Assignment 5

Multiprocessor Programming 521288S

Miika Sikala 2520562, msikala18@student.oulu.fi

Task: Parallel Implementation of Stereo Disparity Algorithm (in C/C++) using pthreads and OpenCL along with profiling. There are two sub-task involved one for pthreads and the other OpenCL-GPU implementation.

Expected results: A working version of the implementation, a brief report (max 2-3 pages), saved output images all together in the form of a compressed folder (.zip file)

The report should contain about the task solved, brief description of your implementation, comparison of profiling information for pthread and OpenCL implementations and the final screenshot of your outputs asked to be displayed under the assignment.

Detailed instructions regarding the task are provided under assignment_5.pdf. Kindly go through them once and get back to me in case there are any questions.

Introduction

The threaded implementation was implemented simply by dividing the image to equal horizontal slices. Each thread would work on a separate strip at the same time, as shown in Figure 1.

Figure 1. Implementation using Pthreads.

Justifications: work group size used is 16. This is optimized to my GPU, which has `MAX_WORK_GROUP_SIZE` of 256 (16x16). Also, the threaded implementation is optimized for my 8-core CPU. This can be changed by changing the `NUM_THREADS` definition in “Application.hpp”.

Usage

Requirements: since windows doesn’t support phtreads natively, you may have to install it manually. I was able to use pthreads out of the box with g++, but not with Visual Studio.

To change the target device, you can change the `COMPUTE_DEVICE` flag in “Application.hpp”. The options are `TARGET_NONE` (no parallelization), `TARGET_PTHREAD` (CPU parallelization using pthread) `TARGET_GPU` (OpenCL on GPU) and `TARGET_CPU` (OpenCL on CPU). You must re-compile the program for the change to take effect.

The application takes two to six input parameters, which are the image file names for the left and right images, and optionally some calculation parameters. See the example below, which uses image files `img/im0.png` and `img/im1.png`.

A makefile is provided for compiling with g++.

<u>Compiling using *g++*:</u>
```
make
```

<u>Running:</u>
```
stereo.exe img/im0.png img/im1.png 15 55 8 4

# Arguments: LEFT_IMG RIGHT_IMG [WINDOW_SIZE=9] [MAX_SEARCH_DIST=32]
#            [CROSS_CHECK_THRESHOLD=8] [DOWNSCALE_FACTOR=4]
```

Testing and benchmarking the implementations

The results (intermediate and final) are stored in “img” folder. Figure 2 shows the execution of the program for three different compute targets.

