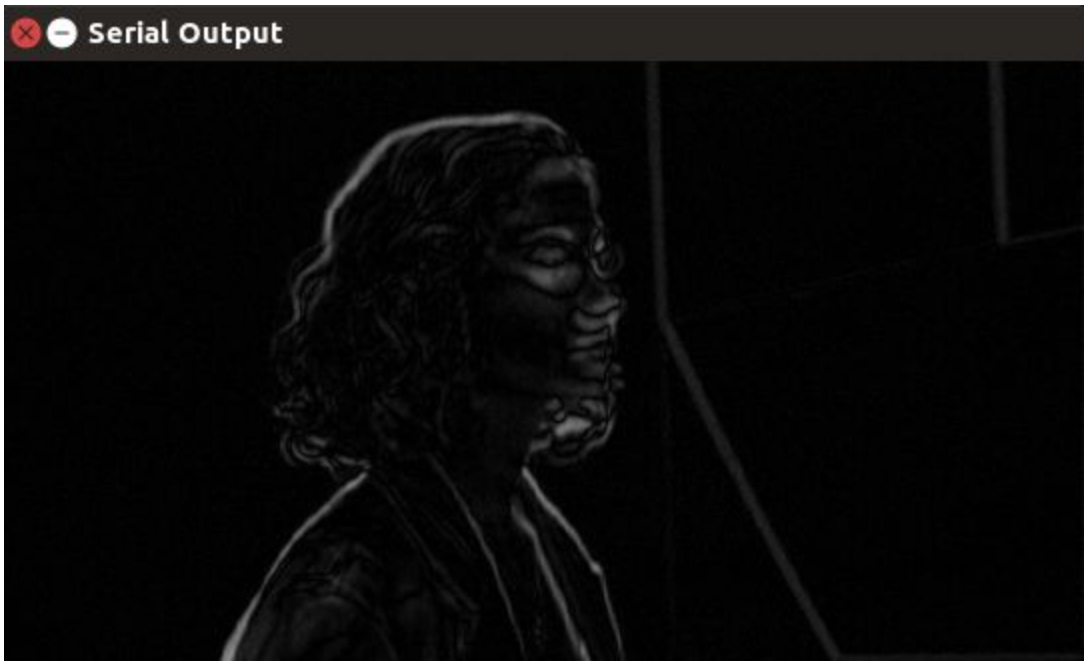
در اینجا بجای آنکه تصویرها را به صورت آرایه ای از بایت بگیریم، به صورت آرایه ای از __m128i__ می گیریم تا بتوانیم پردازش را به صورت موازی انجام دهیم. حال هر سطر از تصویر در بسته های ۱۶ بایتی پردازش می شود و در نتیجه حلقه ی داخلی یک شانزدهم برابر اجرا می شود. در هر اجرای این حلقه، ابتدا با استفاده از دستور `_mm_loadu_si128` یک بسته ۱۶ بایتی از دو تصویر ۱ و ۲ را در متغیری ذخیره می کنیم. سپس با استفاده از دستور `_mm_sub_epi8` مقدار تفاضل دو بایت متناظر از دو تصویر را بدست آورده و با استفاده از دستور `_mm_abs_epi8` مقدار قدر مطلق این تفاضل را بدست می آوریم (چون تفاضل بایت به بایت تصویر مورد نظر است از epi8 استفاده می کنیم). در نهایت نتیجه را با استفاده از دستور `_mm_storeu_si128` در قسمت متناظر از تصویر خروجی ذخیره می کنیم.

