Multiprocessor programming 521288S

Miguel Bordallo
University of Oulu
Center for Machine Vision and Signal Analysis
Faculty of Information Technology and Electrical Engineering
January 21, 2019

GENERAL INFORMATION

Welcome to the Multiprocessor Programming course! This course consists on one exercise, and the teacher is:

Miguel Bordallo, miguel.bordallo@oulu.fi, TS301

In addition to the teacher, the course will have one assistant. He is:

Mehdi Safarpour, mehdi.safarpour@oulu.fi, TS301

This handout contains the exercise instructions for the Multiprocessor Programming course. When the exercise is completed, you will receive 5 ECTS credits. The completed answers, a training diary and all code written should be returned by email to the corresponding teacher. For inquiries, you can primarily contact the teacher but you can also be in contact with the assistant.

Each returned exercise will be either accepted or rejected. If an exercise is rejected, the teacher or assistant will give instructions which will help you improving your answers or programs.

When the exercise is accepted, you will complete a short *exit document or email* containing some questions about the total workload, the distribution of the work and the self-assessment of the grade. The contents of this document/questionnaire will be discussed in a final **exit interview**, where the grades will be given.

Based on the overall quality of the work, you will receive a grade from 1 to 5.

There is no final exam in the course, but during the exit interview, be prepared to answer to a few oral questions about the exercise and its implementation. Additionally, you should be able to demonstrate *(if required by the teacher*) that your program is working properly. If you are unable to finish the work in time, you **SHOULD** explicitly ask for additional time. The compulsory part finished exercise work should be returned before **April 22nd, 2019**.

In the beginning of the course, there was a voluntary initial lecture, where some general topics were discussed. If you did not attend the lecture, you should download the slides to get a general idea of the organization of the course. If they raise any doubts, you can contact the teacher or assistants.

The course can be carried out either alone or in groups of ***maximum*** two students. Each group should carry out the exercises by themselves. The assistants will give advises on request. Copying code or answers from other groups is definitely prohibited, and will lead to penalties!

Before starting the work each group must register by registering the group by sending an email with the names of the components, the student numbers and the access key numbers.

After the initial lecture, the course material and further announces will be placed in the official Noppa website:

https://noppa.oulu.fi/noppa/kurssi/521288s/

Visit it often in order to see possible updates to the instructions or material

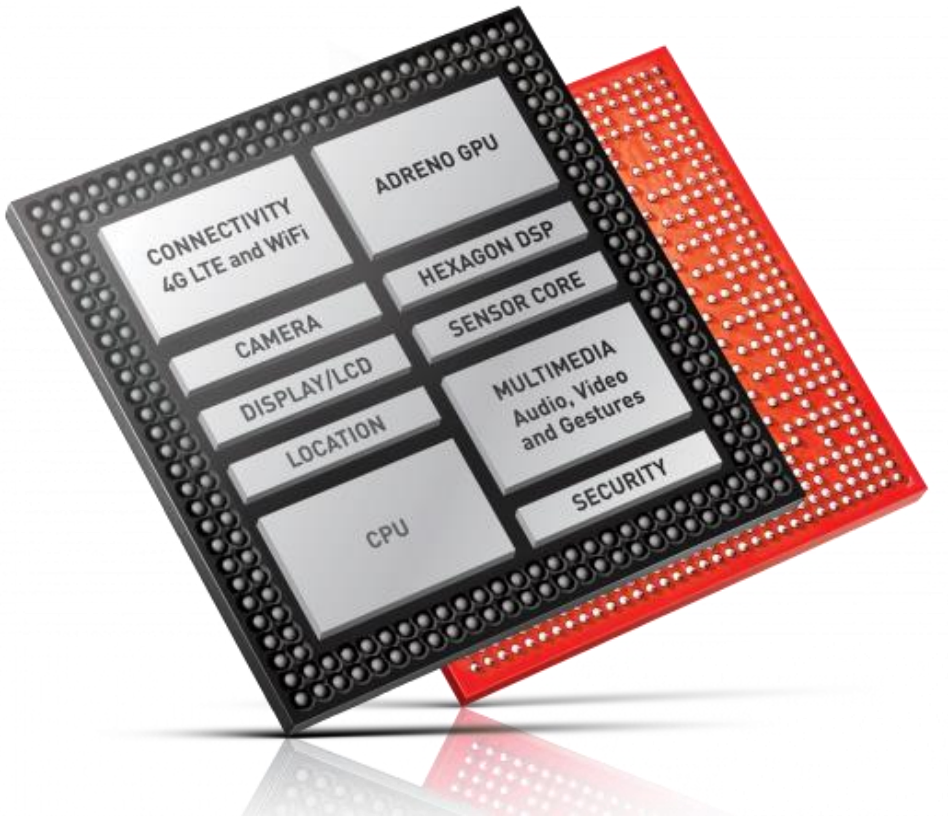
NOTE:
If you are an exchange student and/or you have different grade requirements other than an ECTS grade, you must ask for it when returning the work. If you are planning to graduate soon after completing the course, you should mention it to the teacher *(e.g. this is almost your last course).* Otherwise the completion mark will be given within three weeks after the exit interview.

