****«به نام خدا»

**طراحی سیستم های نهفته مبتنی بر هسته**

**تکلیف کامپیوتری شماره 2:** **CUDA**

پردیس دانشکده­های فنی دانشگاه تهران

دانشکده مهندسي برق و کامپيوتر

استاد: دکتر احمد شعبانی

نیم­سال دوم سال تحصیلی 1401-1400

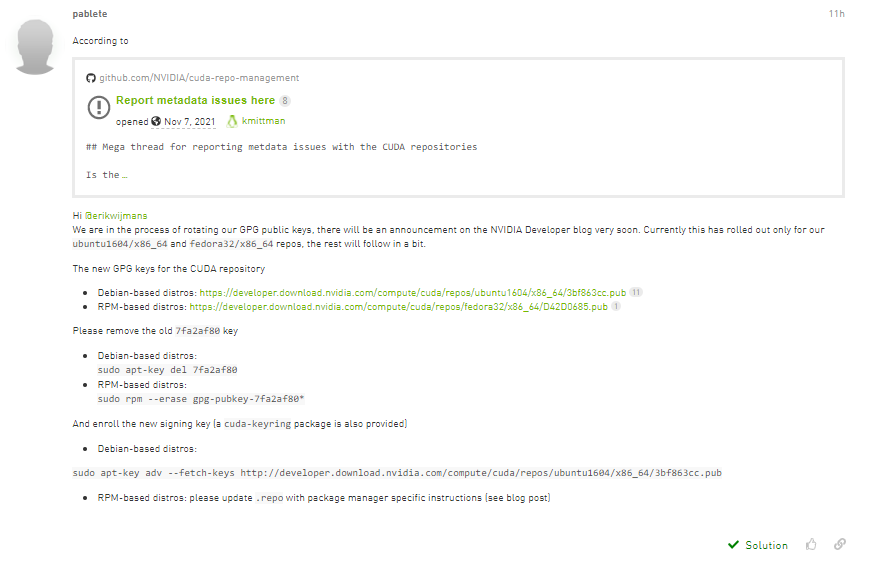
نگارش: علی ایمانقلی 810197692

**مراحل نصب:**

در ابتدا طبق دستورات موجود در صورت پروژه اقدام به نصب شد، ولیکن به علت تغییراتی که در رپوزیتوری مرتبط با NVIDIA به وجود آمده بود در اجرای دستور زیر، ارور ذیل نمایش داده می شد:  



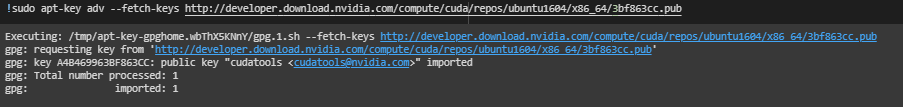

این ارور در هنگام اجرای دستور apt-get update به وجود می آید.  
برای رفع این مشکل طبق بررسی که در اینترنت انجام شد، راهکار زیر یافت شد.



در نتیجه با اجرای دستورات زیر، ارور مربوط به رپوزیتوری برطرف شد:

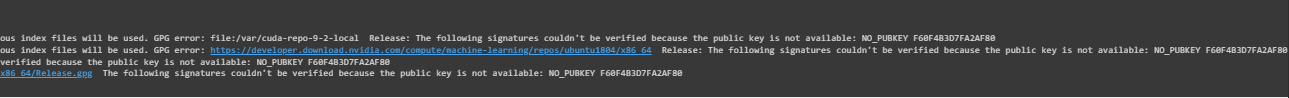
sudo apt-key del 7fa2af80  
sudo apt-key adv --fetch-keys http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86\_64/3bf863cc.pub

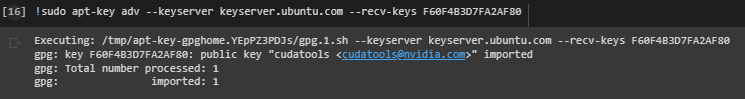




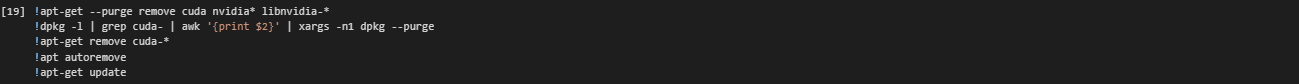
پس از اجرای دستورات بالا ارور زیر نمایش به وجود می آید که توسط اجرای دستور زیر برطرف می شود:

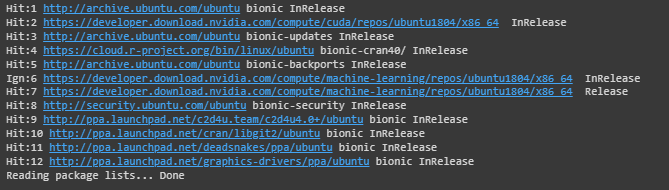
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys F60F4B3D7FA2AF80

