Is Ethereum's ProgPoW ASIC Resistant?

Jason Orender, Ravi Mukkamala and Mohammad Zubair
Department of Computer Science, Old Dominion University, Norfolk, VA, U.S.A.

Keywords: ASIC, Blockchains, Cryptocurrencies, Ethereum, GPU, Ethash, ProgPoW.

Abstract: Cryptocurrencies are more than a decade old and several issues have been discovered since their then. One of these issues is a partial negation of the intent to "democratize" money by decentralizing control of the infrastructure that creates, transmits, and stores monetary data. The Programmatic Proof of Work (ProgPoW) algorithm is intended as a possible solution to this problem for the Ethereum cryptocurrency. This paper examines ProgPow's claim to be Application Specific Integrated Circuit (ASIC) resistant. This is achieved by isolating the proof-of-work code from the Ethereum blockchain, inserting the ProgPoW algorithm, and measuring the performance of the new implementation as a multithread CPU program, as well as a GPU implementation. The most remarkable difference between the ProgPoW algorithm and the currently implemented Ethereum Proof-of Work is the addition of a random sequence of math operations in the main loop that require increased memory bandwidth. Analyzing and comparing the performance of the CPU and GPU implementations should provide an insight into how the ProgPoW algorithm might perform on an ASIC.

1 INTRODUCTION

Decentralized control is one of the founding principles of cryptocurrency adoption (Narayanan, A., Bonneau, J., Felten, E., Miller, A., Goldfeder, S., 2016). However, several cryptocurrencies have been developed that overtly discard this idea in favor of enhanced speed and throughput, XRP (Armknecht, 2015) and Stellar (Mazieres, 2015) are two prominent examples of these so-called "federated" models.

Among those cryptocurrencies that continue to embrace decentralization, ensuring Application Specific Integrated Circuit (ASIC) resistance is becoming a popular theme (Taylor, 2017). An Application Specific Integrated Circuit (ASIC), essentially a custom microprocessor optimized in hardware to run specific code, is able to calculate the SHA-256 hashing algorithm in excess of 100,000 times faster than a top of the line CPU (Balaji, A., 2018). Since computing SHA-256 hashes is the primary computational effort involved in the proof-of-work (PoW) algorithms, use of ASICs has a significant impact on cryptocurrencies that use these algorithms.

PoW algorithm is being used in several cryptocurrencies and its implementation on an ASIC is emblematic of the struggle that cryptocurrency communities are facing to maintain decentralized control. ASICs are expensive to create and, as a result, miners with the most resources tend to use ASIC rigs to mine the cryptocurrency of their choice. This makes all other hardware used by less affluent miners obsolete. This results in the centralization of the hash power in significantly fewer miners than was originally envisioned by the cryptocurrency community (Wang W., Hu, P., Niyato, D., Wen, Y, 2019). The differences between ASIC and GPU cryptocurrencies were summarized in a 2018 article (ComputeNorth, 2018).

As a result of this progressive centralization of hash power, some options are being explored by cryptocurrency communities that would negate this advantage. One option is the designing of algorithms that are ASIC resistant. The other option is the periodic change of the algorithm so that current ASICs quickly become obsolete. This discourages designers and manufacturers from producing new ASICs, as it would not be cost effective or timely. ProgPoW is an example of the first option.

One example of these efforts is Hcash, a plug-and-play blockchain (Hcash, 2019). Another example is HashCore, a set of proof-of-work functions developed for general purpose processors (Gorghiades, Y., Flolid, S., Vishwanath, S., 2019).

