ECSE 420 - Fall 2020

Parallel Computing

Lab 1 Report

Group 05

Group Members	**Student ID**
Nicolas Barreyro	260730549
Erion Hysa	260746699

Work Presented To:

Prof. Zeljko Zilic

McGill University

October 19th, 2020

Lab 1- Logic gates simulation

Assumptions:

- The GPU on which we ran these experiments is a NVIDIA Geforce GTX 1660 Ti (Laptop version) with 6GB memory.
- The number of lines (= #gates) passed as the second command line argument is used to determine the number of threads and, consequently, the number of blocks.
- For the parallel part of the lab:
    - #lines = #gates = #threads
    - #threads/block = 1024
    - #blocks = ceiling($\frac{\#threads}{\#threads/block}$) = $\frac{\#threads + \#threads/block - 1}{\#threads/block}$

Terminology:

- Host: CPU
- Device: GPU

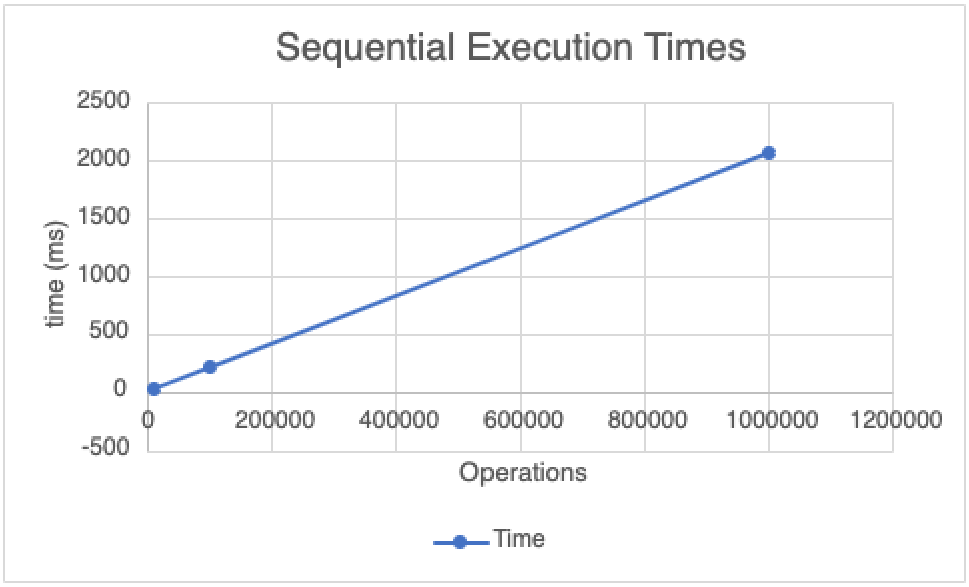
1. Sequential Logic Gate Simulation

To perform the input logic gates sequentially we initially extract the data from the input file. The arguments are <input_file_path> <input_file_length> <output_file_path>. Each line is read from the input file and then stripped by the ‘,’ character to create 3 arguments. The first is the value going into the logic gate, the second the second value and the third holds the id of the logic gate.

These values are passed into the computeGate() function to compute the output as an array. Based on the third value of the array we can then perform the correct logic operation on the two operands. The function returns 0 or 1 as a char which is then printed to the file with the use of fprintf. This is done for every line in the input file.

The time taken to perform the operations for the different files are below:

File Name	Time (ms)
10000.txt	22
100000.txt	208
1000000.txt	2065



We can see what we expect with sequential execution time which is in a linear fashion as each operation takes the same time but is loaded after the previous is finished. However the time to perform the operations is also quite high (compared to parallelized).

