Performance Report

Gabriella Munger, Justin Gomez, Sebastian Fernandez

1.0 Purpose

This program was written to implement a simple blockchain with a tamper-proof ledger and mining through the use of a cryptographic hash puzzle for the purpose of gathering data to compare the efficiency of solving hash puzzles with different difficulty settings and numbers of threads.

2.0 Requirements

There are no further requirements to run this experiment outside of what is typically required to run a Go program.

2.1 Background Knowledge

Bitcoin is a system initially proposed by Satoshi Nakamoto in a white paper published in 2008. It effectively functions as a ledger that serves as a decentralized database for transactions of the system's currency "Bitcoins". Transactions are stored in blocks which are then linked together, similar to a linked list data structure. Miners mine blocks by solving cryptographic puzzles, and the first miner to propose a block and have it verified by a majority of other miners gets the block reward, consisting of bitcoins. It has several key features, however for this project only a couple are necessary to understand: the cryptographic hash puzzle and the tamper proof ledger.

To understand the hash puzzle, examine a hash function with inputs H and n and output h(H|n). H is the previous block's hash (output). A miner will then hash H concatenated with a number n (nonce) until the output hash h(H|n) has a certain number of leading zeros according to the difficulty level. By requiring a larger number of leading zeros, it is more difficult to find an acceptable nonce that will produce the desired hash output.

The use of the previous block's hash as part of the input for the current block's hash means that the ledger is tamper-proof. To alter one block in the chain, all previous blocks would need to be changed as well, through to the genesis block which is typically hardcoded into the program. In this implementation, the Logger keeps track of the blocks, so they are tamper-proof.

Finding a nonce that meets the difficulty requirements can be time consuming. Using more threads can take greater advantage of a machine's computational abilities to make the process more efficient. Go's GOMAXPROCS function is one way to decide how many threads are used by a program.

3.0 Instructions

To run this program, first clone the git repository. Then, type "go run *.go" into the terminal line. The program will then give a brief description of how the program works and ask the user to type an integer input for each of the following: (1) difficulty level, (2) number of miners in the system, (3) number of blocks added to the chain, and (4) number of concurrent threads used.

To gather enough data for analysis, run the program several times. The description of input used in this experiment is described further in 4.2 Data Analysis.

4.0 Output

The output consists of the information for the first block containing the relevant fields mirrored from Bitcoin's repository. Additionally, one the user enters the desired information, the logger will start running and output when a block is verified, when the blockchain height increases, and when the logger finishes running. Once the logger ends, the total puzzle time is given in milliseconds.

4.1 Code Output

This is an example of the program's output with input values of 2 for difficulty level, 2 for number of miners, 1 for blockchain length, and 4 for GOMAXPROCS number. The genesis block information is printed and a message is printed when a new block is verified. When the desired chain length is reached, the message "Ending Logger" and the total runtime to construct the chain is printed.

```
--------------------------------------------------------------------------------
Please choose how many leading bits you would like to be compared.
2
Please input the number of miners you would like to simulate.
2
Please input the number of blocks you would like to add to the blockchain.
1
Please input the number of concurrent threads you would like to use.
4
Thanks! We will start the simulation with 2 miners on difficulty level 2 for 1 rounds using a GOMAXPROCS number o
f 4 .
--------------------------------------------------------------------------------
Started Logger
Printing Block information:
        hash: d04b98f48e8f8bcc15c6ae5ac050801cd6dcfd428fb5f9e65c4e16e7807340fa
        confirmations: 1
        size: 1
        stripped size: 1
        weight: 1
        version: 1
        version hex: 00000001
        merkle root: ce922519a3c3ecaf9b0986c2449c7680895c15f4b0e9818e994e14a4d28b6aaf
        transaction: bde4693e55a336ff81ab238ce20cae1dd9c8ba03b9b8f43963f5569bf3cf5229
        time: 1638322310
        median time: 1638322310
        nonce: 0
        bits: 1d00ffff
        difficulty: 2
        chain work: 00000000000000000000000000000000000000000000000000000001f501f501f5
        prevBlockHash: e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
Block was verified!! from 0
incremented chain height
Ending Logger
the total puzzle time is :  42.008089ms
```

