**Problem 1. For CS4473 (40 points) and CS5473 (20 points)**

We are going to develop a new cryptocurrency, called PDNcoinTM. At the Initial Coin Offering (ICO), everyone will be able to buy one PDNcoin for one US dollar. Only a trillion PNDcoins will be offered to create artificial scarcity. We expect that one PDNcoin will be worth $1000 in 5 years given the current inflation rate of the fiat money. Please help us develop the PDNcoins and Make the World a Better Place!

Mining the PDNcoins will be similar to mining the bitcoins. Please read the blog post below to understand the concepts like **block**, **transactions**, **nonce**, **hash**, and **target**.

<https://medium.com/coinmonks/complex-puzzle-in-bitcoin-mining-aint-complex-no-more-9035b25b2a10>

You may look at a concrete example of mining a block in bitcoin

<https://www.blockchain.com/btc/block/0000000000000000003d05ef31993d1ddb80b6ef5632d0ae939ea1b22a24e150>

This block contains 1068 transactions. The winning nonce value is 554,703,974. The hash of this winning nonce value contains 18 leading zeros, which is smaller than the target. The miner who hit this nonce value earned about half a million dollars for successful mining this block.

You are provided with a serial C program, serial\_mining.c, that mines the PDNcoins. This program takes a block of transactions as an input file and tries a certain number of randomly-generated nonce values. For each nonce value, the program computes a hash value from the block of transaction using a simple hash function. This hash function generates a hash given from the transactions, nonces, and hash-array index. The index was used in order to attain more hash-value ‘randomness’ in the program. For simplicity, a transaction is represented by an integer and a block of transaction is a list of integers. The nonce values are also integers. The hash function is based on the modulus of a large prime number. To successfully mine a block of PDNcoin transactions, the program must find a nonce value that can generate a hash value less than the target (e.g., 10).

After loading a block of transactions, a PDNcoin mining program attempts to mine it in 3 steps:

Step 1: generate the nonce values using a random number generator

Step 2: generate the hash values using the hash function

Step 3: find the nonce with the minimum hash value

Let us accelerate this PDNcoin mining program using GPU in this project. You are provided with a starter CUDA program, gpu\_mining\_starter.cu. This starter CUDA program off-loads the Step 1 to the GPU using a kernel function from nonce\_kernel.cu. It still performs the Step 2 and Step 3 on the CPU using the same code as the serial program. In this project, let’s off-load the Step 2 to GPU in the Problem 1 and then off-load Step 3 to GPU in Problem 2.

In the Problem 1, please write a kernel function called hash\_kernel.cu to generate the hash value array on the GPU. Please change the CUDA program to gpu\_mining\_problem1.cu which performs Step 1 and Step 2 on the GPU and Step 3 on the CPU. The gpu mining program should carry out the following procedure for off-loading the hash computation

1. Keep the nonce value array on the device memory
2. Copy the transaction array to the device memory
3. Allocate the hash value array on the device memory
4. Launch the hash kernel function to compute the hash value array
5. Copy the hash value array to the system memory, which allows you to use the serial code to find the min hash and min nonce

Your CUDA program should be run as the starter CUDA program like this:

gpu\_mining transactions.csv n\_transactions trials out.csv time.csv

Where:

* transactions.csv: A block of transactions
* n\_transactions: The number of transactions in this block (20,000 for the provided transaction.cvs in the test data)
* trials: the number of nonce values to try
* out.csv: the minimum hash value and its nonce value found in this run
* time.csv: the runtime

Please first compile and run the starter CUDA program to make sure that you have the correct programming environment setup. Again, it is recommended that you use the GPEL machines for your computation.

Learning outcome:

This is designed to practice the embarrassingly parallel pattern on GPU.

What to submit:

In the ZIP file, please include submit all the code as specified on the first page, including the new hash\_kernel.cu and gpu\_mining\_problem1.cu.

Please benchmark the performance of your program using the provided transaction block file that contains 20,000 transactions. In the report, please provide the wall-clock runtimes for 5 millions trials and 10 million trials.

|  |  |  |  |
| --- | --- | --- | --- |
| **Implementation** | **Steps on GPU** | **5 million trials** | **10 million trials** |
| serial\_mining.c | None |  |  |
| gpu\_mining\_starter.cu | Step 1 |  |  |
| gpu\_mining\_problem1.cu | Steps 1 and 2 |  |  |

In the report, please discuss the speedup provided by the CUDA parallelization and the parallelization overhead incurred by CUDA, in comparison with the serial program.

Grading rubrics

CS4473

* 25 points for CUDA parallelization
* 15 points for the report

CS5473

* 15 points for CUDA parallelization
* 5 points for the report