PAP - Assignment - 03

Roham Jabbari - a12211638

In this assignment, we were tasked to implement an equation that iterates over a matrix for 100 times in both sequential and parallel (using OpenCL).

The code is provided in four files (also on ALMA in a3 folder):

- assignment-float.c and assignment-float.cl with float variables
- assignment-double.c and assignment-double.cl with double variables

In this report some comparisons on different optimisations and different inputs are provided. Two equations were used but only the main one will be covered in this report:

- Main equation provided in the assignment file:

$$a_{ij}^{t+1} = \sqrt{a^t_{(i-1)(j-1)} + a^t_{(i-1)(j+1)} + a^t_{(i+1)(j+1)} + a^t_{(i+1)(j-1)}} \cdot \sqrt{a^t_{(i)(j-2)} + a^t_{(i-2)(j)} + a^t_{(i)(j+2)} + a^t_{(i+2)(j)}}$$

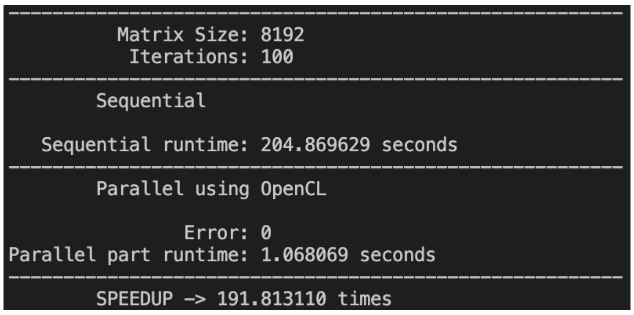
- Alexander Hecke's equation on Moodle (Secondary equation):

$$\sqrt{\frac{(a+b+c+d)}{4}} \cdot \sqrt{\frac{(e+f+g+h)}{4}} = \sqrt{\frac{(a+b+c+d)\cdot(e+f+g+h)}{16}} = \frac{\sqrt{(a+b+c+d)\cdot(e+f+g+h)}}{4}$$

Better precision and no inf or nan values was observed on Alexander's equation. The speedup was the same.

Sequential Code:

The sequential code was provided beforehand. Therefore, it will be redundant to explain anything. However, it is worth mentioning that in the sequential code the matrix is navigated in a row-major order. The sequential part finishes faster by using row-major, so that is what will be used for a fair comparison. (Compared to column-major sequential code, speedup of 204x was achieved).

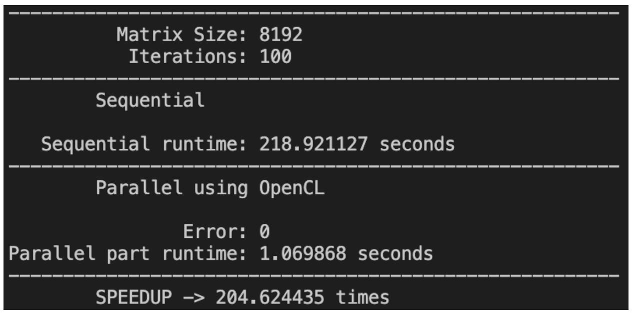


```
----------------------------------------------------
         Matrix Size: 8192
          Iterations: 100
----------------------------------------------------
       Sequential

  Sequential runtime: 204.869629 seconds
----------------------------------------------------
       Parallel using OpenCL

               Error: 0
Parallel part runtime: 1.068069 seconds
----------------------------------------------------
       SPEEDUP -> 191.813110 times
```

Sequential with column-major on Secondary Equation



```
----------------------------------------------------
         Matrix Size: 8192
          Iterations: 100
----------------------------------------------------
       Sequential

  Sequential runtime: 218.921127 seconds
----------------------------------------------------
       Parallel using OpenCL

               Error: 0
Parallel part runtime: 1.069868 seconds
----------------------------------------------------
       SPEEDUP -> 204.624435 times
```

Sequential with column-major on Main Equation

Parallel Code:

For the parallel section of this assignment, OpenCl in C-language was used.

- Kernel The main calculation happens in a kernel (located in assignment03.cl) with two arguments. The arguments are two matrices:
    - The mainMatrix, which is the last last filled matrix.
    - The secondaryMatrix, which is the matrix to be filled using the mainMatrix.

  Rest of the kernel code is self-explanatory.
- The iteration part and buffer swap was moved out of the kernel to avoid data race.
- For the introduction of buffers CL_MEM_USE_HOST_PTR flag was used since we are operating with pointers to access matrix elements.

