Weekly Report - Explaining and Testing Mining Algorithms

MSc Project
Runchao Han

December 1, 2017

1 Progress of the Last Week

1. Run all chosen mining algorithms and got results
2. Understood chosen mining algorithms deeper
3. Learned about the mining software and the mining pool
4. Learned about CUDA, which is the de facto standard of implementing mining algorithms

2 Explaining Chosen Mining Algorithms

2.1 Ethash

Ethash is the PoW algorithm adopted by Ethereum, the latest version of which is also called Dagger-Hashimoto[1]. Dagger-Hashimoto is build upon two previous projects:

- Dagger[2] an algorithm by Vitalik Buterin which uses directed acyclic graphs to simultaneously achieve memory-hard computation but memory-easy validation.
- Hashimoto[3] an algorithm by Thaddeus Dryja which intends to achieve ASIC resistance by being IO-bound, ie. making memory reads the limiting factor in the mining process.

Dagger-Hashimoto is designed for two purposes:

- ASIC-resistance: the benefit from creating specialized hardware for the algorithm should be as small as possible, ideally to the point that even in an economy where ASICs have been developed the speedup is sufficiently small that it is still marginally profitable for users on ordinary computers to mine with spare CPU power.
- Light client verifiability: a block should be relatively efficiently verifiable by a light client.
- Full chain storage: mining should require storage of the complete blockchain state (due to the irregular structure of the Ethereum state trie, we anticipate that some pruning will be possible, particularly of some often-used contracts, but we want to minimize this).

[1]https://github.com/ethereum/wiki/blob/master/Dagger-Hashimoto.md
[2]http://www.hashcash.org/papers/dagger.html
[3]https://pdfs.semanticscholar.org/3b23/7cc60c1b9650e260318d33bec471b8202d5e.pdf

The general route of Ethash is as follows:

1. Getting the seed: There exists a seed which can be computed for each block by scanning through the block headers up until that point.
2. 16MB cache generation: From the seed, one can compute a 16 MB pseudorandom cache. Light clients store the cache.
3. 1GB dataset generation: From the cache, we can generate a 1 GB dataset, with the property that each item in the dataset depends on only a small number of items from the cache. Full clients and miners store the dataset. The dataset grows linearly with time.
4. Mining: Mining involves grabbing random slices of the dataset and hashing them together. Verification can be done with low memory by using the cache to regenerate the specific pieces of the dataset that you need, so you only need to store the cache.

The seed only depends on the number of the block on the blockchain. The 16MB cache only depends on the seed, and the 1GB dataset only depends on the 16MB cache. That is, with a seed the cache and the dataset can be generated. That is, every block on the blockchain has a corresponding seed calculated, which can generate the cache and the whole dataset.

2.1.1 Getting the seed

The process of generating the seed is shown below:

```
def get_seedhash(block):
    s = '\x00' * 32
    for i in range(block.number // EPOCH_LENGTH):
        s = serialize_hash(sha3_256(s))
    return s
```

2.1.2 Size definitions of the cache and the dataset

The sizes of cache and dataset are not fixed but stable, the definitions of which are shown below:

```
def get_cache_size(block_number):
    sz = CACHE_BYTES_INIT
              + CACHE_BYTES_GROWTH * (block_number // EPOCH_LENGTH)
    sz -= HASH_BYTES
    while not isprime(sz / HASH_BYTES):
        sz -= 2 * HASH_BYTES
    return sz

def get_full_size(block_number):
    sz = DATASET_BYTES_INIT
              + DATASET_BYTES_GROWTH * (block_number // EPOCH_LENGTH)
    sz -= MIX_BYTES
    while not isprime(sz / MIX_BYTES):
```

```
        sz -= 2 * MIX_BYTES
    return sz
```

2.1.3 The cache generation

The cache generation relies on the cache size and the seed, shown below:

```
def mkcache(cache_size, seed):
    n = cache_size // HASH_BYTES

    # Sequentially produce the initial dataset
    o = [sha3_512(seed)]
    for i in range(1, n):
        o.append(sha3_512(o[-1]))

    # Use a low-round version of randmemohash
    for _ in range(CACHE_ROUNDS):
        for i in range(n):
            v = o[i][0] % n
            o[i] = sha3_512(map(xor, o[(i-1+n) % n], o[v]))

    return o
```

A lightweight node which can perform verifications of transactions but cannot mine only needs to generate the cache, but not the whole dataset.