BACKGROUND

Mobile communication devices are becoming attractive platforms for multimedia applications as their display and imaging capabilities are improving together with the computational resources. Many of the devices have increasingly been equipped with several sensors i.e. motion sensors, environmental sensors and position sensors that allow the users to capture several signals at different rates.

To process these signals, with power efficiency and reduced space in mind, most mobile device manufacturers integrate several chips and subsystems on a System on Chip (SoC). Figure 1 illustrates organization of a Snapdragon 805 system-on-chip (SoC) featured e.g. on Nexus 6 mobile phones. However, the same basic structure can be found on any high-end mobile phone or tablet.

For long, it was a tedious task to develop programs that could exploit all available hardware. However, since Open Computing Language (OpenCL) was introduced, the development work was simplified significantly. However, the code still needs to be optimized using approaches that depend on the architectural differences, using strategies such as assembly macros embedded in the actual OpenCL code. The program development itself has changed quite dramatically. The same code can be run and tested on multiple devices. In addition, the code can even be partioned to run on multiple devices simultaneously, providing true heterogeneous computing capabilities with a single programming framework.



Qualcomm Snapdragon 805

https://www.qualcomm.com/products/snapdragon/processors/805

In this project, we will concentrate on a simple heterogeneous computing setup using only a multicore CPU and a GPU. The implementation will be benchmarked on both devices individually as well as with a heterogeneous setup using both devices. The CPU works as the "host"-processor in all cases according to the OpenCL platform model. You will make the final optimization choice on which parts of the algorithms to offload to the GPU.

THE ASSIGNMENT

Your task is to **implement and accelerate** a stereo disparity algorithm in **OpenCL** by offloading most of the computations GPU. For comparative reasons, the same OpenCL implementation should be benchmarked also on a CPU-only setup. Also a heterogeneous CPU-GPU version will be implemented based on the kernel execution times.

The algorithm to be implemented is "Depth estimation based on Zero-mean Normalized Cross Correlation (**ZNCC**)".

ZNCC is function that expresses the correlation between two grayscale images (or patches). The result of the function is an integer that becomes larger the more that these two patches are correlated. Estimating depth with ZNCC can be done evaluating the following expression.

$$ZNCC(x,y,d) \triangleq \frac{\sum_{j=-\frac{1}{2}(B-1)}^{\frac{1}{2}(B-1)} \sum_{i=-\frac{1}{2}(B-1)}^{\frac{1}{2}(B-1)} [I_L(x+i,y+j) - \overline{I_L})] * [I_R(x+i-d,y+j) - \overline{I_R}]}{\sqrt{\sum_{j=-\frac{1}{2}(B-1)}^{\frac{1}{2}(B-1)} \sum_{i=-\frac{1}{2}(B-1)}^{\frac{1}{2}(B-1)} [(I_L(x+i,y+j) - \overline{I_L}]]^2} * \sqrt{\sum_{j=-\frac{1}{2}(B-1)}^{\frac{1}{2}(B-1)} \sum_{i=-\frac{1}{2}(B-1)}^{\frac{1}{2}(B-1)} [I_R(x+i-d,y+j) - \overline{I_R}]^2}}$$

B = window size, I_L, I_R= left and right images, $\overline{I_L}$, $\overline{I_R}$ window average, d = disparity value (0 to d)

The final disparity value will be chosen based on the Winner-takes-it-all-approach, where the value of d with largest ZNCC(x,y,d) value is chosen for the pixel value (x,y) of the final disparity map.