In this paper, we focus on Ethereum's ProgPoW algorithm. We look into the ASIC resistance properties of this algorithm by exploring the scalability of the algorithm. This is achieved by

Orender, J., Mukkamala, R. and Zubair, M.
Is Ethereum's ProgPoW ASIC Resistant?.
DOI: 10.5220/0008909203100316
In *Proceedings of the 6th International Conference on Information Systems Security and Privacy (ICISSP 2020)*, pages 310-316
ISBN: 978-989-758-399-5; ISSN: 2184-4356

isolating the PoW code and comparing its performance on a multi-threaded CPU implementation and a GPU implementation. Analyzing the memory access patterns and the scalability of the algorithm will provide insights into its claimed ASIC resistance properties. In fact, in the completely different context of N-body simulation, a similar study as ours was conducted earlier to compare the efficacy of ASICs, FPGAs, GPUs, and general Purpose Processors on specific problems (Hamada, T., Benkrid, K., Nitadori, K., Makoto, T., 2009). Their conclusion, measured in terms of Mflops/$, is that GPUs far outperform their ASIC, FPGA, and CPU counterparts. This is in line with our conclusions, even though in a different context.

The paper is organized as follows. In section 2, we look into the background and related work that prompted the current work. Section 3 highlights some of the important aspects of ProgPoW that make it suitable for GPUs. In section 4, we discuss the adopted methodology. Section 5 describes some of the implementation details. In section 6, we discuss the results from our experiments. Finally, section 7 summarizes the contributions of the paper and discusses some future work.

2 BACKGROUND

The cryptocurrency and blockchain revolution started with the Bitcoin proposal by the pseudonymous Satoshi Nakomoto in 2008. The primary motivation for this proposal was to introduce an electronic monetary system with decentralized control. To this end, it is based on a peer-to-peer system of nodes that together maintain Bitcoin and its underlying blockchain infrastructure. A Proof-of-Work (PoW) algorithm is used here to discourage malicious actors from inundating network nodes with denial of service attacks, thereby damaging the trustworthiness of the Bitcoin network. This is also an integral part of the Bitcoin consensus algorithm. Here, the suggested proof-of-work algorithm requires computing the hash of a block header that is restricted to be within certain bounds. A field in the block header, called a nonce, is a 4-byte random integer that could be chosen by a data miner to produce a 32-byte hash for a block within the given bounds. The range, expressed in terms of the number of leading zeros in a hash, dictates the computational difficulty in mining a block. Bitcoin uses Hashcash as its PoW algorithm. Ethereum has also adopted proof-of-work as its consensus algorithm; Ethash is the PoW algorithm used by Ethereum.

To give a short overview of how the Ethash PoW works, the following generalized procedure applies:

1. Calculate the seed, which is generated by scanning through all block headers up to that point.
2. Compute a 16MB pseudorandom cache based on that seed.
3. Generate a 1GB Directed Acyclic Graph (DAG) based on the cache. This DAG will grow linearly with time as the blockchain expands.
4. The mining algorithm will systematically select pseudorandom slices of the DAG and hash them together.
5. Specific pieces of the DAG can be regenerated at will from the cache for quick verification of the resultant hash.

With the increasing popularity and price of cryptocurrencies, data mining activity has become very attractive for miners. Since the first data miner who mines the next block in the blockchain gets the reward for mining the block, there is a competition among the miners to be the first one to mine. In the context of PoW, this translates directly to having more computational power. The crux of mining difficulty with the PoW used in Bitcoin lies with the SHA-256 hash function. Similarly, Ethereum uses the Keccak-256 algorithm for its proof-of-work. This algorithm is related to the widely used SHA3. Once again, the difficulty lies with the hash computation. Since these computations require manipulating 32-bit words, performing 32-bit modular additions, and some bitwise logic, it is easy to implement them in hardware.

The first generation of mining used CPUs with ability to compute about 20 million hashes/second. The second generation replaced CPUs with GPUs. These are designed with parallelism in mind, so several of the hash computations can be done simultaneously. The third generation started with the advent of FPGAs, or Field Programmable Gate Arrays. With a careful configuration, one can obtain 1 Ghash/second hash rates. The fourth generation is the Application-Specific Integrated Circuits, or ASICs. These are ICs designed, built, and optimized for a specific purpose. But the cost of ASIC mining, due to the expense of developing and manufacturing the ASIC, is not friendly to small miners. They are primarily used by professional mining centres, also termed "mining farms" (Narayanan, A., Bonneau, J., Felten, E., Miller, A., Goldfeder, S., 2016). This has completely changed the original intent of decentralized peer-to-peer data mining attributed to Satoshi Nakomoto, as well as other early developers of cryptocurrencies.