2.1.4 The dataset generation

The dataset consists of multiple dataset items generated by the function *calc_dataset_item*(), which relies on the data aggregation function *fnv*().

```
FNV_PRIME = 0x01000193

def fnv(v1, v2):
    return ((v1 * FNV_PRIME) ^ v2) % 2**32

def calc_dataset_item(cache, i):
    n = len(cache)
    r = HASH_BYTES // WORD_BYTES
    # initialize the mix
    mix = copy.copy(cache[i % n])
    mix[0] ^= i
    mix = sha3_512(mix)
    # fnv it with a lot of random cache nodes based on i
    for j in range(DATASET_PARENTS):
        cache_index = fnv(i ^ j, mix[j % r])
        mix = map(fnv, mix, cache[cache_index % n])
```

```
    return sha3_512(mix)

def calc_dataset(full_size, cache):
    return [calc_dataset_item(cache, i) for i in range(full_size // HASH_BYTES)]
```

To mine on the network, a node should generate or download the whole dataset.

2.1.5 Mining

Mining is to iteratively select slices in the dataset to find a slice which is smaller than the difficulty, where the slice selection is a function $hashimoto_full()$ which frequently accesses the dataset. This makes the mining memory-bound.

```
def hashimoto_full(full_size, dataset, header, nonce):
    return hashimoto(header, nonce, full_size, lambda x: dataset[x])

def mine(full_size, dataset, header, difficulty):
    target = zpad(encode_int(2**256 // difficulty), 64)[::-1]
    from random import randint
    nonce = randint(0, 2**64)
    while hashimoto_full(full_size, dataset, header, nonce) > target:
        nonce = (nonce + 1) % 2**64
    return nonce
```

2.1.6 Verifying

Verifying a nonce only needs the cache.

```
def hashimoto_light(full_size, cache, header, nonce):
    return hashimoto(header, nonce
                , full_size, lambda x: calc_dataset_item(cache, x))
```

2.2 CryptoNight

CryptoNight[4] is the adopted mining algorithm of Monero, whose transactions are unlinkable and untracable by ring signature[4]. Its process is divided into three steps:

1. Scratchpad initialisation
2. Memory-hard loop
3. Result calculation

2.2.1 The scratchpad initialisation

To mine Monero, a 2MB scratchpad is initialised at first. $Keccak$[1] is a hash function (will be adoped as SHA-3), and AES stands for the Advanced Encryption Standard[2]. Given an input, the scratchpad is generated as shown in Fig. 1.

[4] https://da-data.blogspot.co.uk/2014/08/minting-money-with-monero-and-cpu.html

