



به نام خدا



دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده مهندسی برق و کامپیوتر

سیستم های نهفته مبتنی بر هسته

تکلیف کامپیوتری ۲

محمد تقی زاده گیوری

۸۱۰۱۹۸۳۷۳

بهار ۱۴۰۲

خروجی مراحل نصب nvcc

خروجی نصب cuda-9-2:

```
✓ 15m ▶ | 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
| 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

Setting up libcheese-gtk25:amd64 (3.34.0-1ubuntu1) ...
Setting up libdataserver-1.2-24:amd64 (3.36.5-0ubuntu1) ...
Setting up libecal-2.0-1:amd64 (3.36.5-0ubuntu1) ...
Setting up libebook-contacts-1.2-3:amd64 (3.36.5-0ubuntu1) ...
Setting up gnome-control-center (1:3.36.5-0ubuntu4) ...
Setting up libdataserverui-1.2-2:amd64 (3.36.5-0ubuntu1) ...
Setting up libebook-1.2-10:amd64 (3.36.5-0ubuntu1) ...
Setting up libedata-cal-2.0-1:amd64 (3.36.5-0ubuntu1) ...
Setting up libedata-book-1.2-26:amd64 (3.36.5-0ubuntu1) ...
Setting up libebook-1.2-20:amd64 (3.36.5-0ubuntu1) ...
Setting up evolution-data-server (3.36.5-0ubuntu1) ...
Setting up gnome-shell (3.36.0-0ubuntu0.20.04.2) ...
Setting up screen-resolution-extra (0.18build1) ...
Setting up ubuntu-session (3.36.0-2ubuntu1) ...
Setting up gdm3 (3.36.3-0ubuntu0.20.04.4) ...

Creating config file /etc/gdm3/greeter.dconf-defaults with new version
(udevadm settle could not determine current state)
✓ 15m 28s completed at 14:55

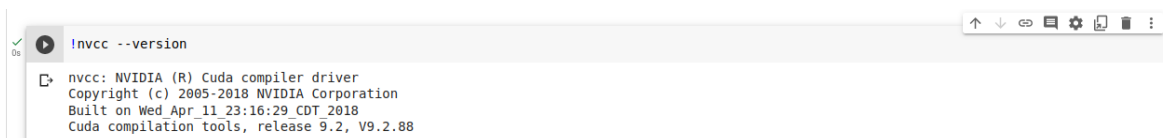
Processing triggers for sgml-base (1.29.1) ...
Setting up docbook-xml (4.5-9) ...
Processing triggers for dictionaries-common (1.28.1) ...
aspell-autobuildhash: processing: en [en-common].
aspell-autobuildhash: processing: en [en-variant 0].
aspell-autobuildhash: processing: en [en-variant 1].
aspell-autobuildhash: processing: en [en-variant 2].
aspell-autobuildhash: processing: en [en-w accents-only].
aspell-autobuildhash: processing: en [en-wo accents-only].
aspell-autobuildhash: processing: en [en-AU-variant 0].
aspell-autobuildhash: processing: en [en-AU-variant 1].
aspell-autobuildhash: processing: en [en-AU-w accents-only].
aspell-autobuildhash: processing: en [en-AU-wo accents-only].
aspell-autobuildhash: processing: en [en-CA-variant 0].
aspell-autobuildhash: processing: en [en-CA-variant 1].
aspell-autobuildhash: processing: en [en-CA-w accents-only].
aspell-autobuildhash: processing: en [en-CA-wo accents-only].
aspell-autobuildhash: processing: en [en-GB-ise-w accents-only].
aspell-autobuildhash: processing: en [en-GB-ise-wo accents-only].
aspell-autobuildhash: processing: en [en-GB-ize-w accents-only].
aspell-autobuildhash: processing: en [en-GB-ize-wo accents-only].
aspell-autobuildhash: processing: en [en-GB-variant 0].
aspell-autobuildhash: processing: en [en-GB-variant 1].
aspell-autobuildhash: processing: en [en-US-w accents-only].
aspell-autobuildhash: processing: en [en-US-wo accents-only].
Processing triggers for rygel (0.38.3-1ubuntu1) ...
Processing triggers for libc-bin (2.31-0ubuntu9.9) ...
Processing triggers for dbus (1.12.16-2ubuntu2.3) ...
Processing triggers for systemd (245.4-4ubuntu3.21) ...
Processing triggers for sgml-base (1.29.1) ...
```

