Soma Szabo

Simulating Parallel Bitcoin Mining Using OpenMP
CPR E 527 Project Report

1. Introduction

Cryptocurrencies are digital currencies that use cryptography to secure their transactions and control the creation of new units. They are decentralized, meaning they are not issued or regulated by any central authority like a bank or a government. Cryptocurrencies have become popular in recent years because they offer some advantages over traditional currencies such as:

- Transparency: Cryptocurrencies typically run on a distributed public ledger called the blockchain. This is continuously updated by recording every transaction and make it possible to verify the authenticity and history of any coin or transaction.
- Anonymity: Cryptocurrencies allow users to send and receive money without revealing their identities or personal information.
- Innovation: Cryptocurrencies enable new business models, applications, and services that are based on the principles of decentralization, peer-to-peer exchange, and smart contracts (self-executing agreements that are encoded on the blockchain).
- Inclusion: Cryptocurrencies can provide access to financial services for people who are unbanked or underbanked, especially in developing countries where traditional banking systems are inefficient or corrupt.

One of the most popular and influential cryptocurrencies is Bitcoin, which was created in 2009 by an anonymous person or group using the pseudonym Satoshi Nakamoto. Bitcoin is based on a proof-of-work system, meaning that miners must compete to solve complex mathematical problems using computers to generate new bitcoins and validate transactions. This process is called mining, and it rewards miners with bitcoins for their work. Minig is very resource-intensive and competitive. As more miners join the network and more Bitcoin is mined, the difficulty of the problems increases, requiring more computing power and electricity to solve them. This makes mining less profitable and accessible for individual miners, who must compete with large mining pools and specialized hardware (such as ASICs) that are designed for mining.

Since Bitcoin mining is a highly repetitive task, parallel computing and algorithm optimizations can greatly speed up the mining process. In this project, OpenMP was used to parallelize a custom-built simulation of the Bitcoin mining process in C++. OpenMP is an API that supports multi-platform shared-memory parallel programming in C/C++ and Fortran. It allows programmers to create parallel regions of code that can be executed by multiple threads on different cores/processors or offloaded to target devices such as GPUs. This project explores OpenMP's parallelization procedures, how they distribute the workload, and the performance improvements gained from parallelizing a simulated Bitcoin mining process. It also explores the efficiency and granularity of target offloading compared to conventional parallelization on the CPU. Ultimately, the scalability of the Bitcoin mining process is evaluated which is used to demonstrate how parallel computing improves the efficiency and profitability of cryptocurrency mining.

2. Implementation

To implement the Bitcoin mining process in software, the SHA-256 algorithm is used according to the pseudocode and examples in [1] and [2]. This is verified by the OpenSSL library [3] to ensure correctness. Since this project does not mine actual Bitcoin nor connect to the network, the Bitcoin mining process will be slightly modified for the purpose of this project. It will consist of generating cryptographic hashes using the double SHA-256 algorithm of custom block data structures appended with nonces (a number only used once for finding a valid digest). Like Bitcoin, the goal is to generate a digest that passes a certain criterion, specifically the digest must have a certain number of leading 0s to be valid. If the digest is not valid, a new nonce value is tried, and the data is re-hashed to get a new digest. This continues until a valid digest is obtained. Then, the block is verified to ensure the current data and nonce do produce a valid digest and the block is added to the blockchain.

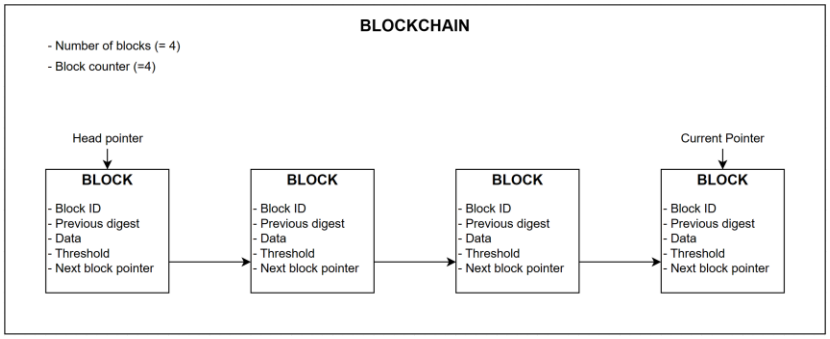

Figure 1. Blockchain object data structure.