میزان تسریع

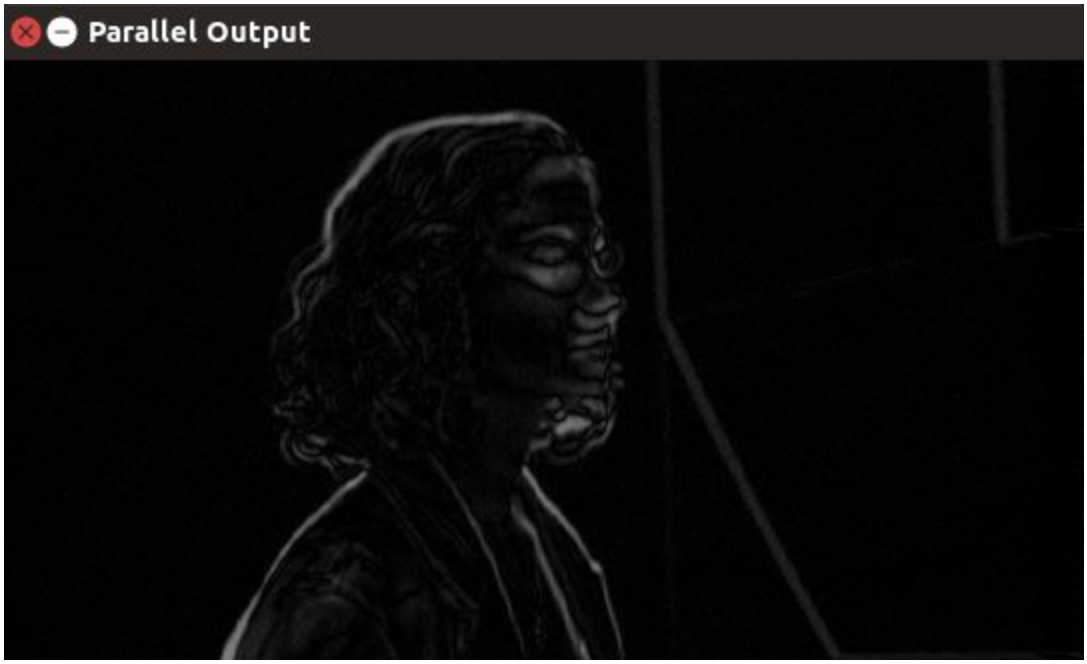
برای محاسبه زمان اجرا از کتابخانه ی sys/time استفاده شده و تفاضل زمان قبل و بعد از پردازش تصویر در هر دو حالت سری و موازی گزارش شده است. در نهایت با تقسیم زمان اجرای حالت سری بر زمان اجرای حالت موازی میزان تسریع بدست آمده است. زمان اجرای حالت سری حدود ۷۰۰ میکروثانیه و زمان اجرای حالت موازی حدود ۱۵۸ میکروثانیه و میزان تسریع حدود ۴.۵ برابر بدست آمد.

نتایج

تصویر خروجی حالت سری:



تصویر خروجی حالت موازی:



تصویر خروجی کد:

```
mobina@mobina: ~/Desktop/CA3/P1
mobina@mobina:~/Desktop/CA3/P1$ ./main
Serial implementation took 721 micro seconds

Parallel implementation took 163 micro seconds

Speedup: 4.423313

mobina@mobina:~/Desktop/CA3/P1$ ./main
Serial implementation took 671 micro seconds

Parallel implementation took 153 micro seconds

Speedup: 4.385621

mobina@mobina:~/Desktop/CA3/P1$ ./main
Serial implementation took 723 micro seconds

Parallel implementation took 157 micro seconds

Speedup: 4.605095

mobina@mobina:~/Desktop/CA3/P1$ ./main
Serial implementation took 724 micro seconds

Parallel implementation took 163 micro seconds

Speedup: 4.441718

mobina@mobina:~/Desktop/CA3/P1$
```

overlay کردن یک تصویر روی تصویر دیگر

ابتدا، با استفاده از تابع `imread`، تصاویر را به صورت GrayScale(هر پیکسل یک عدد بین ۰ تا ۲۵۵) می‌خوانیم. حال، تصویر دوم را یکبار به صورت سریال و یکبار به صورت موازی، به شکل گفته‌شده به تصویر اول اضافه می‌کنیم.

با توجه به فرض مطرح‌شده در گروه درس، تصور شده‌است که تصویر دوم همواره کوچکتر از تصویر اول می‌باشد.

بنابراین، هم در اجرای سریال و هم در اجرای موازی، در ابتدا، تصویر نهایی همان تصویر پس‌زمینه می‌باشد و حلقه‌ها فقط به اندازه تصویر کاور اجرا می‌شوند و آن قسمت را می‌توانند تغییر بدهند.

پیاده سازی سری

با توجه به فرض گفته‌شده در ابتدا، در آغاز تصویر نهایی را همان تصویر پس‌زمینه قرار می‌دهیم. برای این‌کار، از تابع clone() استفاده شده‌است که به صورت deep copy عمل می‌کند.

