**به نام خدا**

تمرین کامپیوتری یک – موازی سازی در سطح داده

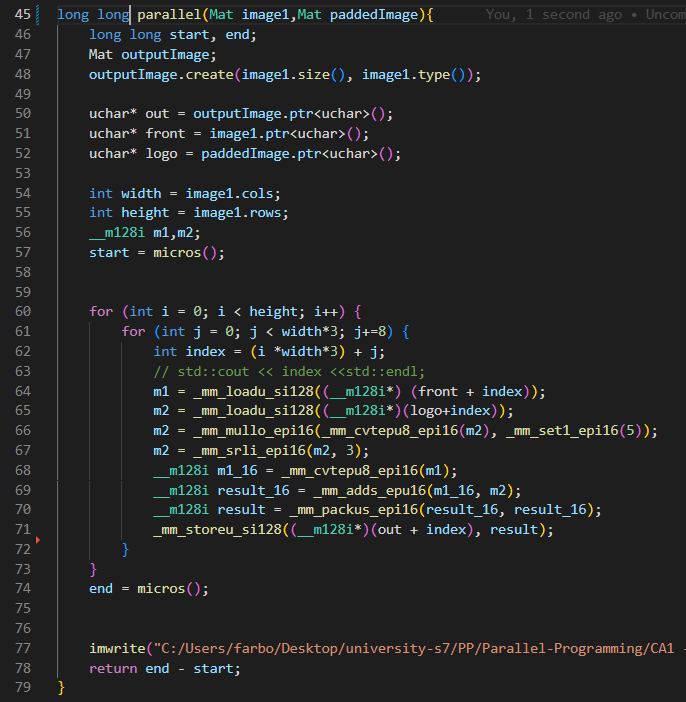
فربد عظیم محسنی 810100187

امیرحسین راحتی 81010144

* سوال اول:

در این بخش ، با استفاده از opencv ابتدا تصاویر را میخوانیم. و سپس لوگو را به اندازه عکس pad می کنیم. در حالت سریال ، با استفاده از حلقه های تو در تو ، روی پیکسل های عکس پیمایش میکنیم و لوگو را اضافه میکنیم. به این صورت که ابتدا لوگو را در 5 ضرب می کنیم و در نهایت با سه بار شیفت دادن تقسیم بر 8 می کنیم تا بتوانیم در کل لوگو را در 0.625 ضرب کرده باشیم. در نهایت نیز پیکسل به پیکسل جمع می کنیم. این جمع به صورت saturation انجام می شوند تا داده ها در محدوده 0 تا 255 قرار بگیرند. تمام عملیات ها به صورت short انجام شده و در نهایت به uchar تبدیل می شوند.



در حالت موازی ، حلقه های تو در تو وجود دارند ولی محاسبات بوسیله دستورات موازی simd انجام میشود : ابتدا ایندکس مورد نظر برای قرار گرفتن لوگو محاسبه میشود. نکته ای که وجود دارد این است که ما داده ها را 128 بیت میخوانیم که شامل 16 پیکسل از عکس است. ولی باید محاسبات را با داده های 16 بیتی short انجام دهیم. به همین منظور از دستور \_mm\_cvtepu8\_epi16 استفاده می کنیم که به این صورت کار می کند که 8 بایت بالا را خوانده و هر کدام را به داده های 16 بیتی تبدیل می کند و در نهایت ما 8 پیکسل را در هر دور پردازش می کنیم. و عملیات ها را به صورت word انجام می دهیم. عملیات ها نیز شبیه حالت سریال هستند. در نهایت از \_mm\_packus\_epi16 برای اینکه داده ها را به صورت uchar و غیر علامت دار تبدیل کنیم استفاده می کنیم. این عملیات به صورت saturation انجام می شود و داده ها را در محدوده 0 تا 255 قرار می دهد. 