A custom blockchain data structure is used which resembles a linked list. Each node is a block object that contains:

1. The block ID (index).
2. The (valid) digest of the previous block.
3. The concatenated data/digests up to the current block.
4. The current block threshold (number of leading 0's to make the digest valid).
5. A pointer to the next block.

These are linked together in a blockchain object which has several functions. One of the most important is determining when a new block can be added to the blockchain. In this project, if the number of leading 0s in the digest matches the number of blocks mined so far, it is said to be valid; otherwise, a new nonce must be used. Once the block is added to the blockchain, the newly concatenated data/digests will be hashed with unique nonces per thread to find another valid digest. Since the digest must have more leading 0s, the computation will be more difficult and time consuming, similar to the Bitcoin specification. The overall structure and contents of the Blockchain object is shown in Figure 1.

Three versions of the simulated Bitcoin mining process were created to evaluate their differences. These are: serial, parallel, and target offloading to the GPU(s). The serial and parallel versions used the OpenSSL library's SHA256 functions to hash the data and nonce for each block. However, the GPU version required a manual implementation of the SHA256 algorithm due to compilation issues when offloading and translating the OpenSSL library functions to the GPU. A high-level flow diagram for the simulated Bitcoin mining process is

shown in Figure 2. The CPU initializes the data, previous digest, and appends an initial block to the blockchain. It then gets the current block information to mine and offloads it to the GPU to hash the data with nonces. Once a valid nonce is found, the GPU sends the data to the CPU, stops execution, and the CPU validates the data and once combination using the OpenSSL SHA256 library. Lastly, the block is appended to the blockchain and the process repeats.

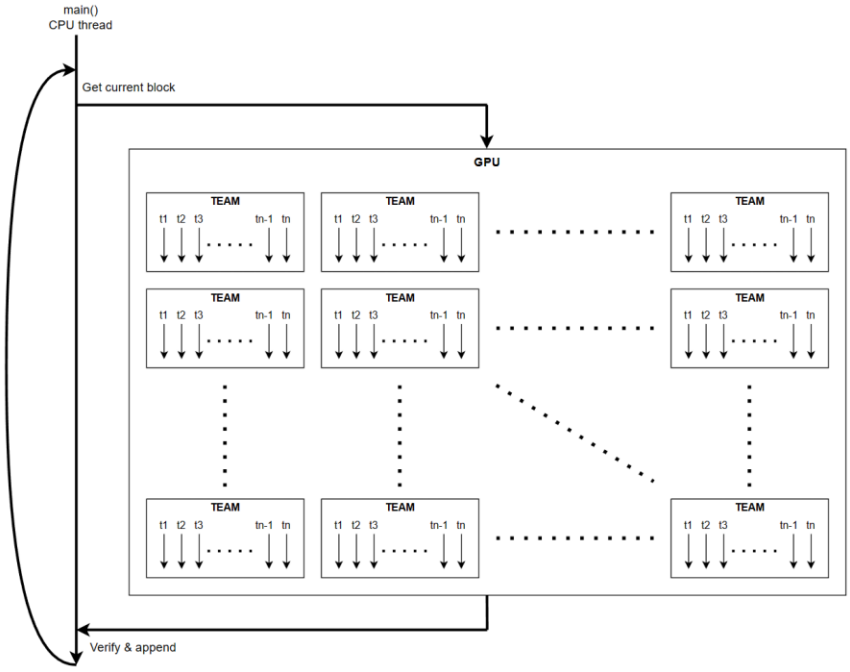

Figure 2. Simulated Bitcoin mining implementation flow for offloading to GPU.

The GPU version of the program distributes nonces per team on the device and each team increments its base team nonce which is assigned to it before the nonce incrementation loop. Specifically, the "size_t" (unsigned long) primitive is used to hold a nonce, so each team on the GPU is assigned a team nonce according to the equation below. Each thread in the team is assigned a continuously incrementing team nonce; thus, each thread hashes the data with a unique nonce.

$$Team\ nonce = \left(\frac{(\max size\_t)}{(number\ of\ teams)}\right) * (team\ number)$$

The CPU versions (serial and parallel) use a global nonce "size_t" variable that is continuously incremented. Each thread is assigned a private nonce based on the current value of the global nonce so that every thread works with a unique nonce value.

3. Evaluation

Two testing platforms were used as outlined below. The ISU HPC Nova cluster was used to benchmark the serial and parallel versions of the program as it supports up to 64 threads. The local machine was used to benchmark the GPU version of the program since the HPC cluster did not allow for GPU allocation when running programs. During the CPU tests on the local machine and the evaluated GPU trials, HWInfo [4] was used to monitor the system to ensure and confirm that the hardware was fully utilized by the software.