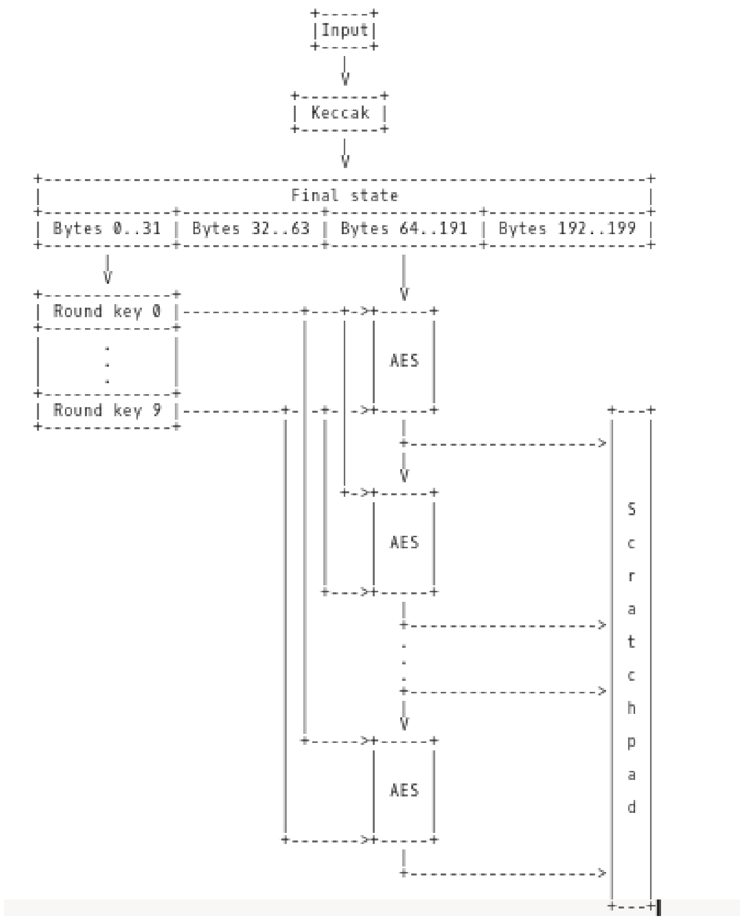

Figure 1: The Scratchpad Initialisation of the Monero Mining Process

2.2.2 Memory-hard Loop

The memory-hard loop (the main loop) is an iterative process with frequent memory access to make the mining process memory-bound, which is shown in Fig.2.

2.2.3 Result Calculation

After the memory-hard loop, the result is calculated as shown in Fig. 3.

2.3 Scrypt

Scrypt[3] is defined as Fig. 4, where MF is a pluggable sequential memory-hard function.

PBKDF2[3] is the abbreviation of Password-Based Key Deviation Function 2, which takes a string and a Pseudo Random Function (PRF) as inputs and outputs a key (no need to be sequential or memory-hard).

The chosen MF is the SMix function family proposed by the Scrypt paper. An implementation of SMix functions is called $BlockMix$, the process of which is shown in Fig. 5.

```
+-------------------------------------------------------------+
|                         Final state                         |
+-------------+--------------+---------------+----------------+
| Bytes 0..31 | Bytes 32..63 | Bytes 64..191 | Bytes 192..199 |
+-------------+--------------+---------------+----------------+
       |             |
       |   +-----+   |
       +-->| XOR |<--+
           +-----+
            |   |
       +----+   +----+
       |             |
       V             V
     +---+         +---+
     | a |         | b |
     +---+         +---+
       |             |
--------------------- REPEAT 524288 TIMES ---------------------
       |             |                            address +---+
       +-------------|----------------------------------->|   |
       |   +-----+   |                               read |   |
       +-->| AES |<--|------------------------------------|   |
       |   +-----+   V                                    |   |
       |      |   +-----+                                 | S |
       |      +-->| XOR |                                 |   |
       |      |   +-----+                           write | c |
       |      |      |    +------------------------------>|   |
       |      |      +----+                       address | r |
       |      +------------------------------------------>|   |
       |      |  +-----------+                       read | a |
       |      +->| 8byte_mul |<--+------------------------|   |
       |      |  +-----------+   |                        | t |
       |      |        |         |                        |   |
       |      |        V         |                        | c |
       |      |  +-----------+   |                        |   |
       +------|->| 8byte_add |   |                        | h |
              |  +-----------+   |                        |   |
              |        |         |                  write | p |
              |        +---------|----------------------->|   |
              |        |         |                        | a |
              |        V         |                        |   |
              |     +-----+      |                        | d |
              |     | XOR |<-----+                        |   |
              |     +-----+                               |   |
              +------+ |                                  |   |
       +-------------|-+                                  |   |
       |             |                                    +---+
------------------------- END REPEAT --------------------------
```

Figure 2: The Memory-hard Loop of Monero Mining Process

3 Testing and Performance Analysis

3.1 Ethash

The chosen Ethash implementation is `Ethminer`[5], which is the official mining software for Ethereum with CUDA and OpenCL support.

The benchmark result of the Ethash CUDA implementation is shown in Fig. 6, while the OpenCL implementation is shown in Fig. 7. The benchmark process is to generate the whole DAG first, then test the hashrate for five times, finally get the results and make statistics.

According to the testing, the CUDA version performs better than the OpenCL version. This is because OpenCL targets at implementing a general-purpose parallel computing platform, which inevitably causes poorer performance.

According to the statistics about the cryptocurrency market, Ethereum is the second most popular cryptocurrency combined with state-of-the-art features Bitcoin does not have, like the

[5]Ethminer: https://github.com/ethereum-mining/ethminer

memory-bound Ethash mining, the Smart Contract and the new Zero Knowledge Proof privacy protection. It is reasonable to put the Ethash in a high priority.

3.2 Scrypt

The `ccminer` implementation of Scrypt CUDA is chosen, where `ccminer` is a miner supporting a wide range of mining algorithms[6]. It is noted that the Scrypt performance is much poorer than Ethash. This is because Scrypt is unparallelisable but Ethash is parallelisable, while both of them are memory-bound.