خروجی:

خروجی حالت موازی

خروجی حالت سریال

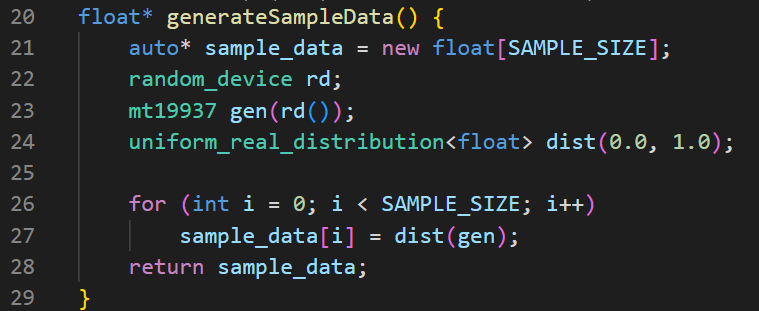
مقدار speed up:



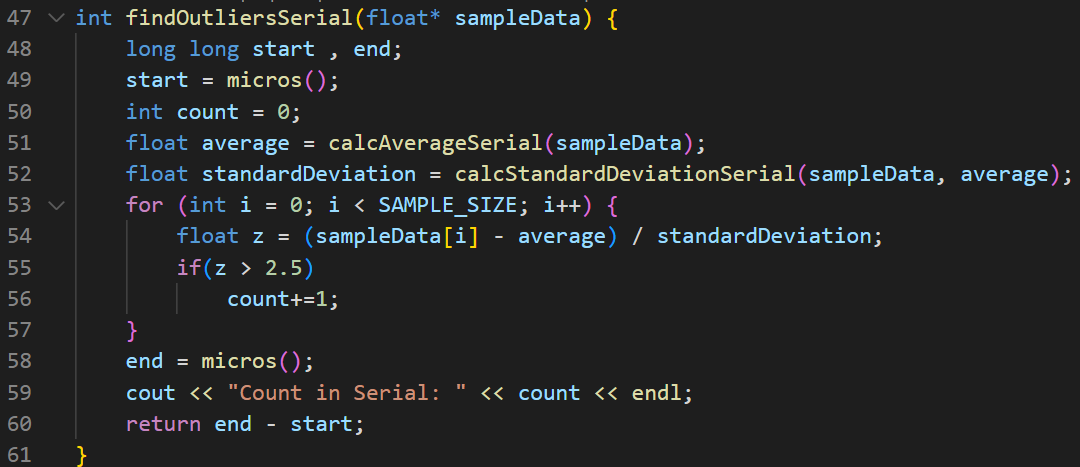
در اینجا ما speeup خیلی خوبی میگیریم و منطقی هم هست چرا که ما هر دفعه به جای اینکه یک پیکسل را پردازش کنیم داریم 8 پیکسل را پردازش می کنیم و حداکثر speedup ما می تواند برابر 8 باشد که به دلیل overhead های انتقال بین simd و حافظه و همچنین کند بودن save و load کردن نمی توان به حداکثر speedup رسید.

* سوال دوم :

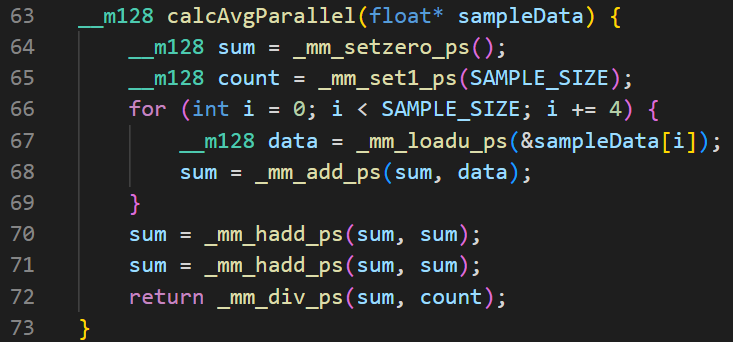
در این بخش ، ابتدا یک آرایه از اعداد اعشاری بین 1 و صفر میسازیم . سپس این داده را برای هر دو حالت موازی و سریال بررسی می کنیم.



