Bitcoin Mining Algorithm with Parallel Computing

Gaurav Singh - 191IT218
Information Technology
National Institute of Technology, Karnataka
Surathkal, India 575025
gauravsingh.191it218@nitk.edu.in

Navyasree B - 191IT135
Information Technology
National Institute of Technology, Karnataka
Surathkal, India 575025
navya.191it135@nitk.edu.in

Abhinav Dugar - 191IT202
Information Technology
National Institute of Technology, Karnataka
Surathkal, India 575025
abhinavdugar191it202@nitk.edu.in

Niraj Nandish - 191IT234
Information Technology
National Institute of Technology, Karnataka
Surathkal, India 575025
nirajn.191it234@nitk.edu.in

***Abstract*—The project involves implementing bitcoin mining using parallel computing using OpenMP, MPI and CUDA and comparing the three methods. Bitcoin mining involves finding the NONCE value which is the target hash using trial and error(brute force). This is a very heavy computational work as the number of possibilities are 2^{256} which is huge. Hence, parallelizing this process will help in achieving the NONCE faster in bitcoin mining. The project was done in C++. The output of each library(OpenMP, MPI and CUDA) is the number of hashes per second which represents the number of hashes the algorithm generated per second. Since this is a trial and error algorithm, the higher hashes per second, the faster the target hash is achieved. For the sake of simplicity, the NONCE is any hash divisible by 800000. To calculate the hash, SHA256 algorithm is used, which is the best hashing algorithm known and is extensively used in blockchains.**

I. Introduction

Bitcoin is a digital currency which was created in 2009. The main motive behind the creation of bitcoin was to arrive at an agreement in a valid transaction in a decentralized manner.Bitcoin was the first decentralised network which enabled peer to peer transfer of value. Bitcoin provides an alternate body of control other than the government or a central bank, here the activities and balances are stored on a public ledger, which is called the blockchain. Transactions involving the blockchain do not require the presence of any middlemen. With the help of the blockchain, the entire transaction history is readily available to verify.This transparency along with the cryptographically secure protocol helps limits external manipulation to almost none.

Bitcoin uses blockchain technology to maintain a peer-to-peer network and the nodes (or peers) use a distributed consensus mechanism called Proof of Work to verify and confirm the transactions. However, the transaction speed with the help of this process is much slower than with the centralised digital transaction systems. In this report, a parallel Proof of Work model is proposed in order to increase the scalability and speed of the processing of the transactions.

Key Contributions of our work are:

- Developing a iterative algorithm for proof of work
- Developing a parallel solution for proof of work using OpenMP, MPI and CUDA

A. Proof of Work

Proof of Work is one of the mechanisms using which a blockchain can achieve consensus. It ensures that all the nodes of a system can agree on the state of the public ledger and value can only be transferred when all the actors in the exchange agree upon what is being exchanged. In PoW system, consensus is agreed upon via the work done by the network miners, in return for this they receive the transaction fees as an incentive. Miners check the validity of the transactions before adding the block to the blockchain network.

A cryptographic puzzle is required to be solved for every block, for which some processing power is spent by the miner to find the valid solution. When a miner finds a solution, it is able to create the new block and broadcast it to everyone else.

II. Literature Survey

Parallel mining in blockchain for bitcoin using game theory by M. Tote, A. Kumar, M. Mahankal, S. Khadse and V. Uprikar describes the usage of game theory paired with parallel computing to increase the hash rate efficiency so as to mine blocks faster. Game theory is used to generate nonce values with multi-threading.

Design and Development of a Parallel Proof of Work for Permissionless Blockchain Systems by Shihab Shahriar Hazari explains the working of blockchain technology with reference to cryptocurrency and the proof of work algorithm to be parallelized. However, the network architecture requires a Manager to coordinate with multiple users which ideally should not be present in decentralized systems.

From our literature survey we have identified the ideal portion of the algorithm that can be parallelized which is the computation of the hashes. Increasing rate of hash generation

improves the time complexity of the brute force approach to finding the nonce.

