به‌نام خدا

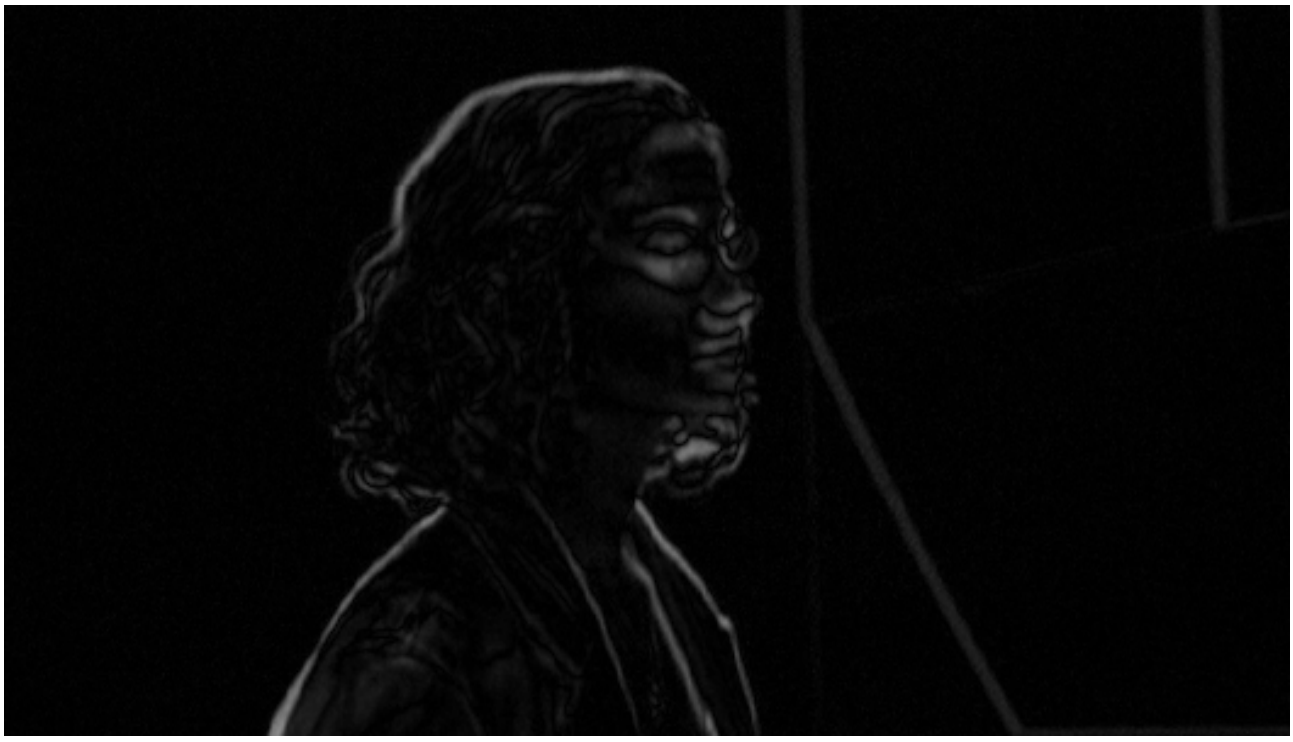
۱. در این بخش ابتدا دو تصویر را می‌خوانیم و در ماتریس‌های مربوطه ذخیره می‌کنیم. سپس در قسمت محاسبه‌ی سریال، روی هر دو ماتریس پیمایش می‌کنیم، پیکسل مربوطه را از هر دو ماتریس می‌خوانیم و سپس با استفاده از تابع std::abs، قدرمطلق تفاضل آن‌ها را محاسبه کرده و در خروجی می‌نویسیم.