خروجی نصب افزونه مورد نیاز برای نوشتن کد های CUDA:

```
✓ 6s [9] | pip install git+https://github.com/andreinechaev/nvcc4jupyter.git

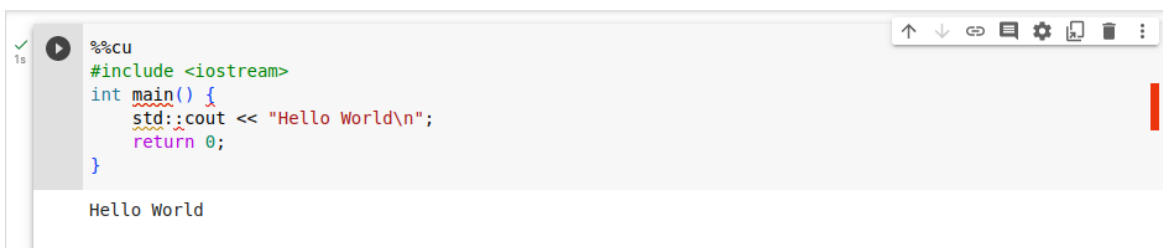
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting git+https://github.com/andreinechaev/nvcc4jupyter.git
  Cloning https://github.com/andreinechaev/nvcc4jupyter.git to /tmp/pip-req-build-d_se4exr
  Running command git clone --filter=blob:none --quiet https://github.com/andreinechaev/nvcc4jupyter.git /tmp/pip-req-build-d_se4exr
  Resolved https://github.com/andreinechaev/nvcc4jupyter.git to commit aac710a35f52bb78ab34d2e52517237941399eff
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: NVCCPlugin
  Building wheel for NVCCPlugin (setup.py) ... done
  Created wheel for NVCCPlugin: filename=NVCCPlugin-0.0.2-py3-none-any.whl size=4287 sha256=0ed3174b46290f739f71d03c34b3c780f50f4b0377622fef
  Stored in directory: /tmp/pip-ephem-wheel-cache-km8skkez/wheels/a8/b9/18/23f8ef71ceb0f63297dd1903aedd067e6243a68ea756d6feea
Successfully built NVCCPlugin
Installing collected packages: NVCCPlugin
Successfully installed NVCCPlugin-0.0.2
```

خروجی ورژن nvcc نصب شده جهت بررسی صحت نصب شدن nvcc:



```
nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2018 NVIDIA Corporation
Built on Wed Apr 11 23:16:29 CDT 2018
Cuda compilation tools, release 9.2, V9.2.88
```

خروجی اجرا کد نمونه (برنامه Hello World) بر روی nvcc نصب شده:



```
%cu
#include <iostream>
int main() {
    std::cout << "Hello World\n";
    return 0;
}

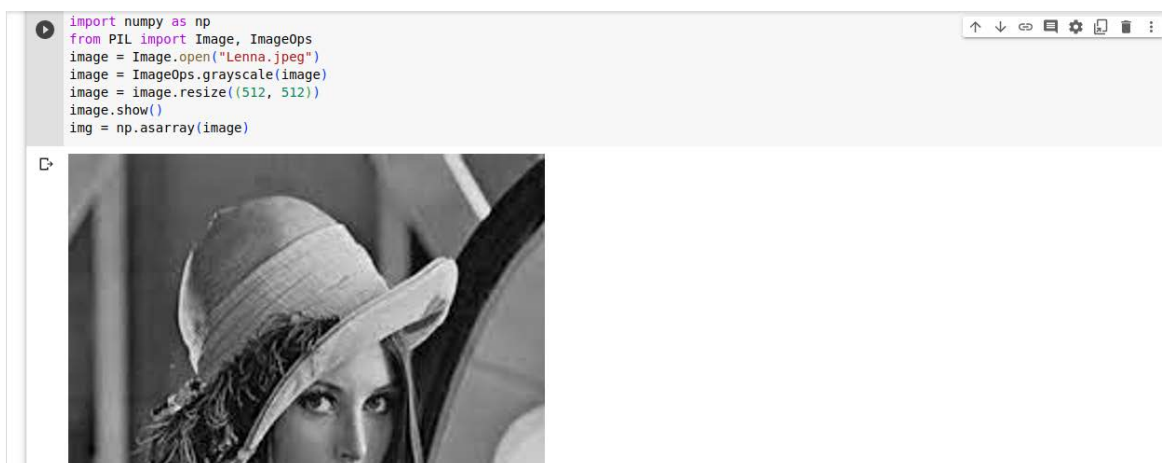
Hello World
```

آپلود کردن عکس بر روی Google Colab جهت تست فیلتر Sharpening:

```
[1] from google.colab import files
image = files.upload()

Browse... Lenna.jpeg
Lenna.jpeg(image/jpeg) - 7301 bytes, last modified: n/a - 100% done
Saving Lenna.jpeg to Lenna.jpeg
```

خواندن عکس ورودی، Gray Scale کردن و تغییر سایز آن به $512 * 512$:



