گزارش پروژه سوم برنامه نویسی موازی

امین عارف زاده

امیرحسین محمودی

810195424-810195578

سوال یک:

اول دو عکس را با کمک توابع openCV به صورت gray scale شده لود کردیم.

در قطعه کد زیر قسمت سریال برنامه نمایش داده شده است.

```
gettimeofday(&start, NULL);
for (int row = 0; row < image_rows; row++) {
  for (int col = 0; col < image_cols; col++) {
    serial_output_mat[row * image_cols + col] = abs(
      serial_first_image_mat[row * image_cols + col] - serial_second_image_mat[row * image_cols + col]
    );
  }
}
gettimeofday(&end, NULL);
serial_time = (end.tv_sec - start.tv_sec) * 1000000 + (end.tv_usec - start.tv_usec);
```

مورد خاصی در این کد نیست و همانطور که مشاهده می‌شود از تابع abs در کتابخوانه‌ی cstdlib استفاده شده برای محاسبه‌ی قدر مطلق تفاضل.

در قسمت پارالل، ابتدا از تابع abs_epi8 استفاده کردیم. با استفاده از این تابع اسپیدآپی تقریبا حدود ۵ و ۶ گرفتیم که خیلی اسپید آپ مناسبی به حساب می‌آید. اما این تابع در sse3 تعریف نشده و یک نسخه بالاتر است و در ssse3 تعریف شده در نتیجه مجاز به استفاده از آن نبودیم.

```
gettimeofday(&start, NULL);
for (int row = 0; row < image_rows; row++) {
  for (int col = 0; col < image_cols / 16; col++) {
    xReg1 = _mm_loadu_si128(parallel_first_image_mat + row * image_cols / 16 + col);
    xReg2 = _mm_loadu_si128(parallel_second_image_mat + row * image_cols / 16 + col);
    xReg1 = _mm_sub_epi8(xReg1, xReg2);
    xReg1 = _mm_abs_epi8(xReg1);
    _mm_storeu_si128(parallel_output_mat + row * image_cols / 16 + col, xReg1);
  }
}
gettimeofday(&end, NULL);
parallel_time = (end.tv_sec - start.tv_sec) * 1000000 + (end.tv_usec - start.tv_usec);
```

در ادامه سعی شد با استفاده از روشی که استاد در کلاس گفتند. یعنی با استفاده از عملگر های and, andnot و or در واقع یک نوع Vectorization of Cond بنویسیم که همانند عملگر abs در کد قبلی عمل کند. که کد زیر نوشته شد.

```
gettimeofday(&start, NULL);
for (int row = 0; row < image_rows; row++) {
  for (int col = 0; col < image_cols / 16; col++) {
    xReg1 = _mm_loadu_si128(parallel_first_image_mat + row * image_cols / 16 + col);
    xReg2 = _mm_loadu_si128(parallel_second_image_mat + row * image_cols / 16 + col);
    mask = _mm_cmpgt_epi8(xReg1, xReg2);
    res = _mm_or_si128(_mm_and_si128(mask, _mm_sub_epi8(xReg1, xReg2)), _mm_andnot_si128(mask, _mm_sub_epi8(xReg2, xReg1)));
    _mm_storeu_si128(parallel_output_mat + row * image_cols / 16 + col, res);
  }
}
gettimeofday(&end, NULL);
parallel_time = (end.tv_sec - start.tv_sec) * 1000000 + (end.tv_usec - start.tv_usec);
```

اما این کد در محاسبه مشکل داشت و خروجی شبیه خروجی سریال نبود. متوجه شدیم مشکل این است که از توابع cmpgt_epi8 و sub_epi8 استفاده کردیم و این توابع به صورت signed عملیات را انجام میدهد. در نتیجه الگوریتم را دچار مشکل میکنند. در نتیجه تغییراتی دادیم که در قطعه کد زیر مشاهده می‌شود، و توانستیم مشکل را حل کنیم.

