HW4
Bitcoin Miner

Parallel Programming
2025/10/31

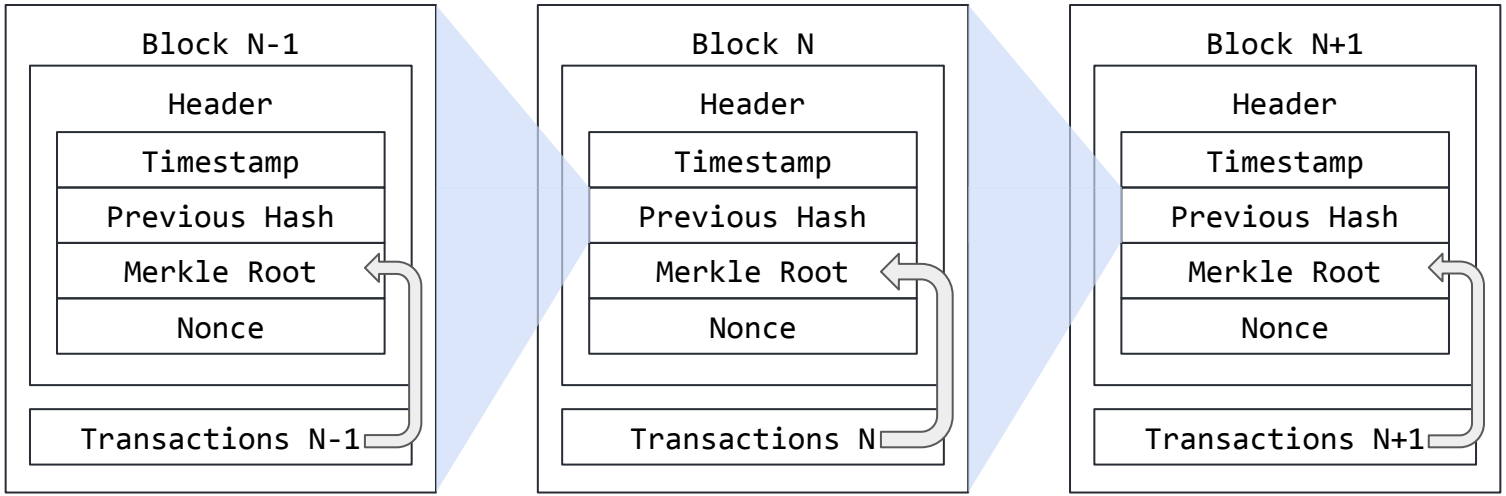
Spec

In this assignment, you are going to **parallelize a bitcoin miner with CUDA**. You may use any techniques that you've learned in this class!

Please refer to the assignment's specification for detailed information about the assignment.

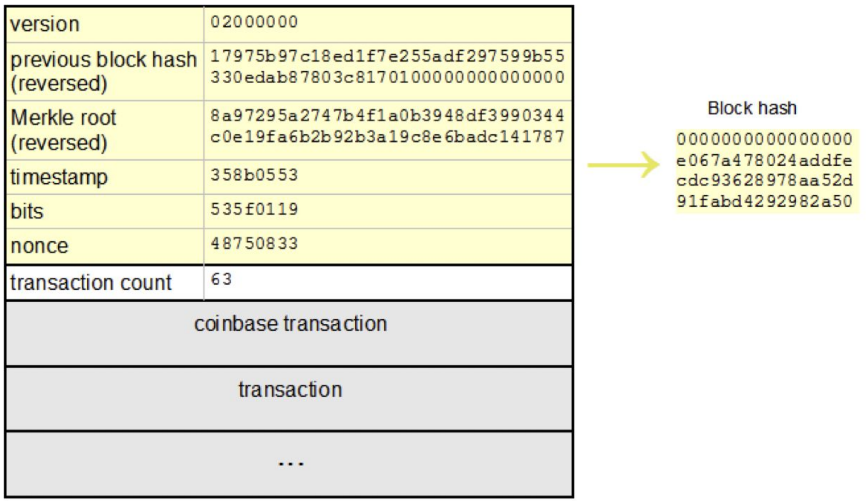
What is Blockchain?