مقداردهی تصویر ورودی (عکس Lenna) در تابع main:

```
int main() {
    unsigned char image[num_of_rows][num_of_columns] =
    { { 161, 161, 161, 161, 161, 160, 160, 160, 160, 159, 159, 159, 159, 159, 159, 159, 158, 159, 161, 163, 162, 160, 158,
      { 161, 161, 161, 161, 161, 160, 160, 160, 160, 159, 159, 159, 159, 159, 159, 159, 158, 159, 161, 163, 162, 160, 158,
      { 160, 160, 160, 160, 160, 160, 160, 159, 159, 159, 159, 158, 158, 158, 158, 158, 159, 161, 162, 161, 160, 159,
      { 160, 160, 160, 160, 160, 160, 160, 159, 159, 159, 159, 158, 158, 158, 158, 158, 158, 159, 161, 162, 161, 160, 159,
      { 160, 160, 159, 159, 159, 159, 159, 159, 159, 158, 158, 158, 158, 157, 157, 157, 158, 160, 161, 160, 159, 158,
      { 159, 159, 158, 158, 158, 158, 158, 158, 158, 157, 157, 157, 157, 156, 156, 156, 157, 159, 160, 159, 158, 157,
      { 159, 159, 158, 158, 158, 158, 158, 158, 157, 157, 157, 157, 156, 156, 156, 157, 159, 160, 159, 158, 157,
      { 158, 158, 157, 157, 157, 157, 157, 157, 156, 156, 156, 156, 155, 155, 155, 155, 156, 158, 159, 158, 157, 156,
      { 157, 157, 157, 157, 157, 157, 156, 156, 156, 156, 155, 155, 155, 155, 155, 155, 157, 158, 158, 157, 156,
      { 156, 156, 156, 156, 156, 156, 155, 155, 155, 155, 154, 154, 154, 154, 154, 154, 156, 157, 157, 156, 155,
      { 156, 156, 156, 156, 156, 156, 155, 155, 155, 155, 154, 154, 154, 154, 154, 154, 156, 157, 156, 155, 155,
      { 156, 156, 155, 155, 155, 155, 155, 155, 154, 154, 154, 154, 153, 153, 153, 153, 155, 156, 156, 155, 154,
      { 156, 156, 155, 155, 155, 155, 155, 155, 154, 154, 154, 154, 153, 153, 153, 153, 154, 155, 156, 155, 154,
      { 155, 155, 155, 155, 155, 155, 154, 154, 154, 154, 153, 153, 153, 153, 153, 153, 154, 155, 155, 154, 154,
```

فیلتر Sharp_Given_Image:

در این فیلتر ابتدا مقادیر کرنل فیلتر، مطابق با مقادیر داده شده در صورت پروژه، تعیین می شود:

```
void Sharp_Given_Image(int img[][512], int num_of_rows, int num_of_columns) {  
    int Sharp_Filter[3][3] = {{0, -1, 0}, {-1, 5, -1}, {0, -1, 0}};  
    for(int row = 0; row < num_of_rows - 3; row++) {  
        for(int column = 0; column < num_of_columns - 3; column++) {  
            int temp = 0;  
            for(int i = 0; i < 3; i++)  
                for(int j = 0; j < 3; j++)  
                    temp = temp + (img[row + i][column + j] * Sharp_Filter[i][j]);  
            img[row][column] = temp;  
        }  
    }  
}
```

سپس طی دو حلقه تو در تو، تصویر ورودی اسکن می شود:

```
void Sharp_Given_Image(int img[][512], int num_of_rows, int num_of_columns) {  
    int Sharp_Filter[3][3] = {{0, -1, 0}, {-1, 5, -1}, {0, -1, 0}};  
    for(int row = 0; row < num_of_rows - 3; row++) {  
        for(int column = 0; column < num_of_columns - 3; column++) {  
            int temp = 0;  
            for(int i = 0; i < 3; i++)  
                for(int j = 0; j < 3; j++)  
                    temp = temp + (img[row + i][column + j] * Sharp_Filter[i][j]);  
            img[row][column] = temp;  
        }  
    }  
}
```

به هر پیکسل از تصویر که می رسیم، طی دو حلقه تو در تو، پیکسل های مجاور خوانده شده و در مقادیر متناظر با کرنل sharp، ضرب می شود:

```
void Sharp_Given_Image(int img[][512], int num_of_rows, int num_of_columns) {  
    int Sharp_Filter[3][3] = {{0, -1, 0}, {-1, 5, -1}, {0, -1, 0}};  
    for(int row = 0; row < num_of_rows - 3; row++) {  
        for(int column = 0; column < num_of_columns - 3; column++) {  
            int temp = 0;  
            for(int i = 0; i < 3; i++)  
                for(int j = 0; j < 3; j++)  
                    temp = temp + (img[row + i][column + j] * Sharp_Filter[i][j]);  
            img[row][column] = temp;  
        }  
    }  
}
```