In order to reduce the cost of mining, which is quite exorbitant with the current proof-of-work algorithms, Ethereum has plans to shift to Proof-of-Stake (PoS). As part of this transition, Ethereum developers are proposing solutions to ward off the professional mining centres from taking over the network and to maintain a truly peer-to-peer mining environment. ProgPoW, or programmatic proof-of-work, is an attempt to eliminate the gap between the GPU miners and the ASIC miners by making ASIC mining less efficient and tailoring the algorithm to be more easily exploitable by GPUs. This is achieved by introducing the following changes which are intended to make mining with ASICs impractical. The first change is adding a random sequence of calculations which make it impossible to create an ASIC chip with a fixed workflow. Second is adding reads from a small low-latency cache that supports random addresses, which limits ASIC's capabilities and performance. Third, the dynamic random access memory (DRAM) read size is increased from 128 to 256 bytes to favor GPUs.

In the rest of the paper, we look into the operational efficiency aspects of ProgPoW measured through experimentation.

3 KEY FEATURES OF ProgPoW

Ethereum uses the Keccak-256 algorithm for its proof of work. This algorithm is related to the widely used SHA3 algorithm and has numerous technical advantages over the previously mentioned SHA-256 hashing algorithm, including enhanced collision resistance and preimage resistance. This preimage resistance has been explicitly quantified between SHA-256 and SHA3 for the purpose of evaluating each algorithm for implementation on quantum computers (Amy 2016), and the multiple advantages of Keccak over SHA-256 are spelled out in great detail in the original paper detailing the comparison and analysis study of SHA3 finalists (Alshaikhli, 2012).

The ProgPoW uses the same Keccak-256 hashing algorithm, but splits up the words into 32 bit chunks instead of 64 bit chunks to make the algorithm more GPU friendly and reduce total power consumption.

The Ethereum PoW uses a large Directed Acyclic Graph (DAG) to direct the PoW calculation, and ProgPoW increases the so-called "mix state" of this DAG. This governs the complexity of how the DAG is utilized to process the block and produce the arguments that are submitted to the hash function. This ultimately results in more off-chip memory

The following pseudocode gives a general idea regarding how the algorithm functions:

```
progPowHash(nonce, header, DAG, block):

   // initializing 256 bit digest
   dgst[8] <- 0 // 8 4-byte integers


    // use keccak hashing algorithm to
    // generate seed
    seed <- keccak(header, nonce, dgst)

    // use pseudorandom numbers based on
    // seed to fill mix table
    mix[16 x 32] <- rand(seed)

    // cycle through the inner loop
    for (i in 1:64):
        innerLoop(mix, DAG, block)

   // initializing lanes array
   l[16] <- 0

   BASIS <- 0x811c9dc5
   PRIME <- 0x1000193
   // using the mix to generate lane digest
    for (i in 0:15):
        l[i] <- BASIS
        for (j in 1:dim(mix)):
            l[i] <- (l[i] ^ mix[j])*PRIME

   // creating 256 bit digest from the
   // lane digest array
    for (i in 0:7):
        dgst[i] <- BASIS
    for (j in 0:15):
        dgst[j%8] <- (dgst[j%8]^l[j])*PRIME

    hash <- keccak(header, nonce, dgst)
    return(hash)


innerLoop(mix, DAG, block):
   initilizePoW(mix)
   n1 <- 0
   n2 <- 0
   m <- dim(mix)
   for (i in 1:18):
       for (j in 1:11):
           src <- mix[(n1++)%m]
           dst <- mix[(n2++)%m]
               for (k in 1:16):
                   o <-
                     mix[k*src]%CacheSize
                   mix[k*dst] <-
                     mix[k*dst] ⊕ DAG[o]

   for (i in 1:18):
       r <- rand(m^2)
       src1 <- r%m
       src2 <- r/m
       sel  <- rand(block)
       dst  <- mix[(n2++)%m]
       for (j in 1:16):
           d <- randmath(mix[j*src1],
               mix[j*src2], sel)
           mix[j*dst] <- mix[j*dst] ⊕ d
```

```
// ⊕ = merge operation
// CacheSize = 16KB
// rand(x) = random int in the range 0:x
// randmath(x,y,n)= random combination of
//     x and y using the nth arrangement
//     of arithmetic operators defined
//     separately

//    = notable difference between
// ProgPoW and standard Ethereum PoW



// initialize the mix
initilizePoW(mix):
    for (i in dim(mix)):
        mix[i] <- i
    for (i in dim(mix)):
        n <- rand(i)
        swap(mix[n],mix[i])
```