III. Methodology

A. Nonce

A Nonce, or 'number only used once' is the number that a bitcoin (or any blockchain) miner has to find. Every blockchain network has a predefined difficulty level requirements set to it. When the nonce is added to a hashed block and rehashed, the target hash must conform to the aforementioned requirements. Usually, it means that the first x digits of the hash must be 0's, or in our case, the hash value modulus of 800000 should be 0. Hash generation in the algorithm must be deterministic with low chances of collision, which is why we use the SHA-256 algorithm to generate hashes (further details provided in section B). The only method of finding the nonce for an encrypted (or hashed) block is by employing a brute force approach where the miner runs through all possible numbers till the nonce is found.

B. Sequential Program

Bitcoin Mining is the process of finding the nonce value using brute force i.e. trial and error. Nonce value is set up by the blockchain network, and miners try to find the nonce. The first miner to find the nonce gets to add the block to the blockchain network and the miner gets rewarded for that.

1) Block: Each block in the blockchain contains the timestamp, transaction information, previous hash(this isn't used for this project for simplicity) and the NONCE that the miner needs to find. Each block is hashed using the SHA-256 algorithm. NONCE is already explained in the previous sections.

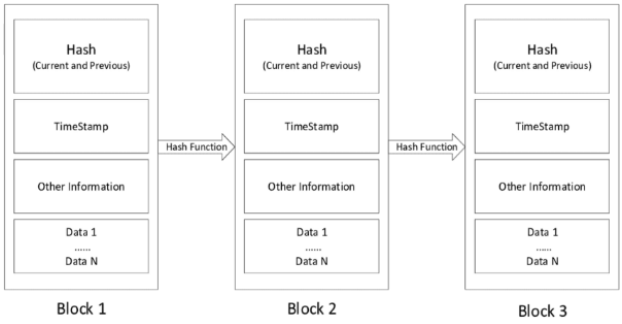

Fig. 1. Structure of a block

2) SHA-256: SHA-256 is the hashing algorithm used to hash the block. It takes in the block as the input and returns a 256 bit number. SHA-256 is said to be collision free algorithm, where the chance of a collision is negligibly low, and hence it's widely used in blockchains. The steps in SHA-256 algorithm are:

- **Padding Bits** This step adds some extra bits to the message to ensure that the length is 64 bits short of the multiple of 512. While adding, the first bit should be 1 while the rest are 0's.
- **Padding Length** Now 64 bits of data is added to make the final plaintext multiple of 512.
- **Buffers Initialising** The default values of eight buffers are initialised to be used in the rounds as shown in the fig.2.

a = 0x6a09e667

b = 0xbb67ae85

c = 0x3c6ef372

d = 0xa54ff53a

e = 0x510e527f

f = 0x9b05688c

g = 0x1f83d9ab

h = 0x5be0cd19

Fig. 2. SHA-256 Buffer Initialising

- **Compression Functions** The entire message is then broken down into multiple blocks of 512 bits each. Each block is put through 64 rounds of operation, and the output of each block serving as the input of the next block. The process is explained visually in fig 3.

Additionally, in bitcoins, the addresses also include the checksums so they can be checked to verify that they have been typed in correctly. Checksums are created by hashing the message with SHA-256 twice, and then using the first 4 bytes of the result. Then the data and the checksum is kept together to check that the whole thing is typed in correctly next time its used. If a small mistake is made in any part, the data will not match the checksum.

3) The Algorithm: The algorithm for Bitcoin mining in this project in most simple terms is that the nonce value starting from 0 is incremented in a while loop which breaks only when the target hash is achieved and the time taken is more than 1 minute for the process of mining. The target hash in this project is any hash which is divisible by 800000. The pseudo code is given below and the flowchart is given in the fig 4.