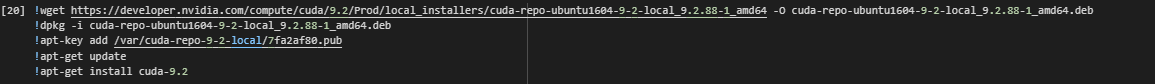


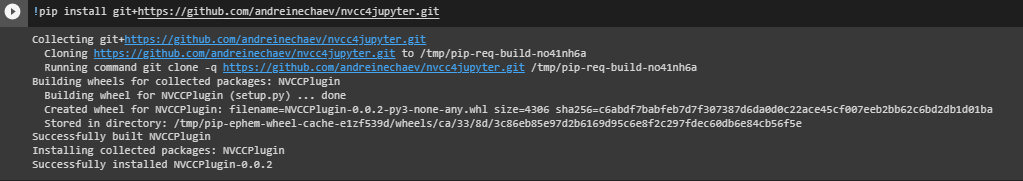


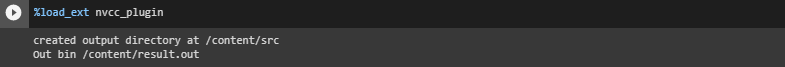
و در نهایت دستور apt-get update بدون ارور اجرا می شود.  
در ادامه تصاویر مربوط به اجرای کامل مراحل نصب آورده ده است:





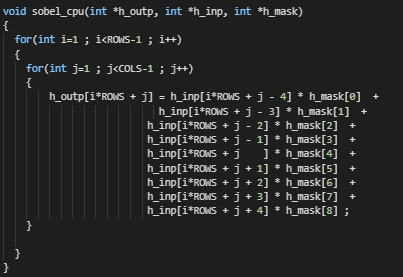






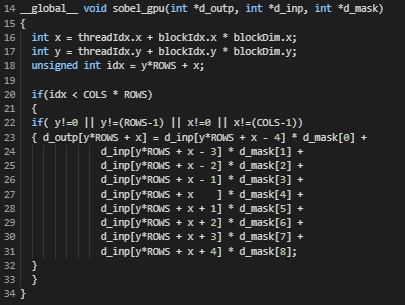
**کد قسمت:CPU(serial)**

در این قسمت کد مربوط به اجرای الگوریتم سوبل توسط CPU آورده شده است.  
با اجرای این تابع حلقه‌های for تو در تو در کل به تعداد 227 \* 227 مرتبه اجرا می شوند و کانولوشن بین تصویر و ماسک عمودی سوبل را انجام می دهند.  
ورودی h\_inp ارایه ی تک بعدی است که پیکسل های تصویر را در خود نگه می دارد و در main تعریف شده است.  
با وجود اینکه تصویر 2 بعدی می باشد، ولیکن مقادیر آن به صورت تک بعدی و خطی ذخیره شده اند و به همین علت در هنگام دستری به پیکسل های تصویر و ماسک سوبل خانه ی مورد نظر را با انجام عملیات بر روی x و y تعیین می نماییم.