در آخر مجموع ضرب پیکسل های مجاور در کرنل sharp، در پیکسل متناظر تصویر خروجی ذخیره می شود:

```
void Sharp_Given_Image(int img[][512], int num_of_rows, int num_of_columns) {
    int Sharp_Filter[3][3] = {{0, -1, 0}, {-1, 5, -1}, {0, -1, 0}};
    for(int row = 0; row < num_of_rows - 3; row++) {
        for(int column = 0; column < num_of_columns - 3; column++) {
            int temp = 0;
            for(int i = 0; i < 3; i++)
                for(int j = 0; j < 3; j++)
                    temp = temp + (img[row + i][column + j] * Sharp_Filter[i][j]);
            img[row][column] = temp;
        }
    }
}
```

فراخوانی فیلتر Sharp_Given_Image در تابع main و اندازه گیری زمان اجرا سریال:

```
auto start = high_resolution_clock::now();
Sharp_Given_Image(image, num_of_rows, num_of_columns);
auto end = high_resolution_clock::now();

auto execution_time = duration_cast<microseconds>(end - start);
cout << "Execution Time: " << execution_time.count() << " Micro Seconds\n";
```

با استفاده از تابع high_resolution_clock، موجود در کتابخانه (library) chrono.

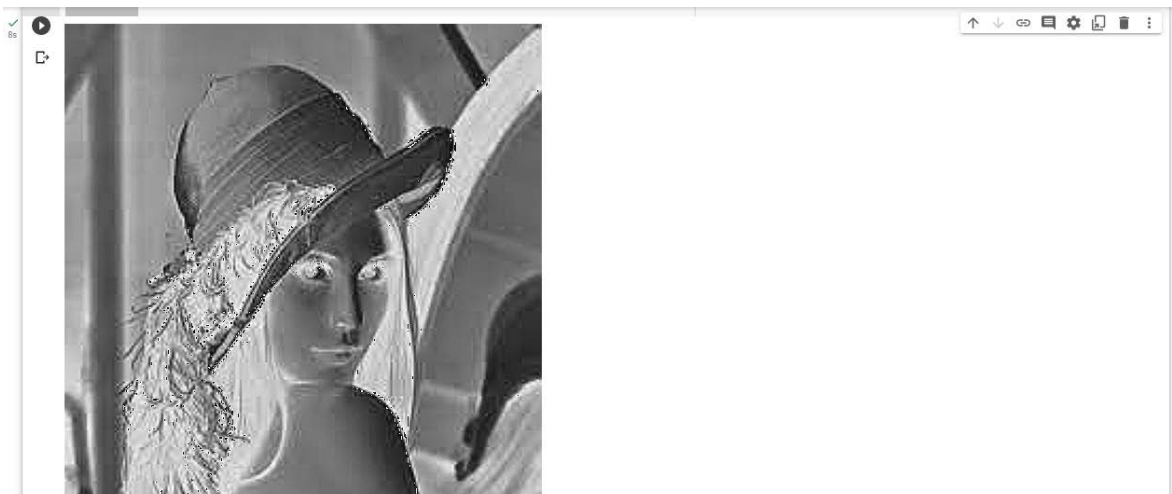
چاپ کردن مقادیر پیکسل های تصویر فیلتر شده:

```
cout << "[";
for(int row = 0; row < num_of_rows; row++){
    cout << "[";
    for(int column = 0; column < num_of_columns; column++){
        cout << image[row][column];
        if(column != (num_of_columns - 1))
            cout << ",";
    }
    if(row != num_of_rows - 1)
        cout << "],\n";
    else
        cout << "]]\n";
}
```