Optimizations:

- **Building Options:**
    - -cl-finite-math-only: Allow optimizations for floating-point arithmetic that assume that arguments and results are not NaNs or ±∞.
    - -cl-mad-enable: The *mad* computes *a * b + c* with reduced accuracy.
    - -cl-single-precision-constant: Treat double precision floating-point constant as single precision constant.
    - -cl-denorms-are-zero: single precision denormalized numbers may be flushed to zero.
    - -cl-unsafe-math-optimizations: Allow optimizations for floating-point arithmetic for unsafe arguments and also may violate IEEE standards.

  Used building options did not cause a major speedup in the parallel section, the speedup to negligence of precision and regulations is not desirable.
- **Native Math Functions:**
    - Instead of normal square root function, native_sqrt() was implemented. This did not cause a major speedup. However, it makes more sense to use native functions when using gpu.
    - native_sqrt() was not used in float version to get smaller error value.
- **Work-Groups:**
    - For this task, three local work sizes of 8, 16, 32 were used. Differences will be explored in comparison.
    - Does not work with 64. (Exceeds existing work group numbers)
- **Different Cache Order Layout:**
    - It was noticed that parallel systems with OpenCL work remarkably with column-major order layout rather than row-major. For this assignment, the parallel section works with a column-major order layout and the sequential part with a row-major order layout since the goal is to find a fair speedup.
- **Double or Float:**
    - Using double will result an outstanding precision and no *inf* values, however, it will be too slow since gpu has more units allocated for float values. For the main equation, float will be used with lower iterations to avoid getting *inf* values.

Comparisons:

- Main Equation:

- Different matrix sizes with the same local work size
  - Double
  - Iterations: 70
  - Local Work Size: 32
  - Matrix Sizes: 1024, 2048, 4096, 8192

Matrix Size:	**1024**	**2048**	**4096**	**8192**
Sequential Time:	0.64	2.41	9.28	36.65
Parallel Time:	0.024	0.094	0.37	1.50
Speedup:	26.43	25.50	24.58	24.28

- Different matrix sizes with the same local work size
  - Double
  - Iterations: 100
  - Local Work Size: 16
  - Matrix Sizes: 1024, 2048, 4096, 8192

Matrix Size:	**1024**	**2048**	**4096**	**8192**
Sequential Time:	0.64	2.39	9.28	37.35
Parallel Time:	0.023	0.092	0.36	1.47
Speedup:	27.02	25.83	25.17	25.27

- Different matrix sizes with the same local work size
  - Double
  - Iterations: 100
  - Local Work Size: 8
  - Matrix Sizes: 1024, 2048, 4096, 8192

Matrix Size:	**1024**	**2048**	**4096**	**8192**
Sequential Time:	0.64	2.45	9.52	36.60
Parallel Time:	0.03	0.12	0.48	1.93
Speedup:	20.79	20.22	19.71	18.92

- Different matrix sizes with the same local work size
  - Float
  - Iterations: 70
  - Local Work Size: 32
  - Matrix Sizes: 1024, 2048, 4096, 8192
  - Error Margin: 0.001

Matrix Size:	1024	2048	4096	8192
Sequential Time:	0.83	2.28	9.03	35.86
Parallel Time:	0.012	0.49	0.19	0.78
Speedup:	64.18	46.02	46.20	45.56
Error (%)	19	10	5	2

- Different matrix sizes with the same local work size
  - Float
  - Iterations: 70
  - Local Work Size: 16
  - Matrix Sizes: 1024, 2048, 4096, 8192
  - Error Margin: 0.001. Error did not change.

