CUDA Exercises

Lab 2 (26th Aug 2019)

Logistics

- ~~Block A (Friday / 23rd Aug)~~
  - ~~How to use CUDA~~
  - ~~How to launch a CUDA kernel and array indexing~~

- Block B (Monday / 26th Aug)
  - How to index 2D indexes
  - How simple convolution works
  - (Bonus) How to use shared memory

Clarification

- Time difference computation should now be fixed
- Tegner supports nvprof
  - srun -n 1 nvprof ./a.out

Setting up...

1. kinit --forwardable your_username@NADA.KTH.SE
2. ssh -Y your_username@tegner.pdc.kth.se
3. cd /cfs/klemming/nobackup/[your_initial]/[your_username]
4. module load git
5. module load cuda/7.0
    1. FORTRAN: module load pgi
6. git clone https://github.com/PDC-support/cuda-lab-exercises.git
7. Go inside the code directories
    1. C: cd cuda-lab-exercises/lab_2/C
    2. FORTRAN: cd cuda-lab-exercise/lab_2/Fortran

README.md

CUDA Laboratory 2

(Again) Instructions also available in README on Github page

Introduction to High-Performance Computing

Introduction

In this second laboratory about GPU programming in CUDA, we are going to continue building your skills in order to develop more advanced GPU-accelerated applications. As a friendly reminder, the laboratory is divided in two different blocks and exercises:

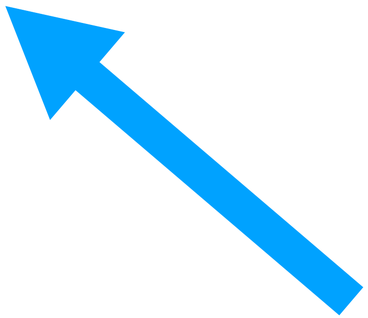
Compiling

```
nvcc -arch=sm_30 lab02_ex3_6.cu -o lab02_ex3_6.out
```

Compiling with Fortran

~~nvcc -arch=sm_30 lab02_ex3_6.cu -o lab02_ex3_6.out~~

```
make
```



Automatically creates binary

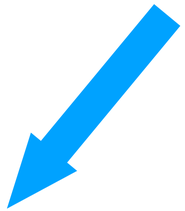
Compiling with Fortran

Main program and helper functions (timer) exists as C file, compiled and linked during compilation!

Don’t bother 🙂

Running CUDA Program

We are going to use the “thin nodes” on Tegner



```
salloc --nodes=1 -C Haswell --gres=gpu:K420:1 -t 00:05:00 -A edu19.summer --reservation=summer-2019-08-26
```

Request for one K420 GPU on the node

Running CUDA Program

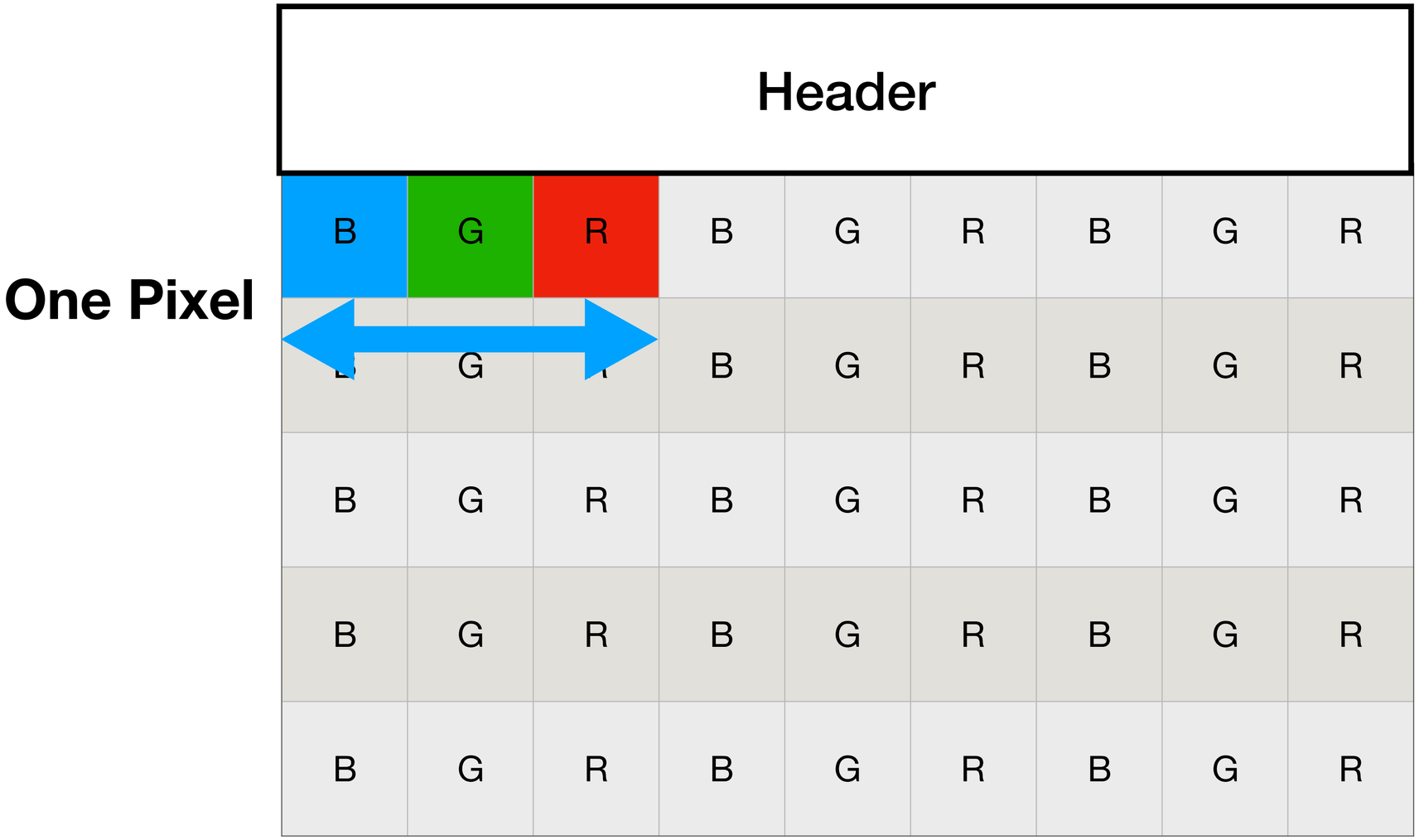
```
srun -n 1  ./lab02-ex3-6.out image/lab-02.bmp
```