زمان اجرا تابع فیلتر به صورت سریال و مقادیر پیکسل های تصویر فیلتر شده:

```
Execution Time: 8181 Micro Seconds
[[162,162,162,163,159,160,161,162,158,159,160,160,160,160,161,156,158,161,167,164,160,156,158,154,154,153,152,150,150,153,151,154,152,154,153,159,159,159,160,161,157,158,159,160,156,157,157,157,157,157,158,162,163,160,160,161,157,154,154,153,152,150,150,153,150,153,151,153,151,159,160,160,160,160,161,158,159,159,160,157,158,158,158,158,157,158,162,164,161,160,160,157,154,154,153,152,150,149,151,153,155,153,155,154,160,161,161,161,161,162,158,159,160,161,157,158,158,159,159,158,159,163,165,162,161,161,157,154,154,153,152,148,152,153,154,156,154,156,156,161,157,158,158,158,159,160,156,157,158,158,159,155,156,155,156,160,162,159,158,157,159,154,154,153,151,152,154,155,156,158,155,157,157,162,159,160,160,160,160,160,161,158,159,159,159,160,157,158,157,158,162,164,161,160,160,155,156,155,153,150,150,152,153,154,155,157,158,158,159,156,157,157,157,157,157,158,155,156,156,156,157,154,155,154,155,159,161,158,157,157,153,153,151,153,154,153,155,156,157,158,158,159,159,161,158,159,159,159,159,160,157,158,158,158,159,156,157,156,157,161,163,160,159,159,155,154,152,153,153,151,153,154,155,155,160,160,161,159,155,156,156,156,157,158,154,155,156,156,157,153,154,153,155,159,161,157,156,156,151,155,152,153,152,155,156,156,157,157,161,161,162,157,158,158,158,158,159,155,156,157,158,154,155,155,156,156,156,153,158,159,160,158,158,152,155,152,153,152,153,154,158,158,157,161,162,163,155,155,155,155,155,156,153,154,154,155,152,153,153,153,153,153,151,156,157,158,156,154,153,155,152,153,150,158,151,161,158,162,166,162,164,156,156,156,156,156,157,154,155,155,156,153,154,154,154,154,154,152,157,159,155,153,156,153,155,152,152,155,155,155,157,160,162,166,163,165,156,157,157,157,158,154,155,156,157,153,154,154,155,155,155,153,158,160,156,154,157,154,156,152,152,153,153,157,159,162,164,168,164,166,157,153,154,154,154,154,155,156,152,153,154,154,155,151,152,152,150,156,157,157,155,153,151,152,153,151,157,156,159,162,160,163,166,167,168,158,154,155,155,155,155,156,157,153,154,155,155,156,152,153,153,152,153,154,159,156,154,152,153,153,151,156,155,158,161,159,162,165,165,166,154,155,155,155,155,156,152,153,154,155,151,152,152,152,152,152,151,153,155,154,151,155,152,153,153,151,156,155,158,160,163,165,168,168,170,155,154,154,154,154,155,152,153,152,153,149,150,149,154,153,153,152,154,156,156,154,152,151,152,153,151,157,156,159,161,163,165,168,168,170,154,158,157,157,157,158,154,154,158,157,158,158,158,156,156,155,157,159,160,158,158,152,156,152,152,153,153,156,158,160,162,165,165,167,153,157,156,156,156,156,156,155,159,158,157,157,156,161,160,160,159,161,164,160,159,159,154,153,151,151,154,154,157,159,161,163,166,166,168]]
```

نمایش تصویر خروجی sharp شده:



فیلتر Sharp_Given_Image نوشته شده با *CUDA*:

از جایی که تابع فیلتر، توسط چندین thread، اجرا می شود، ابتدا مشخص

می کنیم که thread ای که این تابع را اجرا می کند، مربوط به کدام سطر

و ستون (row و column) است:

```
#define num_of_rows 512
#define num_of_columns 512

__global__ void Sharp_Given_Image(unsigned char* input_image, unsigned char* output_image) {
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int column = blockIdx.x * blockDim.x + threadIdx.x;
    if(row >= num_of_rows - 3 || column >= num_of_columns - 3)
        return;
    int Sharp_Filter[3][3] = {{0, -1, 0}, {-1, 5, -1}, {0, -1, 0}};
    int sharpened_pixel = 0;
    for(int i = 0; i < 3; i++){
        for(int j = 0; j < 3; j++){
            int current_pixel = input_image[row * num_of_rows + column];
            sharpened_pixel = sharpened_pixel + (current_pixel * Sharp_Filter[i][j]);
        }
    }
    output_image[row * num_of_rows + column] = max(0, min(255, sharpened_pixel));
}
```

سپس طی یک حلقه تو در تو، تمامی پیکسل های مجاور، در کرنل متناظر با

فیلتر sharp، ضرب می شود:

```
#define num_of_rows 512
#define num_of_columns 512

__global__ void Sharp_Given_Image(unsigned char* input_image, unsigned char* output_image) {
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int column = blockIdx.x * blockDim.x + threadIdx.x;
    if(row >= num_of_rows - 3 || column >= num_of_columns - 3)
        return;
    int Sharp_Filter[3][3] = {{0, -1, 0}, {-1, 5, -1}, {0, -1, 0}};
    int sharpened_pixel = 0;
    for(int i = 0; i < 3; i++){
        for(int j = 0; j < 3; j++){
            int current_pixel = input_image[row * num_of_rows + column];
            sharpened_pixel = sharpened_pixel + (current_pixel * Sharp_Filter[i][j]);
        }
    }
    output_image[row * num_of_rows + column] = max(0, min(255, sharpened_pixel));
}
```


در آخر، مجموع این ضرب ها، در پیکسل متناظر در تصویر خروجی (output_image)، نوشته و ذخیره می شود.

```
#define num_of_rows 512
#define num_of_columns 512

__global__ void Sharp_Given_Image(unsigned char* input_image, unsigned char* output_image) {
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int column = blockIdx.x * blockDim.x + threadIdx.x;
    if(row >= num_of_rows - 3 || column >= num_of_columns - 3)
        return;
    int Sharp_Filter[3][3] = {{0, -1, 0}, {-1, 5, -1}, {0, -1, 0}};
    int sharpened_pixel = 0;
    for(int i = 0; i < 3; i++){
        for(int j = 0; j < 3; j++){
            int current_pixel = input_image[row * num_of_rows + column];
            sharpened_pixel = sharpened_pixel + (current_pixel * Sharp_Filter[i][j]);
        }
    }
    output_image[row * num_of_rows + column] = max(0, min(255, sharpened_pixel));
}
```

