ZelHash Specification

An ASIC/FPGA-resistant implementation of Equihash for the Zel cryptocurrency

Wilke Trei

lolMiner developer

June 26th, 2019

1 Overview

ZelHash is a proof-of-work (PoW) algorithm based on Equihash 125/4 with a modification that closes a potential compute/memory trade-off which could be applied in future specialized hardware. This document describes the basics of Equihash and offers an analysis of the potential feasibility of a hardware implementation of said algorithm on FPGAs and ASICs. In summary, the modification of ZelHash over standard Equihash 125/4 and why ZelHash is a benefit to the Zel cryptocurrency chain is described.

Note that this documentation has many outlines in common with the description of the Beam crypt currency variant of Equihash 150/5, which is a close sibling to the ZelHash PoW algorithm.

2 What is Equihash?

Despite its name, Equihash is not a hash function in the traditional sense; instead it references the solution to a mathematical problem. The problem can be formulated as follows:

Definition 1 *Let n and k be the Equihash parameters and let work and nonce be given bit streams. Then we define $m := \frac{n}{k+1}$ to be the* collision *length for n and k. Furthermore, let:*

$$B(k) := Blake2B\left(\text{concat}\left(work, nonce, l\right)\right)$$

be the output of the Blake2B hash function for $l = 0 \ldots \frac{2^{m+1}}{\lfloor \frac{512}{n} \rfloor}$.

In Equihash, one computes $s := \lfloor \frac{512}{n} \rfloor$ disjoint sections of length n bits out of each $B(l)$ and index them first in order of l and then in their local position within $B(l)$. Note

that the size of each segment is padded to a full byte length. The padding bytes will be ignored later.

Then a valid solution of the Equihash problem is a set of 2^k *indices such that:*

- *all indices are pairwise distinct,*
- *for any* $1 \leq i < k$ *the exclusive or (xor) of all elements referenced by an* 2^{i+1} *elements index block is* 0 *on the first* $i \cdot m$ *bits ,*
- *the exclusive or of all elements indexed by the solution equals 0.*
- *the indexes are sorted in a way such that for two index blocks with* 2^i *indexes each, the one with the lowest leading element leads first. E.g.* $I_{2 \cdot j} < I_{2 \cdot j+1}$ *for all* $0 \leq j \leq 2^{k-1}$.

Note that this is not a complete formal description but is sufficient to discuss the characteristics of Equihash. Note that an Equihash problem instance may or may not have a solution; in fact there are on average approximately two solutions for each discrete problem instance.

The 2^{m+1} bit strings generated in the above definition are also called *step rows*. Figure 1 shows the location of the $4 = \lfloor \frac{512}{125} \rfloor$ step rows generated out of one Blake2b output for Equihash 125/4. This includes the padding bytes to full byte multiples denoted in gray in Figure 1.

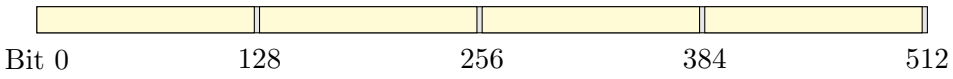

Figure 1: Positioning of Equihash 125/4 step rows in a single 512 bit Blake2b output

The algorithm behind the computation of an Equihash solution is the Wagner algorithm. The simplified process is to find pairs of bit streams matching on its lowest m bits. The XOR of each such pair will then give a step row that is shorter by m bits and is the input to the next round. Iterating this process $k - 1$ times will create step rows of length $2m$ that each have 2^{k-1} full length ancestors. If two such elements match on the remaining $2m$ bits, and its ancestors are pairwise distinct, then they will generate an Equihash solution.

The matching can be achieved by sorting the elements by their lowest bits in each of the overall k rounds. The complexity of sorting is quasi-linear in the number of elements to be sorted, which equals 2^{m+1} in the beginning. It can be shown that the expected number of pairs generated in each round is again 2^{m+1}, except for the last round where we only expect two outputs since we match elements on twice the number of bits.