2. Parallel Logic Gate Simulation (Explicit Memory Allocation)

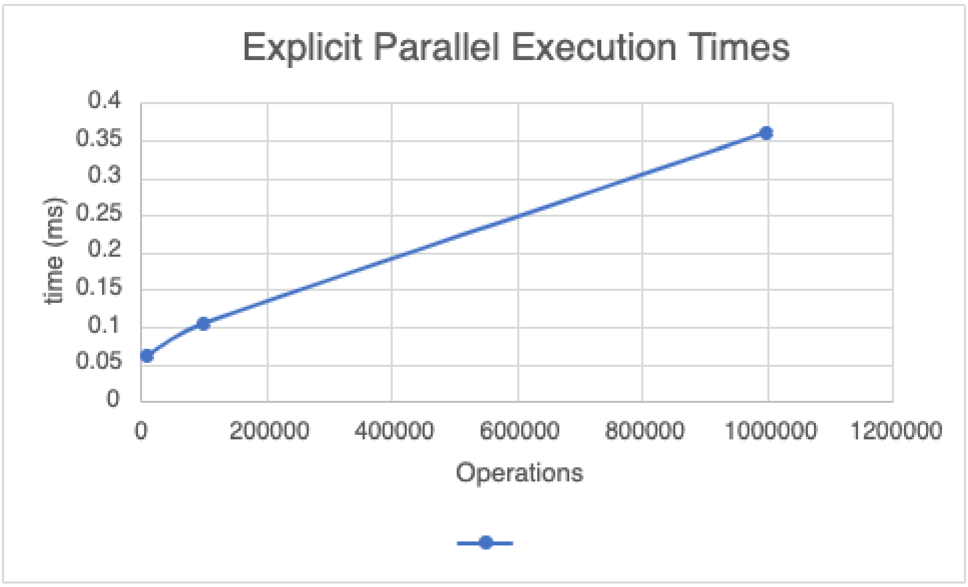
The threads per block were defined at 1024 for both versions of the parallel logic gate simulation. The gathering of information works in a similar way to sequential except we store all the inputs in an array before running the outputs in parallel.

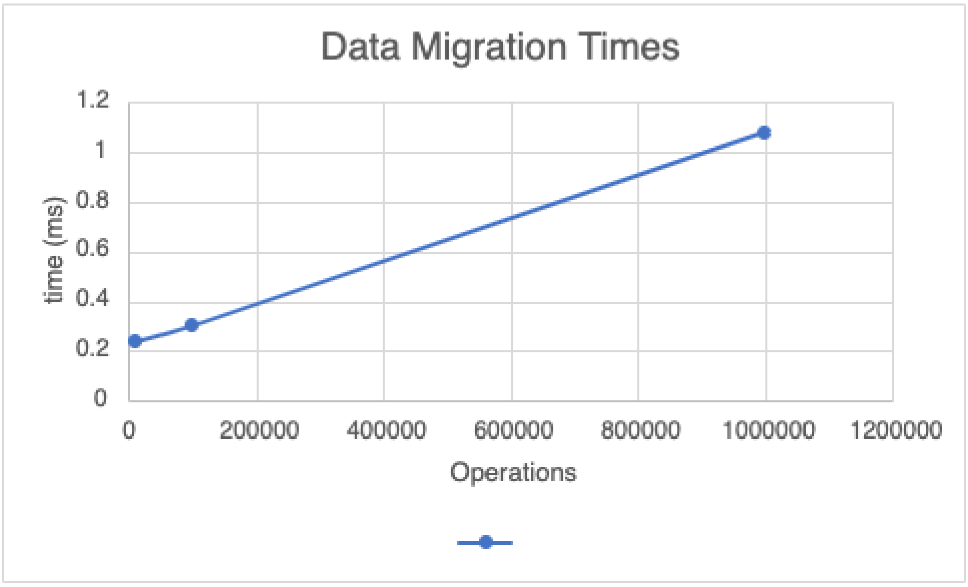
The host allocates two arrays (input/output) in the host memory using malloc() and two arrays in the device memory using cudaMalloc(). We know the size to allocate in memory as it is 3 characters per line for the inputs and 1 character per line for the output. The input values in the host memory are populated by going through the file one line at a time until the end. We then copy the input data from host to device and we launch the kernel.