در تابع main:

ابتدا به تعداد $\text{num_of_rows} * \text{num_of_columns}$ (تعداد پیکسل های تصویر) حافظه اختصاص می دهیم (چون هر پیکسل در بازه 0 تا 255 است، پس هر پیکسل را به صورت `unsigned char` در نظر می گرفتیم).

```
unsigned char *d_input, *d_output;

cudaMalloc((void**)&d_input, num_of_rows * num_of_columns * sizeof(unsigned char));
cudaMalloc((void**)&d_output, num_of_rows * num_of_columns * sizeof(unsigned char));

cudaMemcpy(d_input, image, num_of_rows * num_of_columns * sizeof(unsigned char), cudaMemcpyHostToDevice);

auto start = high_resolution_clock::now();
Sharp_Given_Image <<<1, 1>>> (d_input, d_output);
auto end = high_resolution_clock::now();

cudaMemcpy(image, d_output, num_of_rows * num_of_columns * sizeof(unsigned char), cudaMemcpyDeviceToHost);

cudaFree(d_input);
cudaFree(d_output);

auto execution_time = duration_cast<microseconds>(end - start);
cout << "Execution Time: " << execution_time.count() << " Micro Seconds\n";
}
```

سپس تصویر ورودی را با استفاده از `cudaMemcpy` به `device` یا همان GPU منتقل می کنیم:

```
unsigned char *d_input, *d_output;

cudaMalloc((void**)&d_input, num_of_rows * num_of_columns * sizeof(unsigned char));
cudaMalloc((void**)&d_output, num_of_rows * num_of_columns * sizeof(unsigned char));

cudaMemcpy(d_input, image, num_of_rows * num_of_columns * sizeof(unsigned char), cudaMemcpyHostToDevice);

auto start = high_resolution_clock::now();
Sharp_Given_Image <<<1, 1>>> (d_input, d_output);
auto end = high_resolution_clock::now();

cudaMemcpy(image, d_output, num_of_rows * num_of_columns * sizeof(unsigned char), cudaMemcpyDeviceToHost);

cudaFree(d_input);
cudaFree(d_output);

auto execution_time = duration_cast<microseconds>(end - start);
cout << "Execution Time: " << execution_time.count() << " Micro Seconds\n";
}
```

با فراخوانی تابع `Sharp_Given_Image`، تصویر فیلتر شده در `d_output` قرار داده شده و ذخیره می شود:

```
unsigned char *d_input, *d_output;

cudaMalloc((void**)&d_input, num_of_rows * num_of_columns * sizeof(unsigned char));
cudaMalloc((void**)&d_output, num_of_rows * num_of_columns * sizeof(unsigned char));

cudaMemcpy(d_input, image, num_of_rows * num_of_columns * sizeof(unsigned char), cudaMemcpyHostToDevice);

auto start = high_resolution_clock::now();
Sharp_Given_Image <<<1, 1>>> (d_input, d_output);
auto end = high_resolution_clock::now();

cudaMemcpy(image, d_output, num_of_rows * num_of_columns * sizeof(unsigned char), cudaMemcpyDeviceToHost);

cudaFree(d_input);
cudaFree(d_output);

auto execution_time = duration_cast<microseconds>(end - start);
cout << "Execution Time: " << execution_time.count() << " Micro Seconds\n";
}
```

در آخر با استفاده از `cudaMemcpy`، تصویر فیلتر شده به `host` یا همان CPU منتقل شده و در آرایه دوبعدی `image` ذخیره می شود.

```
unsigned char *d_input, *d_output;

cudaMalloc((void**)&d_input, num_of_rows * num_of_columns * sizeof(unsigned char));
cudaMalloc((void**)&d_output, num_of_rows * num_of_columns * sizeof(unsigned char));

cudaMemcpy(d_input, image, num_of_rows * num_of_columns * sizeof(unsigned char), cudaMemcpyHostToDevice);

auto start = high_resolution_clock::now();
Sharp_Given_Image <<<1, 1>>> (d_input, d_output);
auto end = high_resolution_clock::now();

cudaMemcpy(image, d_output, num_of_rows * num_of_columns * sizeof(unsigned char), cudaMemcpyDeviceToHost);

cudaFree(d_input);
cudaFree(d_output);

auto execution_time = duration_cast<microseconds>(end - start);
cout << "Execution Time: " << execution_time.count() << " Micro Seconds\n";
}
```