retrievals and increases the bandwidth required to complete a calculation.

A set of pseudorandom mathematical operations was also added in the main loop, which would increase the complexity of any ASIC design in addition to requiring more off-chip memory accesses since the results of the random math govern where the next memory access might occur.

In addition, the DRAM read size was increased from 128 bytes to 256 bytes, also increasing the bandwidth required to complete a calculation.

The inputs to the *progPowHash* function essentially consist of a header generated from block data (zeroes were used in this case for experimentation), a nonce, and a pointer to the DAG. There is an additional "seed" argument in the C++ code that is not shown here. This argument is deterministic and is based on the block number.

4 METHODOLOGY

In order to carry out our study of ProgPoW's effectiveness in making effective use of GPUs but not so effective for ASICs, we have followed the below methodology in five steps.

4.1 Extract the PoW Code

First, the code from the GitHub site hosting proposed ProgPoW code (Ifdefelse, 2019) was cloned. Elements of this code that were solely used for the proof of work algorithm were extracted and implemented in a stand-alone version that only ran the PoW algorithm with specific inputs for experimental repeatability. There were several dependencies that were required from the main cpp-Ethereum code (Ethereum, 2019), and those were also extracted and placed in the stand-alone version. The goal was to create a simple version whose only purpose was to run the PoW algorithm for a specific block number. This would allow more accurate analysis of memory access patterns and benchmarking.

4.2 Test the Extracted PoW Code

To ensure that the assembled pieces of code performed in the same manner as the original code, test vectors provided on the ProgPoW GitHub site (Ifdefelse, 2019) were utilized. All code produced output identical to the test vectors page.

4.3 Generate the Directed Acyclic Graph

As previously introduced, the Ethereum PoW uses a Directed Acyclic Graph (DAG) to govern the process of the PoW calculation. In addition, ProgPoW uses the DAG to generate pseudorandom mathematical operations in an attempt to make any ASIC design more complex and less efficient, as well as require that the algorithm request randomized blocks of memory from the DAG which increases memory bandwidth (this element is not present in the current Ethereum PoW algorithm). The DAG is approximately 1 GB in size at the 30,000th block (as analyzed), though that size changes as the number of blocks in the blockchain increases. The code for generating the DAG was extracted from the Ethereum project and implemented in isolation. It takes several hours on current state of the art machines to generate this DAG.

4.4 Implement the PoW Code on Multithreaded CPU

Once the code was tested to ensure that it produced the same results as expected, a version was created to run multi-threaded on a CPU. OpenMP was used as the programming standard, and it performed as expected, meaning that as the number of threads doubled, the time required to calculate a hash very nearly halved. More details will be explored in the results section.

Creation of this version had a dual use both as a stepping-stone to get familiar with the code before attempting a GPU implementation, and as a point of

comparison for how an ASIC might manage memory accesses.