- A blockchain is essentially a sequence of verified **blocks**
  - Each block contains the information of transactions and metadata, including the hash value of its previous block.
  - Due to the property of [cryptographic hash function], If something is modified, it will be discovered immediately.



What is Block?

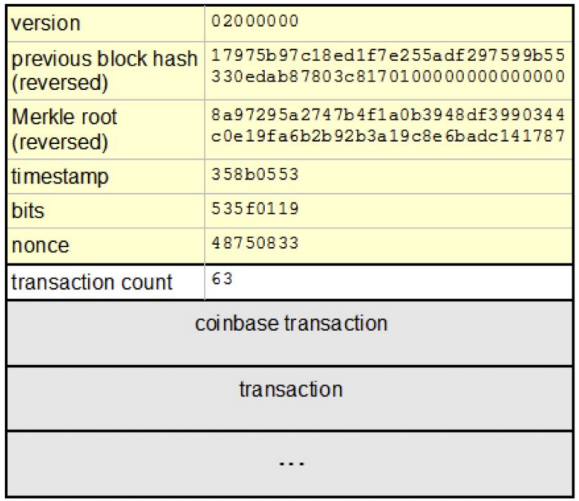
- A block is a data structure containing the information of transactions and metadata. It consists of two parts:
  - Block Header
    - Version number
    - Hash of the previous block
    - Merkle root
    - Timestamp
    - Target Difficulty (encoded in Bits)
    - Nonce
  - Block Body
    - Transaction counter
    - List of transaction records
    - ...



Job of Bitcoin Miner

The job of a miner is to create and add blocks to the blockchain by following steps:

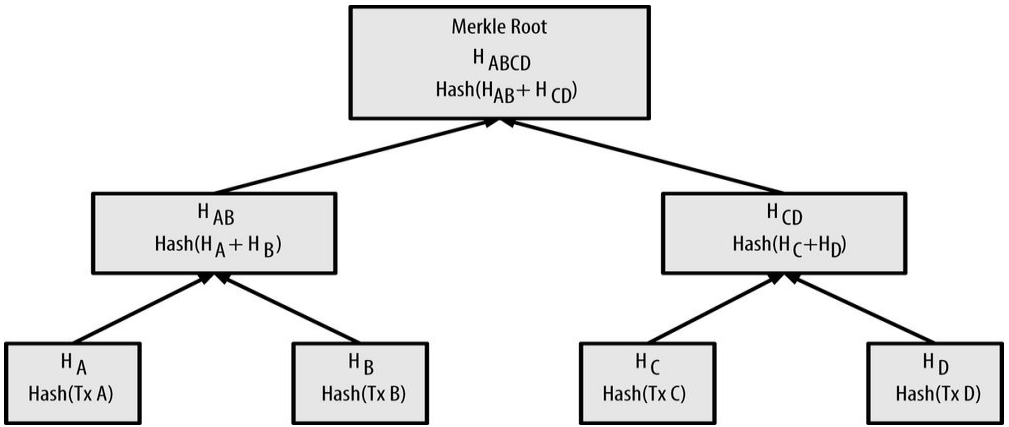
1. Gather transactions
2. Create the block header
3. Find the proper nonce
4. Broadcast the block
5. Receive rewards



hash < Target Difficulty

What is Merkle Tree?

A Merkle tree is a binary tree in which every leaf node is the hash of a data block, and every inner node is the hash its children. Merkle tree allows **efficient and secure verification of the contents of a large data structure**.