1. Local machine with CPU: 8-core AMD Ryzen 7 5800x @ 3.8GHz (base), GPU: 1xNvidia RTX 3080, RAM: 32GB. Running Ubuntu 20.04.6 LTS on WSL.
2. ISU HPC Nova cluster with CPU: 2x32-core Intel Xeon Gold 6140 @ 2.3GHz (base), GPU: N/A, RAM: 16 GB.

OpenMP's built in timing method (omp_get_wtime()) was used to capture the time taken to compute a set number of blocks. The current block mining time and the total time taken to mine up to each subsequent block was recorded for up to 7 blocks. This was due to the long time taken to mine past 7 blocks, especially on lower thread counts, which was beyond the 15-minute runtime threshold provided on the HPC clusters. A total of $7 * 3 = 21$ different CPU trials were performed where the threads were doubled starting from 1 to 64 and each trail was repeated 3 times for accuracy. Each CPU trial ran for 14 minutes maximum with 2 extra threads allocated beyond the required amount to handle any other processes. Thus, a parallel 8 thread version would be allocated 10 threads to ensure each thread may be fully utilized for mining. The only exception to this was when using 64 threads since 66 threads could not be allocated. The raw results are shown in Figure 3. The GPU version of the program utilized a slightly different implementation as outlined in Section 2. Therefore, 6 trials were performed due to the great variability in finding valid nonces and the digests affecting the subsequent block mining times afterward.

4. Results

The CPU results illustrate excellent scaling going from 1 to 16 threads; however, 32 and 64 threads have issues with maintaining the same order of hashes and digest computations. Therefore, the computation time per block varies greatly since the previous data block was not identical to the lower thread count trials. This also occurs on the GPU because there is even less synchronization between threads on that platform. On the CPU, going from 1 to 2 threads shows a ~0.52x reduction in average total computation time, 2 to 4 is ~0.55x, 4 to 8 is ~0.53x, and 8 to 16 is ~0.72x reduction. When using 32 or more threads, the compute node likely used threads from both CPUs resulting in nonce incrementation inconsistencies due to the communication overhead. This is likely why a different valid nonce was recorded and changed the entire structure of the blockchain from that point. Due to this, the total time to compute 7 blocks is much larger on 32 or more threads. However, looking at the per block computation times, we can see that block 3 on 64 threads was faster than all others, hinting that scaling should continue. However, the synchronization overhead will become more apparent and only see benefits after long block mining periods.

The GPU could most closely be compared to the 32 and 64 thread parallel versions of the results since it also had great variability in the blockchain structure due to different valid nonces that were found first. It illustrates how small periods of block mining like those for blocks 1-4 usually take longer due to the offloading overhead and lower per-thread performance. Once mining runs for longer periods, one may see improvements in block computation time; however, the results are still far behind the optimal 16 thread scenario when block 7 is reached. The bad performance on the GPU may be due to OpenMP's ability to effectively compile code for the target device. There were several warnings, specifically when using the "sprintf" function (in one place) and outputting that: "HAS does not support functions with variadic arguments". Another bottleneck may be the use of WSL on Windows 10 to run and compile the GPU program. This

was done because Windows does not have a native compiler with OpenMP. Lastly, the custom implementation of the SHA256 algorithm may not have been translated natively to the GPU, making computation much more resource intensive than necessary.

Looking at Figures 5 and 6, we can see that the 16 and 8 thread versions are the best after 3 or more blocks have been mined because more computation is required. On the other hand, when computation is simple and short, the single and dual threaded versions are the best. The largest issue is regarding the scaling of the GPU and higher thread versions of the program. Those should be much faster; however, the limited dataset and inconsistencies with the blockchain make it difficult to establish whether they are truly faster and would mine more Bitcoin in the long run.