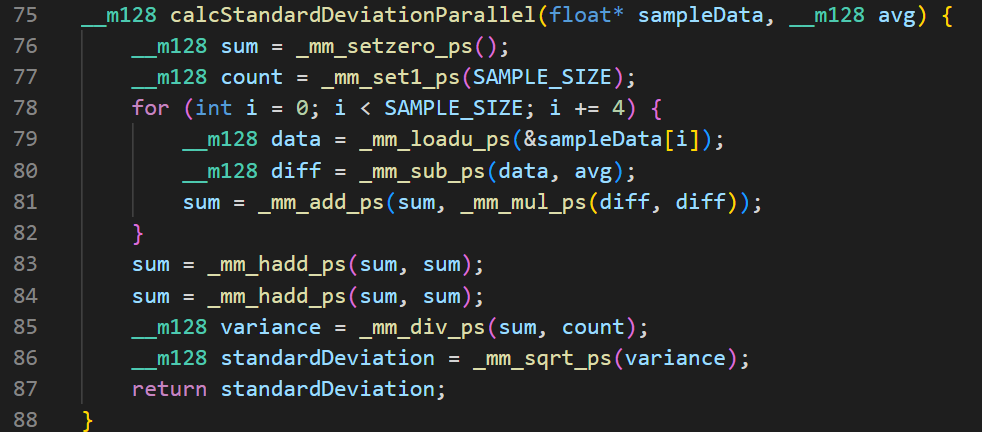
در حالت سریال ، مقدار انحراف معیار و میانگین جداگانه محاسبه میشود و تعداد اعداد پرت در تابع زیر بدست می آید :



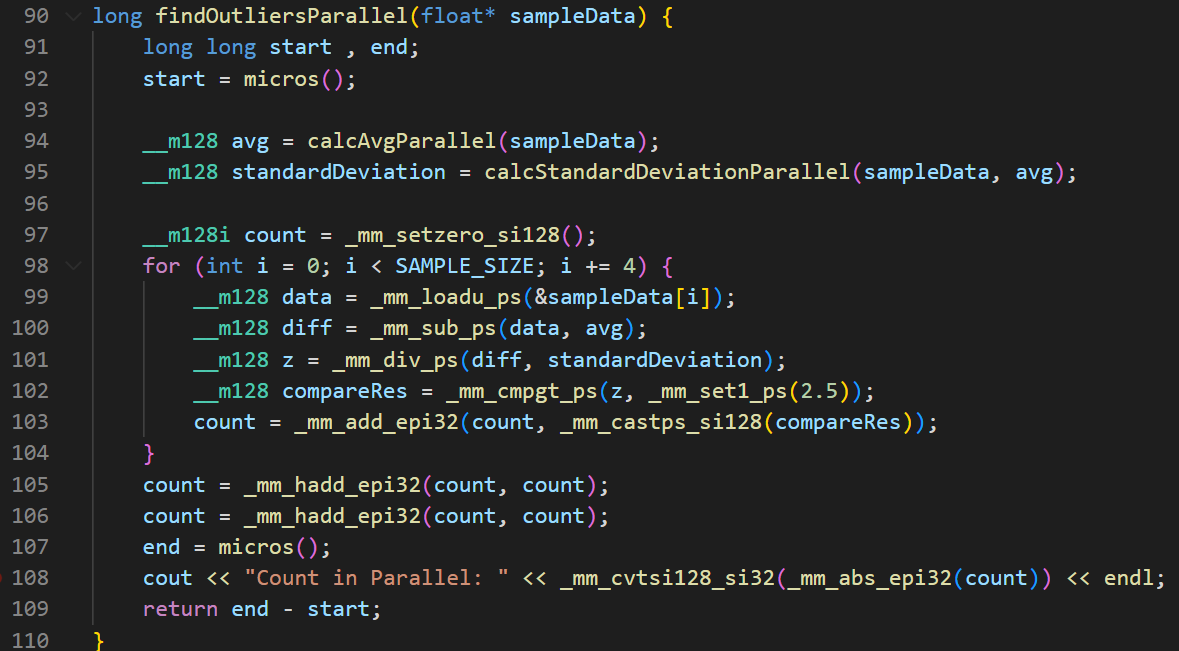
در بخش موازی ، ابتدا میانگین و انحراف معیار را می یابیم و در نهایت تعداد داده های پرت را محاسبه میکنیم : اعداد در یک متغیر \_\_m128 جمع میشوند و در اخر همه مقادیر متغیر sum با خودش جمع میشود .