To view an image

```
display -resize 1280x720 images/lab02.bmp
```


ESPRESSO HOUSE
ESPRESSO HOUSE

Exercise 4: Greyscale “filter”



Exercise 4: Greyscale “filter”

- Colorimetric method (luminance-preserving) method

$$Y_{out} = 0.0722 \times B_{in} + 0.7152 \times G_{in} + 0.2126 \times R_{in}$$

Exercise 4: Greyscale “filter”

$$Y_{out} = 0.0722 \times B_{in} + 0.7152 \times G_{in} + 0.2126 \times R_{in}$$

Exercise 4: Greyscale “filter”

- Understand cpu_greyscale()
- Implement gpu_greyscale()
  - Block and Grid in ***2D*** is already setup with you
  - Assume each thread responsible for one output pixel
  - ***Boundary check!!!***

Exercise 4: Greyscale “filter”

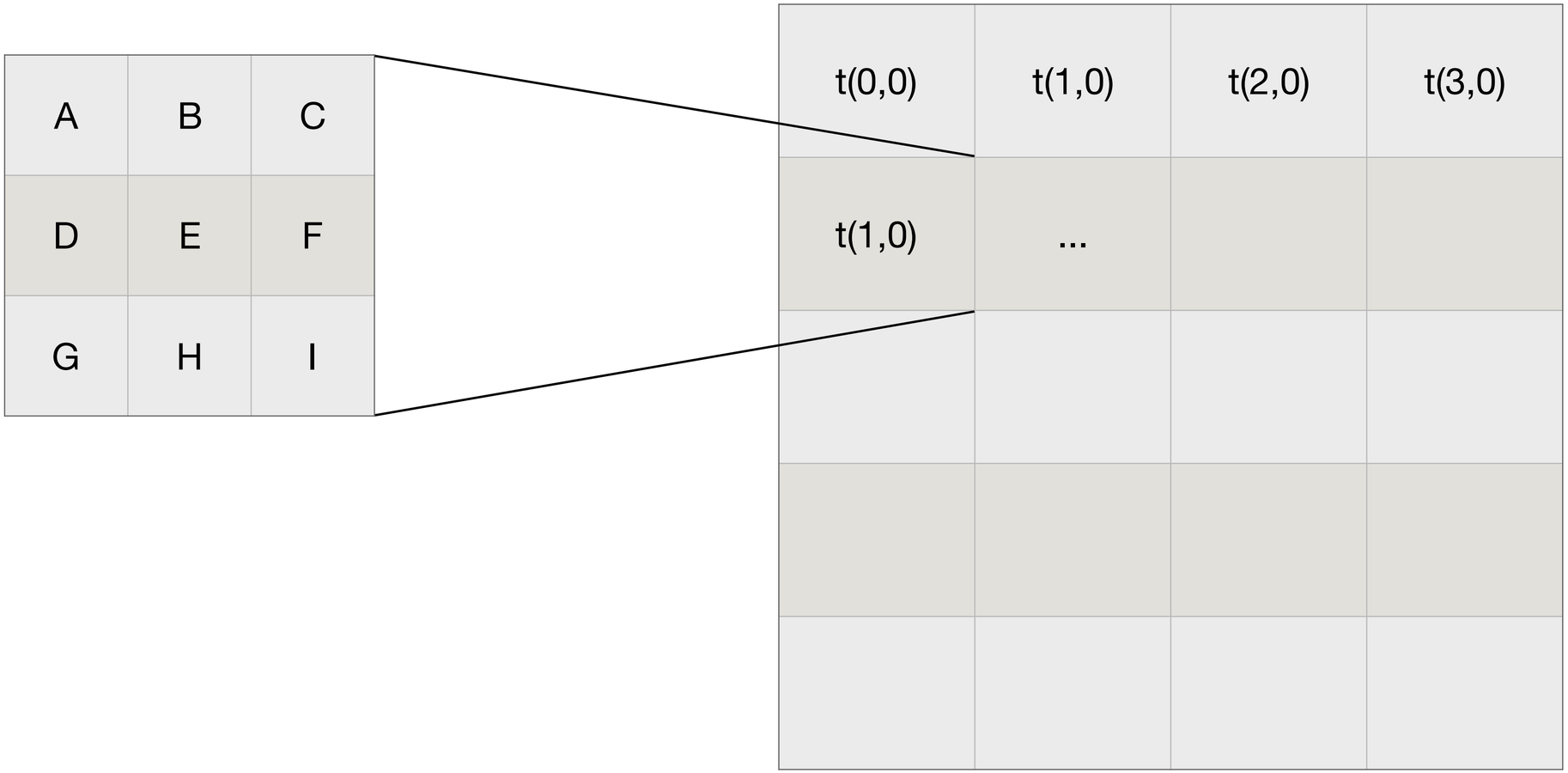
```
display -resize 1280x720 images/lab02_result_1.bmp
```

Exercise 5: Gaussian filter

- Apply a 3x3 convolution matrix on all the pixels of the image
  - Map each pixel as the center of the 3x3 matrix
  - Apply weights with surrounding pixels

$$\left(\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}\right)[2,2] = (i*1) + (h*2) + (g*3) + (f*4) + (e*5) + (d*6) + (c*7) + (b*8) + (a*9).$$

Exercise 5: Gaussian filter



Exercise 5: Gaussian filter

- Understand cpu_applyFilter()
  - Understand how a convolution matrix is applied to a certain pixel
- Implement gpu_applyFilter()
  - The kernel is launched from kernel function from gpu_gaussian()
  - What is the modifier for functions launched by another kernel?

Exercise 5: Gaussian filter

```
montage -tile 2x1 -crop 320x180+512+512 -geometry 640x360   \
        images/lab02_result_1.bmp images/lab02_result_2.bmp \