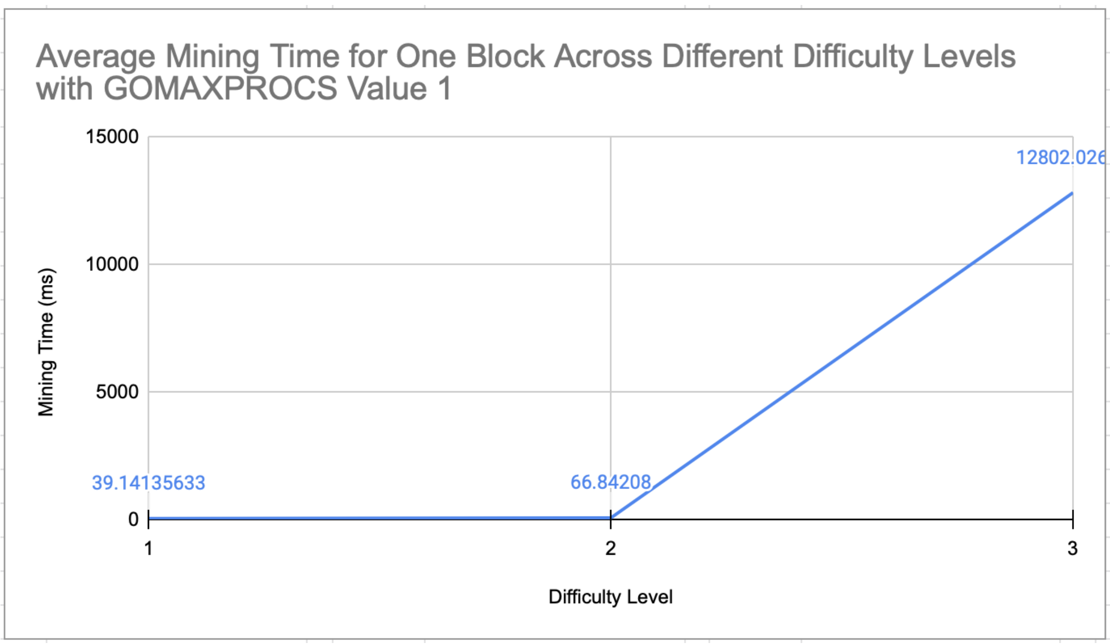
4.2 Data Analysis

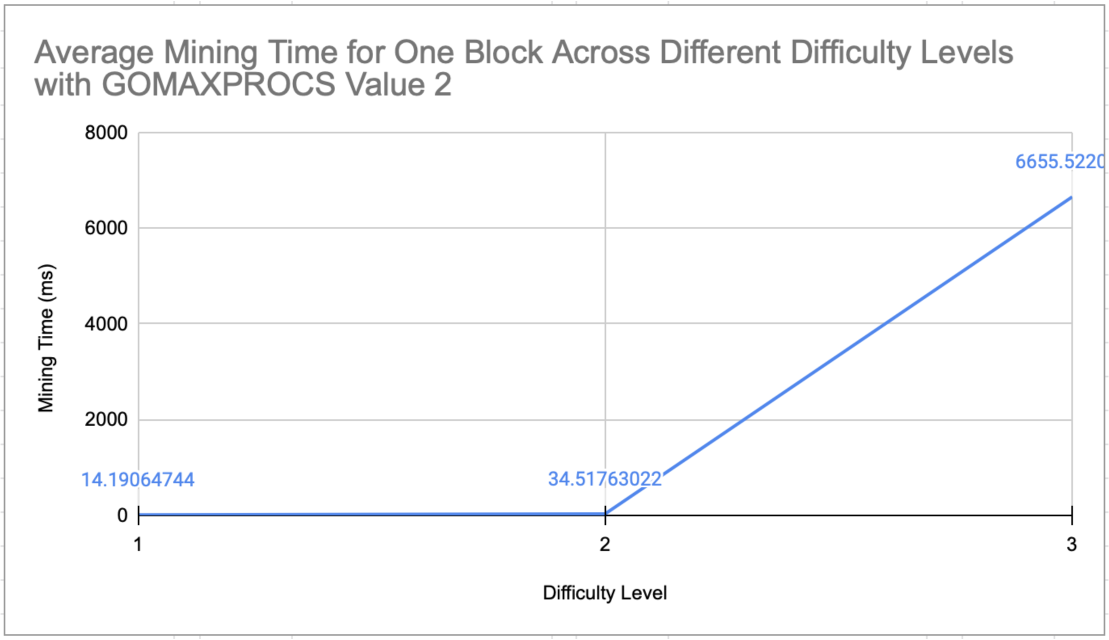
This project investigates the difference in mining time for one block between systems with different difficulty levels and systems with different GOMAXPROCS values. To maintain consistency, the tests were always run on a system with two miners mining for a chain with length one. Data was collected by running the program nine times for each combination of GOMAXPROCS numbers up through eight and difficulty levels up through three. Because of the runtime when using difficulty levels higher than three, it was not feasible to perform the experiment on higher difficulty levels while gathering enough data to draw meaningful conclusions. The collected data was then put into a Google Sheet for creation of charts.

