Ethereum Academy

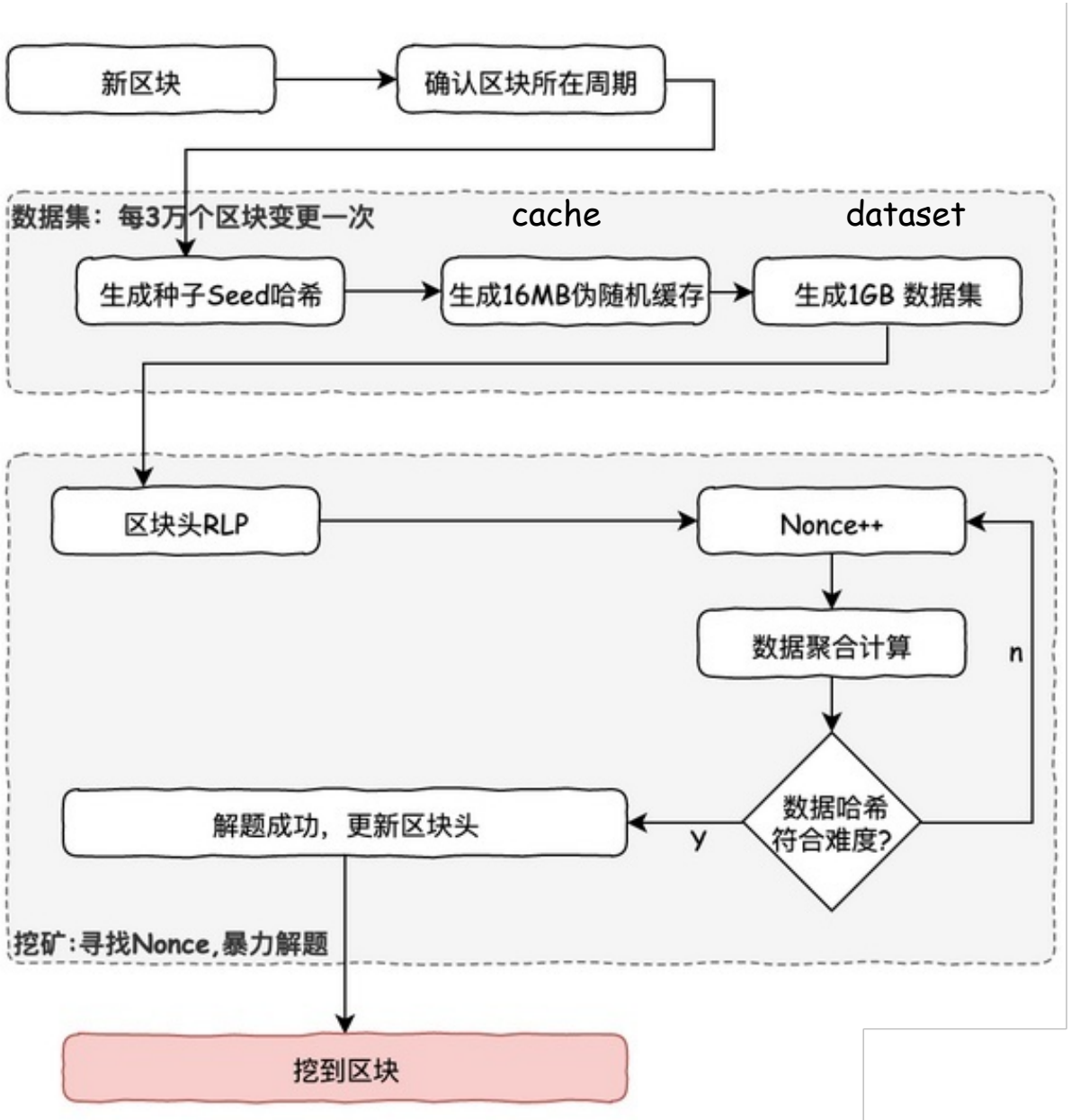
2021.11.15

Ethash

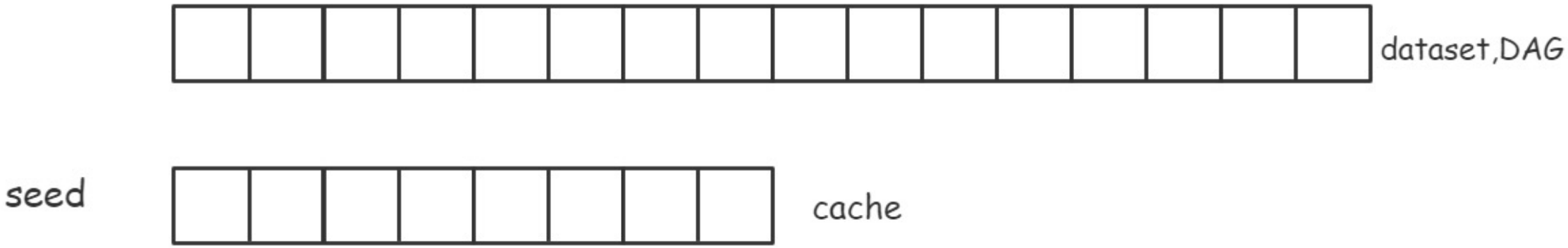
Ethash设计目标

- 抗ASIC性：为算法创建专用硬件的优势应尽可能小，让普通计算机用户也能使用CPU进行开采。
  - 通过内存限制来抵制（ASIC使用矿机内存昂贵）
  - 大量随机读取内存数据时计算速度就不仅仅受限于计算单元，更受限于内存的读出速度。
- 轻客户端可验证性: 一个区块应能被轻客户端快速有效校验。
- 矿工应该要求存储完整的区块链状态。

挖矿算法Ethash

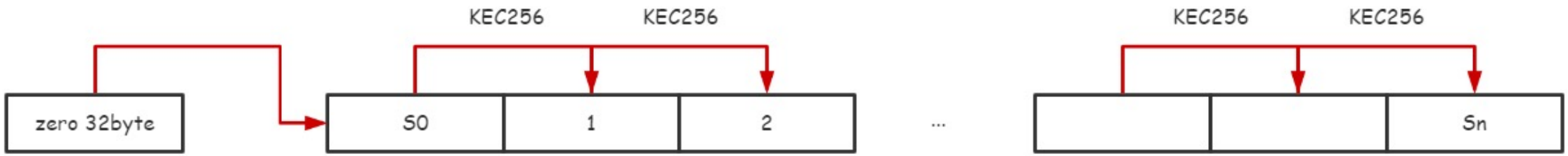

新区块
确认区块所在周期