Version-Thread-Round	Block 1	Block 2	Block 3	Block 4	Block 5	Block 6	Block 7	Total
Serial-1-1	0.000076	0.000716	0.022275	0.007466	0.103259	16.194077	178.776371	195.104240
Serial-1-2	0.000332	0.001277	0.022277	0.007460	0.103172	16.170601	178.519476	194.824595
Serial-1-3	0.000076	0.000719	0.022279	0.007464	0.103246	16.179242	178.605121	194.918147
Average-1	**0.000161**	**0.000904**	**0.022277**	**0.007463**	**0.103226**	**16.181307**	**178.633656**	**194.948994**
Parallel-2-1	0.000095	0.000387	0.011381	0.003832	0.053392	8.382784	91.921619	100.373490
Parallel-2-2	0.000095	0.000383	0.011373	0.003878	0.053470	8.325325	91.971715	100.366239
Parallel-2-3	0.000095	0.000380	0.011592	0.003974	0.058623	9.362379	91.605328	101.042371
Average-2	**0.000095**	**0.000383**	**0.011449**	**0.003895**	**0.055162**	**8.690163**	**91.832887**	**100.594033**
Parallel-4-1	0.001142	0.000255	0.006426	0.002031	0.029512	4.578708	49.977982	54.596056
Parallel-4-2	0.008405	0.000269	0.007953	0.002670	0.036714	4.602882	50.432032	55.090925
Parallel-4-3	0.009113	0.000287	0.008103	0.002702	0.037318	4.623183	50.914203	55.594909
Average-4	**0.006220**	**0.000270**	**0.007494**	**0.002468**	**0.034515**	**4.601591**	**50.441406**	**55.093963**
Parallel-8-1	0.000331	0.000116	0.003193	0.001064	0.014594	2.199274	25.303444	27.522016
Parallel-8-2	0.000369	0.000115	0.003181	0.001068	0.014768	2.241608	24.934345	27.195454
Parallel-8-3	0.000269	0.000131	0.003949	0.001310	0.017984	2.740295	30.231344	32.995282
Average-8	**0.000323**	**0.000121**	**0.003441**	**0.001147**	**0.015782**	**2.393726**	**26.823044**	**29.237584**
Parallel-16-1	0.000564	0.000181	0.002806	0.000864	0.010717	1.186917	14.012097	15.214146
Parallel-16-2	0.008854	0.000109	0.002786	0.000936	0.012604	1.328713	14.418916	15.772918
Parallel-16-3	0.000635	0.000177	0.004069	0.001311	0.017787	2.557128	29.282915	31.864022
Average-16	**0.003351**	**0.000156**	**0.003220**	**0.001037**	**0.013703**	**1.690919**	**19.237976**	**20.950362**
Parallel-32-1	0.001615	0.000272	0.007615	0.029724	1.198912	30.096425	194.334631	225.669194
Parallel-32-2	0.001325	0.000210	0.004682	0.018359	0.720814	16.074555	123.304675	140.124620
Parallel-32-3	0.001378	0.000275	0.007280	0.028589	1.126632	27.095695	196.513991	224.773840
Average-32	**0.001439**	**0.000252**	**0.006526**	**0.025557**	**1.015453**	**24.422225**	**171.384432**	**196.855885**
Parallel-64-1	0.002662	0.000372	0.001944	0.123391	3.003956	26.337025	220.848665	250.318015
Parallel-64-2	0.002646	0.000308	0.001586	0.117973	2.934098	29.137958	242.532685	274.727254
Parallel-64-3	0.002977	0.000412	0.002095	0.129208	3.157864	34.632563	264.072124	301.997243
Average-64	**0.002762**	**0.000364**	**0.001875**	**0.123524**	**3.031973**	**30.035849**	**242.484491**	**275.680837**

Figure 3. CPU block computation time for each round and their averages. The total column on the right shows the total time to compute all 7 blocks.

Version-Round	Block 1	Block 2	Block 3	Block 4	Block 5	Block 6	Block 7
GPU-1	0.268132	0.010174	0.012876	0.068571	0.213775	100.521264	N/A
GPU-2	0.293886	0.012846	0.014653	0.071626	0.928156	23.800998	521.647613
GPU-3	0.375549	0.013009	0.036673	0.126372	4.199760	31.616015	N/A
GPU-4	0.251697	0.010058	0.023764	0.010279	0.769245	1.114231	117.389423
GPU-5	0.269128	0.012149	0.025532	0.082860	2.613284	30.755869	86.591953
GPU-6	0.253049	0.013294	0.013154	0.023225	1.526046	1.649155	66.193755
Average	**0.285240**	**0.011922**	**0.021109**	**0.063822**	**1.708378**	**31.576255**	**197.955686**

Figure 4. GPU block computation time for each round and their averages.

