

ASSIGNMENT #2

PROGRAMMING MASSIVELY PARALLEL PROCESSORS

Due Date:

As specified on the moodle

Topics covered:

Tiled convolution, shared memory, constant cache/memory

, Deliverables:

The submitted project file is kirsch-gpu.cu. The file should be submitted according to the stipulations set out in the course syllabus.

Objectives:

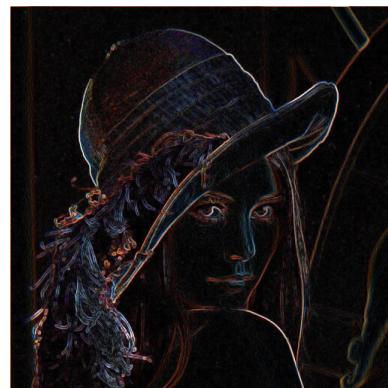
Implementation of Kirsch edge detection in CUDA C. Learn how to use shared memory, constant cache/memory, tiled convolution, and loop unrolling to write more efficient kernel functions.

Implementing the Kirsch Edge Detection Filter

The aim of this assignment is to implement a GPU version of the Kirsch Filter, which is a kind of filter used in image processing for edge detection. The purpose of edge detection is to identify points in the image where there are discontinuities or major shifts in colour change (hence detecting the “edges” of the image). This is useful for applications such as feature extraction or face detection etc. For example, the following pictures show the effect of edge detection.



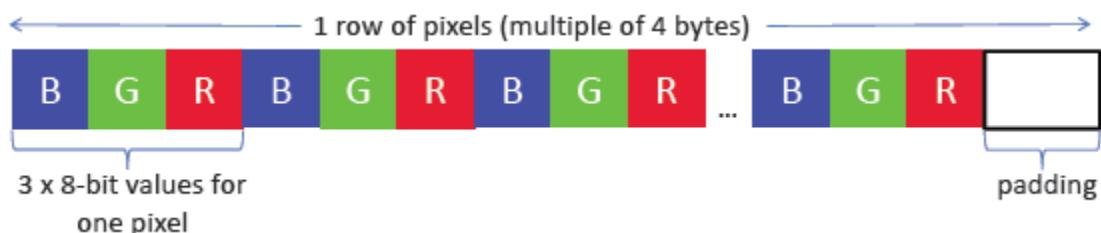
Before Edge Detection



Kirsch Edge Detection

The BMP file format

The 24-bit colour bitmap file consists of two parts. A 54-byte header that contains information such as image height and width, and pixel array that contains the colour information of each pixel. Every row in the pixel array has the following format:



Each pixel contains 3 8-bits value for Blue, Green and Red respectively. Each of these 8-bits value should be considered as an unsigned byte ranging between 0 to 255 in values. Different combinations of the RGB values determine the colour of the particular pixel. The number of pixels in a row is determined by the width of the picture. For example, if the picture has a width of 50 pixels, the pixel row will be $50 \times 3B + 2B = 152B$. The additional 2 bytes of padding was included to make entire row a multiple of 4 bytes. Of course, the height of a picture will determine the number of rows that we have.

However, the original format of the file is not convenient for data manipulation. We have included a function in the file bmp.cpp that has the following two functions:

```
void bmp_read(char *filename, bmp_header *header, unsigned char **data);
void bmp_write(char *filename, bmp_header *header, unsigned char *data);
```

The `bmp_read` takes in a filename and fill in the header information into the header object passed by reference into the function. It also allocates an array for the data variable and fills it in with a reformatted pixel information in one single array. In particular, the array is presented in the following form:



All the paddings have been removed and now instead, we have a single array instead. The `bmp write` function performs the reverse of the `bmp read` function and writes a given single array into the bitmap file with the correct format. Therefore, the following code simply copies the bitmap file from one to another:

```
bmp_header h;
unsigned char *data;

bmp_read("in.bmp", &h, &data );
bmp_write("out.bmp", &h, &data);
```

The definition of the `bmp header` and the functions are given in the files `bmp.cpp` and `bmp.h`. You may turn on interleave option by defining `COLOR_INTERLEAVE` as shown in `bmp.cpp`.

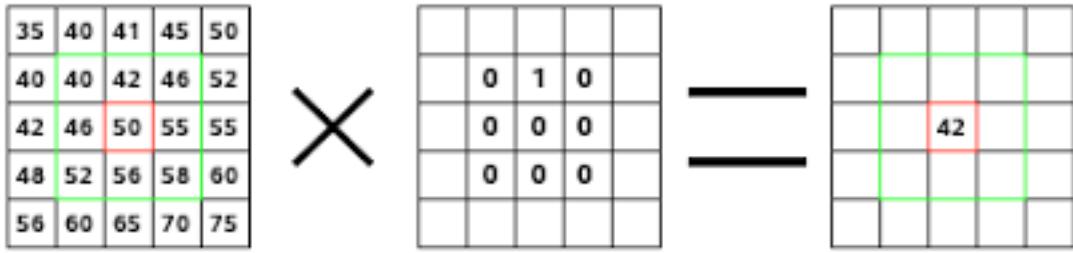
Kirsch Edge Detection Algorithm

At the heart of the Kirsch edge detection algorithm is one mask that is rotated 8 times and applied. The following are 8 masks that are used in Kirsch edge detection:

