CS3210 Assignment 2
Parallelising Hash-Based Proof-of-Work System Using CUDA

Jervis Chan Jun Yong
A0134191M

1. Problem Description and Parallel Strategy

In this problem, we are required to find a value X for which the digest of X (in this case, we will be using SHA256(X) to create a 256-bit digest) has a 64-bit prefix that is less than a target value of n, in order to validate the transaction associated with X.

X is 416 bits in length and comprises of

1. The previous digest of 64 characters (256 bits)
2. t - the UNIX timestamp (32 bits)
3. Transaction ID of 8 characters (64 bits)
4. Nonce (64 bits)

We will brute force through the 2^{64} possible values of nonce to generate the required prefix that satisfies the condition. This will be implemented with parallelism in the following ways:

1. We will generate different values of nonce to generate different values of X for SHA256 computation.
2. We will perform the same SHA256 computation of X for different values of X.

2. Measurements

The measurements below was done on ***xgpc5,*** which is a node with **Tesla V100** with **compute capability of 7.0**. The output of the parallel program was verified to be correct and the GPU time taken for the program to complete the proof-of-work against different target values (n) is shown in the table below. In the initial program, we experimented using 10 blocks with 320 threads.

Target Size	**GPU Time Taken (ms)**
1073741824	17023.5
24513400340	156.00
432635866006	455.84
6215925831232	47.019
21346507927713	8.6301

From these results, we could infer that the size of n does not directly affect the time taken for the program to perform a valid proof-of-work. This could be due to the randomness of the problem. Thus, rather than the size of n, we can perform the modifications in the following section to see if the there are other factors that can affect the time taken to run the program.

3. Modifications

After implementing the CUDA-based program, the following modifications were performed on the code to investigate how they affect the GPU time taken to run the program:

1. Different Blocks and Grid Sizes
2. Different Nodes and Compute Capability

Compute Capability (2)	Grid Size (4)	Block Size (5)
1. xgpc5 (Tesla V100 with c.c 7.0) 2. xgpf5 (Tesla T4 with c.c 7.5)	5, 10, 20, 40, 80	32, 64, 128, 256, 320

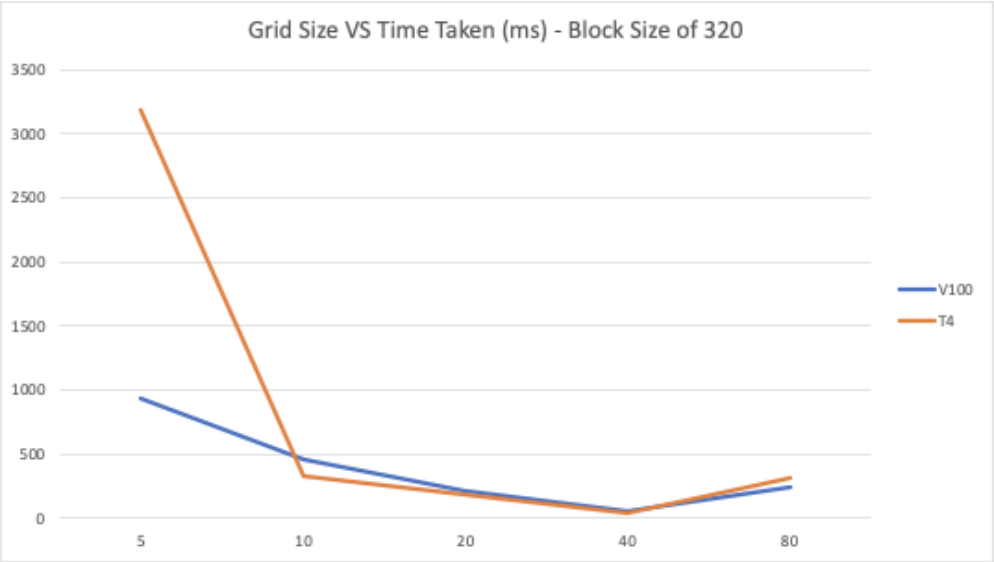

Figure 1: Grid Size VS GPU Time Taken (ms)

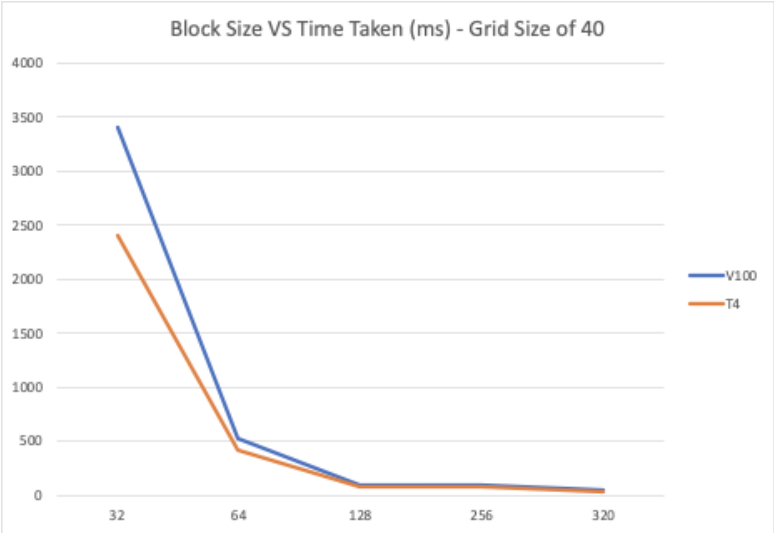

Figure 2: Block Size VS GPU Time Taken (ms)

From the results shown in the figures above, we can see that increasing the block size and grid size does lead to an improvement in the GPU time taken for the program to perform a valid proof-of-work. This is with the exception of a grid size of 80 where the time taken increased for both nodes. This could be a sign that at even higher grid sizes, there could be more overhead incurred, resulting in a higher time taken. Next, we can also see that having a higher compute capability does lead to an improvement in the GPU time taken where the T4 takes a slightly lower time to perform the proof-of-work. However, it must be noted that the difference is very small, and due to the random nature of the results, there were also occurrences where the node with the higher compute capability took a longer time.

4. Conclusion

In conclusion, we found a parallel implementation that allows the computation of a proof-of-work to be much faster as compared to a regular sequential algorithm. We experimented between between different grid and block sizes and found that having more grids and blocks led to improvements in the time taken for the program. Lastly, we compared between two nodes with different compute capabilities and found that the T4 with a higher compute capability did prove to be better.