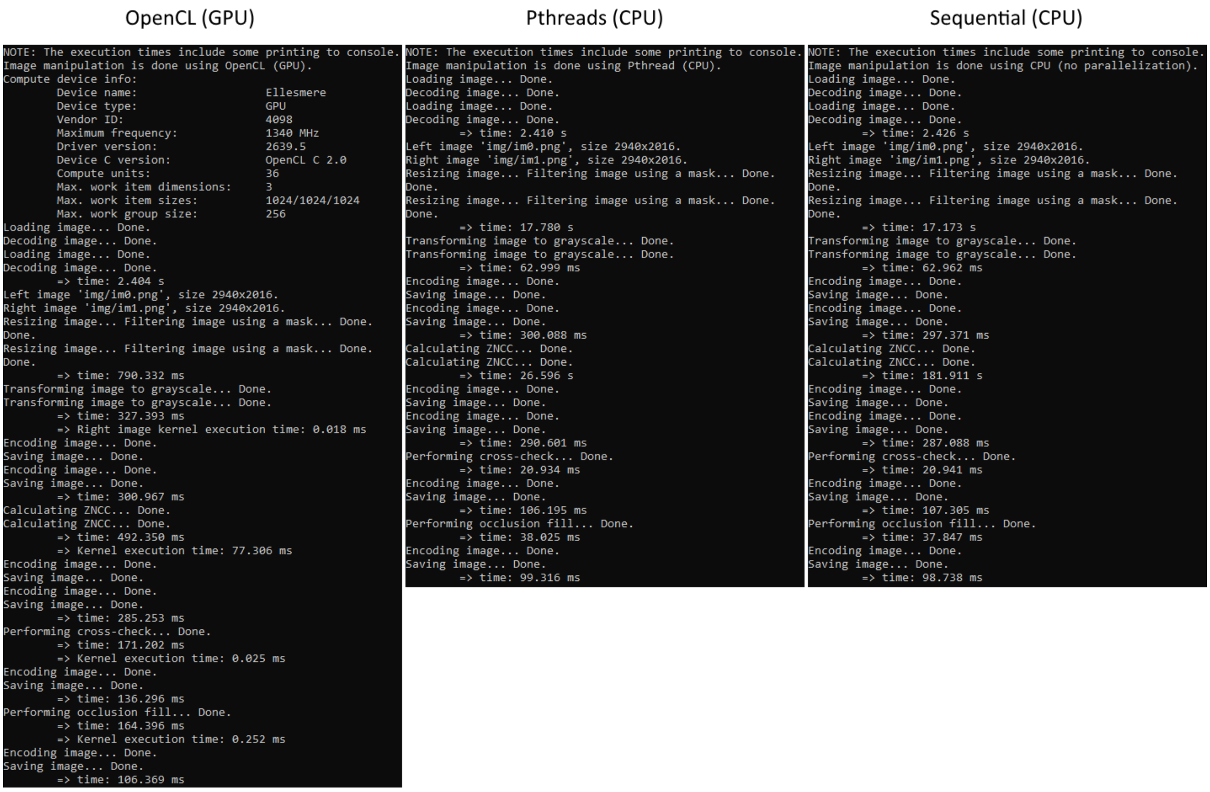

Figure 2. Execution of the program using all three compute targets (OpenCL, Pthreads, sequential).

The benchmarking was done using a simple occlusion fill algorithm, which simply seeks the nearest non-zero pixel on the left. However, another occlusion fill kernel was implemented, which seeks the nearest non-zero pixel in square-like spiral.

Used configuration:

window size: 15
maximum search distance: 55
cross-checking threshold: 8
downscaling factor: 4

Execution times:

Operation	CPU (sequential)	CPU (Pthreads)	GPU (OpenCL)	Kernel only
Load + decode	2.431 s	2.426 s	2.435 s	-
Resize	17.566 s	17.017 s	748.777 ms	-
Convert to grayscale	62.078 ms	62.659 ms	294.004 ms	0.034 ms
Encode + save	298.543 ms	300.209 ms	301.397 ms	-
Calculate ZNCC	**181.237 s**	**26.474 s**	**469.242 ms**	156.45 ms
Encode + save	333.823 ms	334.044 ms	285.672 ms	-
Cross-check	**21.999 ms**	21.991 ms	**148.863 ms**	0.026 ms
Encode + save	122.874 ms	124.158 ms	137.731 ms	-
Occlusion fill	**38.629 ms**	39.909 ms	**144.385 ms**	0.200 ms
Encode + save	109.111 ms	111.126 ms	105.542 ms	-
Total	**202.221 s**	**46.911 s**	**5.071 s**	**156.710 ms**

As can be seen, the threaded implementation already improves ZNCC calculation significantly: the eight-thread implementation of ZNCC calculation took around 15 % of the sequential implementation's time. Let alone the OpenCL implementation, which only took 0.26 % of that!

Kernel only shows the kernel execution time when applicable. Notice the significant overhead of transferring the images between host and device memory (total of 900 ms). Using OpenCL in cross-checking and other very simple algorithms actually worsens the performance due to the transfer times. This is one point of improvement in the future.

The *benchmark.txt* file shows more details of the benchmarking.

Discussion

The current implementation requires that the intermediate calculation results are transferred between the host and compute device memory multiple times. This is due to the modularity of the implementation (easy to make different computations). If we wanted to make the implementation as optimal as possible, we could minimize the number of transfers between the memories. This may be done in future implementations.

Reporting

Task	Hours
Pthreads implementation	3 h
OpenCL implementation	8 h
Refactoring	1 h
Benchmarking & writing the report	2 h
Total	**14 h**