«به نام خدا»

درس

Core-Based Embedded System Design

تکلیف کامپیوتری دوم

پردیس دانشکده‌های فنی دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

دکتر احمد شعبانی

نیم‌سال دوم سال تحصیلی 1400

نگارش: **ابوالفضل سجادی** (sajady.abolfazl@gmail.com)

شرح تمرین

هدف از این تمرین آشنایی با برنامه نویسی Cuda و اجرای آن بر روی سرور Google colab است. پس از آشنایی با نحوه نوشتن برنامه به صورت Cuda تمرین زیر را انجام دهید و بر روی Google Colab اجرا کنید. زمان اجرای برنامه سریال و برنامه Cuda را به دست آورده و Speed up حاصل را محاسبه کنید.

الگوریتم sobel

کد الگوریتم سوبل متد سوبل لبه ها را با استفاده از تخمین زدن مشتق پیدا می کند، که لبه ها را در آن نقاطی بر می گرداند که گرادیان تصویر I ، maxاست. در فیلتر سوبل دو ماسک به صورت زیر وجود دارد:

$$G_y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

ماسک سوبل عمودی

$$G_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

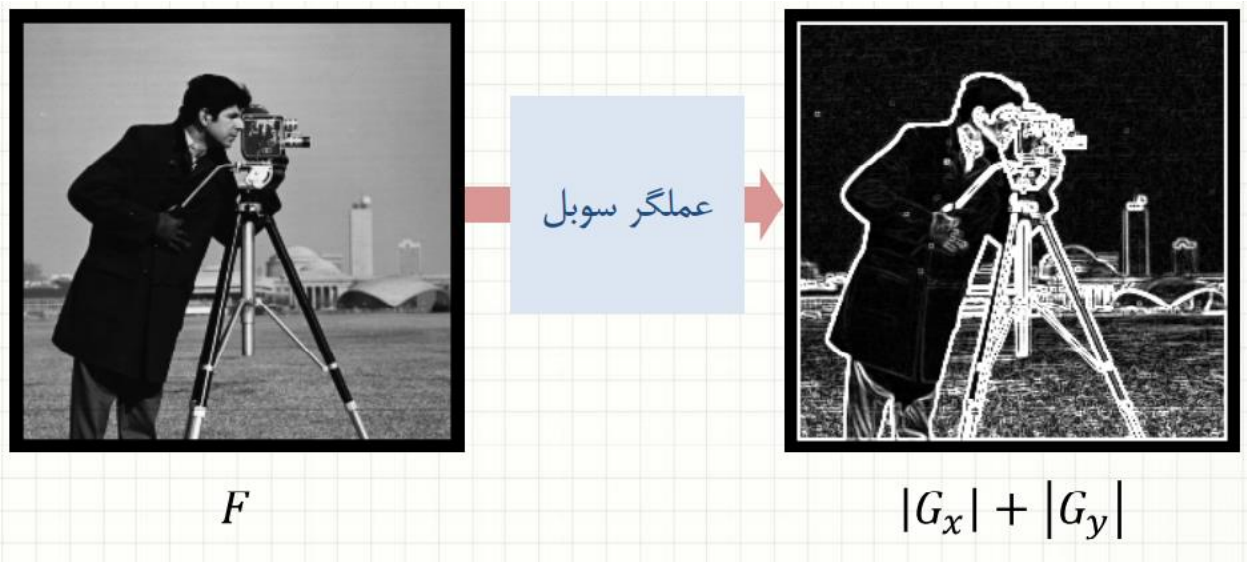
ماسک سوبل افقی

ماسک سوبل افقی بیشتر لبه های افقی را مشخص میکند و ماسک سوبل عمودی،لبه های عمودی را مشخص میکند.

برای مشخص شدن کلیه لبه ها:

اگر G_x و G_y تصاویر فیلتر شده به وسیله ماسک افقی و عمودی باشند، آنگاه تصویر $\sqrt{[G_x]^2+[G_y]^2}$ لبه های تصویر را بهتر نشان میدهد. روال فوق به عملگر یا الگوریتم سوبل موسوم است.