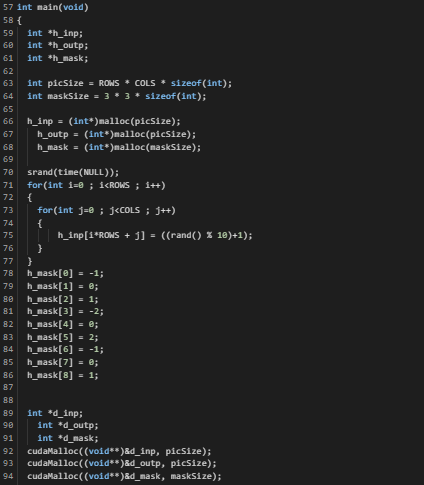
**کد قسمت:GPU(parallel)**

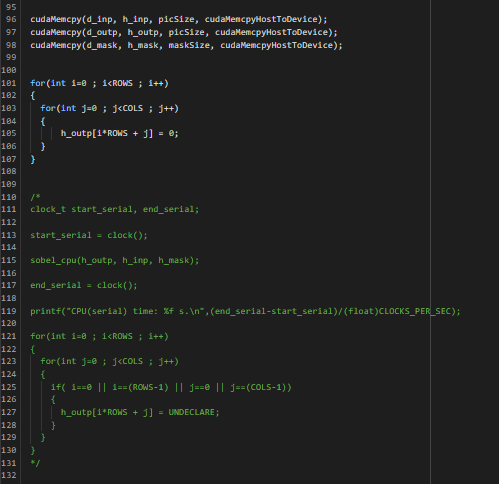
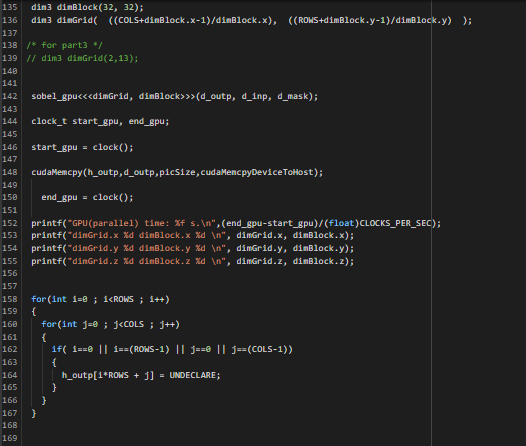
در ای قسمت کد مربوط به الگوریتم سوبل در سمت GPU قرار دارد.  
در ابتدای کد اندیس المان را به وسیله‌ی اطلاعات thread مورد نظر می یابیم و سپس با بررسی شرط مشخص میکنیم که این اندیس از مقدار 227\*227 بیشتر نباشد تا محاسبات به اشتباه انجام نشود و سپس بررسی می کنیم که ماسک سوبل در حاشیه‌ی تصویر قرار نگیرد. زیر در این صورت باید ضرب هایی در مقادیر نامعلوم که جزء پیکسل های تصاویر نیستند انجام شود.



**کد قسمت:main**

در قطعه کد زیر در ابتدا ماسک عمودی سوبل و آرایه‌ی مربوط به تصویر را تعریف می نماییم، برای مدریت راحت تر این آرایه ها با اینکه مفهوم دوبعدی دارند، آن ها را به صورت تک بعدی تعریف می نماییم.  
سپس آرایه ی مورد نظر را به وسیله‌ی دستور srand با مقدار random پر می نماییم.  
و در ادامه حافظه ها را هم به پارمتر های Device و هم به پارامتر های Host تخصیص می دهیم.  
در میانه های کد نیز انتقال حافظه های از Host به Device و از Device به Host قرار دارد.  
در انتها فراخوانی عملیات sobel برای CPU و GPU قرار دارد، که قسمت CPU آن کامنت شده است.  
و در اخر پس از انتقال نتیجه از Device به Host ، حافظه های تخصیص داده شده را حذف می نماییم.



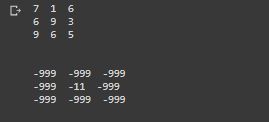
  


در این حلقه‌ی for حاشیه های تصویر را که الگوریتم سوبل آن ها را تعیین نکرده است با مقدار UNDECLARE پر می کنیم.  
مقدار UNDECLARE را به دلخواه برابر با -999 در نظر گرفته شده است.

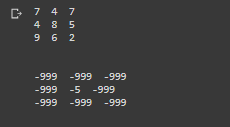


**صحت عملیات:**

برای بررسی اینکه الگوریتم سوبل به درستی اجرای می شود، تست زیر را انجام شد:  
برای راحتی بررسی، ابعاد تصویر را 3\*3 در نظر می گیریم:  
شکل زیر مربوط به اجرای عملیات سوبل توسط CPU می باشد:



شکل زیر مربوط به اجرای عملیات سوبل توسط CPU می باشد:



همانطور که مشخص است الگوریتم سوبل به درستی اجرا می شود.

**بخش1:**

نتیجه اجرای الگوریتم سوبل در حالت CPU:  
\*کد بخش CPU به همراه توضیح، بالاتر آورده شده است.



**بخش2:**

نتیجه اجرای الگوریتم سوبل در حالت GPU:  
\*کد بخش GPU به همراه توضیح، بالاتر آورده شده است.