4.5 Implement the PoW Code on GPUs

The final coding step was to implement the program on a GPU. CUDA was used as the programming standard, and this was a straightforward implementation. No extra optimization was attempted using local memory or cooperative groups, though that could be considered as a next step if desired. To get a general idea of the performance difference between a single (simple) GPU core and a single (more complex) CPU core, the hashing algorithm when run on a single GPU core takes about 92 seconds to complete given a specific set of inputs using the hardware described in Section 5. Given the same inputs, the CPU finished in about 3 seconds. The advantage of using the GPU is clearly the large amount of parallelization possible. The CPU was about 30 times faster on a core for core basis with the specific machine used for this test, however the GPU had 1280 cores available on the machine tested and this algorithm is highly parallelizable, creating a vastly scalable implementation.

5 IMPLEMENTATION

The parallelized implementation of the algorithm described consisted of creating multiple threads that run the *progPowHash* function in its entirety with the only difference being the nonce that is supplied. A sequential number was used for the nonce to ensure that no threads calculated a hash using the same nonce.

When each group of threads was executed, the results were compared, and the hash that met the criteria (14 leading binary zeroes, in the example case) was selected as the result. If more than one hash met the criteria, the hash associated with the lowest nonce was selected as the result. This ensured that results were repeatable and therefore comparable, since there is an element of randomness to this process. In an actual cryptocurrency mining setting, this would be irrelevant.

The only difference between the GPU implementation and the CPU implementation from this perspective is that a great many of the threads for the GPU implementation were submitted at once, and the GPU firmware was left to decide which threads to execute in what order, given the number of cores available in hardware; 64 blocks of 32 threads each, for a total of 2048 threads were submitted in each batch. The CPU implementation, by contrast, submitted the threads in groups of 2, 4, 8, 16, or 32 threads, and then the results were evaluated after completion of execution of each group. This allowed for greater flexibility on the CPU implementation since the number of threads actually executed to achieve a satisfactory result were closer to the minimal number of threads required. The GPU implementation needed to submit thousands of threads at a time in order to minimize the overhead involved with communicating results back to the CPU host; this resulted in thousands more threads being executed by the GPU than were absolutely required. This negative effect was largely overshadowed by the extreme efficiency gains of the GPU execution, however.

6 EXPERIMENTAL RESULTS

As previously stated, the OpenMP version was completed first, and the results were as expected (Figure 1). The execution time did not quite reduce by half when the number of threads doubled, and this can be attributed to the bandwidth limitations of the processor retrieving DAG elements from main memory.

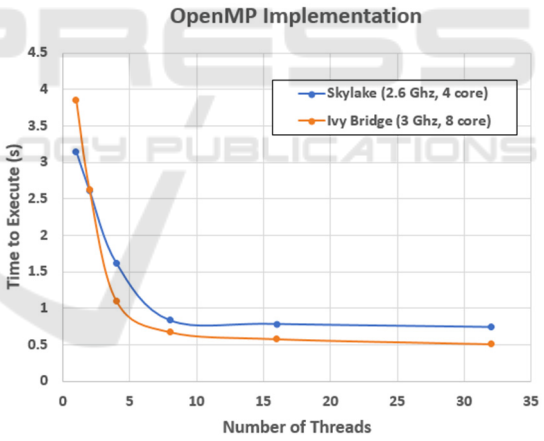

Figure 1: Execution time on two different multithreaded CPU architectures.

The advantage of GPUs over CPUs for this problem becomes evident in Figure 2. Note that the GPU becomes more efficient very early. What is immediately noticeable from here is that the efficiency gains on the GPU become more significant as the difficulty increases. One of the primary reasons for the advantage of GPU over CPUs is the memory access dynamics in a GPU (Figure 3). This fundamental difference is not replicable in an ASIC, which retains a more traditional CPU type memory access (Ren, L., Devadas, S., 2017).