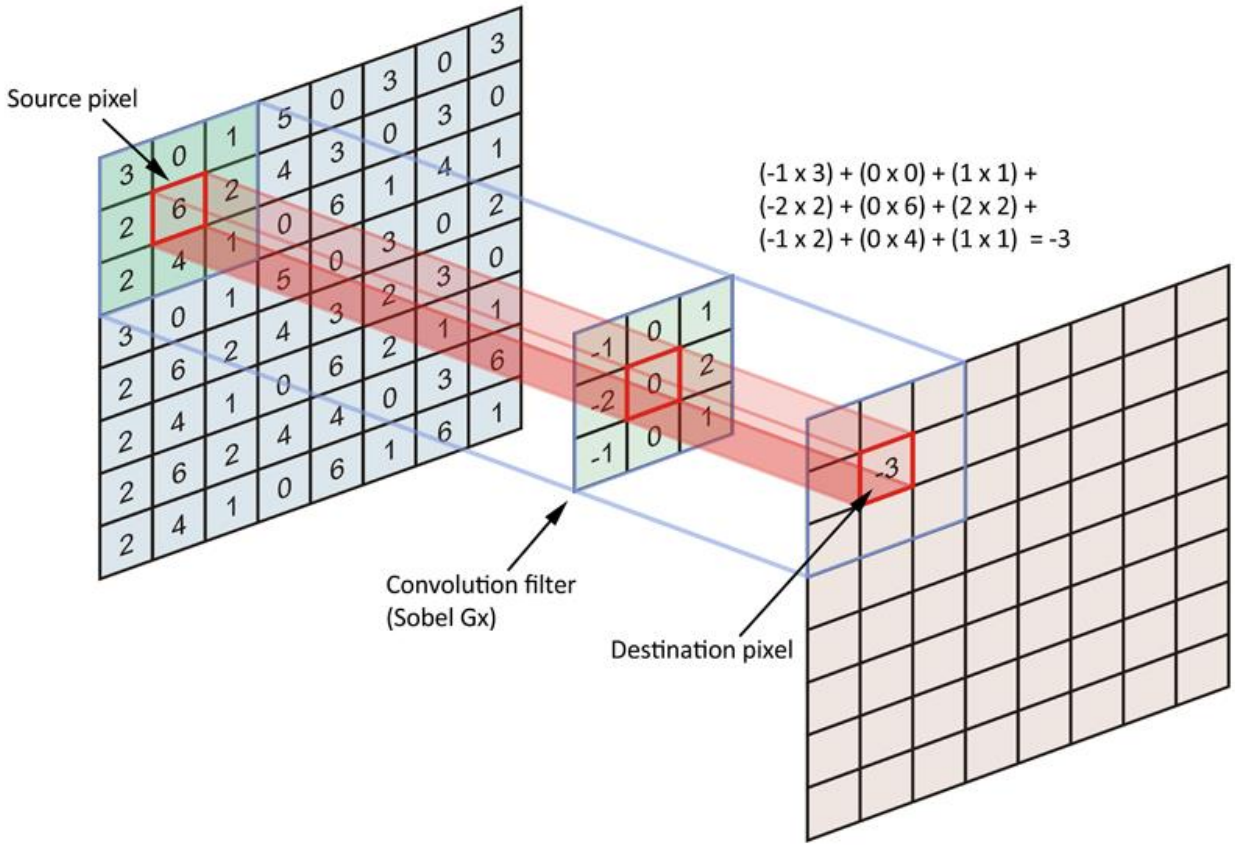
در عمل، به منظور کاهش هزینه محاسبات، به جای $\sqrt{[G_x]^2+[G_y]^2}$ میتوان از تقریب $[G_x] + [G_y]$ استفاده میشود. توجه شود که نتیجه این دو فرمول تقریبا یکسان است ولی فرمول دوم با هزینه کمتری قابل محاسبه می باشد.



تابعی بنویسید که فیلتر دو بعدی ای (3*3) (فیلتر سوبل عمودی) را روی یک تصویر دو بعدی (227*227) حرکت دهد و در هر مرحله به صورت یک به یک مقادیر آن را در هم ضرب کند و با هم جمع کند و در خروجی قرار دهد. در واقع این تابع مانند کانولوشن بر روی تصویر های دو بعدی است.

در این سایت (https://setosa.io/ev/image-kernels/) میتوانید به صورت عملی نحوه عملکرد این فیلترها را روی تصویر شبیه سازی نمایید.

مقادیر هر پیکسل در تصویر را به صورت رندم و مقادیر کرنل (فیلتر) را مانند تصویر زیر تعیین کنید.