**بخش3:**









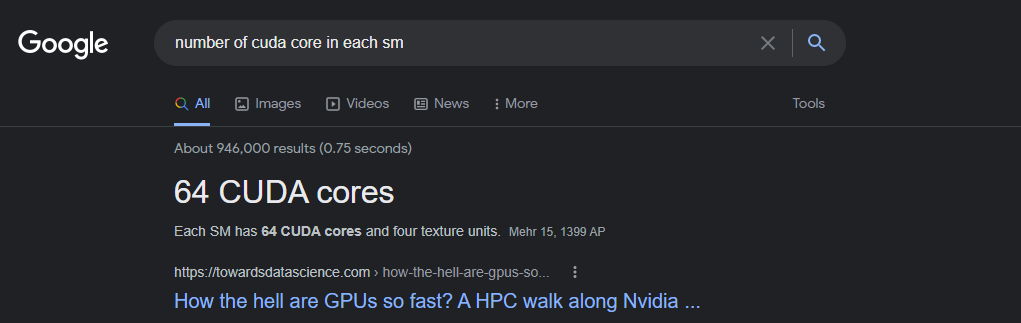


در جدول زیر تمامی مشخصات بالا به همراه speed up حاصل نسبت به اجرای سریال آورده شده است.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Speed up | time | block size | Grid size | threads per block |
| 8.6049 | 0.000081s | (32,32,1) | (8,8,1) | 1024 |
| 9.0519 | 0.000077s | (32,16,1) | (8,15,1) | 512 |
| 8.9359 | 0.000078s | (16,16,1) | (15,15,1) | 256 |
| 8.7125 | 0.000080s | (8,16,1) | (29,15,1) | 128 |
| 7.2604 | 0.000096s | (8,4,1) | (29,57,1) | 32 |

**بخش4:**

برای رسید به بیشترین تسریع ممکن باید تعداد thread در هر block را به نحوی تنظیم نماییم که،  
از یک طرف CUDA core ای بی کار نماند و از طرف دیگری تعداد thread ها به قدری نباشد که load فراتر از حد بر روی CUDA core قرار بگیرد و برای اجرای thread های بایند شده به CUDA core نیازمند به توقف بشویم تا تمامی دستورات توسط CUDA core اجرا شود.  
بدین منظور بهترین حالت به نحوی است که حد واسط دو حالت بالا قرار بگیرد. حالت مناسب می تواند به نحوی باشد که چینش thread ها مضربی از warp باشد.  
همانطور که می دانیم هر block به یک SMبایند می شود و هر SM به تعداد 64 CUDA core دارا می باشد.



از طرف دیگری می دانیم که هر warp تعداد 32 عدد thread را شامل می شود؛  
بنابراین می توان block size را به صورت 32\*64 در نظر گرفت.  
حال برای grid size به نحوی تعیین می نماییم که تعداد thread های مازاد برای تعداد element کمینه شود که با درنظر گرفتن ابعاد 2\*13 برای grid می توان به این امر دست یافت.  
در شکل زیر نتیجه‌ی اجرا به ازای ابعاد بالا شده نمایش داده شده است:



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Speed up | time | block size | Grid size | threads per block |
| 9.9571 | 0.000070s | (32,64,1) | (2,13,1) | 32\*64=2048 |

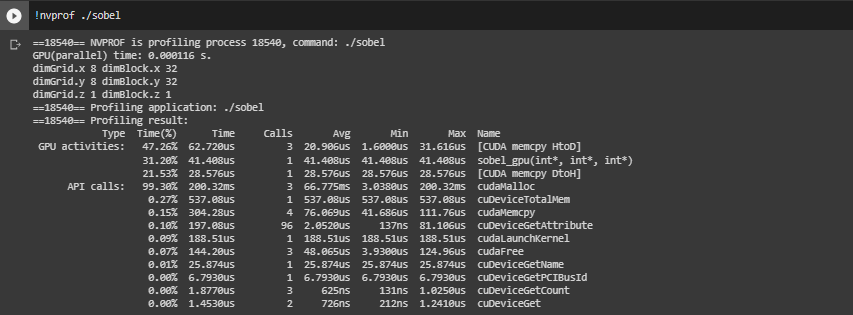
Speed up حاصل از تمامی Speed up های بدست آمده در بخش 3 بالاتر است.

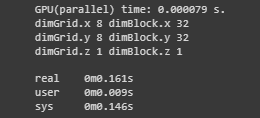
**بخش5:**

\* در بخش GPU activities بیشترین زمان صرف انتقال داده از Host به Device و بلعکس شده است. (حدود %68.79 درصد از کل زمان صرف شده توسط Device)  
و همانطور که آموختیم این بخش bottleneck برنامه می باشد.

\* در قسمت API calls نیز بیشترین زمان صرف شده مربوط به allocate کردن حافظه می باشد که حدود 99.30% از کل زمان صرف شده توسط API calls می باشد.

\* برای بهبود شرایط فوق می توان از تکنیک هایی برای کاهش دفعات انتقال داده از Host به Device و بلعکس استفاده نمود به طور مثال همزمان که داده ها در حال ارسال می باشند، GPU پردازش انجام دهد.  
\* می توان برنامه را به صورت طراحی کرد که از یک مرتبه kernel از سمت Host، lunch شود و در مراتب بعدی kernel از سمت Device، lunch شود.  
\* راه دیگر ذخیره سازی تصویر به نحوی است که چینش پیکسل های مجاور به نحوی باشد که دسترسی به آن ها با سرعت بالاتری انجام شود.





**با تشکر از توجه شما**