Note that for our simple case, the disparity can be considered one-dymensional (horizontal only) and there is no disparity in y-axis. This is due to the fact that the sample images that we use for our computation are already *rectified*, as it is usually the case from the input of stereo cameras.

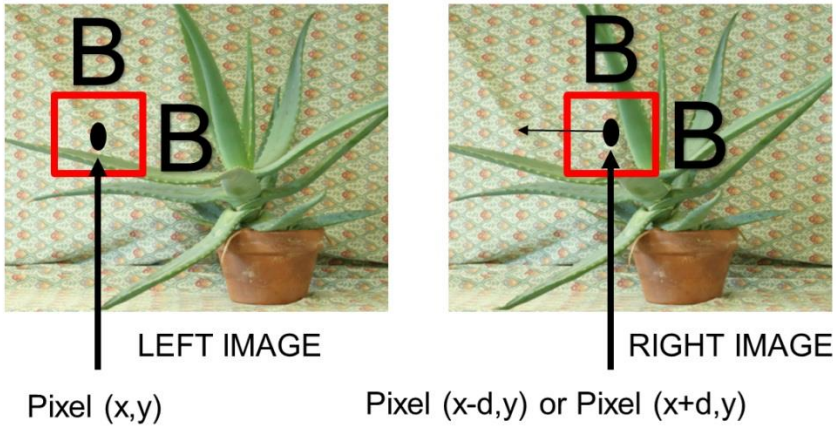
The previous equation can be also expressed using the following Pseudo code:

```
FOR J = 0 to image height
    FOR I = 0 to image width
        FOR d = 0 to MAX_DISP
            FOR WIN_Y = 0 to WIN_SIZE
                FOR WIN_X = 0 to WIN_SIZE
                    CALCULATE THE MEAN VALUE FOR EACH WINDOW
                END FOR
            END FOR

            FOR WIN_Y = 0 to WIN_SIZE
                FOR WIN_X = 0 to WIN_SIZE
                    CALCULATE THE ZNCC VALUE FOR EACH WINDOW
                END FOR
            END FOR

            IF ZNCC VALUE > CURRENT MAXIMUM SUM THEN
                UPDATE CURRENT MAXIMUM SUM
                UPDATE BEST DISPARITY VALUE
            END IF
        END FOR

    DISPARITY_IMAGE_PIXEL = BEST DISPARITY VALUE
    END_FOR
END FOR
```



POST PROCESSING:

Once the ZNCC equation is evaluated and a first estimation of the depth map is obtained, it should be post processed for refinement.

The post processing stage consists of two phases: cross-checking and occlusion filling.

Cross checking: (Two input images, one output image)

Cross checking is a process where you compare two depth maps. The left disparity map is obtained by mapping the image on the left against the one on the right. The right disparity map is obtained analogously by mapping the image on the right against the one on the left.

To obtain a consolidated map, the process consists in checking that the corresponding pixels in the left and right disparity images are consistent. This can be done by comparing their absolute difference to a threshold value. For our case, it is recommended to start with a threshold value of 8, while the best threshold value for your implementation can be obtained by experimentation. If the absolute difference is larger than the threshold, then replace the pixel value with zero. Otherwise the pixel value remains unchanged. This process helps in removing the probable lack of consistency between the depth maps due to occlusions, noise, or algorithmic limitation.

Occlusion filling: (One input image, one output image)

Occlusion filling is the process of eliminating the pixels that have been assigned to zero by the previously calculated cross-checking. In the simplest form it can be done by replacing each pixel with zero value with the nearest non-zero pixel value. However, in this exercise it is recommended that you experiment with other more complex post-processing approaches that obtain the pixel value in different forms.