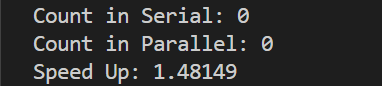
محاسبه انحراف معیار : بوسیله میانگین در بخش قبل ، طبق فرمول انحراف معیار ، آن را محاسبه میکنیم



در نهایت اعداد پرت شمارش میشوند : روند شمارش به این صورت است که هر مرحله مقدار z محاسبه میشود و با 2.5 مقایسه میشود . نتیجه خروجی مقایسه را در count ذخیره میکنیم و در آخر مقدار خانه های count را با هم جمع میکنیم:



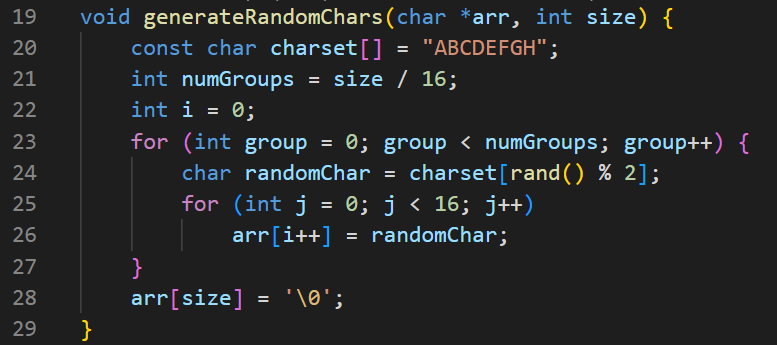
در اخر، خروجی و speedup برای یک دیتاست با توزیع یونیفرم به صورت زیر خواهد بود :



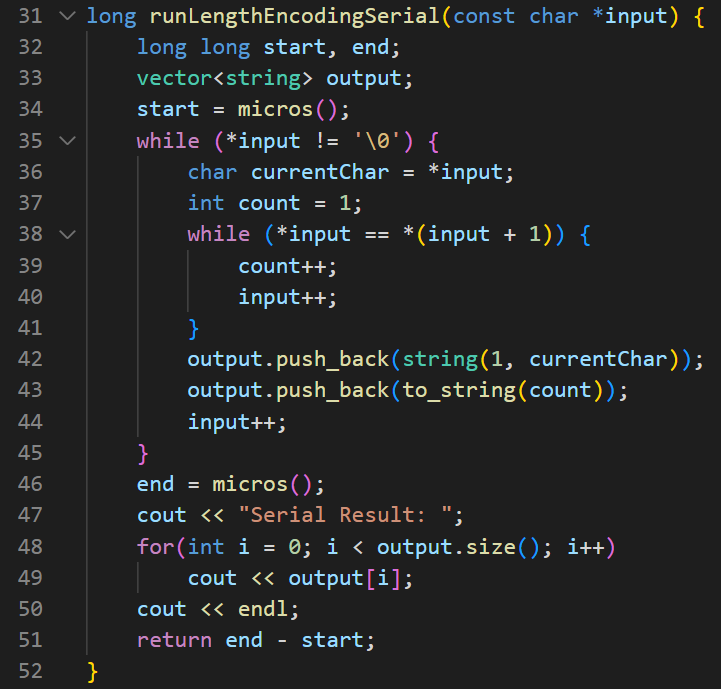
از آنجایی که شرایط ایده آل نیست ، مقدار رایط ایده آل نیست ، مقدار speedup عددی کمتر از 4 برابر خواهد بود .

* سوال سوم :

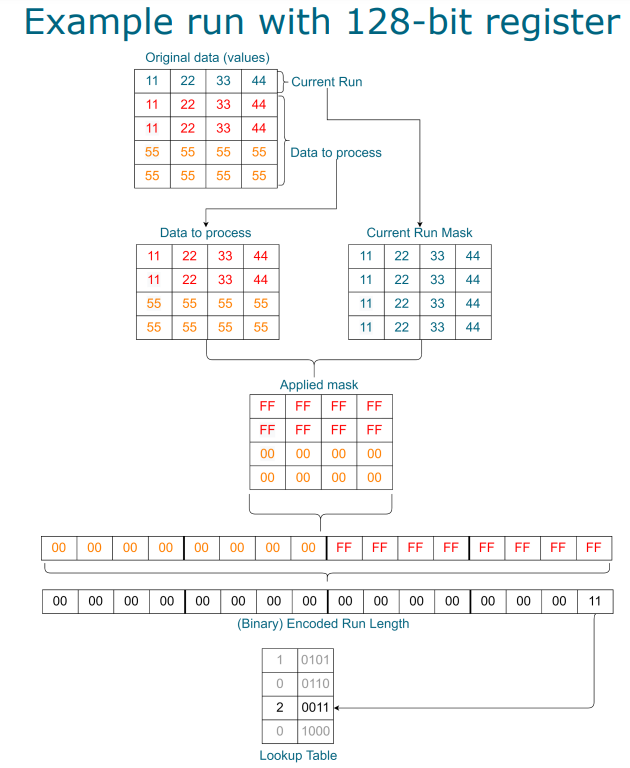
ابتدا یک آرایه از کاراکتر ها ایجاد میکنیم . از آنجایی که این الگوریتم برای فشرده سازی داده های متنی خوب نیست و بیشتر مناسب داده هایی است که دیتای پشت سر هم یکسان زیاد دارند ، سعی میشود که داده ای ایجاد کنیم که این خاصیت را داشته باشد تا اثر موازی سازی به درستی مشاهده شود.