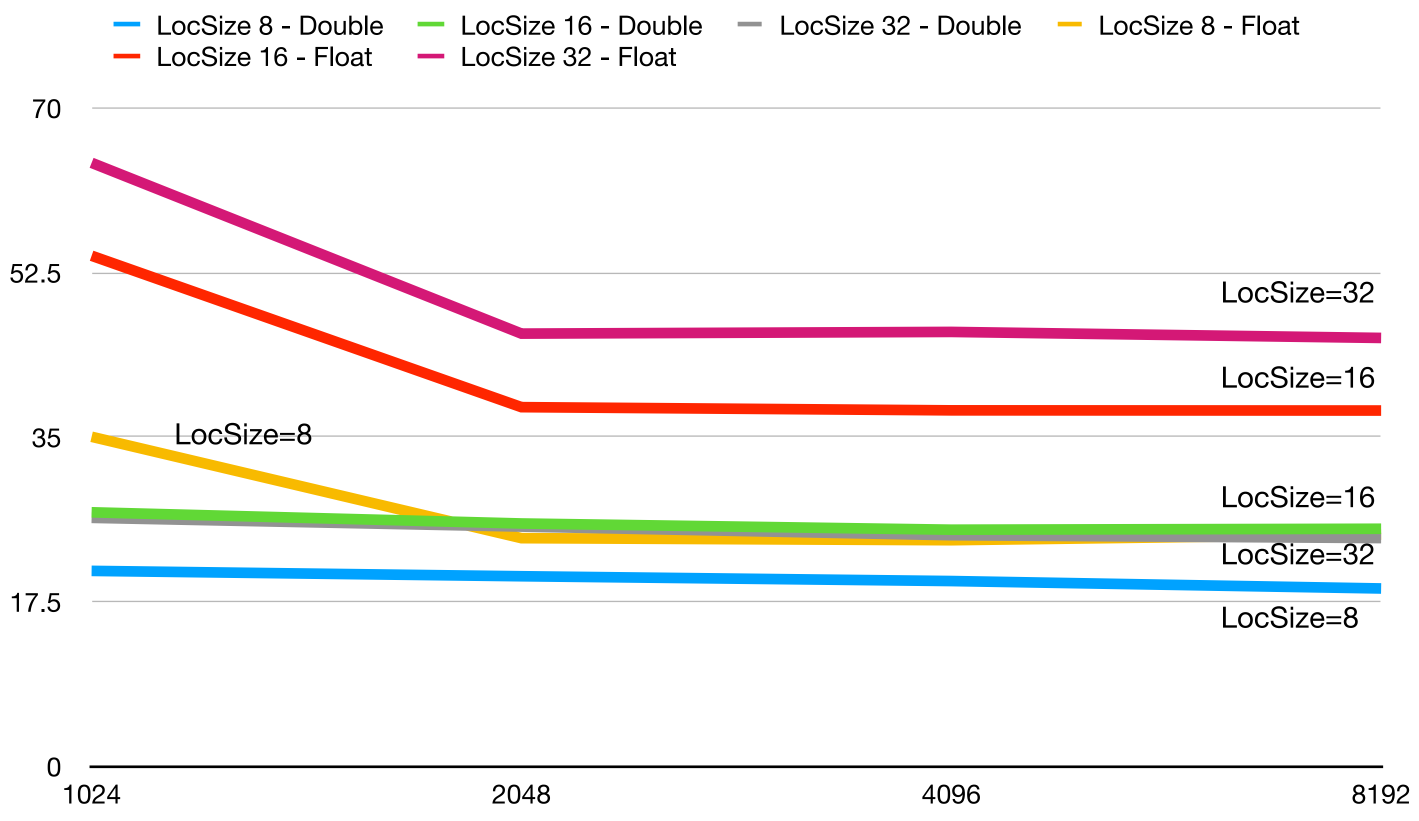
Matrix Size:	1024	2048	4096	8192
Sequential Time:	0.84	2.28	8.99	35.86
Parallel Time:	0.015	0.05	0.23	0.94
Speedup:	54.30	38.20	37.86	37.84

- Different matrix sizes with the same local work size
  - Float
  - Iterations: 70
  - Local Work Size: 8
  - Matrix Sizes: 1024, 2048, 4096, 8192
  - Error Margin: 0.001. Error did not change.

Matrix Size:	1024	2048	4096	8192
Sequential Time:	0.84	2.27	9.01	36.87
Parallel Time:	0.024	0.094	0.37	1.49
Speedup:	35.05	24.27	24.05	24.66

- Speedup Table:

Matrix Size:	1024	2048	4096	8192
LocSize 8 - Double	20.79	20.22	19.71	18.92
LocSize 16 - Double	27.02	25.83	25.17	25.27
LocSize 32 - Double	26.43	25.50	24.58	24.28
LocSize 8 - Float	35.05	24.27	24.05	24.66
LocSize 16 - Float	54.30	38.20	37.86	37.84
LocSize 32 - Float	64.18	46.02	46.20	45.56



- **Build:**
  - Float:

    ```
    g++ -O3 -I/usr/local/cuda/include -L/usr/local/cuda/lib64/ -o float assignment-float.c -lOpenCL
    ./float
    ```

  - Double:

    ```
    g++ -O3 -I/usr/local/cuda/include -L/usr/local/cuda/lib64/ -o double assignment-double.c -lOpenCL
    ./double
    ```

Note: srun is not suggested since interactive version was requested.