The previously depicted description of the algorithm and the enclosed pseudocode are just meant to help you get started with the algorithm implementation. However, notice that this is not the only (or best) solution. You are free to experiment with other approaches. Once you understand the algorithm so you are able to design a sequential implementation, you can start evaluating what are the opportunities of parallelization that could be exploited in the several phases of the exercise.

DATA SET UTILIZATION:

In this exercise, for benchmarking purposes we are going to utilize images from the following dataset:
http://vision.middlebury.edu/stereo/data/scenes2014/datasets/Backpack-perfect/

As a starting point, download images *im0.png* and *im1.png*. The maximum disparity value for the two images included in the package can be found on the *calib.txt* file and is *ndisp=260* for the above mentioned two images. Take this disparity value into account if you do any image downsizing and reduce it accordingly.

When starting to code the algorithm, it is recommended that you use LodePNG to read the images, You will only need to include *lodepng.c*- or *lodepng.cpp*-source file and the *lodepng.h*-header file to your project and use the appropriate functions provided in these files to read and write the image files. If you feel it is more convenient, you are welcomed to use any other image-reading library or function.

GETTING STARTED

The exercise is divided in six phases, each of them having its own checkpoint:

Phase 1: Self study and platform installation
Expected result: A working development environment, knowledge of the open CL basics

Phase 2: Algorithm implementation in sequentialm single thread C/C++-code
Expected result: A simple but correct serial implementation using C/C++ code

Phase 3: Algorithm implementation using in multi-threaded C/C++-code
Expected result: A C/C++ implementation that utilize more than one core of the CPU

Phase 4: Algorithm implementation using OpenCL for a GPU
Expected result: A simple but correct parallel implementation of the algorithm runnings on GPU

Phase 5: Algorithmic and implementation optimization for OpenCL in CPU/GPU
Expected result: An optimized implementatuion that considers the GPU architecture
Expected documents: Training diary, complete source code, final report

Phase 6 (Optional): Porting of the algorithms to a mobile platform:
Expected result: An OpenCL implementation able to run on an ORDROID platform ***OR***
An OpenCL implementation that uses efficiently the memory hierarchy

PHASE 1: Installation of the environment and self study

Checkpoint:

- *21st of January 2019*

Important tasks:

- *Environment installed and working*
- *Registration of the group with the teacher via email*

This phase is dedicated to self study and installation of the work environment. To install the environment, you can follow these steps:

STEP 1.

Start your work by setting up the development environment. Since you will be using your own computer, you will need to first install a C/C++ compiler. You will want to install also an integrated development environment (IDE) such as Eclipse or Visual Studio.

> *Note: In TS135 there are workstations equipped with AMD GPUs and Visual Studio 201x. You can check how a working evironment should feel and even test some of your code there.*

For Windows users that want to use Visual Studio, the 2015 version (recommended) can be obtained signing in with your O365 university credentials and joining the developing program.

For Linux users GCC and G++ compilers are enough, but if you want to use an IDE you can install i.e. Eclipse or QtCreator. GDB might also be useful for debugging your code.

For Mac users, please check https://developer.apple.com/opencl/

Note that you can use a virtual Linux machine running on your Windows/Mac machine, but there is no quarantee that you will be able to use the GPU, but CPU becnhmarking should work.

STEP 2.

After you have the compiler and IDE installed, you will also have to install the at least one OpenCL software development kit (SDK), depending on your computer set-up.

The typical SDK includes OpenCL headers and an installable client driver (ICD) loader amongst other things needed to compile and run OpenCL programs. A short listing on SDKs for different set ups can be found below. If you are not sure which SDK you should install or if you encounter problems in Step 1. or 2. contact the teacher or assistants.

For a CPU only set-up:

AMD CPU : AMD SDK
Intel CPU: AMD or Intel SDK

For a CPU+GPU set-up (check that the GPU has OpenCL support from the vendor list)

Intel Integrated Graphics: Intel SDK
AMD GPU: AMD SDK
NVIDIA GPU: NVIDIA CUDA SDK + SDK for the CPU. NVIDIA SDK does not provide support for running code on the CPU.