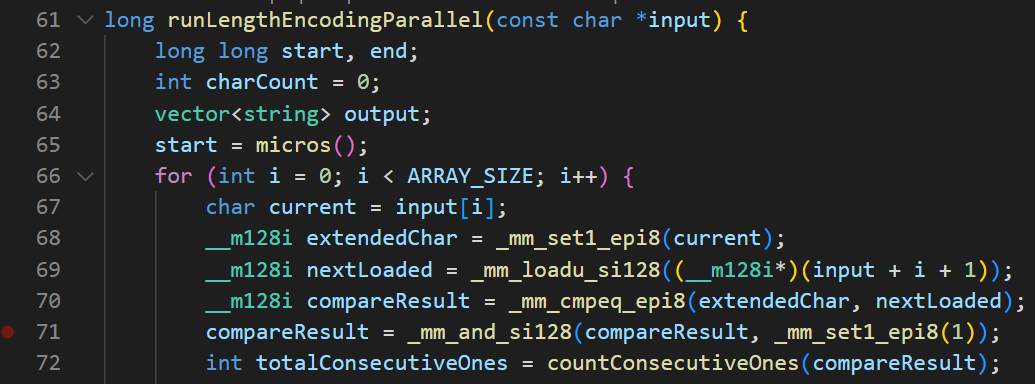
در حالت سریال ، نیاز به یک پیمایش با اردر o(n) داریم که در هر ایندکس ، مقدار کاراکتر فعلی را با قبلی مقایسه میکند. پیاده سازی آن به صورت زیر است :



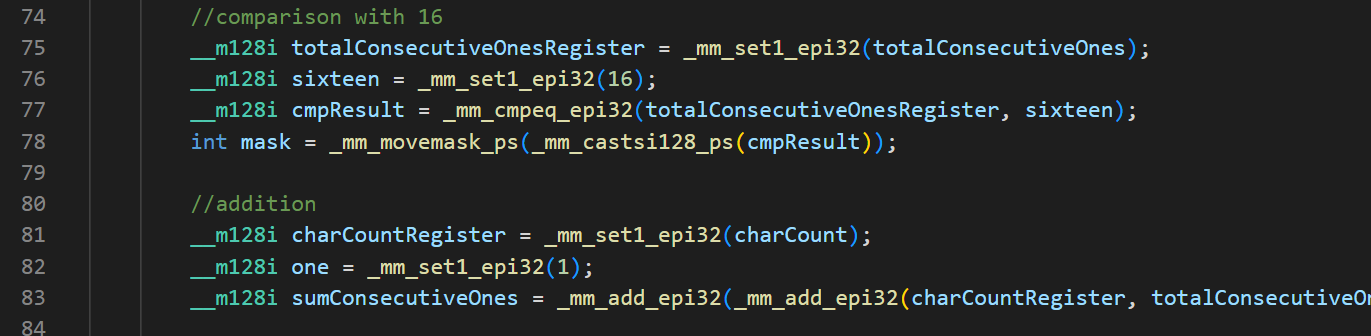
برای بخش موازی ، نیاز به تغییراتی در بخش سریال داریم . به این صورت که ابتدا یک کاراکتر از ورودی میخوانیم و آن را در یک متغیر \_\_m128i ذخیره میکنیم و آن را در متغیر بسط میدهیم . سپس 16 کاراکتر بعدی را از آرایه میخوانیم و در یک متغیر \_\_m128i دیگر ذخیره میکنیم .حالا مقادیر این دو متغیر را با هم مقایسه میکنیم و خروجی تعدادی یک به معنی مساوی بودن و صفر به معنی مساوی نبودن است . حالا برای اینکه بفهمیم چند کاراکتر بعد از کاراکتر فعلی ، برابر با کاراکتر فعلی هستند ، باید بیت های یک متوالی از اول آرایه نتیجه را بشماریم و مقدار شمارنده i را به اندازه آن جلو ببریم. این ایده در تصویر زیر به نمایش گذاشته شده است :