3.3 CryptoNight

`ccminer` has the CryptoNight CUDA implementation, too. However, unfortunately when I tried to run it, my computer crashed and rebooted. The next step is to limit its GPU usage and run again.

4 Miscellaneous

- Deleted the Cuckoo algorithm because it is not widely used, but added Cryptonight of Monero

5 Next Week's Plan

- Benchmark chosen algorithms with specified parameters(in hardware simulators?)
- Try to understand source code of chosen algorithms
- Learn CUDA programming and other related tools

References

[1] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak specifications. *Submission to NIST (Round 2)*, 2009.

[2] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[3] Colin Percival and Simon Josefsson. The scrypt password-based key derivation function. Technical report, 2016.

[4] Ronald L Rivest, Adi Shamir, and Yael Tauman. How to leak a secret: Theory and applications of ring signatures. *Essays in memory of Shimon Even*, 3895:164–186, 2006.

[6] ccminer: https://github.com/cbuchner1/ccminer

```
+-------------------------------------------------------------+
|                         Final state                         |
+-------------+--------------+---------------+----------------+
| Bytes 0..31 | Bytes 32..63 | Bytes 64..191 | Bytes 192..199 |
+-------------+--------------+---------------+----------------+
      |                |             |                |
      |       +--------+             |                |
      |       V        |             |                |
      |+-------------+ |             |                |
      || Round key 0 |-|---+---+     |                |
      |+-------------+ |   |   |     |                |
      ||      .      | |   |   |     |                |
      ||      .      | |   |   |     |                |
      ||      .      | |   |   |     |                |
      |+-------------+ |   |   |     |                |
+---+ || Round key 9 |-|-+-|-+ |     V                |
|   | |+-------------+ | | | | | |  +-----+             |
|   |-|----------------|-|-|-|-|->| XOR |             |
|   | |                | | | | | |  +-----+             |
| S | |                | | | | |     |                |
|   | |                | | | | |     V                |
| c | |                | | | | +->+-----+             |
|   | |                | | | |    |     |             |
| r | |                | | | |    |     |             |
|   | |                | | | |    | AES |             |
| a | |                | | | |    |     |             |
|   | |                | | | |    |     |             |
| t | |                | | | +--->+-----+             |
|   | |                | | |         |                |
| c | |                | | |         V                |
|   | |                | | |      +-----+             |
| h |-|----------------|-|-|----->| XOR |             |
|   | |                | | |      +-----+             |
| p | |                | | |         |                |
|   | |                | | |         .                |
| a | |                | | |         .                |
|   | |                | | |         .                |
| d | |                | | |         |                |
|   | |                | | |         V                |
|   | |                | | |      +-----+             |
|   |-|----------------|-|-|----->| XOR |             |
|   | |                | | |      +-----+             |
+---+ |                | | |         |                |
      |                | | |         V                |
      |                | | +----->+-----+             |
      |                | |      |     |             |
      |                | |      |     |             |
      |                | |      | AES |             |
      |                | |      |     |             |
      |                | |      |     |             |
      |                | +------->+-----+             |
      |                |             |                |
      V                V             V                V
+-------------+--------------+---------------+----------------+
| Bytes 0..31 | Bytes 32..63 | Bytes 64..191 | Bytes 192..199 |
+-------------+--------------+---------------+----------------+
|                       Modified state                        |
+-------------------------------------------------------------+
                               |
                               V
                         +----------+
                         | Keccak-f |
                         +----------+
                          |    |
              +-----------+    |
              |                |
              V                V
       +-------------+  +-------------+
       | Select hash |->| Chosen hash |
       +-------------+  +-------------+
                               |
                               V
                       +--------------+
                       | Final result |
                       +--------------+
```

Figure 3: The Result Calculation of Monero Mining Process

Algorithm $\mathbf{MFcrypt}_{H,MF}(P, S, N, p, dkLen)$

Parameters:

	PRF	A pseudorandom function.
	$hLen$	Length of output produced by PRF, in octets.
	MF	A sequential memory-hard function from $\mathbb{Z}_{256}^{MFLen} \times \mathbb{N}$ to $\mathbb{Z}_{256}^{MFLen}$.
	$MFLen$	Length of block mixed by MF, in octets.

Intput:

	P	Passphrase, an octet string.
	S	Salt, an octet string.
	N	CPU/memory cost parameter.
	p	Parallelization parameter; a positive integer satisfying $p \leq (2^{32} - 1)hLen/MFLen$.
	$dkLen$	Intended output length in octets of the derived key; a positive integer satisfying $dkLen \leq (2^{32} - 1)hLen$.