4.3 Discussion of Results

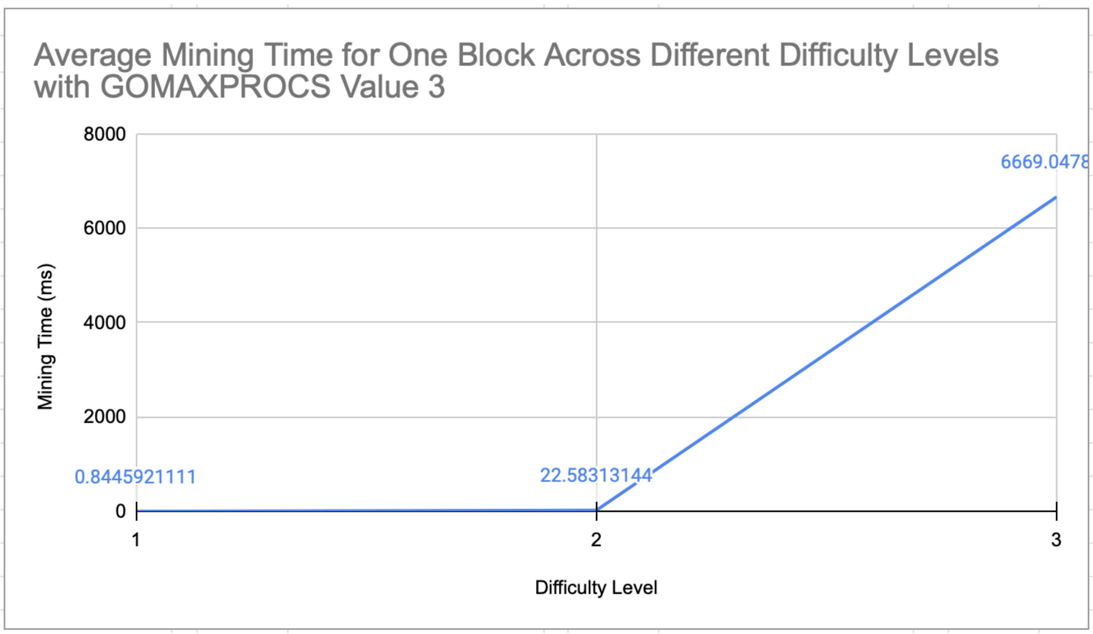
4.3.1 Difficulty Level vs. Time

Overall, the results indicate that the relationship between difficulty level and mining time for systems with two miners using any GOMAXPROCS value follows a similar trajectory to a parabola. Using nine repetitions to find the average runtime for each difficulty level may result in error due to a small sample size, however it was the most feasible number of repetitions for this execution of the experiment as the data had to be manually transferred to a spreadsheet. Figures are included for each GOMAXPROCS value, however the same trend is followed for each and the graphs appear almost identical, save for the data labels describing the actual mining time value.