H_A = SHA256(SHA256(TxA))
H_B = SHA256(SHA256(TxB))
H_{AB} = SHA256(SHA256(H_A+H_B))

Credit: https://www.oreilly.com/library/view/mastering-bitcoin/9781491902639/ch07.html

Node Generation

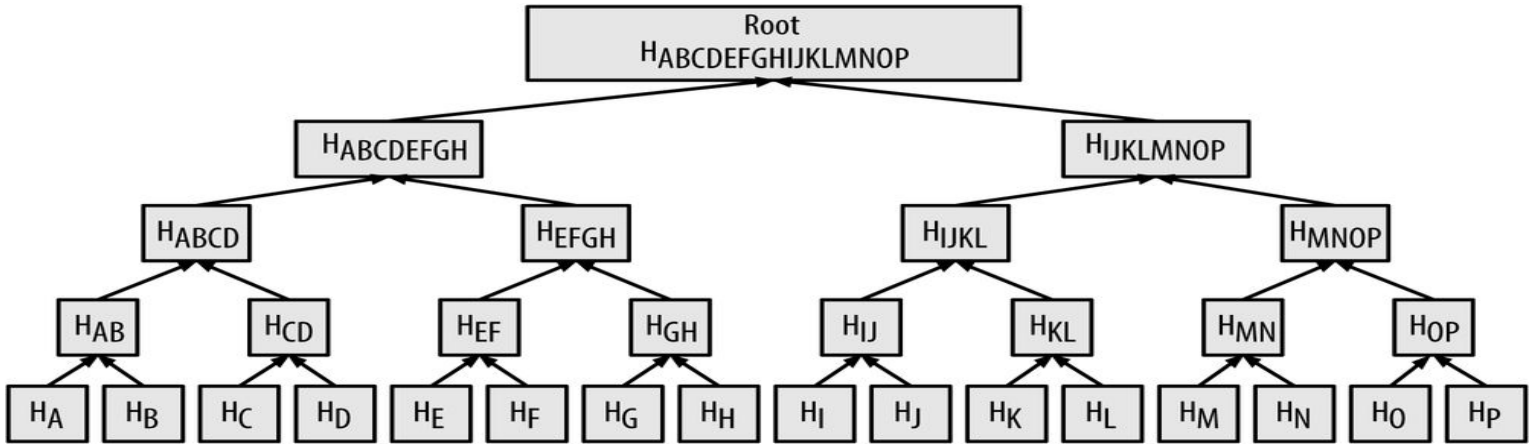
- Each parent node is generated by **double SHA-256 function**
  1. Each child node is hashed by SHA-256 TWICE.
  2. The results (HA and HB) are summed together.
  3. The sum is then hashed AGAIN by SHA-256 TWICE.
- The procedure is repeated again and again, until the root node is reached

$$H_A = \text{SHA256}(\text{SHA256}(\text{TxA})) \qquad H_B = \text{SHA256}(\text{SHA256}(\text{TxB}))$$

$$H_{AB} = \text{SHA256}(\text{SHA256}(H_A + H_B))$$

Merkle Root

The root is called Merkle root. It represents the entire data set, and any change in the data would result in a different Merkle root.



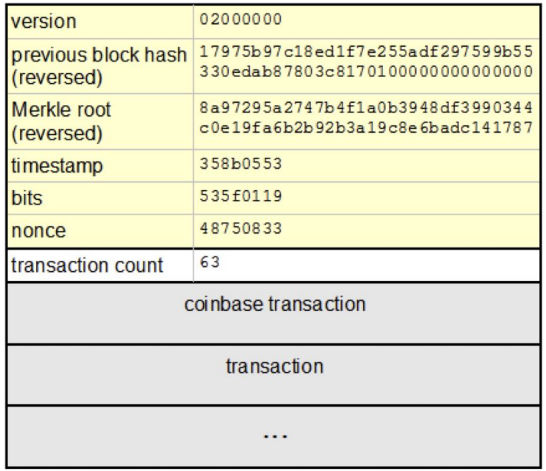
Credit: https://www.oreilly.com/library/view/mastering-bitcoin/9781491902639/ch07.html

