Project OpenCL: Convolution

Assistant: Christophe De Troyer
Mail: cdetroye@vub.be
Office: 10F731

Deadline: 11/06/2017 23:59

Abstract

The goal of this programming project is for you to demonstrate that you know how to write a massively parallel program in OpenCL and apply this knowledge to implement well-performing algorithms in OpenCL.

You are asked to implement a convolution filter in OpenCL. A convolution filter allows you, amonst other things, to apply filters on images, such as blurring, edge detection, and sharpening.

For the project you will receive a sequential implementation which you will then port to OpenCL. The sequential implementation can then serve as a baseline for your benchmarks, as well as a verification of your parallel implementation.

1 Procedure

The project consists of four parts: a sequential implementation in your language of choice, two implementations in OpenCL, a bechmark of your three solutions, and finally a report in which you explain your choices and implementation.

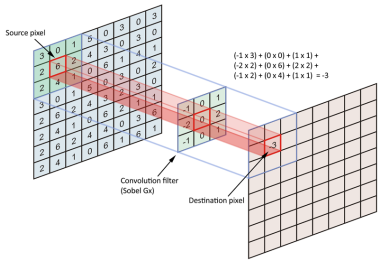
Deadline 11th of June 2017 at 23:59 CEST.

Deliverables Package the implementation into a single ZIP file, including the report as a PDF. The ZIP file should be named `Firstname-Lastname-gpgpu.zip`. On the PointCarré page of the course, go to *Assignments (Opdrachten)* > *Project OpenCL* and submit the ZIP file.

Grading This project accounts for one third of your final grade. It will be graded based on the *submitted code*, the accompanying *report and its evaluation*, and the *project defense* at the end of the year.

2 Convolution

Figure 1: Source: community.arm.com



For this project you will implement a convolution filter; a mathematical operation on two matrices; the input matrix A, and the kernel K. You "slide" the kernel over the input matrix while multiplying all the matching values, and adding them together to yield the value of the center pixel of the kernel. Consider the example below.

$$K = \begin{bmatrix} k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 \\ k_7 & k_8 & k_9 \end{bmatrix} \qquad A = \begin{bmatrix} i_1 & i_2 & i_3 \\ i_4 & i_5 & i_6 \\ i_7 & i_8 & i_9 \end{bmatrix} \qquad Result = \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix}$$

In the $Result$ matrix above, we have computed $Result(1,1)$ as follows.

$$r_5 = (k_1 * i_1) + (k_2 * i_2) + (k_3 * i_3) + (k_4 * i_4) + (k_5 * i_5) + (k_6 * i_6) + (k_7 * i_7) + (k_8 * i_8) + (k_9 * i_9)$$

The compute the $(1,1)$ value in the $Result$ matrix we would need values outside of the matrix. To solve this problem there are a few approaches. This technique is called edge handling. Extending, Wrapping and Cropping are three possible techniques[1]. You are free to choose which technique you apply. However, keep in mind that you will need to modify the sequential algorithm to account for the edge detection that you have chosen.

2.1 Example

Figure 2: Input image

Figure 3: Output image

The left image above is the input image. We have applied a convolution kernel on the image to produce the right image. The convolution kernel is a $5x5$ Gaussian blur kernel.

$$K = \begin{bmatrix} 0.00296902 & 0.01330548 & 0.02193164 & 0.0132969 & 0.00296624 \\ 0.01329306 & 0.05956148 & 0.09809933 & 0.05940565 & 0.01324369 \\ 0.02183118 & 0.09781164 & 0.16105536 & 0.09749752 & 0.02173345 \\ 0.01317942 & 0.05905921 & 0.09732217 & 0.05898191 & 0.01315492 \\ 0.00293499 & 0.01315345 & 0.02168443 & 0.01315035 & 0.00293399 \end{bmatrix}$$

3 Implementation

For the project you are required to convert a given sequential algorithm to OpenCL. Secondly you are required to create an optimized version of this algorithm. The algorithm should read

[1] https://en.wikipedia.org/wiki/Kernel_(image_processing)#Edge_Handling

in an image from your disk, apply the convolution and then write out a new image of the same dimensions.

First of all, you will port the sequential algorithm to the GPU by means of OpenCL. You should make sure that this algorithm produces the exact same result as the sequential algorithm. Be sure to use the relative error as we have seen in the lab sessions.

Finally, you will optimize this algorithm for the GPU. The vector for optimisations on this algorithm is quite large so we do not expect you to write a fully optimized version. The goal of this optimized algorithm is for you show that you grasp the concepts of GPU programming and how an algorithm can be optimized. Possible optimizations are using private memory, local memory, tweaking work-group sizes, changing the workload per work-item, vectorization etc.

4 Evaluation

Next to your implementation, you should perform a thorough evaluation of your application. Your evaluation should focus on two points: validating the correctness of your implementation and evaluating its performance.

To validate **correctness** you can simply verify your OpenCL algorithms against the sequential implementation.

To evaluate the **performance** you can use any image of sufficient dimension (minimum 2 million pixels) and measure the amount of time it takes to process the image. You will have to do this for the sequential implementation, and both OpenCL implementations. For the OpenCL implementations you are required to benchmark on the GPU as well as on the CPU. You can show the increased performace by timing the execution. Note that you should always optimize for execution on the GPU, and not for execution on the CPU. The most important part of the evaluation of performance is that you can explain and reason why your naive solution is slower than your optimized solution, as well as the difference in performance on the GPU and CPU.

5 Reporting

Your source code should be accompanied by a brief report. Please structure your report according to the following outline:

1. Overview (max. 100 words)

Briefly summarize your implementation approach and the experiments you performed.

2. Implementation (ca. 2 pages)

Describe the implementation on an abstract level. Use illustrations or code snippets to guide the reader where necessary. Answer the following questions:

- How did you port the sequential algorithm to the naive OpenCL version?
- What is the work a work-item did?
- What are the potential performance bottlenecks?
- Which techniques did you use to optimize your naive implementation? Did they have the expected effect on performance?

3. Evaluation (ca. 3 pages, without graphs/illustrations)

- How did you verify correctness of your sequential and OpenCL implementations? Which tests did you use?
- Report results appropriately, graphically. Explain these results.

- Interpret the results: provide an explanation for the observed behavior.
  - Why are you seeing the performance you see?
  - Could you optimize further by smarter use of the local and private memory?
  - Explain your choice of parameters of your kernel (i.e., work-group size, shape, private memory size..)
- Which tools did you use to assess performance?

4. Insight questions (max. 250 words each)

Briefly answer the questions below. They relate to variations of the problem and how you'd change your implementation to deal with them. You do not have to implement these variations or perform any experiments.

- If you had more time, which optimizations would you apply to your OpenCL implementation, and why do you think they are a good idea?
- Consider the portability of your OpenCL algorithms. For example, the Iris Pro GPU in an Intel Core i7 has 64 KB of local memory, and a maximum of 512 work-items in a work group. On the other hand, an AMD Radeon R9 M370X has 32KB of local memory, and only supports up to 256 work-items in a work-group. Explain how these different possible specifications could harm or improve the performance of your implementation.