Theoretically, one should be able to use any SDK for all GPUs. However, in any case, vendor specific ICD and drivers are always required. The safe bet (recommended) is to go with the processor vendors' own SDK. The SDKs always include OpenCL samples. You should check that your system is set up correctly by re-compiling and running a few of these samples.

PHASE 2: Sequential implementation of the algorithm (in C/C++)

Checkpoint:

- *25th of February*

Important tasks:

- *Implement a working version of the algorithm using C/C++*
- *Send the code and training diary to the teacher and assistants via email*

This phase is dedicated to the implementation of the code in C/C++ in a sequential manner, using only one thread and one processor core. The code should be implemented in a way that the most of the parameters (such as e.g. the window size) can be easily changed. This implementation will be used by you throughout the exercise as the *golden implementation*, and the results of the next phases should be comparable to the ones obtained here.

To proceed with the implementation, you can follow these steps:

- Read/decode the images using *LodePNG*. (*This is the recommended way although you might use others. You just need to download and add the appropriate .c or .cpp files and the header file to you project*). Use the *lodepng_decode32_file*-function to decode the images into 32-bit RGBA raw images and *lodepng_encode_file*-function to encode the output image(s). Notice that you need to normalize the results to grayscale 0..255 as the output image should be in that format.

- Resize the images to $\frac{1}{16}$ of the original size (From 2940x2016 to 735x504). For this work it is enough to simply take pixels from every fourth row and column. You are free to use more advanced approach if you wish.

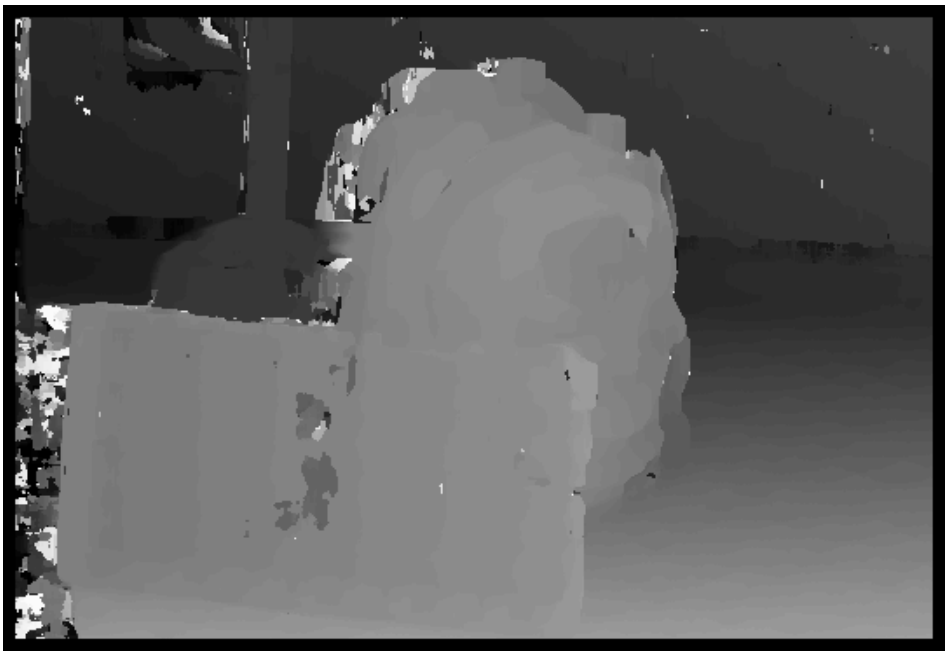
- Transform the images to grey scale images (8-bits per pixel). E.g. Y=0.2126R+0.7152G+0.0722B.

- Implement the ZNCC algorithm with the grey scale images, start with 9x9 window size. Since the image size has been downscaled, you also need to scale the *ndisp*-value found in the calib.txt. The *ndisp* value is referred to as MAX_DISP in the pseudo-code. Use the resized grayscale images as input. You are free to choose how you process the image borders.

- Compute the disparity images for both input images in order to get two disparity maps for post-processing. After post-processing one final disparity map is sufficient in this work.