برای قسمت محاسبه‌ی موازی نیز سه رجیستر ۱۲۸ بیتی درنظر می‌گیریم که دوتای آن‌ها برای خواندن از دو ماتریس و دیگری برای ذخیره‌ی نتیجه‌ی قدرمطلق تفاضل آن‌هاست. برای محاسبه‌ی این مقدار و از آن‌جا که تابع _mm_abs_ epu8 که اعداد صحیح بدون علامت هستند وجود ندارد، اقدام به استفاده از روش دیگری می‌کنیم. در این روش دو بار تفاضل اشباع‌شده‌ی دو رجیستر m1 و m2 را محاسبه می‌کنیم که بار اول m1 از m2 کم می‌شود و بار دیگر m2 از m1. از طرفی در تفاضل اشباع‌شده، اعداد منفی تبدیل به صفر می‌شوند. حال برای محاسبه‌ی قدرمطلق این مقدار، بین دو مقداری که پیش‌تر محاسبه کردیم از عملگر or استفاده می‌کنیم و چون مقدار یکی از آن‌ها حتماً صفر است، مقدار بزرگ‌تر را به دست می‌آوریم:

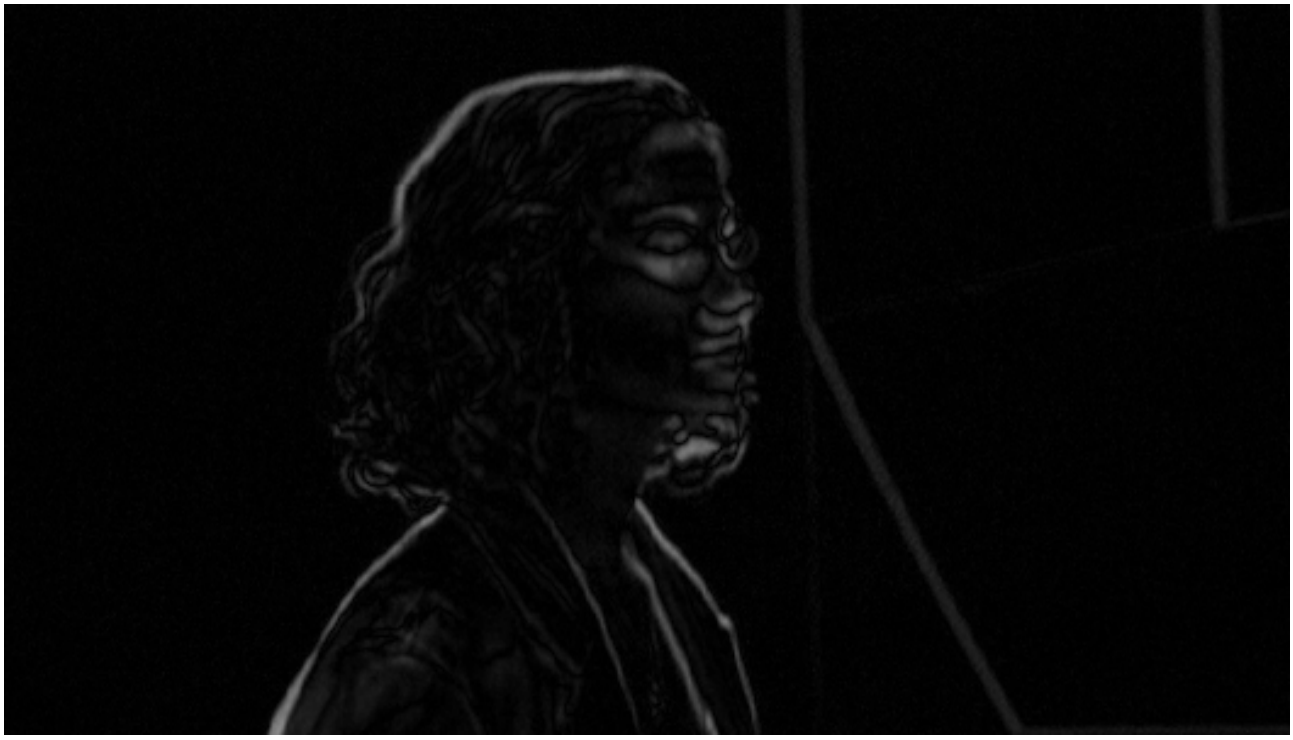
```
_mm_or_si128(_mm_subs_epu8(m1, m2), _mm_subs_epu8(m2, m1));
```