Output:

	DK	Derived key, of length $dkLen$ octets.

Steps:

$(B_0 \ldots B_{p-1}) \leftarrow \mathrm{PBKDF2}_{PRF}(P, S, 1, p \cdot MFLen)$
for $i = 0$ to $p - 1$ **do**
  $B_i \leftarrow MF(B_i, N)$
end for
$DK \leftarrow \mathrm{PBKDF2}_{PRF}(P, B_0 \parallel B_1 \parallel \ldots \parallel B_{p-1}, 1, dkLen)$

Figure 4: The Scrypt Function

Algorithm $\mathbf{BlockMix}_{H,r}(B)$

Parameters:

	H	A hash function.
	r	Block size parameter

Input:

	$B_0 \ldots B_{2r-1}$	Input vector of $2r$ k-bit blocks

Output:

	$B'_0 \ldots B'_{2r-1}$	Output vector of $2r$ k-bit blocks.

Steps:

$X \leftarrow B_{2r-1}$
for $i = 0$ to $2r - 1$ **do**
  $X \leftarrow H(X \oplus B_i)$
  $Y_i \leftarrow X$
end for
$B' \leftarrow (Y_0, Y_2, \ldots Y_{2r-2}, Y_1, Y_3, \ldots Y_{2r-1})$

Figure 5: The process of BlockMix, which is a type of SMix

```
 cu  20:30:10|ethminer  Using grid size 8192 , block size 128
Benchmarking on platform: CUDA
Preparing DAG for block #0
Warming up...
  i  20:30:10|CUDA0     set work; seed: #00000000, target:  #000000000000
  i  20:30:10|CUDA0     Initialising miner...
 cu  20:30:10|CUDA0     Using device: GeForce GTX 960M  (Compute 5.0)
 cu  20:30:10|CUDA0     Generating DAG for GPU # 0
CUDA#0: 0%
CUDA#0: 6%
CUDA#0: 12%
CUDA#0: 19%
CUDA#0: 25%
CUDA#0: 31%
CUDA#0: 38%
CUDA#0: 44%
CUDA#0: 50%
CUDA#0: 56%
CUDA#0: 62%
CUDA#0: 69%
CUDA#0: 75%
CUDA#0: 81%
CUDA#0: 88%
CUDA#0: 94%
Trial 1... 7975558
Trial 2... 8394904
Trial 3... 8815090
Trial 4... 8815090
Trial 5... 8918246
min/mean/max: 7975558/8583777/8918246 H/s
inner mean: 8709413 H/s
```

Figure 6: The benchmark result of the Ethash CUDA implementation

```
 i  20:28:28|ethminer  Found suitable OpenCL device [ GeForce GTX 960M ] with 4240965632  bytes of GPU memory
Benchmarking on platform: CL
Preparing DAG for block #0
 cl  20:28:28|cl-0      No work. Pause for 3 s.
Warming up...
 cl  20:28:31|cl-0      New work: header #50c856ae… target 0000000000000002000000000000000000000000000000000000000000000000
 cl  20:28:31|cl-0      New seed #00000000…
 cl  20:28:32|cl-0      Platform: NVIDIA CUDA
 cl  20:28:32|cl-0      Device:   GeForce GTX 960M  / OpenCL 1.2 CUDA
 cl  20:28:32|cl-0      Build info:

 cl  20:28:32|cl-0      Creating light cache buffer, size 16776896
 cl  20:28:32|cl-0      Creating DAG buffer, size 1073739904
 cl  20:28:32|cl-0      Loading kernels
 cl  20:28:32|cl-0      Writing light cache buffer
 cl  20:28:32|cl-0      Creating buffer for header.
 cl  20:28:32|cl-0      Creating mining buffer
 cl  20:28:32|cl-0      Generating DAG
 cl  20:28:33|cl-0      DAG 0 %
 cl  20:28:33|cl-0      DAG 6 %
 cl  20:28:34|cl-0      DAG 12 %
 cl  20:28:35|cl-0      DAG 18 %
 cl  20:28:36|cl-0      DAG 25 %
 cl  20:28:37|cl-0      DAG 31 %
 cl  20:28:38|cl-0      DAG 37 %
 cl  20:28:39|cl-0      DAG 43 %
 cl  20:28:40|cl-0      DAG 50 %
 cl  20:28:41|cl-0      DAG 56 %
 cl  20:28:41|cl-0      DAG 62 %
 cl  20:28:42|cl-0      DAG 68 %
Trial 1...  cl  20:28:43|cl-0      DAG 75 %
 cl  20:28:44|cl-0      DAG 81 %
 cl  20:28:45|cl-0      DAG 87 %
0
Trial 2...  cl  20:28:46|cl-0      DAG 93 %
 cl  20:28:46|cl-0      Switch time 18002 ms / 15002299 us
8394204
Trial 3... 8394204
Trial 4... 8276697
Trial 5... 8182165
  x  20:28:58|cl-0      OpenCL Error: clEnqueueNDRangeKernel -52
min/mean/max: 0/6649454/8394204 H/s
inner mean: 8284355 H/s
```