Target Difficulty

- Purpose: to maintain a consistent block generation rate and ensure that the blockchain remains secure and stable.
- The Target Difficulty is a value to determine how difficult it is to find out a proper hash for the block header by requiring hash < Target Difficulty.
- The Target Difficulty conforms to the following rules
    - It is a 256-bit number, encoded by a field called “bits” (in 4 bytes)
    - It has N leading zeros. Currently, N equals 76 (year 2022).
    - It is determined by the Bitcoin network.
    - It is adjusted such that a new block is computed every 10 minutes (on average)
    - You may check current difficulty here: https://btc.com/stats/diff

Nonce

- Nonce is a 32-bit value in the block header.
- Miner needs to find a proper nonce such that the hash of the block header is less than the current Target Difficulty.



Block hash

0000000000000000e067a478024addfecdc93628978aa52d91fabd4292982a50

Block header hash < Target Difficulty

Goal of the Assignment

- Find a proper nonce for a given block
  - The hash of the block header has to be less than the given Target Difficulty.
  - The time you spent on finding the hash value has to be accelerated.
  - Use any techniques that you've learned in this class!

Sequential Code

1. Read input.
2. Calculate the merkle root from transactions.
3. Decode the Target Difficulty.
4. Find a proper nonce, such that the hash value of the block header satisfies the requirement.

```
for nonce = 0X00000000 to 0Xffffffff
    calculate a hash value
    if the hash value < Target Difficulty
        done
```

Sequential Code - Header Extraction

- Step 1 : Read input
- There are several fields in the block header that you have to extract.

```
// **** read data *****
char version[9];
char prevhash[65];
char ntime[9];
char nbits[9];
int tx;
char *raw_merkle_branch;
char **merkle_branch;

getline(version, 9, fin);
getline(prevhash, 65, fin);
getline(ntime, 9, fin);
getline(nbits, 9, fin);
fscanf(fin, "%d\n", &tx);
printf("start hashing")

raw_merkle_branch = new char [tx * 65];
merkle_branch = new char *[tx];
for(int i=0;i<tx;++i) {
   merkle_branch[i] = raw_merkle_branch + i * 65;
   getline(merkle_branch[i], 65, fin);
   merkle_branch[i][64] = '\0';
}
```

Sequential Code - Header Fields

- A representative header fields of a block is depicted below
  - Please have a comparison with pages 4.
  - These information has to be extracted first before we carrying out the calculation.
- `case01.in`

```
4                                                                  Number of blocks
20000000                                                           Version
0000000000000000003348540cbfc68b70825e7abcd5a83a48a5f87fa7f1aace   Previous block hash
5ac22f8b                                                           Timestamp
17502ab7                                                           Bits (packed difficulty)
2094                                                               Number of transactions
c6574adb277efbfb972658ab78b1277707a967076cfc90d6af800cd8a915396d
c01c33240ba97fb6db2b98fbaf7e4211fe3b59585372994bdac1b96b1c9be0d3
abce7b2b30ff62b4998864db6a1ea77db8eb33d3f6abe3f981e0d2802c999042
fd463aca59df1302c8f08e8070b56bd64ebe0cf35ba68cd5b39d208a2ff50053   Transactions
4321318516cb6630e3b3caa29bd49227168a69d170e1110c220b3c30d15f913c
f1f452e09dad935a67499a24e388d61fd7f0f38c97a2a92c73b2fd7019a85578
dd24a8e3f419006cd2f7cb3336b605fa6c2898accecac3d0a844c2ae8f5f53d9
```

Sequential Code - Merkle Root

- Step 2 : Calculate the merkle root from transactions
  - The merkle root is calculated by a function provided by the TA
  - You are encouraged to take a look of its implementation

```
// **** calculate merkle root ****

unsigned char merkle_root[32];
calc_merkle_root(merkle_root, tx, merkle_branch);

printf("merkle root(little): ");
print_hex(merkle_root, 32);
printf("\n");

printf("merkle root(big):    ");
print_hex_inverse(merkle_root, 32);
printf("\n");
```