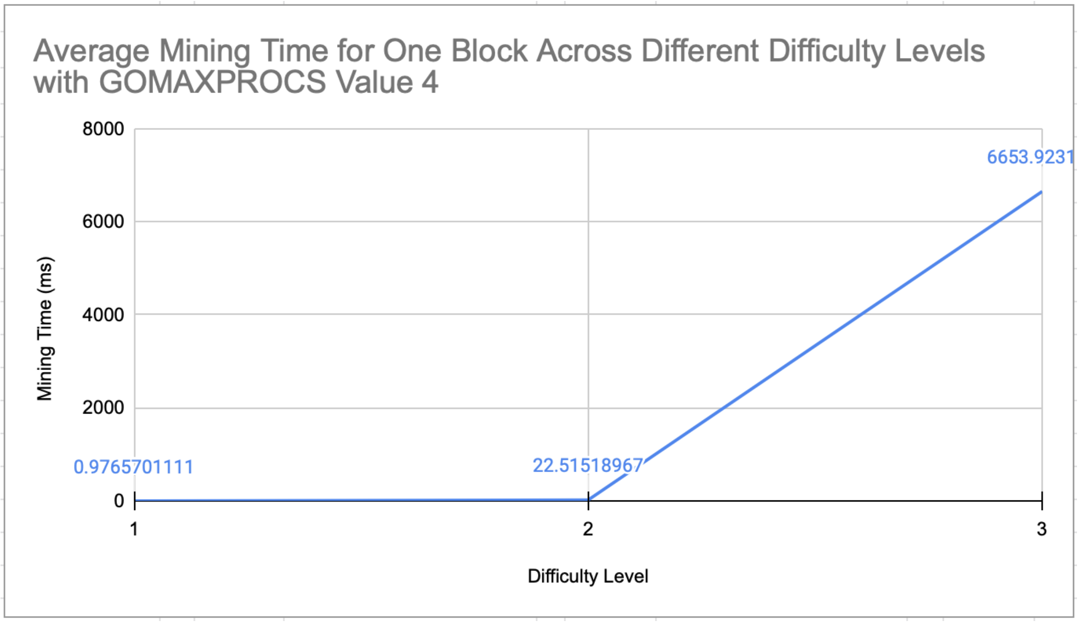


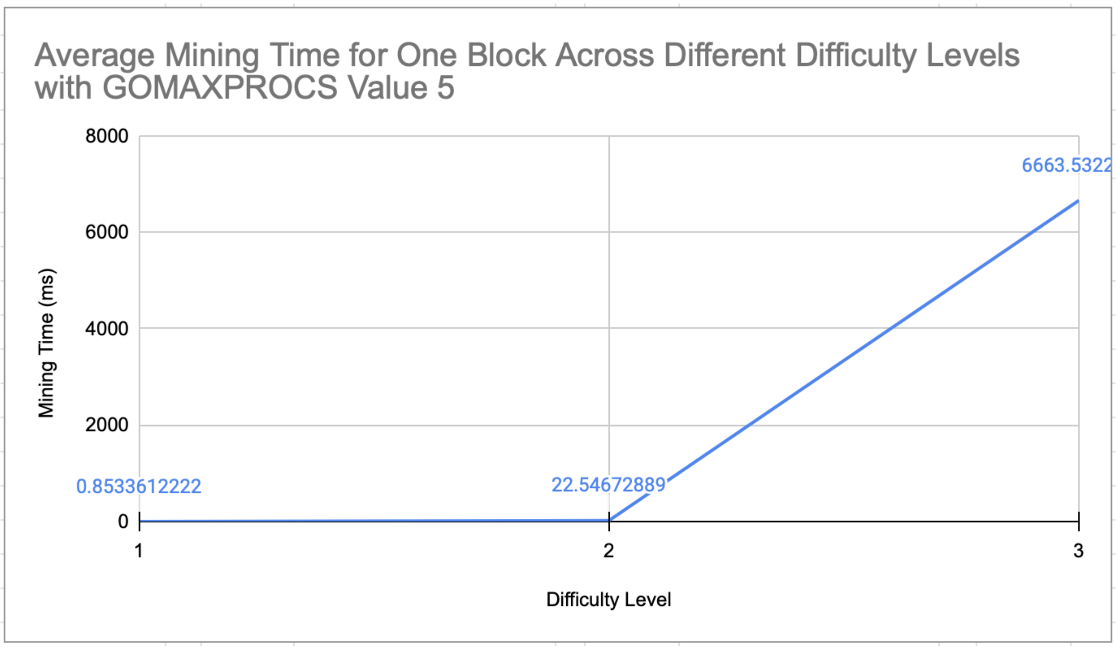


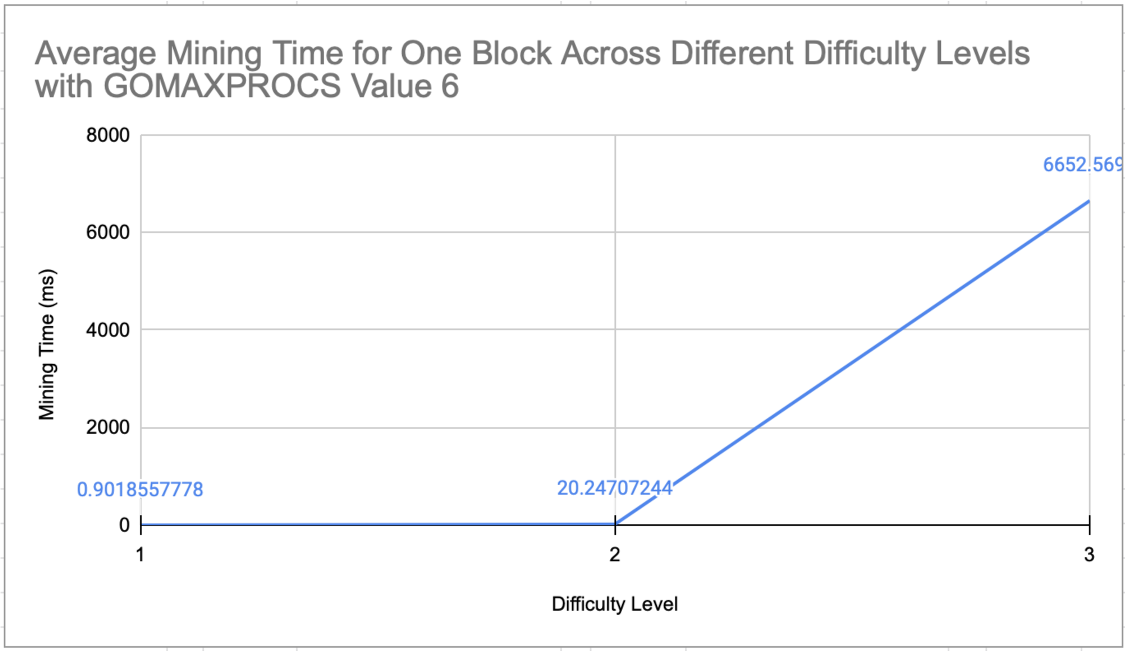
Average Mining Time for One Block Across Different Difficulty Levels with GOMAXPROCS Value 2
Mining Time (ms)
8000
6000
4000
2000
0
14.19064744
34.51763022
6655.5220
1
2
3
Difficulty Level