زمان اجرا تابع نوشته شده با *CUDA*:

```
Execution Time: 25 Micro Seconds
```

گام سوم

برای محاسبه تعداد grid، فرض می کنیم که عملیات فیلترینگ مربوط به هر

پیکسل، قرار است توسط یک thread انجام شود. بنابراین به تعداد

$\text{num_of_rows} * \text{num_of_columns}$ عدد، thread نیاز

داریم. پس اگر تعداد هر thread در هر بلاک را تحت عنوان

ThreadPerBlock بدانیم، مطابق با فرمول زیر تعداد

grid مورد نیاز بدست خواهد آمد:

$$\text{grid} = (\text{num_of_rows} * \text{num_of_columns}) / \text{ThreadPerBlock}$$

زمان اجرا برای حالت $\text{ThreadPerBlock} = 1024$:

```
auto start = high_resolution_clock::now();  
Sharp_Given_Image <<<((num_of_rows * num_of_columns) / 1024), 1024>>> (d_input, d_output);  
auto end = high_resolution_clock::now();
```

Execution Time: 56 Micro Seconds

زمان اجرا برای حالت $\text{ThreadPerBlock} = 512$:

```
auto start = high_resolution_clock::now();  
Sharp_Given_Image <<<((num_of_rows * num_of_columns) / 512), 512>>> (d_input, d_output);  
auto end = high_resolution_clock::now();
```

Execution Time: 24 Micro Seconds

زمان اجرا برای حالت $\text{ThreadPerBlock} = 256$:

```
auto start = high_resolution_clock::now();  
Sharp_Given_Image <<<((num_of_rows * num_of_columns) / 256), 256>>> (d_input, d_output);  
auto end = high_resolution_clock::now();
```

Execution Time: 21 Micro Seconds

زمان اجرا برای حالت $\text{ThreadPerBlock} = 128$:

```
auto start = high_resolution_clock::now();  
Sharp_Given_Image <<<((num_of_rows * num_of_columns) / 128), 128>>> (d_input, d_output);  
auto end = high_resolution_clock::now();
```

Execution Time: 20 Micro Seconds

زمان اجرا برای حالت $\text{ThreadPerBlock} = 32$:

```
auto start = high_resolution_clock::now();  
Sharp_Given_Image <<<((num_of_rows * num_of_columns) / 32), 32>>> (d_input, d_output);  
auto end = high_resolution_clock::now();
```

Execution Time: 33 Micro Seconds

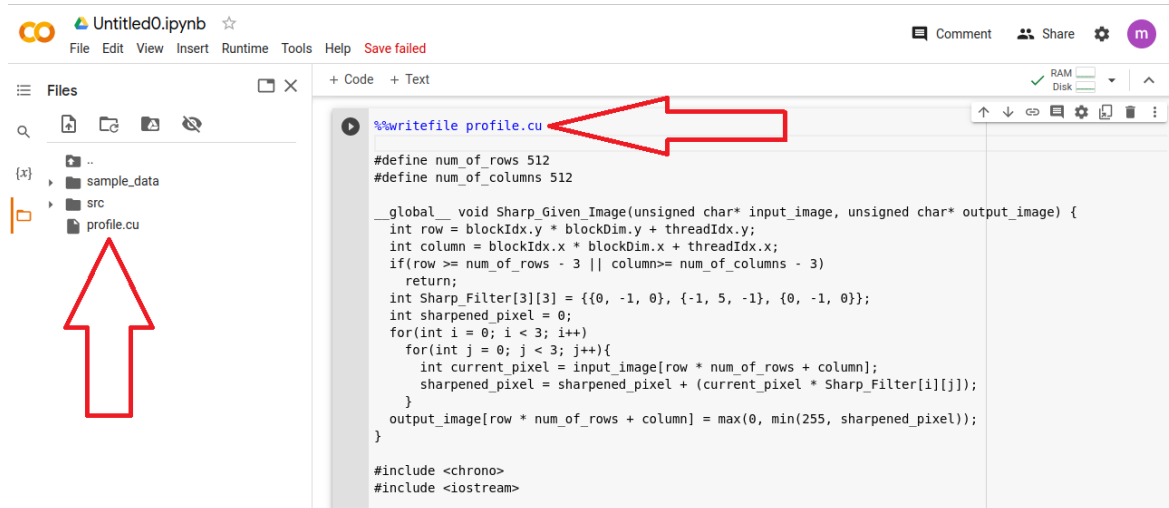
بر اساس زمان اجرا بدست آمده، برای هر حالت، در گام سوم، داریم:

$$1024 > 32 > 512 > 256 > 128$$

زمان اجرا با افزایش thread ها کاهش می یابد ولی از یک حدی به بعد، زمان اجرا شروع به افزایش می کند. در واقع تا زمانی که تعداد thread ها از تعداد core موجود در Streaming Microcontroller کمتر است، هر thread به یک core اختصاص یافته و در نتیجه به صورت موازی اجرا می شود. بنابراین performance افزایش می یابد. اما با افزایش thread ها، زمانی که تعداد thread از تعداد core موجود در SM بیش تر شود، آن وقت به تعدادی از core ها، بیش از یک thread اختصاص داده می شود. در نتیجه بعضی از thread ها باید صبر کنند تا اجرا thread بر روی core خاتمه یابد و بعد از آن، روی core اجرا شوند. در نتیجه بعضی از core ها به صورت سریال اجرا شده و بنابراین، performance کاهش می یابد.