数据集：每3万个区块变更一次
cache
dataset
生成种子Seed哈希
生成16MB伪随机缓存
生成1GB 数据集
区块头RLP
Nonce++
数据聚合计算
n
数据哈希
符合难度?
y
解题成功，更新区块头
挖矿:寻找Nonce,暴力解题
挖到区块

Cache and dataset



- 轻节点仅维护cache
- 全节点两个都维护

生成seed



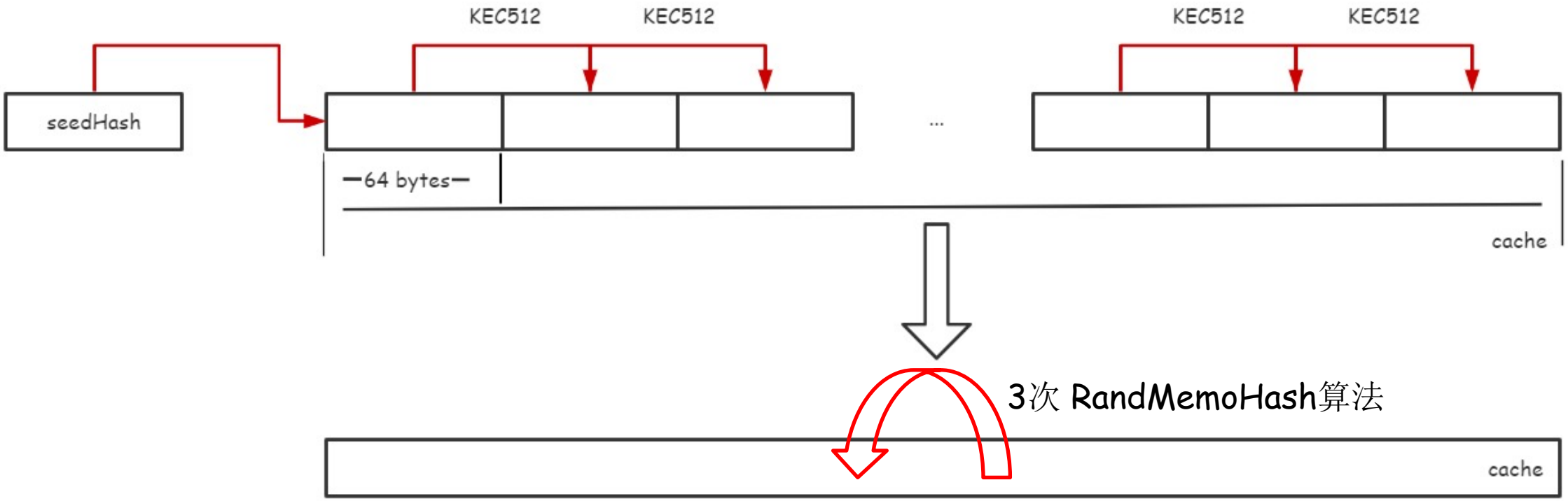
- seed实际是一个哈希值，每个窗口周期（30000个区块）更新一次，它是经过多次叠加Keccak256计算得到的
- 第一个窗口周期内的种子哈希值是一个空的32字节数组，而后续每个周期中的种子哈希值，则对上一个周期的种子哈希值再次进行Keccak256哈希得到