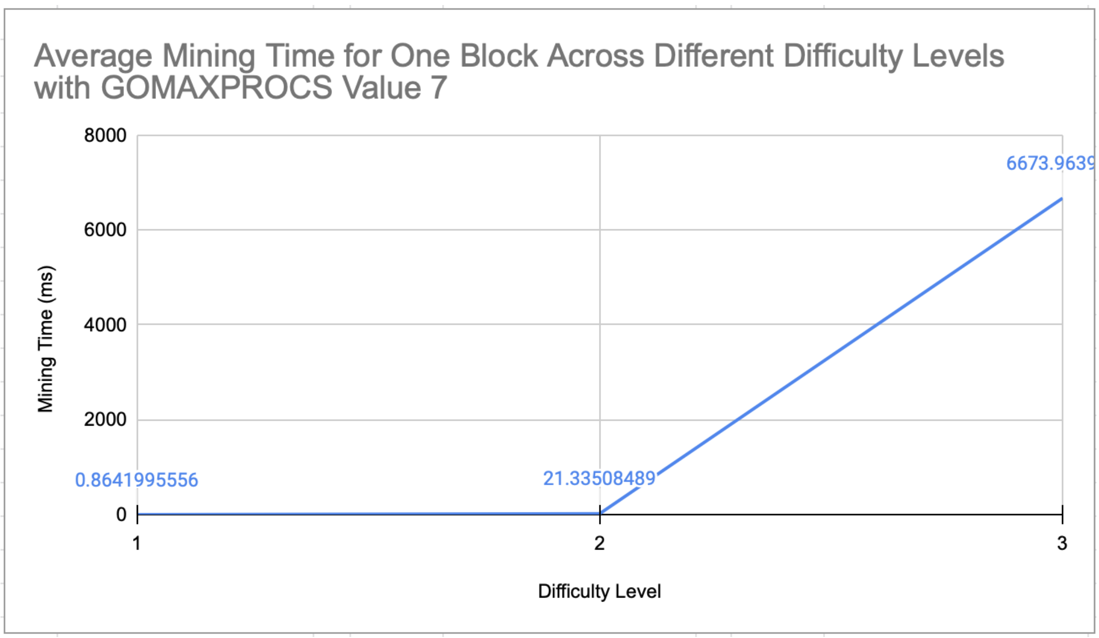


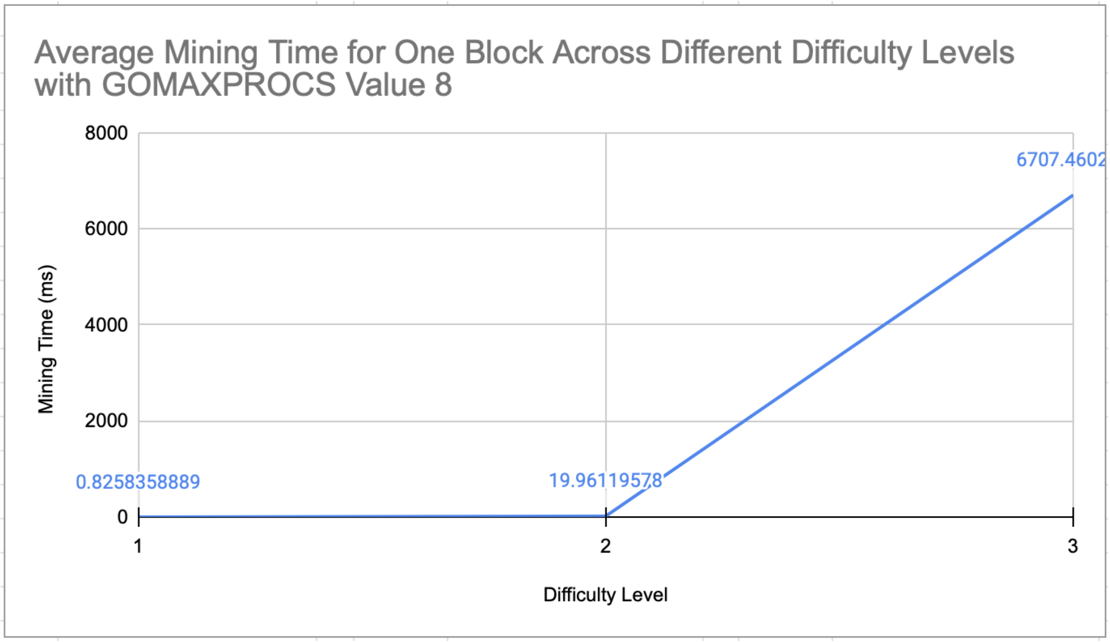
Average Mining Time for One Block Across Different Difficulty Levels with GOMAXPROCS Value 3
Mining Time (ms)
8000
6000
4000
2000
0
0.8445921111
22.58313144
6669.0478
1
2
3
Difficulty Level


Average Mining Time for One Block Across Different Difficulty Levels with GOMAXPROCS Value 4
Mining Time (ms)
8000
6000
4000
2000
0
0.9765701111
22.51518967
6653.9231
1
2
3
Difficulty Level