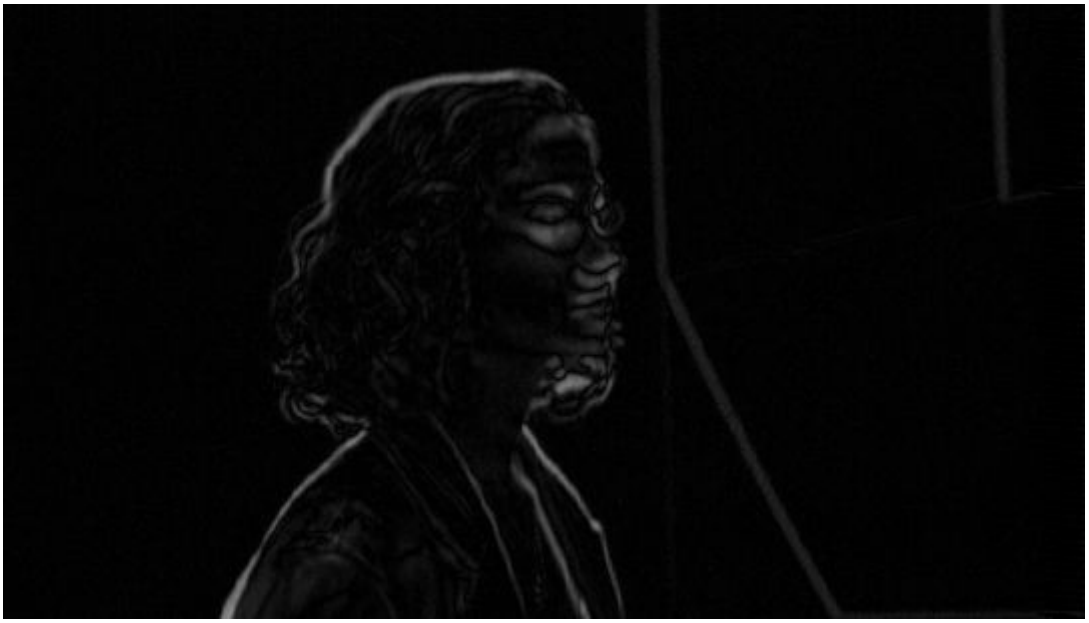
```
gettimeofday(&start, NULL);
for (int row = 0; row < image_rows; row++) {
  for (int col = 0; col < image_cols / 16; col++) {
    xReg1 = _mm_loadu_si128(parallel_first_image_mat + row * image_cols / 16 + col);
    xReg2 = _mm_loadu_si128(parallel_second_image_mat + row * image_cols / 16 + col);
    sub1 = _mm_sub_epi8(xReg1, xReg2);
    mask = _mm_cmpgt_epi8(sub1, zero);
    res = _mm_or_si128(_mm_and_si128(mask, sub1), _mm_andnot_si128(mask, _mm_sub_epi8(xReg2, xReg1)));
    _mm_storeu_si128(parallel_output_mat + row * image_cols / 16 + col, res);
  }
}
gettimeofday(&end, NULL);
parallel_time = (end.tv_sec - start.tv_sec) * 1000000 + (end.tv_usec - start.tv_usec);
```

اما اسپید آپ در روش بالا به مقدار ۳ رسیده بود که کاهش شدیدی نسبت به استفاده از تابع abs نشان میداد. برای افزایش سرعت الگوریتم تصمیم گرفتیم به جای استفاده از Vectorized Cond از تابع max استفاده کنیم. به این شکل که بین تفاضل عکس۱ از عکس۲ و تفاضل عکس۲ از عکس۱ ماکس میگیریم. در واقع لاجیک abs را با استفاده از ماکس بین عدد و قرینه‌ی اون عدد بدست میاریم که اینکار میتونه سرعت الگوریتم رو زیاد کنه. متوجه شدیم در sse2 تنها تابع max_epu8 وجود داد که به صورت unsigned عمل میکند. در نتیجه مجبوریم به جای sub_epi8 از subs_epu8 استفاده کنیم که عملیات تفریق را به صورت unsigned و saturated عمل می‌کند. در نهایت کد زیر پیاده شد که توانست اسپید آپ نزدیک به 6 به ما بدهد و عملکردی شبیه abs در ssse3 داشته باشد.