پس از اجرای این بخش به نتایج زیر می‌رسیم:

اجرای سریال:



اجرای موازی:



نهایتاً بهبود سرعت ۴٫۲۰ برابری را شاهد هستیم:

```
Hossein Soltanloo - 810195407
Serial Run time = 1835277
Parallel Run time = 436454
Speedup = 4.20
```

۲. در این بخش ابتدا دو تصویر را می‌خوانیم و در ماتریس‌های مربوطه ذخیره می‌کنیم. سپس در قسمت محاسبه‌ی سریال، روی ماتریس تصویر دوم پیمایش می‌کنیم، پیکسل مربوطه و متناظرش از تصویر اول را نیز می‌خوانیم و سپس اقدام به محاسبه‌ی خروجی می‌کنیم. از آن‌جا که نیاز به سریع‌ترین روش برای جمع اشباع‌شده‌ی دو عدد داریم، آن را در قالب یک تابع تعریف کرده و استفاده می‌کنیم. در این تابع دو عدد با هم جمع می‌شوند و هرگاه حاصل آن‌ها از یکی از آن‌ها کوچک‌تر بود، یعنی اشباع صورت گرفته و باید مقدار ۲۵۵ را به‌عنوان نتیجه در نظر بگیریم:

```
uint8_t saturatedAdd(uint8_t a, uint8_t b)
{
  uint8_t c = a + b;
  if (c < a)
    c = 255;
  return c;
}
```

سپس به سراغ بخش موازی می‌رویم. در این بخش مانند قبل سه رجیستر ۱۲۸ بیتی درنظر می‌گیریم که دوتای آن‌ها برای خواندن از دو ماتریس و دیگری برای ذخیره‌ی نتیجه‌ی محاسبه روی آن‌هاست. برای محاسبه روی این دو مقدار، ابتدا باید پیکسل‌های تصویر دوم را در ۰٫۵ ضرب کنیم که این کار باید با استفاده از شیفت به راست صورت بگیرد؛ اما تابعی برای شیفت به راست اعداد ۸ بیتی بدون علامت، یعنی _mm_srli_epu8 تعبیه نشده است. از این رو ۴ بایت را هم‌زمان به‌عنوان یک ۳۲ بیتی درنظر می‌گیریم و آن را به راست شیفت می‌دهیم تا هر ۴ بایت به راست شیفت بخورند؛ اما در این حالت بیت پرارزش آن‌ها صفر نخواهد بود و برای حل این مشکل از طرف دیگر ۱۶ عدد ۸ بیتی 01111111 را در یک رجیستر ذخیره می‌کنیم و نهایتاً ۴ مقدار شیفت‌دهی‌شده‌ی ۳۲بیتی را با این ۱۶ بایت and می‌کنیم تا با این روش بتوانیم شیفت به راست را برای epu8 پیاده‌سازی کنیم:

```
_mm_adds_epu8(m1, _mm_and_si128(_mm_set1_epi8(0xFF >> 1), _mm_srli_epi32(m2, 1)));
```

نهایتاً مقدار به‌دست آمده را به مقادیر پیکسل‌های تصویر اول جمع اشباع‌شده می‌کنیم که این کار نیز با _mm_adds_epu8 صورت می‌گیرد.

پس از نوشتن مقادیر روی خروجی، به نتایج زیر می‌رسیم:

اجرای سریال:

اجرای موازی:

نهایتاً بهبود سرعت ۵٫۵۱ برابری را شاهد هستیم:

```
Hossein Soltanloo - 810195407
Serial Run time = 1099010
Parallel Run time = 199624
Speedup = 5.51
```