It is immediately evident that memory use, as well as the number of elements to be processed, is dominated by the number of collision bits m, which is a better indicator for the algorithms complexity than n, which is the initial step row length.

3 Analysis of ASIC and FPGA advantages over GPUs for performing Equihash

Current ASIC implementations of the Equihash 200/9 algorithm (ZCash) make use of large amounts of on-chip memory offering a low latency and high memory bandwidth. For example the Bitmain Z9 Mini features 144 Mbyte of on-chip memory, which is close to the theoretical minimum memory required to perform the calculation on a CPU. Given the performance of the ASIC the used bandwidth exceeds 5 Tbytes per second.

In contrast, an efficient GPU implementation will typically utilize a larger memory footprint because the off-chip memory controllers benefit from read and write access that is at least 32bit, (best performance requires >128 bit). Off-chip memory has a bandwidth ranging of about 256 Gbyte/s on mid-range GPUs to 3 times the value on high-end GPUs equipped with HBM2 memory.

Besides the advantages of large amounts of on-chip memory and efficiently-packed circuit patterns, an ASIC and/or FPGA can trade time spent on parallelizeable compute operations by chip space and power consumption by assigning more circuits to the task. For Equihash the fraction offering most potential for this trade is the Blake2B algorithm.

Massively increasing the performance of the Blake2B algorithm may be beneficial to trade compute power against bandwidth like follows: In the first rounds of the Equihash matching phases, fewer bits than necessary for performing the full algorithm can be stored and loaded. As soon as the abandoned bits are required for continuing with the algorithm, the chip can recover them by computing the hashes again from the indices carried to this round.

This approach is particularly efficient in the early rounds when the bandwidth required to transfer the indices is low compared with the transfer of the original workload bits. Also in these early rounds, fewer different Blake2b passes have to be used to recover the discrete bits. For GPUs this approach is not an option, because adding Blake2b computations increases duration instead of space of the algorithm. These operations also consume too much time to be hidden when performing memory operations and they would increase power consumption as well.

4 Applicability of Equihash 125/4

In the previous section we defined three potential benefits of ASICs compared to GPUs: 1) large on-chip memory capacities, 2) better packing and coalescing of off-chip memory operations and, 3) time-space algorithm trade-off.

Regarding the first issue we have strong confidence that a specialized chip for Equihash 125/4 will not take this exact approach due to its increased memory footprint. When completely computed, the output of the first round is $(26 + 125) \cdot 2^{26}$ bits and thus approx 1.2 Gbytes. We claim that with growing index tables of matched elements this is a sharp lower bound. This even applies when using the trick described in the time

memory trade-off part of the previous section, because the indices written in the first 4 rounds compressed to 36 bits per matched element plus the required remaining bits for performing the final matching round requires at least this amount of space.

In practice we can estimate that the real memory consumption of an efficient implementation is at least twice the size, similar to the ZCash parameters.

Therefore a single chip ASIC holding enough memory to perform all operations at rates of TBytes per second bandwidth would be 10- to 20-times the size of the chip to perform ZCashs Equihash 200/9 algorithm on the same manufacturing process. This would increase the cost for producing the chip dramatically because on the same production processes a higher yield can be expected and also larger chips are more costly.

This approach will likely be feasible in the future with advanced production methods and new technologies regarding fast on-chip memory or 3D stacking chips to maximize bandwidth. Nevertheless the mentioned amount of memory is high enough that the time-to-market for such new inventions will be longer then our planned PoW review and adjustment period.

As of Summer 2019, the PoW mining ASICs with the largest on-chip memory are the Obelisk GRN1 and the G32-500, which are both designed for mining Cuckatoo-31+, and are supposed to be available in Q4 of 2019. Although the specs are not fully public it is known these devices offer 512 MByte of on-chip scratch memory at a 16nm structure and belong to the largest chips that can be created with this structure process. We conclude that even with the smaller 7nm manufacturing process the approximately 2GByte lower bound of an efficient Equihash 125/4 implementation are currently out of reach.

