2022/05/13

Milestone 2: Status Update

Group 3
Michael Altshuler (ALTMIC003)
Taine de Buys (DBYTAI001)
Julian Zille (ZLLJUL001)

Progress Overview:

The team met on Saturday the 7th of May to begin development on the project and ensure we were all on the same page. It was decided that github would be used as a means to store all code and documentation in 1 central repository, allowing for consistent access to the project, regardless of location. In addition, an overleaf page was made for the final project report and the initial skeleton for the report was constructed. We chose to tackle the median filter issue first in isolation, and delegated the work accordingly: Taine was responsible for implementing the sequential implementation of the algorithm in C++ to be used as a golden standard as well as the python preprocessing, Julian would tackle the kernel of the Open CL implementation, and Mike would be responsible for the setting up of the Open Cl implementation as well as producing the block diagram of the system. Each team member agreed that we would all work together on the write up for the project.
The team is attempting to have the Median Filter implementation finished by Sunday the 15th of May so that we can all tackle the Edge detection algorithm early next week in order to finish the assignment well before demonstration.

Current Status:

- Set up Git repository and overleaf for project report - established a means to collaborate on code.
- Finished preprocessing python script. This involves converting a 2D RGB array into a 1D array and writes it to a .txt file (including the dimensions of the image). The .txt can be read into an array in the main.cpp program. OpenCL has been configured in ‘main.cpp’ (creating buffers, established number of work items etc.). The kernel can create a 1D array where each element is the sum of the RGB elements of a pixel in the original image.
- Significant progress has been made with the sequential implementation of Task 1.

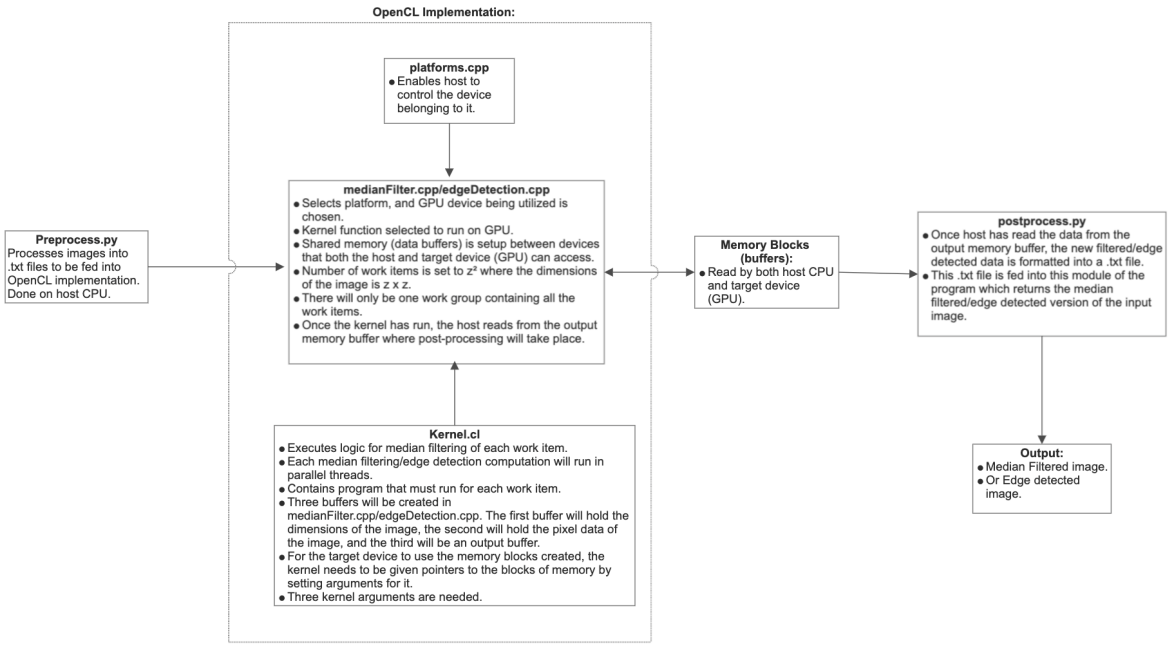
Still to do:

- Sequential and parallel implementations of task 2.

2022/05/13

- Finish the post processing python script.
- Finishtask 1’s kernel to find the median of the surrounding pixel’s for a specific pixel (work-item).
- Finish the report

Block Diagram:



The block diagram above displays the implementation of both the Median Filter and the Edge Detection OpenCL programs. As seen, the image undergoing either edge detection or median filtering undergoes a pre-processing stage first whereby the image is converted into a text file format which is then fed into the respective OpenCL programs. Once the processing on the text file data is complete, a new text file containing either the median filtered or edge detected data is fed into a post-processing python script that reconstructs the filtered/edge detected image.

