**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 . Implementation using Pthreads.

**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 the beginning of main.cpp. 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 five 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++.

Compiling using *g++*:  
make

Running:  
stereo.exe img/im0.png img/im1.png 15 55 4

# Arguments: LEFT\_IMG RIGHT\_IMG [WINDOW\_SIZE=9] [MAX\_SEARCH\_DIST=32] [DOWNSCALE\_FACTOR=4]

**Testing and benchmarking the implementations**

~~The correctness of the implementation was tested by checking the results manually. The screenshots of the outputs of the program are shown below. The resulting images are found in “img” folder.~~

~~Figure 2 shows the execution on CPU. The grayscale conversion and filtering were done using OpenCL in GPU, but other parts are done sequentially. The settings used are:~~

~~maximum search distance: 55  
window size: 15  
cross-check threshold: 8~~

Text

Description automatically generated

Figure 2. Execution of the program.

Execution times:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operation** | **CPU (sequential)** | **CPU (Pthreads)** | **CPU (OpenCL)** | **GPU (OpenCL)** |
| Load + decode | 2.417 s |  |  | - |
| Resize (1/4x) | 607.942 ms |  |  | - |
| Convert to grayscale | - |  |  | 0.264 ms |
| Encode + save | 301.621 ms |  |  | - |
| Calculate ZNCC (both) | **174.390 s** |  |  | - |
| Cross-check | 20.924 ms |  |  | - |
| Occlusion fill | 37.894 ms |  |  | - |
| **Total** |  |  |  |  |

~~Notice that the ZNCC calculation took extremely long time, due to having sequential implementation.~~

Figure 2 shows the resulting image.

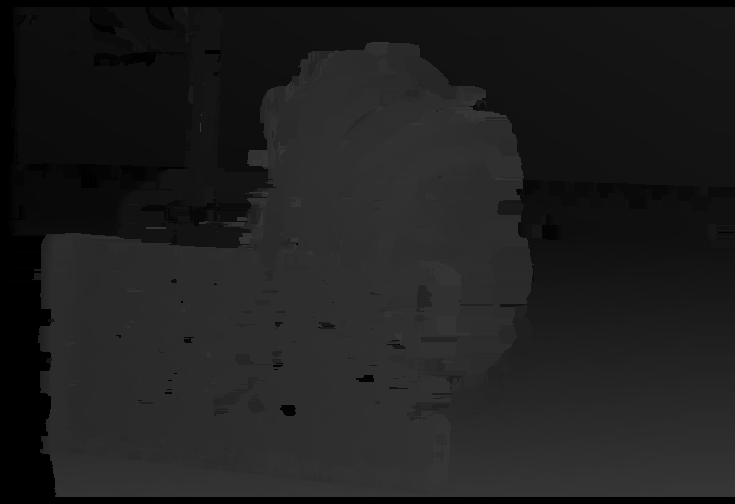


Figure 3. Resulting image.

**Memory**

~~The memory consumption was recorded using Visual Studio debugger. The maximum memory was consumed during ZNCC calculation, which required the left image and right image, as well as the disparity map to reside in the memory at the same time. This consumed total of~~ **~~254 MB~~**~~.~~

~~CPU usage during ZNCC was 12,5 %. This is very inefficient. Since I have 8-core CPU and the implementation only used 1 core, a parallel execution would significantly speed up the calculation.~~

**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.

**Also, as a small note, the disparity map should be scaled to the range of 0-255 based on the minimum and maximum values in the image.**

**Also: using single channel image in calculations would be more efficient, CPU-wise, but especially memory-wise.**

**TODO: (1) SCALING, (2) ONLY COMPILE KERNEL ONCE (3) THRESHOLD TO ARGS (4) NO UNNECESSARY TRANSFERS TO DEVICE!**

Reporting

|  |  |
| --- | --- |
| **Task** | **Hours** |
| Pthreads implementation | 3 h |
| OpenCL implementation | 7 h |
| **Total** | **10 h** |