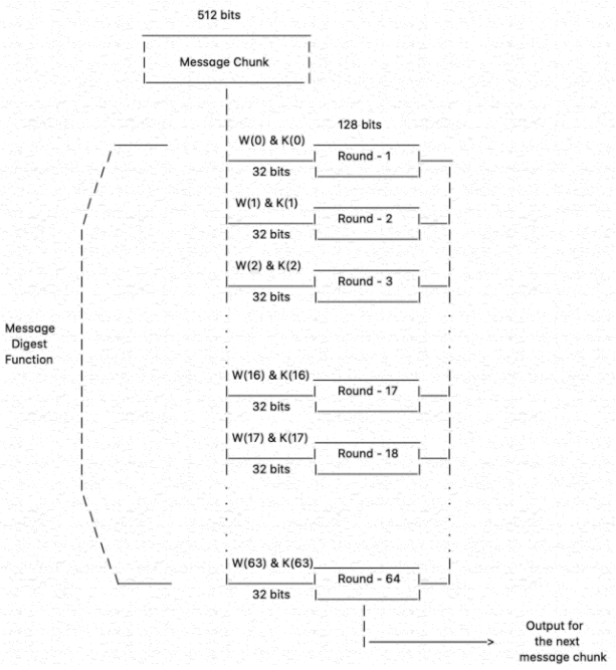

Fig. 3. SHA-256 Algorithm

Algorithm 1 Algorithm for Bitcoin Mining

$nonce \leftarrow 0$
$start \leftarrow currentTime$
$timer \leftarrow currentTime - start$
while $timer \leq 60.0$ **do**
  $nonce \leftarrow nonce + 1$
  if $(nonce\%800000 == 0)$ **then**
    $timer \leftarrow currentTime - start$
  end if
end while

return $nonce$

C. Parallel Techniques

1) OpenMP: OpenMP, which stands for Open Specification for Multi-Processing, is a library used to divide computational work of a program and add parallelism to it by creating threads. Each block will have a header of fixed length. We store the nonce value of each block in the header.
OpenMP contructs used in the program are:

- parallel - This is the construct that created a group of threads used by the program to compute in parallel. Here we are initializing the header as a *private* variable with default values for the threads.
- critical - This construct creates a section within the parallel region where only one thread can execute at a time. In this region we are generating the nonce values for each block and storing it back in the block header.

Next, we make use of the *openssl/sha.h* library to parallely generate the hashes from the data and find the nonce of each block.

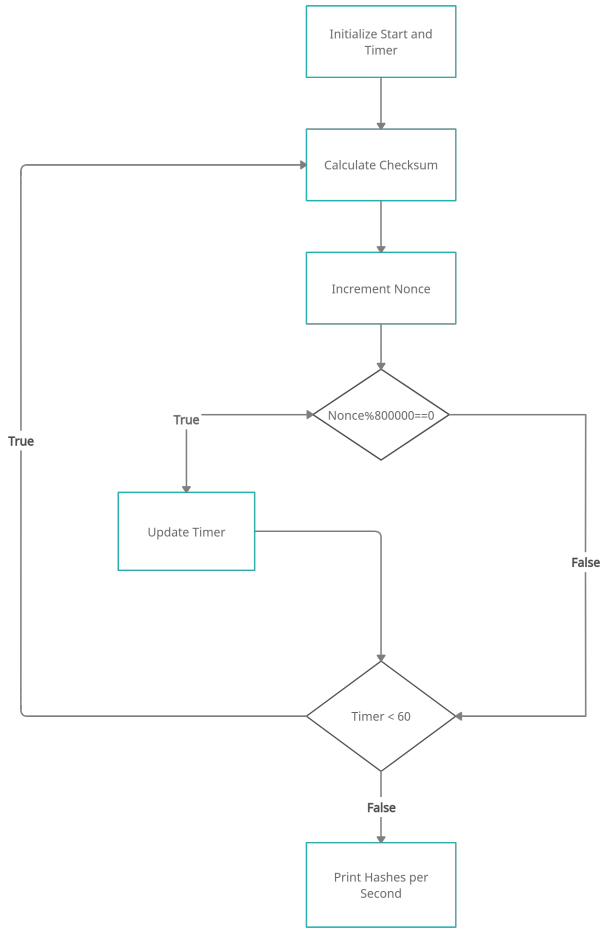

Fig. 4. Flowchart of the algorithm

```
#pragma omp parallel private(header)
    {
        while ( timer < 60.0){
            #pragma omp critical
            {
                header.nonce = counter;
                counter ++;
                if ( counter % 800000 == 0){
                    timer = (When() - start);
                }
            }
        }
    }
```

Fig. 5. OpenMP

2) Message Passing Interface (MPI): MPI stands for Message Passing Interface, it is a library consisting of a set of subroutines,functions and constants which allows for message parsing between processes.