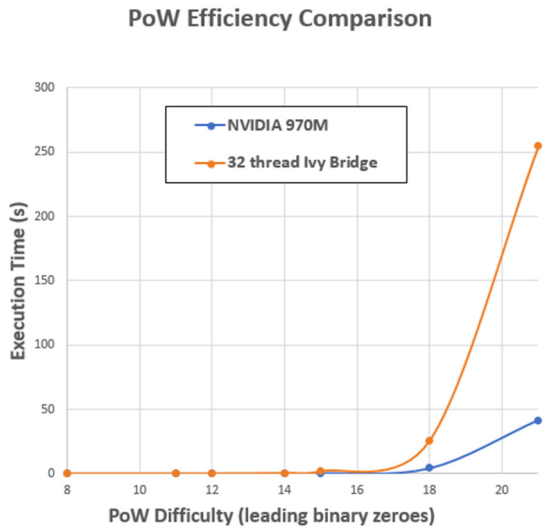

Figure 2: Comparison of execution times on a multithreaded CPU and GPUs with increasing hash difficulty.

What these results have not addressed thus far is the price to performance comparison. To give an indication of where this might lie, however, the Ivy Bridge processor used in the comparison retails for approximately $1350 MSRP, and about a $200 third party reseller sale price at time of this writing (Intel E5-1680); the GPU card used retails for about $650 MSRP, and is available for less than $200 from third party retailers (NVIDIA GTX-970M). ASIC development and production costs are typically much larger than commodity CPU costs because of the narrow applicability (Madore, P. H, 2018). Provided a similar relationship exists for a hypothetical ASIC executing ProgPoW, the cost differential should be fairly large.

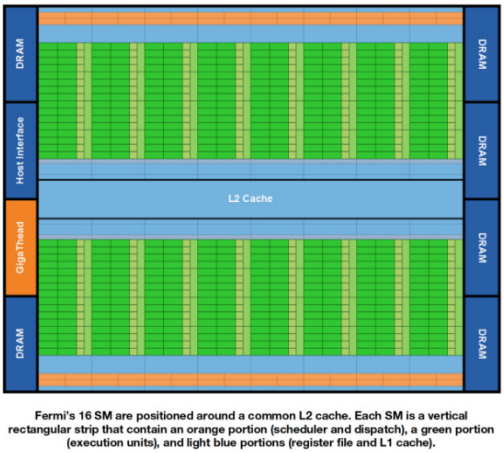

Figure 3: NVIDIA GPU memory architcture (Gao, H., 2017).

This shows the cohesive nature of the GPU memory access heirarchy. Each block of 32 cores (termed a streaming multiproccessor - SM) shares access to the L1 and L2 cache, which gives direct access to DRAM; each SM has independent access to the cache. High bandwidth memory then allows multiple accesses to be fulfilled simultaneously from DRAM.

7 CONCLUSION

What this work shows is that for this particular problem type, a cheaper GPU can outperform a commodity CPU. The reasons why this disparity exists are particularly important:

1. The memory architecture of GPUs allows much more efficient access to on-card memory than a CPU (and by extension an ASIC) can manage.
2. The great number of simple cores allowing massive parallelization for an arbitrary algorithm.

The ProgPoW algorithm is a bandwidth-hard problem. Not only does it require a great deal of off-chip memory (as a memory-hard problem would), the accesses to that memory are essentially random, negating any caching that might mitigate the effect.

Additionally, the broad applicability of GPUs across a broad spectrum of applications implies that development of enhanced bandwidth will likely continue to occur apace and will likely vastly outstrip any similar developments in the ASIC space. The highly parallelizable nature of this algorithm (as mentioned in Section II) also lends a significant GPU advantage when coupled with advances in bandwidth.

By enforcing bandwidth-hardness in the proof-of-work, cryptocurrency mining using the ProgPoW algorithm for Ethereum would likely allow commodity hardware GPUs to retain a cost for performance advantage over a custom ASIC.