2022/05/13

Evidence of Work:

- Preprocessing

First we have a screenshot of our preprocessing python script:

```python
import numpy as np
import sys
from PIL import Image

def readimage(imagename):
    print("Reading image " ,str(imagename))
    img = Image.open(filename)
    width , height = img.size
    pixelarray = np.array(img)
    pixelarray=pixelarray.reshape(-1)

    # Adding dimensions to beginning of array
    pixelarray=np.insert(pixelarray,0,height)
    pixelarray=np.insert(pixelarray,0,width)
    print("Image had width of" , width , " and height of" , height)

    img1d = np.reshape(pixelarray,-1)

    outfile = imagename.split('.')[0] + ".txt"

    np.savetxt(outfile, img1d, fmt = '%d', delimiter = ",")
    print( "Output written to .txt file \n ")

#functionality allowing for multiple files to be read in
print("Welcome to our image processor. ")
running = True
while (running == True):
    filename = input("Please enter name of file to process or type exit: \n")
    if (filename == "exit"):
        quit()
    else:
        try:
            imagearray=readimage(filename)
        except FileNotFoundError:
            print("The file does not exist, try again. ")
```

Users can run the script and insert names of files they want to be preprocessed for the median filter through their command line:

```
(venv) taine_debuys@LAPTOP-8CM3Q26B:~/Taine/YODA/Yoda-Project-$ python3 preprocess.py
Welcome to our image processor.
Please enter name of file to process or type exit:
yoda.jpg
Reading image  yoda.jpg
Image had width of 220  and height of 229
Output written to .txt file

Please enter name of file to process or type exit:
exit
(venv) taine_debuys@LAPTOP-8CM3Q26B:~/Taine/YODA/Yoda-Project-$
```

2022/05/13

- Golden Standard

Here is a screenshot of some of the sequential code for the golden Standard:

```
medianFilter:: medianFilter(std::string filename):
        filename(filename)
{}


bool medianFilter::readFile(std:: string filename)
{
        std::ifstream inputfile(filename) ;
        if (!inputfile){
                std::cout << "The file " << filename << " does not exist. Try again please. " << std::endl ;
                return false ;
        }

        std::string text ; //reads in lines from input buffer
        getline(inputfile , text) ; //reads in dimensions

        std::cout<< text << std::endl ;
```

- medianFilter.cpp
  The following code displays the medianFilter.cpp file used to set up the OpenCL environment, and configure the platform and target device on which the kernel program will be executed. The platform used enables the NVIDIA SDK used for this implementation and the medianFilter.cpp file further configures that the executable will be run on a GPU.

```
int main(void)
{
    //Read pixels from .txt file
    string filename;
    cout << "Enter .txt filename: ";
    cin >> filename;

    ifstream file(filename);
    uint w;
    uint l;
    if (file.is_open())
        file >> w; // width
        file >> l; // length
        int arr[w*l*3];
        int arr_fil[w*l*3];
        for (int i=0;i<w*l*3;i++){
            file >> arr[i];
        }
    cout << ".txt file imported to 1D array" << '\n';
```

  - Read values from .txt array:
  - Create buffers for host and kernel to read/write to:

```
    RGB_buffer = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR, sz*sizeof(int), arr, &err);
    width_buffer = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR, sizeof(uint), &w, &err);
    length_buffer = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR, sizeof(uint), &l, &err);
    //Stores outputs from kernel so host can retrieve what the kernel has calculated.
    grayscale_buffer = clCreateBuffer(context, CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR, global_size*sizeof(int), grayscale, &err);
    filtered_buffer = clCreateBuffer(context, CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR, global_size*sizeof(int), filtered, &err);
```

- median.cl
  The following displays the kernel that will be executed on the target device (GPU) for the median filter operation. A program handler is created for this file in the medianFilter.cpp file , which enables it to be compiled on the target device itself.

```
//Kernel for OpenCL implementation.

__kernel void MedianFilter(__global int* arr, __global uint* w, __global uint* l, __global int* grayscale, __global int* filtered){
    //work item and work groups numbers
    uint n = get_global_id(0); //Work item ID
    //int workGroupNum = get_group_id(0); //Work group ID
    //int localGroupID = get_local_id(0); //Work items ID within each work group

    //memory buffers
    //int arg1 = *test_arr;
    //int arg2 = *argument2;
    uint W=*w;
    uint L=*l;
    uint col=n%W;
    uint row=(n-col)/L;

    //short calculation: work Item Number x argument 1 + argument 2
    //This is a test output to see if the Kernel produces an output.
    //output[global_addr] = global_addr*arg1 + arg2;
    //printf("output[%d] = %d\n", workItemNum, output[workItemNum]);
    //printf("w = %d, l=%d\n",W,L);
    grayscale[n]=arr[n*3]+arr[n*3+1]+arr[n*3+2];
    barrier(CLK_GLOBAL_MEM_FENCE); // Wait for summation of all RGB pixels
    uint med[9];
    //printf("Executed\n");
    if (row!=0 && col!=0 && row!=L-1 && col!=W-1){
        //printf("Executed\n");
        for (int i=-1;i<2;i++){
            med[i+1] = grayscale[n+i];
            med[i+4] = grayscale[n-W+i];
            med[i+7] = grayscale[n+W+i];
        }

        //filtered[n]=(int)(med[4]+med[5])/2;

    }
```