We have taken the master-slave approach to the algorithm. This is accomplished by dividing the processes in MPI_COMM_WORLD into two sets - the master (who manage I/O and controls execution of slave) and the slaves (who execute I/O by contacting and synchronizing with the master).

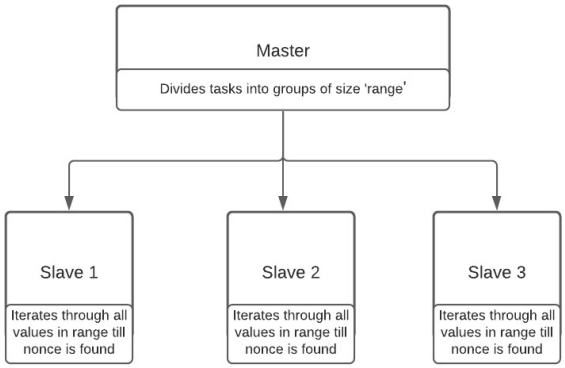

Fig. 6. Master/Slave Scheme

a) Master:

The master seeds all the slaves, the number of slaves seeded depends on the number of tasks. After receiving a result from any of the slave, a dispatch for a new nonce request is sent once all the existing nonce requests have been exhausted. The nonce gets updated in this process. Once all the nonce requests are processed all the slaves are told to exit.

```
void master(int ntasks,int range)    {
    int  rank, nonce = 0 ;
  double       result=0;
  MPI_Status     s;

  for (rank = 1; rank < ntasks; ++rank) {
    MPI_Send(&nonce,1,MPI_INT,rank,WORKTAG,
    MPI_COMM_WORLD);
    nonce += range; ;
  }

  double start = When();
  double timer = When() - start;

  while (timer < 60) {
    MPI_Recv(&result,1,MPI_DOUBLE,MPI_ANY_SOURCE,
    MPI_ANY_TAG,MPI_COMM_WORLD, &s);
    MPI_Send(&nonce, 1, MPI_INT, s.MPI_SOURCE,
    WORKTAG, MPI_COMM_WORLD);
    nonce += range;
    if ( nonce % 800000 == 0){
            timer = (When() - start);
        }
  }

  for (rank = 1; rank < ntasks; ++rank) {
    MPI_Recv(&result, 1, MPI_DOUBLE, MPI_ANY_SOURCE,
    MPI_ANY_TAG, MPI_COMM_WORLD, &s);
  }

  for (rank = 1; rank < ntasks; ++rank) {
    MPI_Send(0, 0, MPI_INT, rank, DIETAG,
    MPI_COMM_WORLD);
  }
}
```

Fig. 7. Master

b) Slaves:

After the request is received from the master, the slave verifies the tag. After verification, the nonce calculation is done in a iterative manner by each of the slave. Once the necessary calculation is completed, the result is then sent to the master thread and the slave waits for the next nonce search to the assigned to it.

```
void slave (int range){
    double result=1;
    MPI_s          s;
  for (;;) {
    MPI_Recv(&header.nonce, 1, MPI_INT, 0,
    MPI_ANY_TAG,
    MPI_COMM_WORLD, &s);

    if (s.MPI_TAG == DIETAG) {
      return;
    }
    for(i = 0 ; i < range ; i++)
    {
            header.nonce ++;
      }
    result = 1 ;
    MPI_Send(&result, 1, MPI_DOUBLE, 0, 0,
    MPI_COMM_WORLD);
  }
}
```

Fig. 8. Slave

3) CUDA: Compute Unified Device Architecture, also known as CUDA, is an application programming interface (API) that employs parallel computing techniques and allows different types of software programs to make use of the graphics processing unit (GPU).