$$g^{(1)} = \begin{pmatrix} +5 & +5 & +5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{pmatrix}, g^{(2)} = \begin{pmatrix} +5 & +5 & -3 \\ +5 & 0 & -3 \\ -3 & -3 & -3 \end{pmatrix}, g^{(3)} = \begin{pmatrix} +5 & -3 & -3 \\ +5 & 0 & -3 \\ +5 & -3 & -3 \end{pmatrix}, g^{(4)} = \begin{pmatrix} -3 & -3 & -3 \\ +5 & 0 & -3 \\ +5 & 5 & -3 \end{pmatrix},$$

$$g^{(5)} = \begin{pmatrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ +5 & +5 & +5 \end{pmatrix}, g^{(6)} = \begin{pmatrix} -3 & -3 & -3 \\ -3 & 0 & +5 \\ -3 & 5 & 5 \end{pmatrix}, g^{(7)} = \begin{pmatrix} -3 & -3 & +5 \\ -3 & 0 & +5 \\ -3 & -3 & +5 \end{pmatrix}, g^{(8)} = \begin{pmatrix} -3 & +5 & +5 \\ -3 & 0 & +5 \\ -3 & -3 & -3 \end{pmatrix}.$$

The filter works by applying the masks as convolution matrices on every 3×3 pixels in the image. The following image shows how the convolution works in general:



We apply the mask to each 3×3 squares of the whole image in a similar fashion to how we apply the mask for the above. For all the convolution results after applying the eight Kirsch edge detection masks, we take the maximum convolution value to be the final value of the center pixel after divided by 8 and clamped to the range between 0 and 255. The value of the ghost cell/pixel is regarded as 0 when the boundary pixels are computed. The student is invited to consult `kirsch_cpu.cpp` to find out more details.

The Assignment

In the given file `kirsch_cpu.cpp`, we have a basic implementation of the kirsch edge detection in a function with the following prototype:

```
void kirschEdgeDetectorCPU(
    const unsigned char *data_in,
    const int *mask,
    unsigned char *data_out,
    const unsigned channels,
    const unsigned width,
    const unsigned height
)
```

Please read the function and understand the code in the light of what was described in the previous section.

You are supposed to use CUDA C to optimize the basic implementation. The function, `convolution`, is supposed to be filled in by you in a `kirsch_gpu.cu` file, where the declaration of `convolution` is as follows:

```

__global__ void convolution(unsigned char *I,
                           const int *__restrict__ M,
                           unsigned char *P,
                           int channels,
                           int width,
                           int height)

```

Please consider the following hints:

- There are two design options as shown in our lecture notes. You are required to implement the following Design 1.
 1. Design 1: The size of each thread block matches the size of an output tile. All threads participate in calculating output elements. You should define a shared memory area that is $(\text{TILE_WIDTH} + \text{Mask_width} - 1) \times (\text{TILE_WIDTH} + \text{Mask_width} - 1)$ and thread block size is $\text{TILE_WIDTH} \times \text{TILE_WIDTH}$. Assume the shared memory area size is smaller than $2 \times \text{TILE_WIDTH} \times \text{TILE_WIDTH}$. `Mask_width` is the width of convolution mask, which is 3. This should be implemented as `convolution`.
- You should associate the 2D coordinates of the thread with the correct location of the shared memory area.
- The operations must be done at least in 16 bits to accommodate for overflow/underflow arithmetic.
- You may use `blockIdx.z` to associate with each channel.

Please compare the timings of your code under Microsoft Visual Studio Release mode. You may use the following utility to collect the measurements¹.

```
ncu -o profile yourApp.exe [yourApp argslist]
```

Assume the application executable is `kirsch.exe` and we are interested in metric of theoretical occupancy given by metric name `sm_maximum_warps_per_active_cycle_pct` and achieved occupancy given by metric name `sm_warps_active.avg.pct_of_peak_sustained_active`. We can use the following command line:

```
ncu --set default --metrics sm_maximum_warps_per_active_cycle_pct,
sm_warps_active.avg.pct_of_peak_sustained_active
kirsch.exe lena.bmp lena.cpu.bmp lena.gpu.bmp
```

The example of the output with regarding to the metric information is

```

==PROF== Disconnected from process 13672
[13672] kirsch.exe@127.0.0.1
convolution(unsigned char *, const int *, unsigned char *, int, int, int)
(32, 32, 3)x(16, 16, 1), Context 1, Stream 7, Device 0, CC 7.5
Section: Command line profiler metrics
-----
```

¹You can use `ncu-ui` as well.

Metric Name	Metric Unit	Metric Value
sm__maximum_warps_per_active_cycle_pct	%	100
sm__warps_active.avg.pct_of_peak_sustained_active	%	91.77

Rubrics

This assignment will be graded over 100 points. Here is the breakdown:

- If your code fails to compile, zero is awarded immediately.
- If you did not use tiled convolution and shared/constant memory to implement the optimized kirsch edge detection, zero is awarded immediately.
- Comments/Code Readability. Comments are important for CUDA C code to enhance readability. Otherwise, C code cannot be easily maintained. Maximum 10 points will be deducted.
- Correctness. You must ensure that the code works for images of all sizes and the output image of CPU and GPU version should be the same. Please refer to VPL test cases on the Moodle.
- Speedup. Please refer to VPL test cases on the Moodle.

The list above is non-exhaustive.

The lecturer reserves the right to impose reasonable penalties for code that violates general practices or does not match the specification in an obvious way that has not been mentioned above. In exceptional cases, the lecturer reserves a discretionary right to allow resubmission or submission after the deadline.