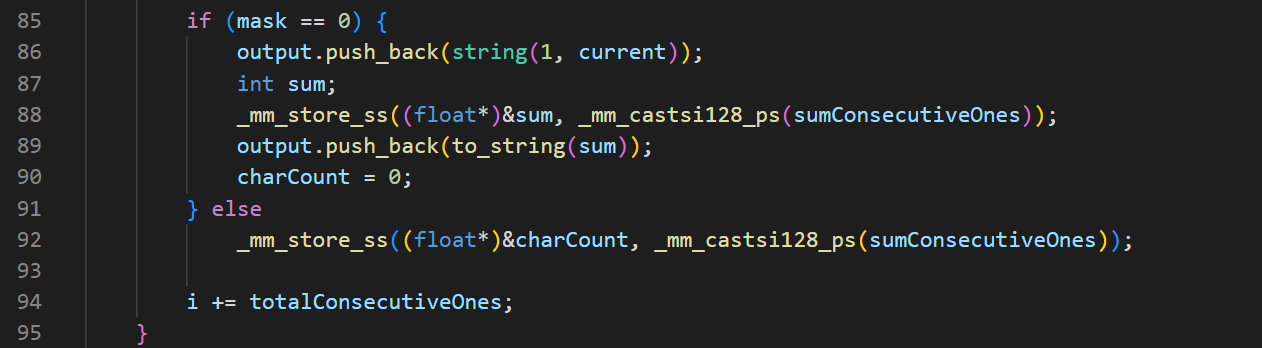
برای حالتی که بیش از 16 کاراکتر یکسان پشت هم داریم ، نیاز است نتیجه مقایسه را چک کنیم . اگر این اتفاق افتاده بود ، 16 کاراکتر بعدی را لود می کنیم و مقایسه را انجام میدهیم . در این حالت نیاز است نتیجه مقایسه قبلی را حفظ کنیم . پیاده سازی سریال این بخش به صورت زیر خواهد بود :



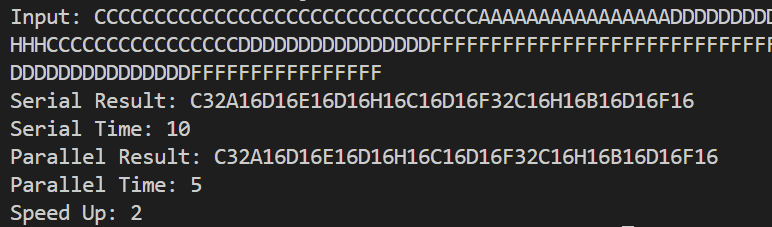
در این قسمت ، کاراکتر خوانده میشود و سپس مقایسه و سپس تعداد یک هاای متوالی شمرده میشود.



حالا چک میکنیم که آیا مقدار یک های متوالی 16 هست یا خیر . اگر نبود ، i را به اندازه تعداد یک ها جلو میبریم . این تعداد را به همراه کاراکتر فعلی در آرایه نتیجه ذخیره میکنیم . وگرنه تعداد یک ها را ذخیره میکنیم و 16 کاراکتر بعدی را لود میکنیم .



در نهایت ، خروجی حالات موازی و سریال با وروردی زیر به این شکل خواهد بود :

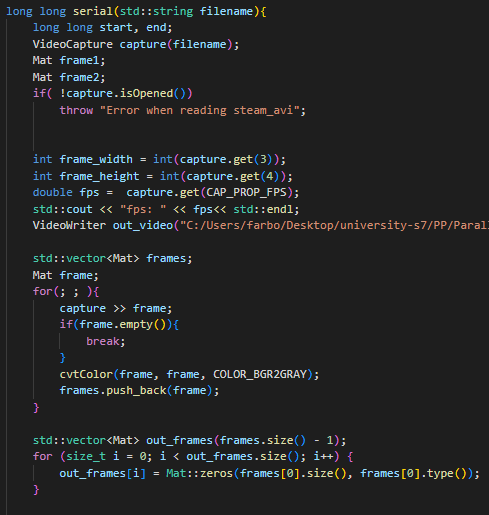


در اجرا های مختلف ، میزان speedup متفاوت خواهد بود. اما به طور متوسط بین 2 تا 3 برابر بهبود سرعت خواهیم داشت.