生成seed


```
121  func seedHash(block uint64) []byte {
122          seed := make([]byte, 32)
123          if block < epochLength {
124                  return seed
125          }
126          keccak256 := makeHasher(sha3.NewLegacyKeccak256())
127          for i := 0; i < int(block/epochLength); i++ {
128                  keccak256(seed, seed)
129          }
130          return seed
131  }
```

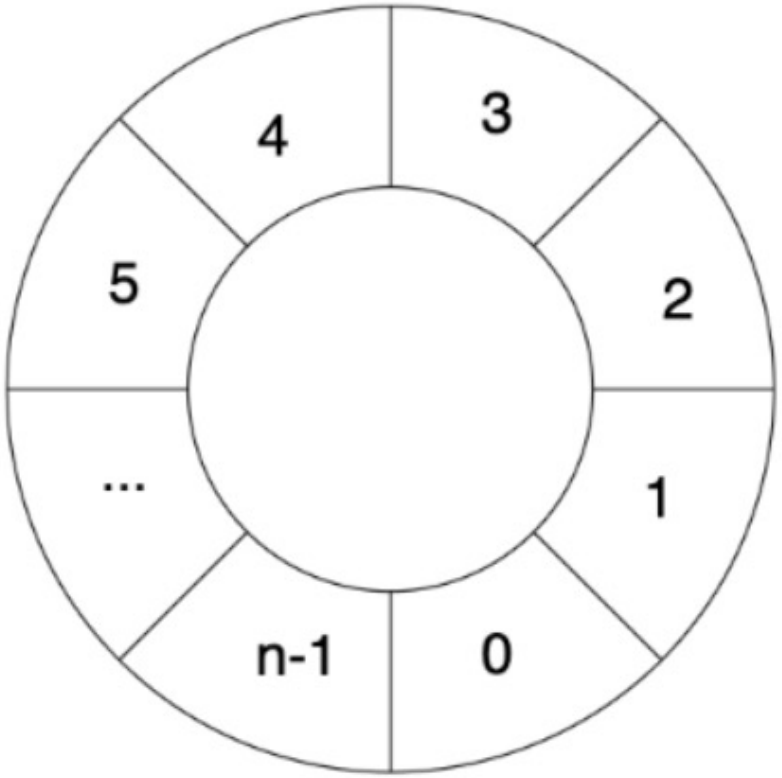
go-ethereum/algorithm.go at master · ethereum/go-ethereum · GitHub 121-131

生成cache



- 先将种子哈希值的Keccak512结果作为初始化值写入第一行中
- 随后，每行的数据用上行数据的Keccak512哈希值填充
- 最后，执行了3次 RandMemoHash算法

生成cache



- RandMemoHash 算法可以理解为将若干行进行首尾连接的环链，其中n为行数
- 每次RandMemoHash 计算是依次对每行进行重新填充。先求第 i 行的前后两行值异或运算结果，再对结果进行Keccak512哈希后填充到第i行中

生成cache


```
51  // cacheSize returns the size of the ethash verification cache that belongs to a certain
52  // block number.
53  func cacheSize(block uint64) uint64 {
54          epoch := int(block / epochLength)
55          if epoch < maxEpoch {
56                  return cacheSizes[epoch]
57          }
58          return calcCacheSize(epoch)
59  }
60
61  // calcCacheSize calculates the cache size for epoch. The cache size grows linearly,
62  // however, we always take the highest prime below the linearly growing threshold in order
63  // to reduce the risk of accidental regularities leading to cyclic behavior.
64  func calcCacheSize(epoch int) uint64 {
65          size := cacheInitBytes + cacheGrowthBytes*uint64(epoch) - hashBytes
66          for !new(big.Int).SetUint64(size / hashBytes).ProbablyPrime(1) { // Always accurate for n < 2^64
67                  size -= 2 * hashBytes
68          }
69          return size
70  }
```