حال ۳ پوینتر از جنس unsigned char (با توجه به اینکه تصویر GrayScale می‌باشد و هر پیکسل، بین ۰ تا ۲۵۵ می‌باشد، می‌توان هر فریم را با یک ۸ بیتی بدون علامت نمایش داد) تعریف شده‌است. یک پوینتر به تصویر زمینه، یک پوینتر به تصویر کاور و یک پوینتر به تصویر نتایج قرار داده شده‌است.

حال زمان اجرای اضافه‌کردن تصویر، که در زیر نوشته شده‌است، محاسبه می‌شود.

اضافه‌کردن تصویر:

مقدار Img1 + Img2*alpha را برای هر پیکسل محاسبه می‌کنیم و در آن قرار می‌دهیم. به منظور اینکه برنامه سریال، در سریعترین حالت خود باشد، بجای ضرب Img2 در مقدار alpha، آن را یک واحد به سمت راست شیفت داده‌ایم.

همچنین دقت‌شود که محاسبه جمع بالا، ممکن است بیشتر از ۲۵۵ شود و در این حالت، منطقا تصویر باید کاملا سفید باشد. اما در حالت عادی، این جمع در cpp به صورت wrap-around محاسبه می‌شود. برای حل این مشکل، تابعی نوشته‌ایم تا محاسبه جمع را به صورت saturated محاسبه‌کند. این تابع، دو ورودی را با یکدیگر جمع می‌کند. در صورتی که حاصل جمع، کمتر از مقدار اول باشد(دقت شود اعداد بدون علامت

می‌باشند) قطعا، overflow رخ داده‌است و مقدار ۲۵۵ بازگردانده می‌شود. در غیر این‌صورت، مقدار جمع بازگردانده می‌شود.

پیاده سازی موازی

در ابتدا، مانند مرحله قبل تصویر را deep copy می‌کنیم و پوینترها را تعریف می‌کنیم (دقت شود در این مرحله، پوینترها از جنس __m128i می‌باشند).

حال زمان اجرای اضافه‌کردن تصویر، که در زیر نوشته شده‌است، محاسبه می‌شود.

اضافه‌کردن تصویر:

همانطور که مشاهده می‌شود، حلقه داخلی، یک شانزدهم حالت قبل، اجرا می‌شود.

یک مجموعه ۱۶ تایی ۸ بیتی را در هر مرحله از حافظه می‌خوانیم.(یکبار از تصویر پس‌زمینه bg1 و یکبار تصویر کاور cv1)

سپس، تصویر کاور را یک واحد به راست شیفت می‌دهیم(Img2*alpha). متاسفانه در SIMD، شیفت برای حالت ۱۶ ۸ بیتی، وجود ندارد. برای حل این مشکل، از شیفت ۸ ۱۶ بیتی استفاده می‌کنیم (خوشبختانه وجود دارد) اما، در این حالت، مشکلی به وجود می‌اید که ممکن است بیت ۸ ام در هر کدام ۸ قسمت، صفر نشود در حالی‌که در ۱۶ قسمت بایت، قطعا در صورت شیفت، تمام بیت‌های هشتم، باید ۰ باشند. برای حل این مشکل، نتیجه حاصل از شیفت ۸ قسمت ورد، در "01111111" AND می‌شود و تمام بیت‌های ۸ ام، صفر می‌شوند.

حال، نتیجه را با bg1 به صورت saturated جمع می‌کنیم. در SIMD، در جمع به صورت Saturated برای دو ورودی ۱۶ تایی ۸ بیتی، از تابع `_mm_adds_epu8` استفاده می‌کنیم (در اینجا، s در adds نشانه‌ی saturated و u در epu، نشانه‌ی بدون علامت بودن می‌باشد).

حال، این ۱۶ پیکسل محاسبه‌شده در یک مرحله از حلقه، همان ۱۶ پیکسل از تصویر نهایی می‌باشند و به صورت Unaligned، با استفاده از تابع `storeu`، در پوینتر گفته‌شده، مقادیر را ذخیره می‌کنیم.