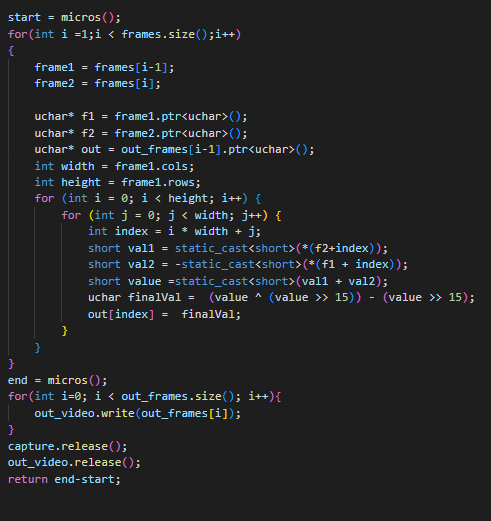
از دلایل نرسیدن به حداکثر speedup ، میتوان به وجود checking در حالت سریال اشاره کرد که چک میکند آیا کاراکتر های مساوی میتوانند بیشتر از 16 تا باشند یا خیر . یک دلیل دیگر ، این است که اگر چه در حالت موازی ما چند عملیات را یک جا با یک دستور انجام میدهیم ، اما برای معادل سازی یک عملیات سریال ، گاهی نیاز به چند عملیات simd داریم که سربار ایجاد میکند. در کل این مسئله به دلیل وابستگی زیاد آن به نتیجه محاسبات مراحل قبل و بعد ، نمیتواند حداکثر speedup آرمانی را بگیرد.

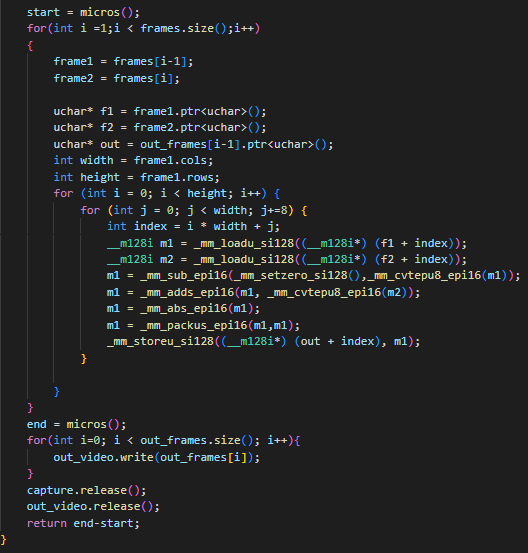
* سوال چهارم :

عملیات کلی به این صورت انجام می شود که ابتدا تمام فریم های ویدیو خوانده می شوند و پس از تبدیل شدن به تصاویر gray در یک وکتور ذخیره می شوند و سپس مقدار مثبت تفاوت هر فریم با فریم قبلی محاسبه می شود.



در اینجا نیز مانند سوال یک ابتدا داده ها را به صورت short یعنی پیکسل های 2 بایتی تبدیل می کنیم و سپس ابتد فریم اول را منفی می کنیم و سپس آنها را با هم جمع می کنیم. برای حساب کردن قدرمطلق می توان از دستور آماده std::abs در کتابخانه cmath استفاده کرد ولی این کار کد را کند می کرد به همین دلیل برای حساب کردن قدر مطلق ابتدا 15 بار به راست شیف می دهیم اگر عدد منفی باشد همه مقادیر بیت ها برابر 1 می شوند و عملا xor ما عملیات not را انجام می دهد و سپس عملا داریم داده را به علاوه یک می کنیم. کلا این عملیات معادل تبدیل در دستگاه 2’s complement است که مقدار مثبت عدد را به ما می دهد. اگر عدد مثبت باشد نیز عملیات xor و منها هیچ تغییری نمی دهند و خود عدد نوشته می شود. همه عملیات ها به صورت saturate انجام می شوند.

