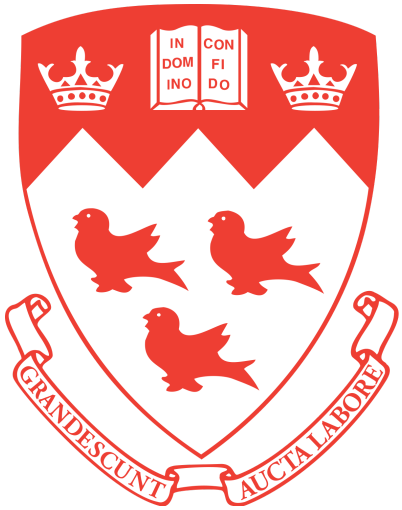
ECSE 420 - Parallel Computing
Fall 2020
Professor Zeljko Zilic

Rebecca Weill - 260804001
Benjamin Szwimer - 260804222
Group 36



Lab 1 - Logic Gates Simulation

October 19th 2020

1. Abstract

The goal of this laboratory experiment was to write code which models simple logic gates, parallelize the code through the use of CUDA and report on the various outcomes depending on which memory allocation technique was used. Three different implementations were tested, sequential, explicit memory allocation and unified memory allocation. The experiment was limited to the gates: AND, NAND, OR, NOR, XOR and XNOR. Finally, all implementations were compared based on their execution times and data migration was measured specifically for the explicit memory allocation technique.

2. Sequential

The sequential algorithm for logic gates uses a simple switch-case function in which every case represents a different logic operation. All operations occur on the CPU sequentially and none of the processing is done in parallel. Just like in the input file, every logic function is mapped to an integer. For every line in the input file, the function will compare the 3rd element of each line with the different cases, find a match and perform the proper bitwise operation on the first 2 elements of the line depending on the case (logic gate).

3. Explicit Memory Allocation

The explicit memory allocation technique, uses both CPU and GPU memory to function. Essentially the algorithm will copy input data from CPU memory, and copy it to GPU memory. The GPU program will then execute on GPU memory. The final results are then copied from the GPU memory back to CPU memory. The explicit memory allocation technique in practice is more efficient than the unified memory allocation technique. That is because an experienced programmer using the technique will be smarter in the way they implement the memory management over unified that does it implicitly. The gate logic was implemented using the same switch-case idea as previously mentioned.

4. Unified Memory Allocation

The unified memory allocation technique is ultimately the automated version of explicit memory management. It will actually lead to a performance deficit, because it cannot outperform expertly written manual data movement implemented by a programmer. Since there is so much that is unknown to the programmer and up to the actual mechanism, the efficiency and fast execution time is very difficult to control. In addition, if the unified memory allocation technique is not implemented properly it can slow the execution time even more. The gate logic was implemented using the same switch-case idea as previously mentioned.

5. Testing

Each implementation of the logic gates (sequential, explicit memory, unified memory) were all tested with 10,000, 100,000, and 1,000,000 threads (lines), and their respective execution times were recorded. Two trials were performed for each number of lines, and their average was obtained.

Number of lines	Trial 1	Trial 2	Average Execution and Data Migration Time (ms)
10,000	1.975936	1.841888	1.908912
100,000	16.884768	17.196032	17.0404
1,000,000	170.139679	169.474426	169.8070525

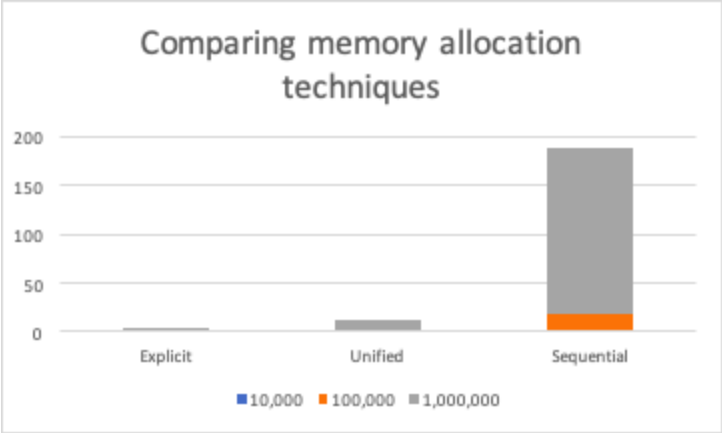
Table 1: Sequential

Number of lines	Trial 1	Trial 2	Average Execution and Data Migration Time (ms)
10,000	0.129024	0.161184	0.145104
100,000	0.472192	0.435264	0.453728
1,000,000	2.471392	2.298656	2.385024

Table 2: Explicit Memory Allocation

Number of lines	Trial 1	Trial 2	Average Execution Time (ms)
10,000	0.378880	0.213248	0.296064
100,000	1.492960	1.888960	1.69096
1,000,000	9.519968	9.394080	9.457024

Table 3: Unified Memory Allocation



Graph 1: Comparing memory allocation techniques by execution time in ms

6. Discussion

As shown by the tables above, the order of slowest execution time to fastest execution time is sequential → unified memory allocation technique → explicit memory allocation technique. The slowest goes to sequential because it must read through the file line by line, and this could take especially long when there are many computations because none of the information is processed in parallel. The fastest execution time goes to the explicit memory allocation technique because, as previously mentioned, this implementation is more controlled by the programmer implementing it rather than the unified allocation technique which is controlled by the mechanism itself, which will lead to a lot of unknowns. Unified has to perform a little more amount of work in order to do what the classic memory allocation (explicit) does.

7. Conclusion

In conclusion, between sequential, explicit memory allocation and unified memory allocation, the explicit memory allocation technique is preferred for its time efficiency and more controlled memory management.