- Check that your output resembles the one presented below (Output the result image to depthmap.png Notice that you need to normalize the pixel values from 0-ndisp to 0-255. However, once you have checked that the image looks as it should, you can move the normalization process and perform it after the post-processing phase)

- Implement the post-processing including cross-check and occlusion filling (Instructions for post-processing are after the pseudo code on the last page)
- Check again that the disparity map looks right
- Measure the total execution time of your implementation including the scaling and greyscale conversion using for example *QueryPerformanceCounter*-function in Windows or *gettimeofday*-function in Linux. Check the processor load. Report all these data in the email to the teacher and include it also in your final report.
- A preliminary depth map looks approximately like this:



PHASE 3: CPU multi-threading and parallelization

Checkpoint:

- *16th of March* *(tentative, might be subject to change)*

Important tasks:

- *Implement the algorithm using CPU parallelization and multi threading*
- *Send the code and training diary to the teacher and assistants via email*
- *Start the implementation of the OpenCL kernels required in the next phase*

This phase is dedicated to the implementation of the code in C/C++ in a parallel manner using more than one thread of execution and more than one processor core. Again, the code should be implemented in a way that the most of the parameters (such as e.g. the window size) can be easily changed. This implementation will be used by you throughout the exercise to understand where are the opportunities for parallelization in the proposed algorithm.

To proceed with the implementation, you can freely utilize any approach you migh consider. The simplest (and still valid approach) is to utilize OpenMP pragma directives. OpenMP is an API supported by multiple vendors programming languages and operating systems that gives the programmer a very simple yet flexible interfaces for the parallelization of applications:
https://en.wikipedia.org/wiki/OpenMP

To use OpenMP, the compiler must link against the library and optionally a header file (omp.h) could be included in the code. The simplest way of using OpenMP is to include a #pragma before the sections of the code that can be parallelized (for example a for loop) and let the compiler create the threads for you. The most advanced use of OpenMP requires using some functions exposed by the library -- in other words, some rewriting -- but it is easy to get some results right away by simply *decorating* your sequential code.

For this phase, any alternative that uses more than one thread and more than one processing core is acceptable, and it is the task of the student to explore the different alternatives (pthreads, C++Threads, boost threads, even just OpenCL already). In addition to multi-threading, vectorization that uses the SIMD units of the processor (for example Intel SSE instructions) can be utilized and will speed up the computation further.

Again, measure the total execution time of your implementation including the scaling and greyscale conversion using for example *QueryPerformanceCounter*-function in Windows or *gettimeofday*-function in Linux. Check the processor load. Report all these data in the email to the teacher and include it also in your final report. A comparative analysis of the parts of the algorithm that have been parallelized and the impact in the final computing time is also recommended.

After you have the multi-threaded version ready, you should start right away the implementation using OpenCL, even before the checkpoint. Note that the time reserved until this checkpoint is selected to be close to two months, since it is meant to give you the chance of revisiting the first algorithm implementations while already coding the next phases of the exercise. If you utilize all the time until this checkpoint for just multithreading C/C++, it will leave you relatively short time for the OpenCL implementation.

PHASE 4: OpenCL implementation for GPU

Checkpoint:

- *6th of April (tentative, might be subject to change)*

Important tasks:

- *Implement the algorithm using OpenCL and a GPU*
- *Send the code and training diary to the teacher and assistants via email*
- *Start the optimization of the OpenCL kernels required in the next phase*

This phase is dedicated to the implementation of the code in OpenCL in a way that can run on a GPU. The code should be implemented in a way that the most of the parameters (such as e.g. the window size) can be easily changed. This implementation will be used by you as a reference to measure the performance of your optimizations, and the results and computation times of the next phases should be compared to ones obtained here.