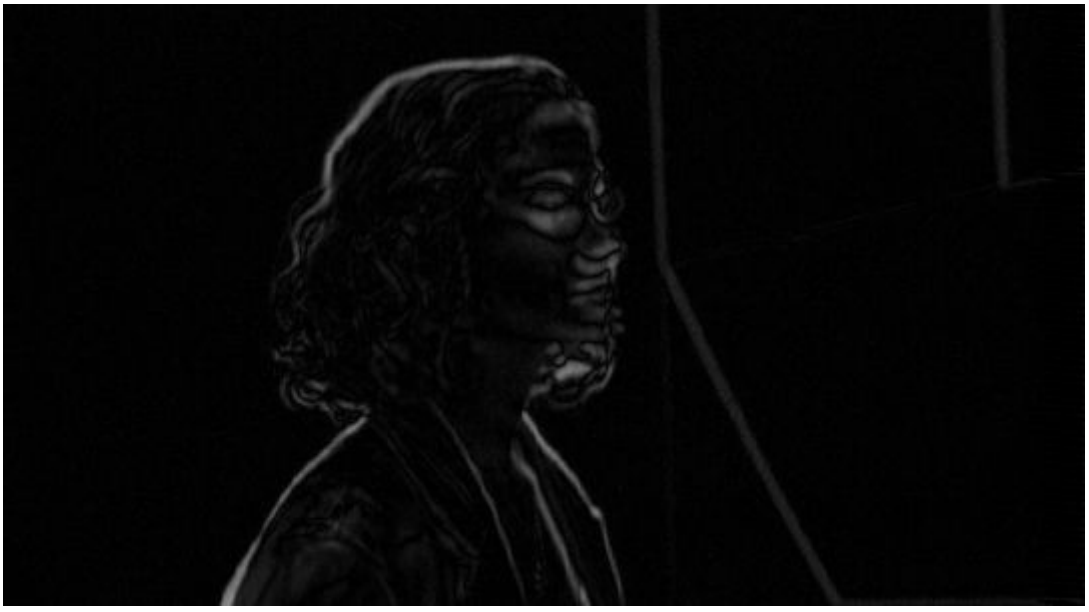
```
gettimeofday(&start, NULL);
for (int row = 0; row < image_rows; row++) {
  for (int col = 0; col < image_cols / 16; col++) {
    xReg1 = _mm_loadu_si128(parallel_first_image_mat + row * image_cols / 16 + col);
    xReg2 = _mm_loadu_si128(parallel_second_image_mat + row * image_cols / 16 + col);
    res = _mm_max_epu8(_mm_subs_epu8(xReg1, xReg2), _mm_subs_epu8(xReg2, xReg1));
    _mm_storeu_si128(parallel_output_mat + row * image_cols / 16 + col, res);
  }
}
gettimeofday(&end, NULL);
parallel_time = (end.tv_sec - start.tv_sec) * 1000000 + (end.tv_usec - start.tv_usec);
```

در زیر تصاویر خروجی دو الگوریتم نمایش داده شده که کاملا شبیه یک دیگر اند

خروجی سریال:



خروجی موازی:



خروجی ترمینال:

```
~/Desktop/uni/pp/parallelprojects/Project3/question1  master  ./main
Amin Arefzade 810195424
Amirhossein Mahmoodi 810195578
Serial time(ns): 1226
Parallel time(ns): 197
Speed up: 6.223350
```

سوال دو:

ابتدا از آنجایی که اندازه تصاویر مضرب 16 نیست، هر دو تصویر را با استفاده از zero padding به مضرب 16 تبدیل میکنیم. دقت کنید که لزوم این کار برای حالت parallel است. زیرا در حالت parallel داده ها در بسته های 16 تایی برداشته می شوند. قطعه کد زیر این عملیات را انجام می دهد.

```
cv::Mat zeroPadding(cv::Mat &inputImage) {
  int padding = (16 - (inputImage.cols % 16)) % 16;

  cv::Mat paddedImage(inputImage.rows, inputImage.cols + padding, inputImage.type());
  paddedImage.setTo(cv::Scalar::all(0));

  inputImage.copyTo(paddedImage(cv::Rect(0, 0, inputImage.cols, inputImage.rows)));
  return paddedImage;
}
```