Sequential Code - Target Difficulty

- Step 3 : Decode the Target Difficulty
  - It is decoded from a number called nbits.
  - The decode algorithm is already implemented by us (shown below).
  - If you are interested in that algorithm, please take a look of this website.

```
// ********** calculate target value *********
// calculate target value from encoded difficulty
which is encoded on "nbits"
unsigned int exp = block.nbits >> 24;
unsigned int mant = block.nbits & 0xffffff;
unsigned char target_hex[32] = {};

unsigned int shift = 8 * (exp - 3);
unsigned int sb = shift / 8;
unsigned int rb = shift % 8;
```

- Example:
  - Bits: 0x1709f8d9
  - Target = 0x09f8d9 * 2 ** (8 * (0x17 - 3))
    = 0x0000000000000000009f8d900000000000000000000000000000000

Sequential Code - Nonce

- Step 4 : Find a proper nonce, such that the hash value of the block header satisfies the requirement
  - Try if you can figure out a way to parallelize and accelerate the code.

```
for(block.nonce=0x00000000; block.nonce<=0xffffffff;++block.nonce){
    //sha256d
    double_sha256(&sha256_ctx, (unsigned char*)&block, sizeof(block));
    if(block.nonce % 1000000 == 0) {
        printf("hash #%10u (big): ", block.nonce);
        print_hex_inverse(sha256_ctx.b, 32);
        printf("\n");
    }
    if(little_endian_bit_comparison(sha256_ctx.b, target_hex, 32) < 0) { // sha256_ctx < target_hex
        printf("Found Solution!!\n");
        printf("hash #%10u (big): ", block.nonce);
        print_hex_inverse(sha256_ctx.b, 32);
        printf("\n\n");
        break;
    }
}
```

Sequential Code - SHA-256

- SHA-256 implementation for the sequential version can be found at:
  - sha256.h
  - sha256.cu
- If you are interested in the implementation details, please take a look at the following website
  - https://en.wikipedia.org/wiki/SHA-2

Test cases

- We provide several test cases for you to test your program
- The execution time of the sequential code are given below:
  - case00.in : about 1 hr 15 min
  - case01.in : about 1 hr 15 min
  - case02.in : about 30 min
  - case03.in : about 11 min
- Please keep in mind that your goal is to parallelize the mining algorithm and accelerating it.
- **It has to be faster than the sequential version**.

Grading

- Correctness (50%)
  - Please make sure to use GPU and verify your answer.
- Performance (20%)
  - We will grade the performance of your assignments against the other students in the class.
- Report (30%)
  - Please describe, in detail, how you parallelize your codes as well as your optimization methodology.
- Advanced CUDA skills
  - You are welcome to use streaming, page-lock memory, asynchronous memory copy, or any other advanced skills.
- Others
  - You will get credits if you are able to optimize the other parts of the source codes. Please justify your solutions and provide a detailed comparison in your report.

Grading - Report

- In your report, please include the following parts:
  - Describe the design/overview of your implementation briefly.
  - The parallelization and optimization techniques you used in your solution
  - Experiments of various combinations of the number of blocks & threads (at least 8 combinations) and plot them with the figures
  - Describe the details if you use advanced CUDA skills
  - (Optional) Any suggestions or feedback for the homework is welcome

Submission

- Please zip the following files to a single archived file named <StudentID>.tar:
  - hw4.cu
  - sha256.cu
  - sha256.h
  - report.pdf
  - Makefile (optional)
- Please make sure your Makefile works properly (Your Makefile should build executable binary hw4) and your program can run before submitting your assignment 4.

Reminder

- Because we are doing hash, there may be a situation multiple nonce can satisfies the requirement hash value < Target Difficulty.
- We accept all nonce satisfies the requirement, so don’t worry if your nonce isn’t same as our provided solutions.
- We ensure that all the testcases have more than one solution.
- Please refer to the assignment’s specification for detailed information about the assignment.

Deadline

- The deadline of assignment 4 is 11/14 (Fri.), 23:59.
- Everyone is welcomed to ask questions on NTU cool.