To launch the kernel we need to calculate the number of blocks and in order to do so, we use the equation written in the Assumptions section of this report. The kernel function takes as arguments pointers to the input and output arrays allocated using cudaMalloc() and maximum size which is going to be #fileLines (= #gates). We then figure out the index for each thread instance using out_i = threadIdx.x + blockIdx.x * blockDim.x and. The thread index is mapped directly to the index of the output array (out_i) as there is only one character per line, but for the input array index (in_i), the thread index has to be multiplied by 3. If the thread index is smaller than the max size, we take the values from the device input array at index in_i and we use the same computeGate function as the sequential version to get the output and we save it to the device output array at index out_i.

To finish off, we copy the output data from device to host and we print to the output file entered as an argument to the command line and free up the space using the free() and cudaFree() functions. This is confirmed to be running correctly with the compareFiles function given to us.

File Name	Execution Time (ms)	Data Migration Time (ms)
10000.txt	0.061427	0.237152
100000.txt	0.104816	0.301632
1000000.txt	0.360790	1.080800





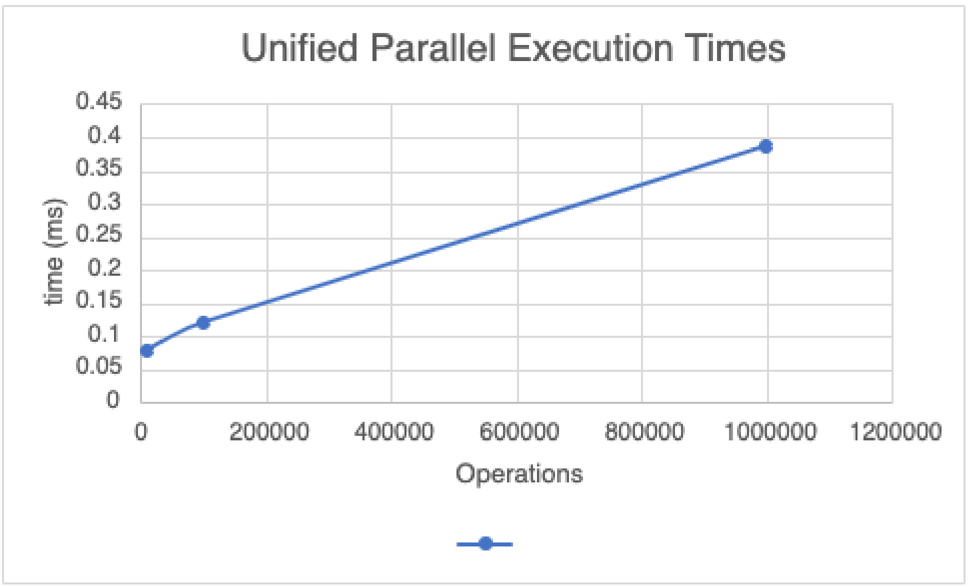
3. Parallel Logic Gate Simulation (Unified Memory Allocation)

The unified memory version of the parallel logic gate simulation is only different to the explicit memory allocation in the way that the memory is accessed by the host and the device. In this version, the host allocates two arrays (one for input of size 3 bytes/gate * #gates and one for output of size of size 1 byte/gate * #gates) using the CUDA function cudaMallocManaged(). By using this function, there is no need to copy the input data from host to device and output data back from device to host. These pointers (arrays) can be dereferenced by both the host and the device, unlike the explicit memory allocation version.

When launching the kernel, these two arrays are passed to the kernel function and the device can access the arrays, operate on the input data and save the results directly to the output array. The data operation (logic gate simulation) is the same as in the other version. The output data is then printed to the filename that was passed to the program in the command line arguments. Finally, we free up the memory by calling the cudaFree() function on the two arrays. This is confirmed to be running correctly with the compareFiles function given to us.

As can be seen from the table and graph below, the execution times grow rather linearly with the input size and they are quite similar to the explicit memory allocation execution times. However, given that there is no data migration, the end result will be faster.

File Name	Time (ms)
10000.txt	0.079264
100000.txt	0.121450
1000000.txt	0.387459


Unified Parallel Execution Times
time (ms)
Operations