پس افزایش thread ها تا یک حدی، باعث افزایش performance می شود و با افزایش بیش از حد تعداد thread ها، Performance کاهش می یابد.

پروفایل گیری برنامه:



The screenshot shows a Jupyter Notebook titled 'Untitled0.ipynb'. The left sidebar displays a file explorer with a folder named 'sample_data' and a file named 'profile.cu'. A red arrow points to 'profile.cu'. The main area shows the code for 'profile.cu', which is a C++ program for image sharpening. A red arrow points to the filename 'profile.cu' in the code editor's title bar.

```

%%writefile profile.cu

#define num_of_rows 512
#define num_of_columns 512

__global__ void Sharp_Given Image(unsigned char* input_image, unsigned char* output_image) {
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int column = blockIdx.x * blockDim.x + threadIdx.x;
    if(row >= num_of_rows - 3 || column >= num_of_columns - 3)
        return;
    int Sharp_Filter[3][3] = {{0, -1, 0}, {-1, 5, -1}, {0, -1, 0}};
    int sharpened_pixel = 0;
    for(int i = 0; i < 3; i++){
        for(int j = 0; j < 3; j++){
            int current_pixel = input_image[row * num_of_rows + column];
            sharpened_pixel = sharpened_pixel + (current_pixel * Sharp_Filter[i][j]);
        }
        output_image[row * num_of_rows + column] = max(0, min(255, sharpened_pixel));
    }

#include <chrono>
#include <iostream>

```

Writing profile.cu

بر اساس profile برنامه در تصویر زیر:

```
[28] !nvcc /content/profile.cu -o profile -Wno-deprecated-gpu-targets
5s

[29] !nvprof ./profile
2s

==19922== NPROF is profiling process 19922, command: ./profile
Execution Time: 41 Micro Seconds
==19922== Profiling application: ./profile
==19922== Profiling result:
```

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	35.12%	24.767us	1	24.767us	24.767us	24.767us	[CUDA memcpy HtoD]
	33.35%	23.519us	1	23.519us	23.519us	23.519us	Sharp_Given_Image(unsigned c
	31.53%	22.240us	1	22.240us	22.240us	22.240us	[CUDA memcpy DtoH]
API calls:	98.29%	434.10ms	2	217.05ms	5.7550us	434.10ms	cudaMalloc
	1.54%	6.8195ms	1	6.8195ms	6.8195ms	6.8195ms	cuDeviceGetPCIBusId
	0.06%	284.34us	2	142.17us	113.19us	171.15us	cudaMemcpy
	0.04%	187.05us	2	93.522us	13.710us	173.34us	cudaFree
	0.04%	175.43us	101	1.7360us	204ns	68.063us	cuDeviceGetAttribute
	0.01%	40.008us	1	40.008us	40.008us	40.008us	cuDeviceGetName
	0.01%	37.426us	1	37.426us	37.426us	37.426us	cudaLaunchKernel
	0.00%	2.3350us	3	778ns	280ns	1.7010us	cuDeviceGetCount
	0.00%	1.1520us	2	576ns	326ns	826ns	cuDeviceGet
	0.00%	785ns	1	785ns	785ns	785ns	cuDeviceGetUuid
	0.00%	646ns	1	646ns	646ns	646ns	cuDeviceTotalMem
	0.00%	544ns	1	544ns	544ns	544ns	cuModuleGetLoadingMode

- 35.12 درصد از زمان اجرا برنامه، به انتقال داده از CPU به GPU اختصاص یافته است.

- 33.35 درصد از زمان اجرا برنامه، به اجرا تابع فیلتر، یعنی Sharp_Given_Image اختصاص یافته است.

- 31.53 درصد از زمان اجرا برنامه، به انتقال داده از GPU به CPU اختصاص یافته است.

در نتیجه حدود 67 درصد برنامه، به جابجایی داده بین CPU و GPU و اختصاص دادن حافظه به GPU، تخصیص داده شده است. بنابراین برای بهبود زمان اجرا می توان:

- داده ها را به صورت یکجا، بین CPU و GPU منتقل کرد تا کلا یک بار سر بار cudaMemcpy داشته باشیم و سر بار cudaMemcpy کاهش یابد.

- حجم داده انتقالی، بین CPU و GPU را کاهش داد تا سربار `cudaMalloc` کاهش یابد.
- خود الگوریتم (در اینجا `sharp` کردن تصویر) را بهبود داد.
- داده ها را از قبل در GPU ذخیره، و به کرات از آن استفاده کرد تا نیاز به انتقال مجدد داده نداشته باشیم و سربار `cudaMemcpy` کاهش یابد.