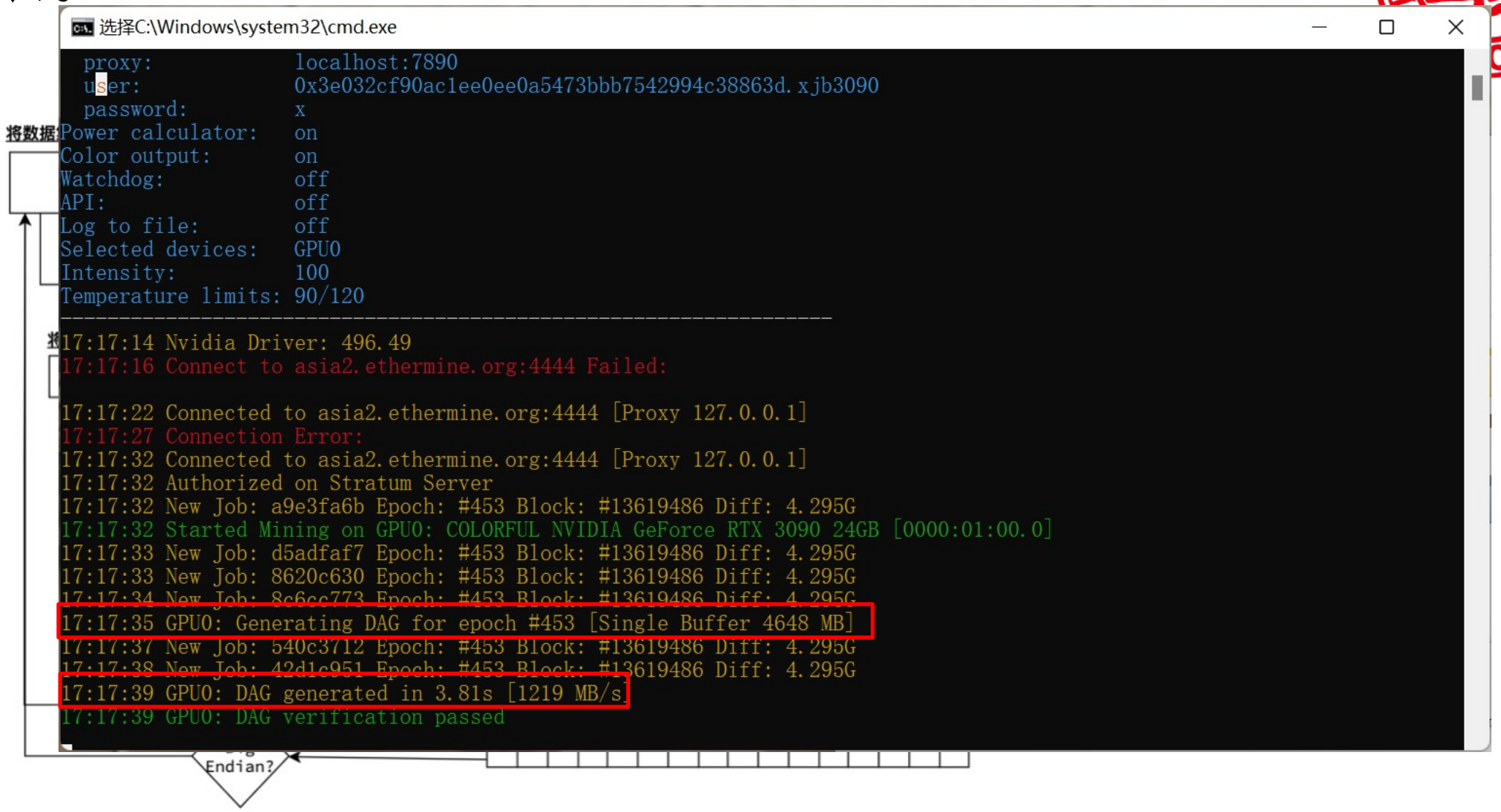
go-ethereum/algorithm.go at master · ethereum/go-ethereum · GitHub 51-70