میزان تسریع

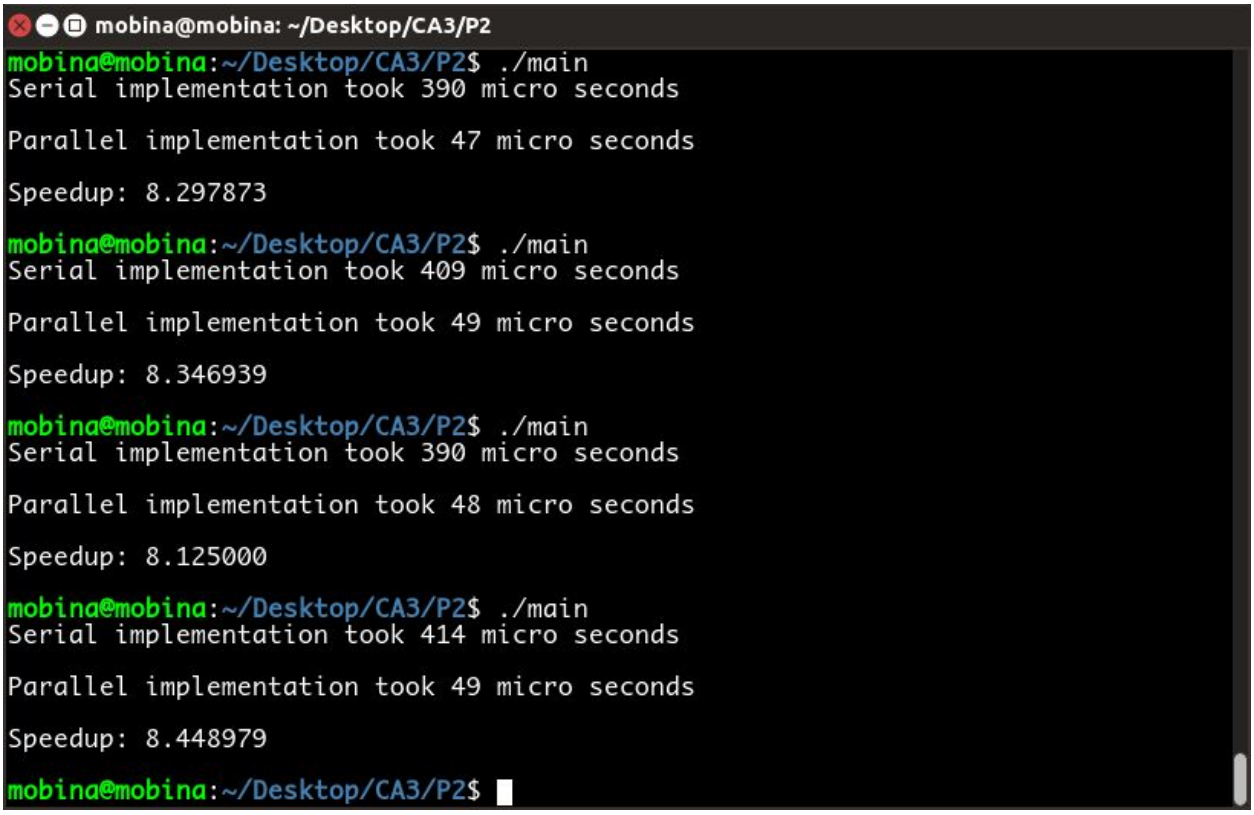
همانطور که در تصاویر اجرا مشاهده می‌شود، به طور متوسط حدود ۸ برابر تسریع در روش موازی داریم که عدد قابل توجهی می‌باشد. روش سریال، حدودا ۴۰۰ میکروثانیه و روش موازی، حدود ۵۰ میکروثانیه برای اجرا زمان نیاز دارند.

نتایج

تصویر خروجی حالت سری:

تصویر خروجی حالت موازی:

تصویر خروجی کد:


mobina@mobina: ~/Desktop/CA3/P2
mobina@mobina:~/Desktop/CA3/P2$ ./main
Serial implementation took 390 micro seconds
Parallel implementation took 47 micro seconds
Speedup: 8.297873
mobina@mobina:~/Desktop/CA3/P2$ ./main
Serial implementation took 409 micro seconds
Parallel implementation took 49 micro seconds
Speedup: 8.346939
mobina@mobina:~/Desktop/CA3/P2$ ./main
Serial implementation took 390 micro seconds
Parallel implementation took 48 micro seconds
Speedup: 8.125000
mobina@mobina:~/Desktop/CA3/P2$ ./main
Serial implementation took 414 micro seconds
Parallel implementation took 49 micro seconds
Speedup: 8.448979
mobina@mobina:~/Desktop/CA3/P2$