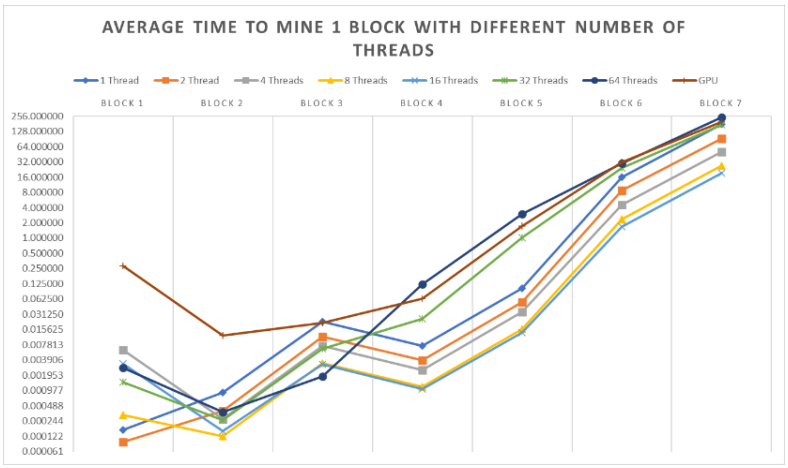

Figure 5. Graph comparing the average time to mine 1 block (lower is better). Note the $\log_2$ scale on the y-axis.

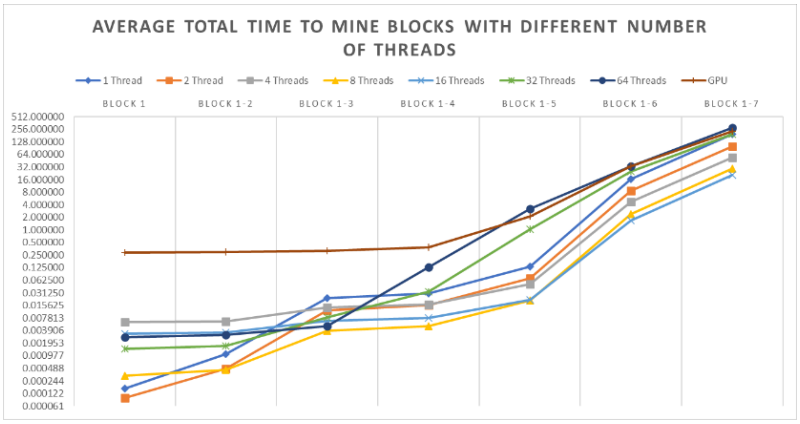

Figure 6. Graph comparing the average total time to mine blocks 1-7 (lower is better). Note the $\log_2$ scale on the y-axis.

5. Conclusion

This work implemented a custom blockchain along with the SHA256 hashing algorithm that allowed new blocks, with the most up-to-date data, to be appended to the blockchain. This successfully simulated the Bitcoin mining process and served as a baseline for comparing parallel and GPU offloaded mining computations. OpenMP was used to parallelize and offload the algorithms and it illustrated excellent scalability; however, it had limited granularity for GPU offloading. It did not permit the same level of parallelism and scalability as on the CPU and provided an inefficient GPU mining computation.

Given more time and resources, the project could be expanded to more trials and constrained mining scenarios to ensure all versions of the programs find the same valid nonce. This would eliminate differences in block mining time as the same nonce would be sought after. However, this method would provide less accurate real-world scenarios. It may be best to count the number of hashed performed per second or evaluate other metrics to see the throughput of each version of the program. Doing so on the GPU would be difficult because a count variable synchronized across all threads would significantly degrade performance. Overall, with more trials and longer mining sessions, one could derive the best version of the simulated Bitcoin mining process and conclude that GPU or ASIC mining for Bitcoin is the most performant.

References

[1] “SHA-2,” *Wikipedia*, Aug. 25, 2022. https://en.wikipedia.org/wiki/SHA-2#Pseudocode

[2] K. Shirriff, “Mining Bitcoin with pencil and paper: 0.67 hashes per day,” *Ken Shirriff’s Blog*. http://www.righto.com/2014/09/mining-bitcoin-with-pencil-and-paper.html

[3] OpenSSL Foundation, Inc, “OpenSSL Cryptography and SSL/TLS Toolkit,” *Openssl.org*, 2019. https://www.openssl.org/

[4] M. Malík, “HWiNFO - Free System Information, Monitoring and Diagnostics,” HWiNFO, 2011. https://www.hwinfo.com/

[5] K. Shirriff, “Bitcoins the hard way: Using the raw Bitcoin protocol,” *Ken Shirriff’s Blog*. http://www.righto.com/2014/02/bitcoins-hard-way-using-raw-bitcoin.html

Source code: https://github.com/WebKingdom/BTC_Miner_OMP