TERAHASH mining algorithm

progr76@gmail.com

Version 0.3
September 2018

Introduction

The purpose of this algorithm is to equalize the people who are mining on a CPU and the people who are mining on a GPU. To achieve this, we suggest to use memory, but unlike other similar algorithms (such as Ethash) in our algorithm, memory does not slow down the GPU, but accelerates a CPU. This can be done by using not an integer nonce when selecting a hash, but a certain value that is time - consuming to calculate-for example, calculated by the sha3 algorithm and allowing this value to be used for enumeration in a wide range of block hash calculations. Thus, it is more profitable to store these values in memory for selection than to recalculate them. This benefit should be maintained even if the speed of computing resources will increase by 1000 times.

Algorithm

The input is a 32-byte hash of the current block CurrentDataHash, you need to find a Nonce (integer), so that the result is a suitable hash of the block with the maximum value of initial zeros.

Restriction:

1. The search should be optimized for memory usage - for protection from GPU mining
2. The check should be performed with a minimum amount of memory and be fast-about the speed of sha3 calculation

Procedure of settlements:

1. Calculated HashNonce= sha3(PrevHashN , Nonce)
2. The Resulting Hash = SimpleMesh(HashTemp,CurrentDataHash)

PrevHashN is a 32-byte hash of some previous block, different from the current block on NDelta (the maximum depth is limited to a certain number, for example 1000 blocks)
Nonce-a number to iterate through values from 0 to max of an integer
SimpleMesh () is a function of the rapid mixing (non-cryptographic). It must satisfy the conditions:

1. The conservation of entropy
2. High speed-1000 times faster than the function calculation speed in step 1 (sha3 in this example).
3. It should be good to mix the data - to prevent a quick search, i.e. guarantee busting of HashNonce

After searching for the most satisfying hash, the blockchain is written:
CurrentDataHash, Nonce, NDelta - which quickly restores the block hash

Small calculations:

- GPU GTX 1060 for 1 second will create and compare the order of 0.5 billion hashes.
- Computer: using the memory DDR4-2133 will have to have speed channel is 17Гб/s and therefore will be able to compare also the order of 0.5 billion hashes

Miner must prove that the base hash has not changed. He needs to find a nonce two:
One that fits the current hash of the block
Different to the previous block hash

The main calculation formula:
Hash = XOR(DataHash, HNonce1)
Hash2 = XOR(PrevHash,HNonce2)
Power=min(Power(Hash),Power(Hash2))

JS code:

```
function GetHash(PrevHash,BlockHash,Miner,BlockNum,
Nonce0,Nonce1,Nonce2,DeltaNum1,DeltaNum2)
{
    if(DeltaNum1>DELTA_LONG_MINING || DeltaNum2>DELTA_LONG_MINING)
        return undefined;

    //calculate the hashes, which will look similar HashNonce
```

```
    var HashBase=sha3(PrevHash);
    var HashCurrent=GetHashFromNum2(BlockHash,Miner,Nonce0);

    //Hash-Nonce
    var HashNonce1=GetHashFromNum3(BlockNum-DeltaNum1,Miner,Nonce1);
    var HashNonce2=GetHashFromNum3(BlockNum-DeltaNum2,Miner,Nonce2);

    //XOR
    var Hash1=XORArr(HashBase,HashNonce1);
    var Hash2=XORArr(HashCurrent,HashNonce2);

    //choose the least POW
    var Ret={Hash:Hash2};
    if(CompareArr(Ret.Hash1,Ret.Hash2)>0)
    {
        Ret.PowHash=Hash1;
    }
    else
    {
        Ret.PowHash=Hash2;
    }

    Ret.Hash=sha3arr2(Hash1,Hash2);

    return Ret;
}
```