The second benefit is the architectural nature with ASICs/FPGAs being able to perform memory operations that use bit length that are not a multiple of 32 more efficiently. Also adding more circuits in the implementation to ensure a better memory coalescing when using external memory chips is exclusive to these chips, while a GPU can improve its memory access patterns only by using software solutions.

We therefore believe this benefit can not be mitigated directly by an algorithm change unless changing the parameters in a way that most memory operations are forced to be in an ideal range for GPUs. This is out of scope for Equihash with nowadays capacities.

Anyways we claim that the total effect of this benefit is limited by the capacities of the external memory controller. For the already well tuned Equihash 144/5 parameters it is known that a single run requires approximately 5 Gbytes of memory bandwidth in total. Each run will produce 2 solutions on average. Modern implementations can run up to 60 solutions per second on an unmodified Nvidia GTX 1080. Therefore on this cards about 150 Gbytes/s of the total available 352 Gbyte/s are effectively utilized. So a specialized chip equipped with the same memory but more efficient memory access patterns may have a gain limited to a factor of about 2.5 if the algorithm behind is forced to write the same data.

This enforcement aligns with the potential space-to-memory and bandwidth trade-off discussed before. ZelHash will modify the default Equihash 125/4 algorithm in a way

that was first described by the Beam cryptocurrency project, which is outlined in the following section. The purpose is to make the described time memory trade-off too costly in terms of chip space and energy efficiency.

Note that devices which are task-programmable (i.e. GPUs) but modified to mine cryptocurrencies by offering additional optimizations for memory access still might hash 2-3x faster than a GPU equipped with the same memory, and that at a potentially lower energy consumption.

5 Modification of Equihash 125/4

The Equihash 125/4 algorithm has a native blocking rate of 4 in its Blake2b phase. This is because if the original algorithm is modified in a way that requires Blake2b algorithm in later compute rounds, each hash string requires 4-times the computational resources of the initial calculation, due to that the first round 4 hash strings of 150 bit length are generated in one computation.

This also applies to the verification of an Equihash 125/4 solution, but is no issue for the verification because the total number of Blake2b runs to verify one solution is 16, while the average number of runs for generating one solution is $\frac{2^{26}}{4} = 2^{24}$.

We propose to increase the blocking factor to 64 by the following strategy:

Algorithm 2 *Let $B(l)$ be the 512 bit string corresponding to the Blake2b hash for input index l and let $B(l)_j$ be the j-th 32 bit component interpreted as 32 bit integer. Then let $B'(l)$ be the 512 bit string computed like follows:* $c \leftarrow 16 \cdot \lfloor \frac{l}{16} \rfloor$
$B'(l) \leftarrow 0$
while $c \leq l$ ***do***
    $B'(l)_j \leftarrow B'(l)_j + B(c)_j$ *for all* $0 \leq j < 16$
    $c \leftarrow c + 1$
end while

Thus in order to compute the modified hash string B' it may be required to know all 15 other hashes in the same group of 16 hash elements. On a GPU this can be cheaply achieved by using the local memory on the device in the initial computation phase. Also on modern GPUs neighbored threads with indices varying only on the lowest four bits run in lock steps, so the sharing over the local memory does not introduce new synchronization barriers.

Therefore for performing the algorithm in the intended manner, the slow down by the extra computation of the sums will be negligible. On the other hand when it is intended to run Blake2b later to recover bits from known indices, this is up to 16 times more costly than before, up to 64 times more costly than in the initial round with an average extra cost factor of 24.

By this approach we aim towards forcing the algorithm to follow the same algorithm implementation as done on GPUs or else to use more chip-space and drastically in-

crease power consumption, since performing the Blake2b algorithm is the most power-consuming component of Equihash.

Note that the verification of solutions becomes more costly by an average factor of eight. But since only a few solutions need to be verified, and due to the still-high asymmetry of generation effort compared to the verification effort, the drawback is acceptable. Overall with this modification the verification of an Equihash 125/4 solution can be done at a lower average cost than the verification of an Equihash 200/9 solution while the worst case costs are equal.