1- این تابع را به صورت سریال اجرا و زمان آن را محاسبه کنید. (برای اجرا تابع را به صورت معمول فراخوانی کنید.) (10)

2- برنامه خود را با Cuda بنویسید و اجرا کنید، زمان اجرا را محاسبه نمایید.(20)

3- برای تعداد ترد در هر بلاک 1024، 512، 256، 128، 32 ، تعداد Grid را بدست آوردید و زمان اجرا هر کدام را گزارش کنید. (20)

4- درمورد تعداد threadها بحث کنید و بیشترین تسریع ممکن را بدست آورید. (20)

5- با دستور nvprof پروفایل برنامه خود را بگیرید و درمورد زمان اجرای هر بخش کد بحث کنید(چه بخشی بیشترین زمان را به خود اختصاص میدهد؟) آیا میتوانید با تغییراتی این زمان را کاهش دهید؟ (20)

10 نمره مربوط به نگارش میباشد.

برای به دست آوردن زمان اجرا میتوانید از دستورات زیر استفاده کنید:

```
clock_t start, end;
start = clock();
```

```
//your serial or cuda program
End = clock();
printf("Serial/Cuda PI calculated in : %f s.\n",(end-start)/(float)CLOCKS_PER_SEC);
```

نحوه نصب:

وارد سایت

https://colab.research.google.com/notebooks/welcome.ipynb#recent=true

شوید و Runtimeرا روی GPU تنظیم نمایید.

ابتدا موارد قبلی را پاک کنید:

```
!apt-get --purge remove cuda nvidia* libnvidia-*
!dpkg -l | grep cuda- | awk '{print $2}' | xargs -n1 dpkg --purge
!apt-get remove cuda-*
!apt autoremove
!apt-get update
```

در مرحله بعد کودا و کامپایلر جدید را نصب کنید:

```
!wget https://developer.nvidia.com/compute/cuda/9.2/Prod/local_installers/
cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64 -O cuda-repo-ubuntu1604-9-2-
local_9.2.88-1_amd64.deb
!dpkg -i cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64.deb
!apt-key add /var/cuda-repo-9-2-local/7fa2af80.pub
!apt-get update
!apt-get install cuda-9.2
```

نصب extention مورد نیاز NVCC در نوتبوک:

```
!pip install git+https://github.com/andreinechaev/nvcc4jupyter.git
```

بررسی نصب موفق

```
!nvcc --version
```

اجرای extention:

```
%load_ext nvcc_plugin
```

در خط اول بالای کدتان %%cu را اضافه کنید و بعد اجرا نمایید:

مثال:

```
%%cu
#include "stdio.h"
#define N 10
//kernel definition
__global__ void add(int *a, int *b, int *c)
{
```

```
int tID = blockIdx.x;
if (tID < N)
{
c[tID] = a[tID] + b[tID];
}
}
int main()
{
int a[N], b[N], c[N];
int *dev_a, *dev_b, *dev_c;
//malloc device global memory
cudaMalloc((void **) &dev_a, N*sizeof(int));
cudaMalloc((void **) &dev_b, N*sizeof(int));
cudaMalloc((void **) &dev_c, N*sizeof(int));
// Fill Arrays
for (int i = 0; i < N; i++)
{
a[i] = i,
b[i] = 1;
}
// transfer data from host to device
cudaMemcpy(dev_a, a, N*sizeof(int), cudaMemcpyHostToDevice);
cudaMemcpy(dev_b, b, N*sizeof(int), cudaMemcpyHostToDevice);
// perform computation on the device here
add<<<N,1>>>(dev_a, dev_b, dev_c);
cudaMemcpy(c, dev_c, N*sizeof(int), cudaMemcpyDeviceToHost);
for (int i = 0; i < N; i++)
{
printf("%d + %d = %d\n", a[i], b[i], c[i]);
}
cudaFree(dev_a);
cudaFree(dev_b);
cudaFree(dev_c);
return 0;
}
```

نحوه پروفایل گیری:

در ابتدا با اضافه کردن دستور زیر به خط اول کدتان آن را با نام دلخواه ذخیره نمایید:

```
%%writefile name.cu
```

سپس با دستور زیر فایل ساخته شده را کامپایل نمایید:

```
!nvcc /content/name.cu -o name -Wno-deprecated-gpu-targets
```

با استفاده از دستور زیر پروفایل گیری کنید:

```
!nvprof ./name
```

همچنین میتوانید با استفاده از دستور زیر نیز زمان اجرا را برآورد نمایید.

```
!time ./name
```

با آرزوی موفقیت و سلامتی برای شما