Figure 7: The benchmark result of the Ethash OpenCL implementation

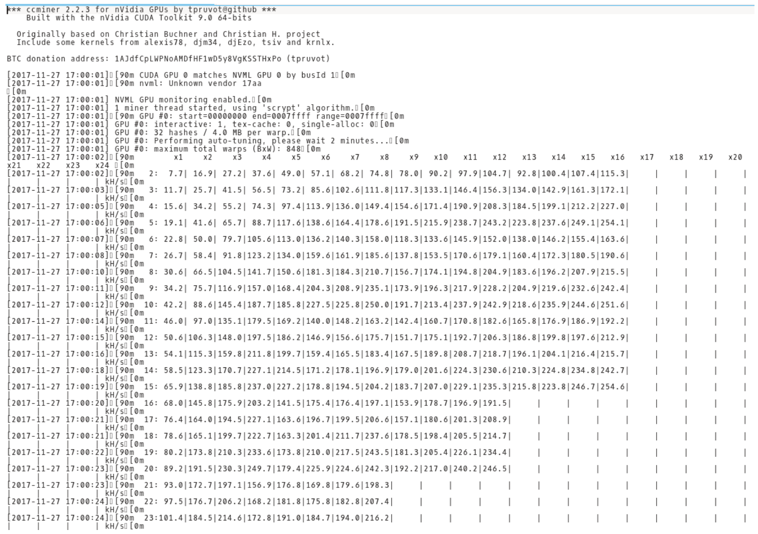

Figure 8: The benchmark result of the Scrypt CUDA implementation 1

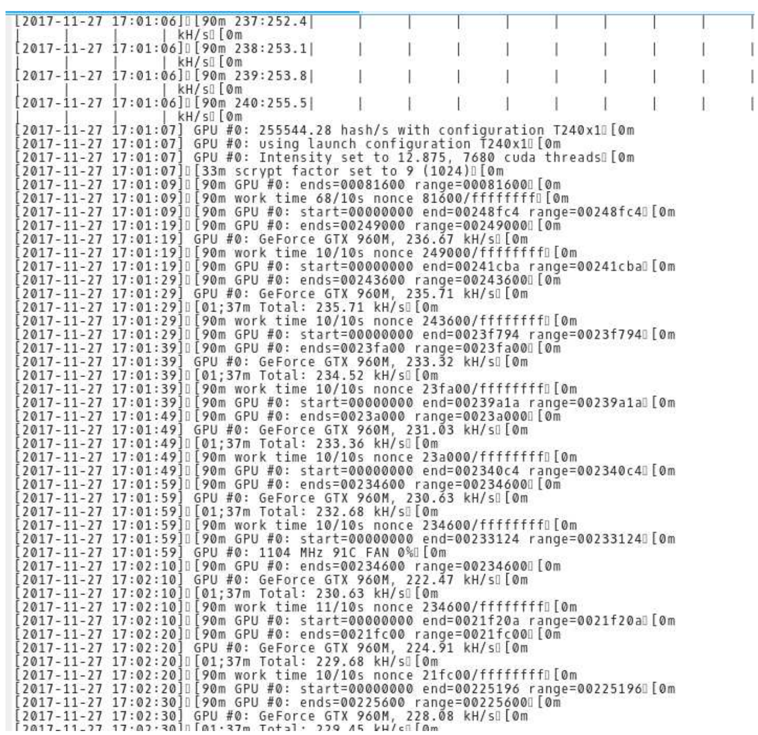

Figure 9: The benchmark result of the Scrypt CUDA implementation 2