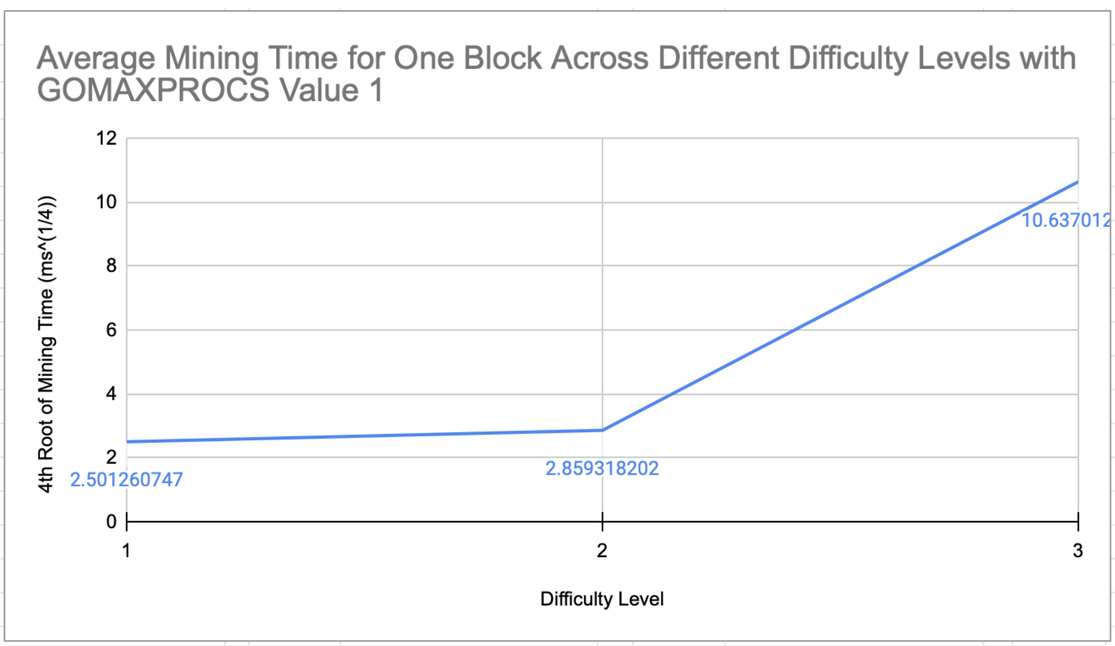

Average Mining Time for One Block Across Different Difficulty Levels with GOMAXPROCS Value 5
Mining Time (ms)
8000
6000
4000
2000
0
0.8533612222
22.54672889
6663.5322
1
2
3
Difficulty Level


Average Mining Time for One Block Across Different Difficulty Levels with GOMAXPROCS Value 6
Mining Time (ms)
8000
6000
4000
2000
0
0.9018557778
20.24707244
6652.569
1
2
3
Difficulty Level

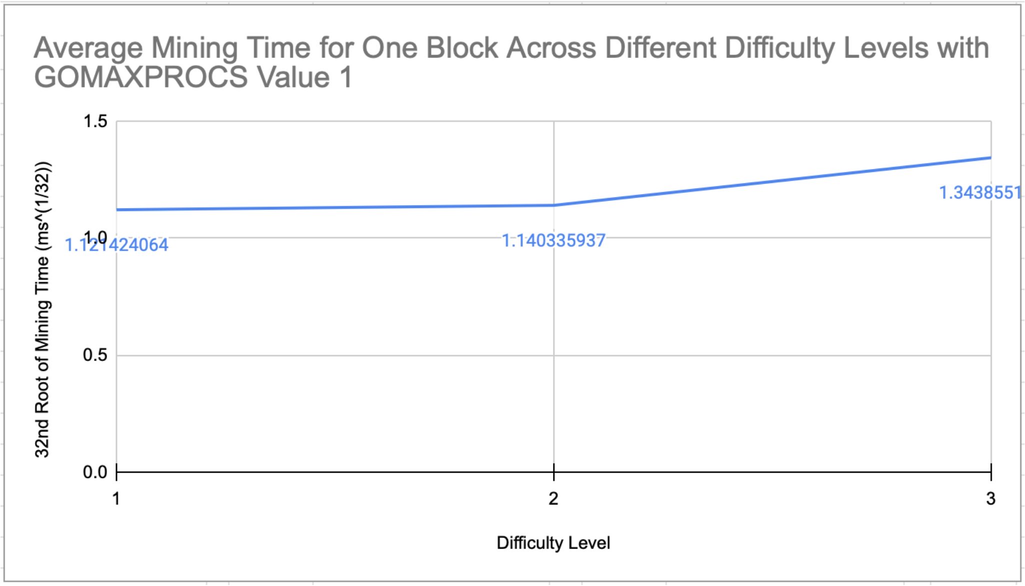

Average Mining Time for One Block Across Different Difficulty Levels with GOMAXPROCS Value 7
Mining Time (ms)
8000
6000
4000
2000
0
0.8641995556
21.33508489
6673.9639
1
2
3
Difficulty Level



Because there is such a large increase in mining time between difficulty levels two and three, graphing the fourth root of the mining time can more clearly show the trajectory between the difficulty levels. Because this still shows a roughly parabolic relationship, it implies that difficulty level and mining time are related to each other through a large exponent number.