To proceed with the implementation, you can follow these steps:

- Read the original images into host (CPU) memory (im0.png and im1.png). Use the ready C-implementation as the starting point for you host-code, but do not overwrite the C-implementation.
    - You are free to use existing host-codes from SDK-example projects or from the internet freely. Just cite where your implementation is from. Note that you still need to **understand** what you are doing
        - E.g. there is a vast amount of examples in *AMDAPPSDK*-folder in the workstations in TS135

- Find out at least the following parameters of your GPU using the *clDeviceInfo*-functions and report them in your final report
    - `CL_DEVICE_LOCAL_MEM_TYPE`
    - `CL_DEVICE_LOCAL_MEM_SIZE`
    - `CL_DEVICE_MAX_COMPUTE_UNITS`
    - `CL_DEVICE_MAX_CLOCK_FREQUENCY`
    - `CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE`
    - `CL_DEVICE_MAX_WORK_GROUP_SIZE`
    - `CL_DEVICE_MAX_WORK_ITEM_SIZES`

- Read the images (original .png-files) into the device as **image-objects**.

- Implement a kernel or kernels for resizing and grey scaling the images

- After resizing and grey scaling, using buffer-objects can be more convenient. NOTE! You only need to read the buffers/images back to the host, if you plan to use them on the host or another device, otherwise keep them on the device in order to avoid unnecessary and costly data transfers.

- Implement the actual ZNCC algorithm. Once the algorithm works proceed to implementing the post processing. You are free to divide the work into multiple kernels. Whatever in your opinion is the best approach, but justify the chosen approach in the final report.
    - Notice that transferring data between host and device is costly, so minimize data transfers

- Benchmark the execution times of each kernel using *clGetEventProfilingInfo* and the total execution time using the appropriate C-functions on the host-side and report them in your final report.

PHASE 5: OpenCL optimization

Checkpoint:

- *22nd of April (tentative, might be subject to change)*

Important tasks:

- *Optimize the OpenCL implementation to exploit the GPU architecture*
- *Write the final report*
- *Send the code final report and training diary to the teacher*
- *Agree with the teacher on a time for the exit interview and grade assignment*

This phase is dedicated to the optimization of the OpenCL code in a way that is able to exploit the architectural features of the GPU. This implementation is the final one that would be mainly used for the evaluation of the course and grades. In addition, this phase includes the writting of the final report and the documentation of the code and coding process.

To proceed with the optimization of the code, you can follow the next steps:

- Make a copy of the implementation before proceeding to the optimization phase! Use this implementation as the basis for optimization for both desktop (and optionally mobile) implementations. Optimize your code. All vendors have their own optimization guides with good examples. Your optimizations should take into account some of the following strategies:
    - Vectorization
    - Full memory architecture utilization (local memory)
    - Memory coalescing
    - Reducing the register usage
    - …

- Report the execution times and the CPU processor load of the optimized code in your final report
- Compare the execution times and the processor load of your final implementation to the ones obtained in the previous phases of the exercise
    - Write a reasoning about the different performance gains and their causes
    - Propose other strategies that could be made for further optimizations

PHASE 6 version 1 (*Optional*): OpenCL on mobile devices

Tasks:

- *Borrow and ORDROID platform from the teacher/assistants*
- *Port, test and optimize the OpenCL implementation for the mobile environment*

This optional phase is dedicated to the utilization of OpenCL in mobile environments. This extra implementation will grant you at least one extra point in the final grade.

To proceed with the mobile implementation, you can follow the next steps:

- Contact the teacher or assistants in order to borrow a mobile platform from them (ORDROID). The ORDROID board has a working environment installed. If you plan to use a different platform that you can provide (e.g. a compatible mobile phone), inform the teacher to plan accordingly.

- Familiarize yourself with the particularities of the mobile GPUs.
- Test your code in the mobile environment and report the execution times.

- Make optimizations taking into account the architecture of the mobile platform.

- Report the results after the optimization and compare with the non-optimized code.

PHASE 6 version 2 (*Optional*): Efficient use of memory hierarchy

Tasks:

- *Test and optimize the OpenCL implementation to use efficiently the multilevel memory hierarchy exposed in the framework (local, private global memory)*
- *Benchmark the memory transfers and possible overhead*

This optional phase is dedicated to the efficient utilization of the exposed memory hierarchy by OpenCL. This extra implementation will grant you at least one extra point in the final grade.

To proceed with the implementation, you can follow the next steps:

- Contact the teacher or assistants in order to inform them that this is the path you have chosen for the phase 6.