Others have evaluated the level of ASIC resistance granted by memory-hard algorithms (Cho, H., 2018), and while they acknowledge that the general strategy has so far proved sound, they caveat their conclusion with the fact that there are several emerging memory technologies that might narrow this gap. As previously stated, the implementation of a bandwidth-hard algorithm, which requires additional resources above simple memory-hardness, will take this concept further and might well prove impossible for ASIC manufacturers to counter. There is little data on this subject so far, since Programmatic Proof of Work is still a relatively new idea and no major cryptocurrency has yet implemented it, but if memory performance in GPUs continues to outstrip that available to CPUs, and by extension ASICs, the idea should have merit.

REFERENCES

Narayanan, A., Bonneau, J., Felten, E., Miller, A., Goldfeder, S., 2016. *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press.

Balaji, A., 2018. A Simple Explanation of ASICs - Crypto Simplified. https://medium.com/crypto-simplified/a-simple-explanation-of-asics-35933d412b2d.

Taylor, M.B., 2017. The evolution of bitcoin hardware. *IEEE Computer.*

Wang, W, Hu, P., Niyato, D., Wen, Y. A survey on consesnsus mechanisms and mining strategy management in blockchain networks. *IEEE Access*. 2019.

Hamada, T., Benkrid, K., Nitadori, K., Makoto, T., 2009. A comparitive study on ASIC, FPGAs, GPUs, and general Purpose Processors in the $O(N^2)$ gravitational N-body simulation. *In 2009 NASA/ESA Conf. Adapative Hardware and Systems*. IEEE.

ComputeNorth, 2018. What is the difference between ASIC and GPU cryptocurrency mining? https://www.computenorth.com/what-is-the-difference-between-asic-and-gpu-cryptocurrency-mining/

Hcash, 2019. Why go ASIC resistant? https:// medium.com/@media_30378/why-go-asic-resistant-7fa1e40f50c4

Gorghiades, Y., Flolid, S., Vishwanath, S., 2019. HashCore: Proof-of-Work functions for general purpose processors. *arXiv. 2019*. https://arxiv.org/pdf/1902.00112.pdf

Ifdefelse, 2019. Ifdefelse/ProgPOW. Retrieved July 13, 2019, from https://github.com/ifdefelse/ProgPOW.

Ethereum, 2019. Ethereum/aleth. Retrieved July 13, 2019, from https://github.com/ethereum/aleth.

Madore, P. H, 2018. Bitmain Releases Antminer S15: How it Stacks Up Against Competitors. Retrieved July 14, 2019, from https://finance.yahoo.com/news/bitmain-releases-antminer-s15-stacks-131706332.html.

Gao, H., 2017. Basic Concepts in GPU Computing. Retrieved July 14, 2019, from https://medium.com/@smallfishbigsea/basic-concepts-in-gpu-computing-3388710e9239

Ren, L., Devadas, S., 2017. Bandwidth hard functions for ASIC resistance. *In Theory of Cryptography Conference*. Springer.

Cho, H., 2018. ASIC-resistance of multi-hash proof-of-work mechanisms for blockchain consensus protocols. *IEEE Access*, 6, 66210-66222.

Amy, M., Di Matteo, O., Gheorghiu, V., Mosca, M., Parent, A., & Schanck, J., 2016. Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3. In International Conference on Selected Areas in Cryptography (pp. 317-337). Springer, Cham.

Alshaikhli, I. F., Alahmad, M. A., & Munthir, K., 2012, November. Comparison and analysis study of SHA-3 finalists. In 2012 International Conference on Advanced Computer Science Applications and Technologies (ACSAT) (pp. 366-371). IEEE.

Armknecht, F., Karame, G. O., Mandal, A., Youssef, F., & Zenner, E., 2015. Ripple: Overview and outlook. In International Conference on Trust and Trustworthy Computing (pp. 163-180). Springer, Cham.

Mazieres, D., 2015. The stellar consensus protocol: A federated model for internet-level consensus. Stellar Development Foundation, 32.