حال ابتدا عملیات را به صورت serial انجام می دهیم. برای این منظور با پیمایش بر روی عکس background هر پیکسل را گرفته و در صورتی که در آن موقعیت، پیکسلی متناظر در foreground باشد، پیکسل foreground را یک شیفت به راست داده و با پیکسل background جمع می کنیم. در غیر این صورت همان پیکس background به تنهایی به عنوان پیکسل خروجی ذخیره می شود. دقت کنید که جمعی که در اینجا استفاده می شود یک saturation addition است. قطعه کد زیر نحوه انجام این جمع را نشان می دهد. اگر مقدار نهایی جمع از هر کدام از پیکسل ها کمتر شود یعنی wrap around رخ داده است. در این حالت مقدار پیکسل خروجی را برابر **منفی یک** (یعنی تماما 1 در نمایش باینری) قرار می دهیم که همان 0xFF است.

```
*(serialFD + row * backgroundCols + col) = *(serialBackgroundFD + row * backgroundCols + col) +
                                           (*(serialForeground + row * foregroundCols + col) >> 1);
if(*(serialFD + row * backgroundCols + col) < *(serialBackgroundFD + row * backgroundCols + col))
  *(serialFD + row * backgroundCols + col) = -1;
```

حال به محاسبه خروجی حالت parallel می پردازیم. همانطور که قبلا گفته شد، عکس ها مضرب 16 شده اند. فلذا کافیست به صورت سطر به سطر عکس ها را پیمایش کرده و در صورتی که موقعیت foreground مناسب باشد، دو عکس را با هم جمع کنیم. تنها نکته ای که وجود دارد نحوه ایجاد پیکسل های عکس foreground است. متاسفانه خانواده SSE شیفت به راست به صورت 8 بیت را ارائه نمی دهد. کوچکترین شیفت به راست این خانواده، شیفت به راست 16 بیتی است. یعنی بسته ی 128 بیتی را به 8 تا بسته ی 16 بیتی تقسیم کرده و هر یک را شیفت به راست می دهد. در این حالت برای اینکه بتوانیم شیفت به راست برای بسته های 8 بیتی را شبیه سازی کنیم، ابتدا بسته های 16 تایی را یکی به راست شیفت می دهیم.

حال با استفاده از یک دستور and بیت اول هر دسته 8 بیتی را صفر می کنیم. قطعه کد زیر این عملیات را نشان می دهد. اینگونه می توان شیفت به راست 8 بیتی را شبیه سازی کرد.

```
if(row < foregroundRows && col < foregroundCols / 16) {
  xReg2 = _mm_loadu_si128(parallelForegroundFD + row * foregroundCols / 16 + col);
  xReg2 = _mm_srli_epi16(xReg2, 1);
  xReg2 = _mm_and_si128(xReg2, _mm_set1_epi8(0x7f));
  xReg1 = _mm_adds_epu8(xReg1, xReg2);
```

در آخر آن مقادیر صفر اضافی موجود در عکس های خروجی را حذف می کنیم. قطعه کد زیر این کار را انجام می دهد.

```
cv::Mat cropZeroes(cv::Mat &inputImage, int padding) {
  cv::Rect cropRegion(0, 0, inputImage.cols - padding, inputImage.rows);
  return inputImage(cropRegion);
}
```

در شکل زیر خروجی parallel و serial برنامه نمایش داده شده است. همانطور که توقع داریم این دو خروجی دقیقا یک شکل هستند.

Parallel Output

Serial Output

خروجی ترمینال: در این تصویر مقدار speed up را نیز مشاهده می کنید که تقریبا ۵.۵ برابر افزایش سرعت داشتیم.