        images/lab02_result_2_comp.jpg
display images/lab02_result_2_comp.jpg
```

Exercise 6: Detecting Edges

- Apply a Sobel filter to our smoothened image
  - Compute an approximation of the gradient of the image intensity
  - Creates an image where the edges are emphasized
  - A base of full edge detection algorithms such as Canny algorithm
  - Apply as two 3x3 convolution filter
    - Computes derivatives on horizontal and vertical directions
    - Resultant pixel value computed as a square root

Exercise 6: Detecting Edges

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

$$\mathbf{G} = \sqrt{{\mathbf{G}_x}^2 + {\mathbf{G}_y}^2}$$

Exercise 6: Detecting Edges

- Understand cpu_sobel()
- Implement gpu_sobel()

Exercise 6: Detecting Edges

```
montage -border 0 -geometry 640x360 -tile 3x1       \
        images/lab02.bmp images/lab02_result_1.bmp \
        images/lab02_result_3.bmp images/lab02_result_3_comp.jpg
display images/lab02_result_3_comp.jpg
```

CPU and GPU Timing

```
Step #1 Completed - Result stored in "images/lab02_result_1.bmp".
Elapsed CPU: 52ms / Elapsed GPU: 16ms
Step #2 Completed - Result stored in "images/lab02_result_2.bmp".
Elapsed CPU: 270ms / Elapsed GPU: 19ms
Step #3 Completed - Result stored in "images/lab02_result_3.bmp".
Elapsed CPU: 570ms / Elapsed GPU: 20ms
```

When testing your GPU functions, comment out the respective CPU functions, otherwise you might be reading the CPU generated file whereas the GPU kernel failed! 🙂

Check README.md from lab Github for more information

Have fun!