- Familiarize yourself with the memory hierarchy of your particular device and how it is implemented in OpenCL

- Make optimizations by efficiently use the memory architecture, including tiling in local memory

- Test your code report the execution times.

- Report the results after the optimization and compare with the non-optimized code.

FINAL REPORT, TRAINING DIARY, GRADING, DEADLINES, HELP

The registration of the course is done as follows:

- Register yourself in WebOodi in order to get course emails and credit points.

- Find yourself a partner to create a group of two students
    - Individual groups of one person are allowed, but the workload remains the same
    - If you are unable to find a partner, contact the teacher and we will do our best to find you a suitable one.

- Register the group by sending the group members name by email to the assistant. This will in practice be the requirement of CheckPoint 1 and will "officially" start your work on the exercise.

As a part of the exercise, you will write a training diary:

- After each checkpoint, send the training diary to the course assistant
- The training diary should only include
    - Number of hours done by every student, including self-studying hours
    - Short description of tasks done by hourly basis

The grading of the course is based on self-assessment. When the exercise is completed, you will be invited for an **Exit Interview** and receive 5 ECTS:

- There are no exams:
    - Each returned exercise will be either accepted or rejected
    - When accepted, a grade 1-5 will be given depending on the overall quality of the work
    - The grade will be based on the mutual agreement and understanding of several factors, including documentation, reasoning, optimization and elegance of solutions
    - The final grades are negotiable during the Exit interview
    - The grades can be the same for both or different for each group member in case the workload has been different during the coding and documentation process
    - If you are aiming for a higher grade than the one decided, a suggestion on how to achieve the desired level will be given

The grades will roughly follow this scale:

- **GRADE 1:** Working implementation **COMPULSORY !!!**
    - *C/C++ implementation and benchmark*
    - *OpenCL-implementation and benchmarks for GPU*
    - *Training diary and final report*
- **GRADE 2**: Complete documentation, comments, code:
    - Design, comments, structure, detailed time management
- **GRADES 3 & 4**: Optimization of your implementation(s)
    - Vectorization, utilizing the full memory architecture, …
    - Profiling
- **GRADE 5**: Mobile implementation
    - Comparisons with desktop implementation

In addition to the grades, there are two modifiers:

- You will get one point less if you miss more than one deadline (CheckPoints).
- You will get one extra point if you use any inventive solution, or you exhaustively explore some of the possibilities one of the phases, e.g.
    - Vectorization with SIMD during the multicore coding
    - Complete profiling of the timing from each part of the algorithms
    - Exhaustive testing of optimization techniques with analysis of the impact
    - Evaluation on multiple platforms (e.g. ARM, or GPUs from different vendors)
    - Evaluation on multiple operating systems (Windows and Linux)
    - …

The tentative deadlines for the course are as follows:

– **January 11th**:

Initial lecture followed by a period of self-study and framework installing

– **January 21st**:

Groups formed and registered and exercise instructions published

– **February 25th**

1st checkpoint: C-implementation

– **March 16th**

2nd checkpoint: Multi-threaded C implementation

– **April 6th**

3rd checkpoint: OpenCL implementation with some optimization

– **April 22th**

Final checkpoint: Fully optimized OpenCL code, mobile implementation, report.

– **May 13th**

Last possible day to return the exercises: Further work could only be extended with explicit permission from the teacher, and help and support is not guaranteed.

Several days before the deadlines and checkpoints, we might hold a **non-compulsory** "group" session in a classroom to discuss problems and strategies face to face. You are free to meet the teacher(s) there and share with them and other students your questions, suggestions, strategies or ideas.

In case of doubt, remember that we are here to help. Do not hesitate contacting us. We will always do our best to point you to the right direction. To contact us:

- Try to find a solution online. Someone has probably faced the same problems. Check the course webpages (Noppa) and look for the F.A.Q. (if/when available).
- If it does not help, email us with your problem
    - Always attach the code, results and possible error messages
    - Specify the problem as much as possible, so it can be reproduced
- We will answer with debugging strategies and suggestions of tests:
    - You SHOULD try these tests that are suggested
    - If everything else fails, we will set up a time to meet you