Our program makes use of 16 blocks with 128 threads each.

```
__global__ void doCalc( unsigned int *seed) {
    \\nonce algorithm
    header.nonce = (*seed * blockDim.x * NUM_BLOCKS)
     + blockIdx.x * blockDim.x + threadIdx.x;
}

\\ Calling of kernel function in the "main"
while ( timer < 10.0){
    cudaMemcpy(d_counter, &counter, sizeof(counter),
     cudaMemcpyHostToDevice);
    doCalc<<< blocksize, threads >>>( d_counter);
    hashes += blocksize*threads;
    counter++;
    timer = When() - start;
    cudaDeviceSynchronize();
}
```

Fig. 9. Kernel Function

We then allocated memory in the GPU for two variables named *dev_prev_block* and *dev_merkle_root*. Next we copied the initial values for the previous two defined variables from the host to the GPU device. Finally we execute the kernel function named *doCalc*. In the kernel function, we generate the nonce value for the respective thread in the respective block using the formula

$$(seed * blockDim.x * NUM\_BLOCKS) + blockIdx.x * blockDim.x + threadIdx.x * repeats$$

where *NUM_BLOCKS* and *repeats* are constants with values 1024 and 1000 respectively. The generation of hashes is done parallely in the same kernel function after the nonce value has been generated.

IV. Results

Hashes per second has been used as a parameter to compare the results obtained by various parallel techniques. The obtained result is shown below:

TABLE I
COMPARISON OF RESULTS OBTAINED USING VARIOUS PARALLEL TECHNIQUES

Type of parallelism	Number of threads	Hashes per second
Iterative	-	6946666
OpenMp	2	6986666
OpenMp	4	6813333
OpenMp	6	7000000
MPI	2	6736313
MPI	4	19276009
MPI	6	32245309
CUDA	-	99619603

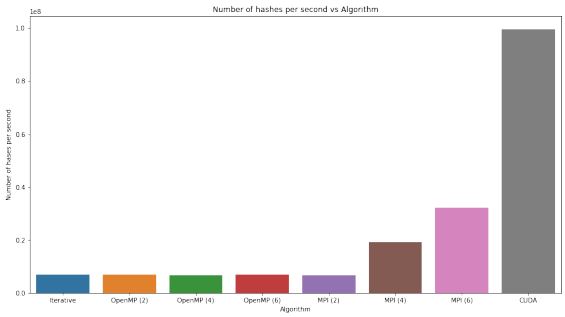

Fig. 10. Master/Slave Scheme

OpenMP, MPI and CUDA were used to parallelize the bitcoin mining algorithm and from the above results table we can observe that the Hashes per second obtained by OpenMp were greater than iterative but by a smaller margin, whereas the Hashes per second obtained by MPI were much more greater than that of OpenMp or iterative.CUDA gave the most optimum result, the number of hashes obtained with CUDA were almost 3x the number of hashes obtained with MPI and more than 13x the number of hashes obtained with OpenMP.

V. Conclusions

This project explores the implementation of parallel computing in bitcoin mining which is a brute force algorithm to find the target hash. The time for each block to be mined was given as 1 minute in this project, which gave some interesting results. We could observe that the performance obtained by OpenMP was better than that of iterative but by not a bigger margin, while with MPI we were able to obtain much better results while increasing the number of threads. However, we were able to achieve the best results with CUDA - three times better than MPI with 6 threads. Overall, we can conclude that CUDA performs the best followed by MPI with more number of threads, while OpenMP gave moderate results and finally iterative performed the worst. Future work for this project would be to simulate a real mining environment and analyse the performances of the different libraries.

VI. Acknowledgement

We would like to thank Professor Geetha V for giving us this opportunity to get a hands-on experience of working with parallel computing in a niche topic, blockchain technology.
The Github Repository can be accessed here:
https://github.com/gaurav2699/Bitcoin-mining-parallel

VII. References

1. S. S. Hazari. Design and Development of a Parallel Proof of Work for Permissionless Blockchain Systems (2019)
2. M. Tote, A. Kumar, M. Mahankal, S. Khadse, V. Uprikar. Parallel mining in blockchain for bitcoin using game theory (2019)
3. OpenMP - https://www.openmp.org/resources/tutorials-articles/
4. MPI - https://www.open-mpi.org/doc/
5. Proof of Work - https://medium.com/utrust/proof-of-work-ab12b8e13c7
6. CUDA - https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html
7. http://mahalayaagri.com/ethereumclassic-address/bitcoin-mining-parallel-computing-bitcoin-mining-pool-concentration/