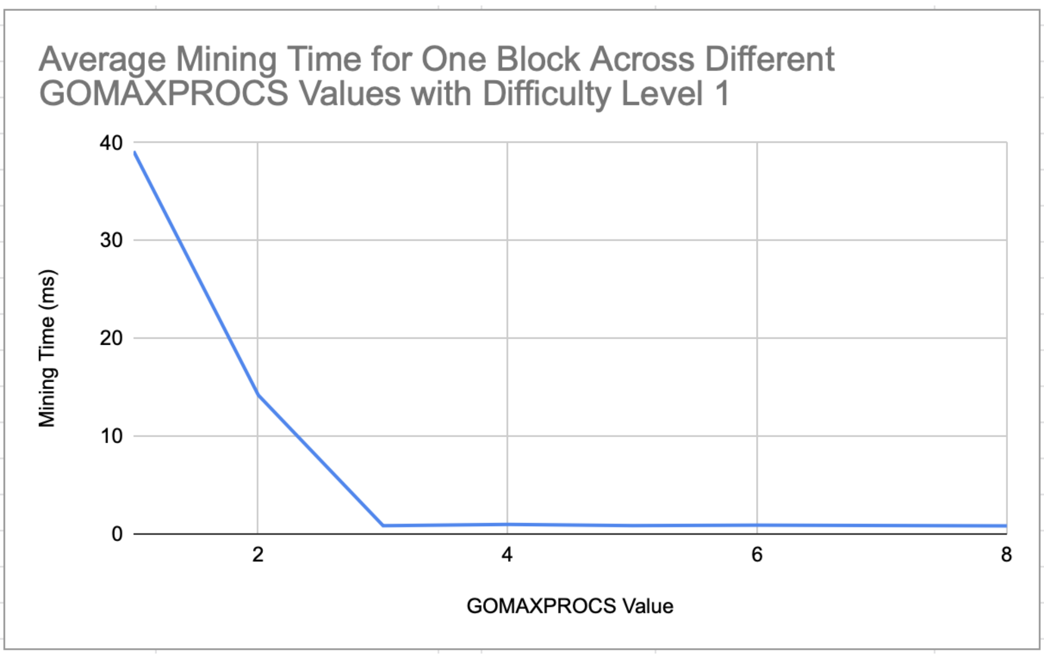


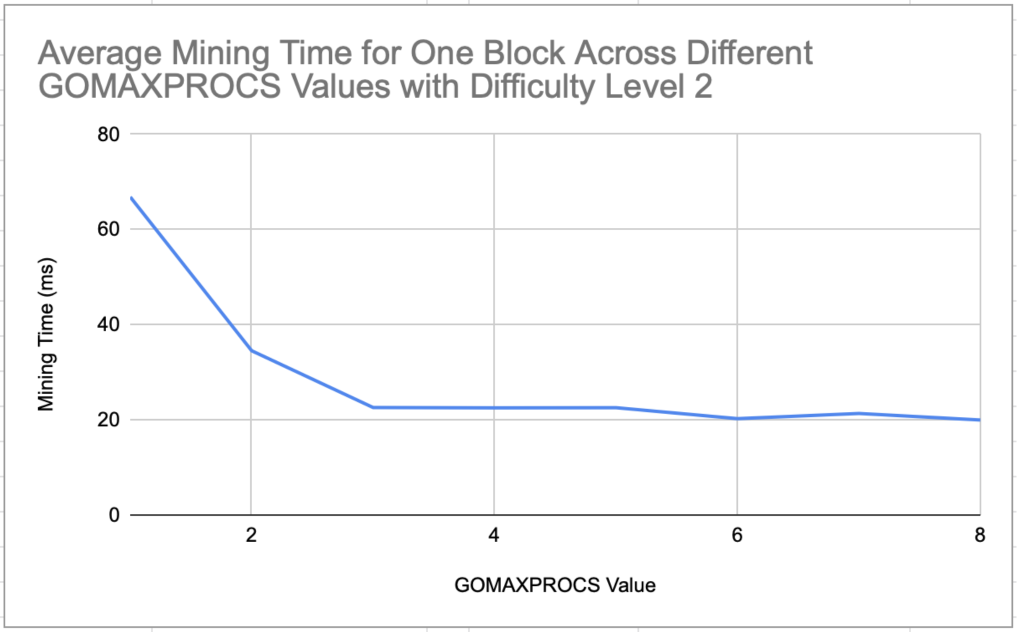
After raising the mining time to a power of 1/32, there appears to be an almost linear relationship with difficulty level, implying that mining time is related to difficulty level raised to the power of 32. More testing would be necessary to definitively assert this relationship.



4.3.2 GOMAXPROCS Number vs. Time

The figures below illustrate the average mining time for one block across different GOMAXPROCS values with difficulty level remaining constant. Consistently for difficulty levels one and two, there is a significant increase in efficiency between GOMAXPROCS values of one, two, and three, but after that point there is no significant gain from using more threads.





Slightly differently from difficulty levels one and two, the significant difference in mining time is only present between GOMAXPROCS values one and two for the figure below.