در حالت موازی به این شکل عملیات انجام می شود که دوباره داده ها را به صورت short مانند سوال 1 وارد می کنیم و پس از منفی کردن فریم اول آن ها را از هم کم می کنیم. تنها تفاوت این است که این جا از عملیات \_mm\_abs\_epi16 استفاده می کنیم چرا که در این حالت سریع تر از وقتی است که عملیات abs که در قسمت سریال پیاده سازی شده بود را موازی کنیم.

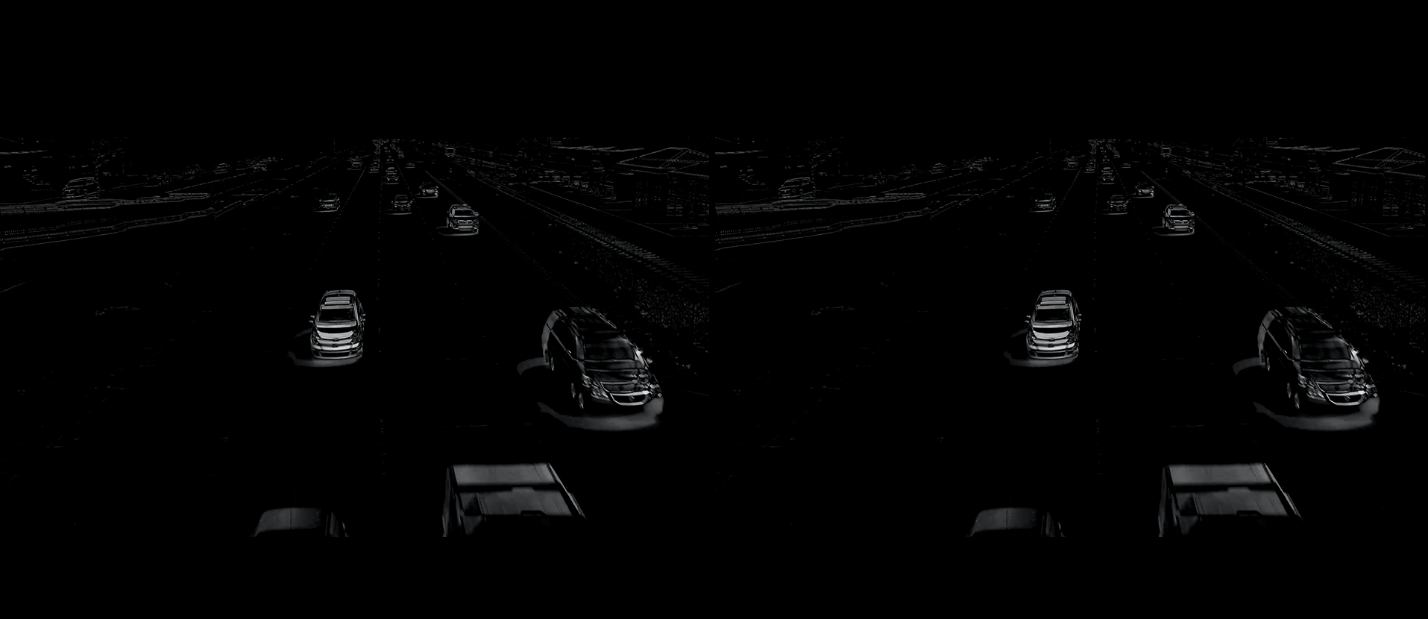


عملیات های تبدیل از 16 بیتی به 8 و برعکس نیز مانند سوال 1 هستند و توضیحات آن ها در همان بخش آورده شده است.

مقدار speedup:

این مسئله نیز بسیار برای موازی سازی خوش تعریف است به این صورت که هر دفعه برای هر دو فریم ما 8 پیکسل از آنها را میخوانیم و پردازش میکنیم ولی به همان دلایل گفته شده در سوال یک امکان رسیدن به حداکثر speedup وجود ندارد.

خروجی:

عکس های مربوط به دو فریم یکسان از ویدیو خروجی. عکس سمت چپ مربوط به حالت سریال و عکس سمت راست مربوط به حالت موازی است