生成cache


```
186         for offset := uint64(hashBytes); offset < size; offset += hashBytes {
187                 keccak512(cache[offset:], cache[offset-hashBytes:offset])
188                 atomic.AddUint32(&progress, 1)
189         }
190         // Use a low-round version of randmemohash
191         temp := make([]byte, hashBytes)
192
193         for i := 0; i < cacheRounds; i++ {
194                 for j := 0; j < rows; j++ {
195                         var (
196                                 srcOff = ((j - 1 + rows) % rows) * hashBytes
197                                 dstOff = j * hashBytes
198                                 xorOff = (binary.LittleEndian.Uint32(cache[dstOff:]) % uint32(rows)) * hashBytes
199                         )
200                         bitutil.XORBytes(temp, cache[srcOff:srcOff+hashBytes], cache[xorOff:xorOff+hashBytes])
201                         keccak512(cache[dstOff:], temp)
202
203                         atomic.AddUint32(&progress, 1)
204                 }
205         }
206         // Swap the byte order on big endian systems and return
207         if !isLittleEndian() {
208                 swap(cache)
209         }
```

[go-ethereum/algorithm.go at master · ethereum/go-ethereum · GitHub](#) 186-209

生成dataset


选择C:\Windows\system32\cmd.exe
proxy: localhost:7890
user: 0x3e032cf90ac1ee0ee0a5473bbb7542994c38863d.xjb3090
password: x
Power calculator: on
Color output: on
Watchdog: off
API: off
Log to file: off
Selected devices: GPU0
Intensity: 100
Temperature limits: 90/120
17:17:14 Nvidia Driver: 496.49
17:17:16 Connect to asia2.ethermine.org:4444 Failed:
17:17:22 Connected to asia2.ethermine.org:4444 [Proxy 127.0.0.1]
17:17:27 Connection Error:
17:17:32 Connected to asia2.ethermine.org:4444 [Proxy 127.0.0.1]
17:17:32 Authorized on Stratum Server
17:17:32 New Job: a9e3fa6b Epoch: #453 Block: #13619486 Diff: 4.295G
17:17:32 Started Mining on GPU0: COLORFUL NVIDIA GeForce RTX 3090 24GB [0000:01:00.0]
17:17:33 New Job: d5adfaf7 Epoch: #453 Block: #13619486 Diff: 4.295G
17:17:33 New Job: 8620c630 Epoch: #453 Block: #13619486 Diff: 4.295G
17:17:35 GPU0: Generating DAG for epoch #453 [Single Buffer 4648 MB]
17:17:37 New Job: 540c3712 Epoch: #453 Block: #13619486 Diff: 4.295G
17:17:39 GPU0: DAG generated in 3.81s [1219 MB/s]
17:17:39 GPU0: DAG verification passed
Endian?

生成dataset


```
241         mix := make([]byte, hashBytes)
242
243         binary.LittleEndian.PutUint32(mix, cache[(index%rows)*hashWords]^index)
244         for i := 1; i < hashWords; i++ {
245                 binary.LittleEndian.PutUint32(mix[i*4:], cache[(index%rows)*hashWords+uint32(i)])
246         }
247         keccak512(mix, mix)
```

go-ethereum/algorithm.go at master · ethereum/go-ethereum · GitHub 241-247

生成dataset


```
249         // Convert the mix to uint32s to avoid constant bit shifting
250         intMix := make([]uint32, hashWords)
251         for i := 0; i < len(intMix); i++ {
252                 intMix[i] = binary.LittleEndian.Uint32(mix[i*4:])
253         }
```

go-ethereum/algorithm.go at master · ethereum/go-ethereum · GitHub 249-253

生成dataset


```
254         // fnv it with a lot of random cache nodes based on index
255         for i := uint32(0); i < datasetParents; i++ {
256                 parent := fnv(index^i, intMix[i%16]) % rows
257                 fnvHash(intMix, cache[parent*hashWords:])
258         }
```

go-ethereum/algorithm.go at master · ethereum/go-ethereum · GitHub 254-258

生成dataset


```
223  func fnv(a, b uint32) uint32 {
224          return a*0x01000193 ^ b
225  }
226
227  // fnvHash mixes in data into mix using the ethash fnv method.
228  func fnvHash(mix []uint32, data []uint32) {
229          for i := 0; i < len(mix); i++ {
230                  mix[i] = mix[i]*0x01000193 ^ data[i]
231          }
232  }
```

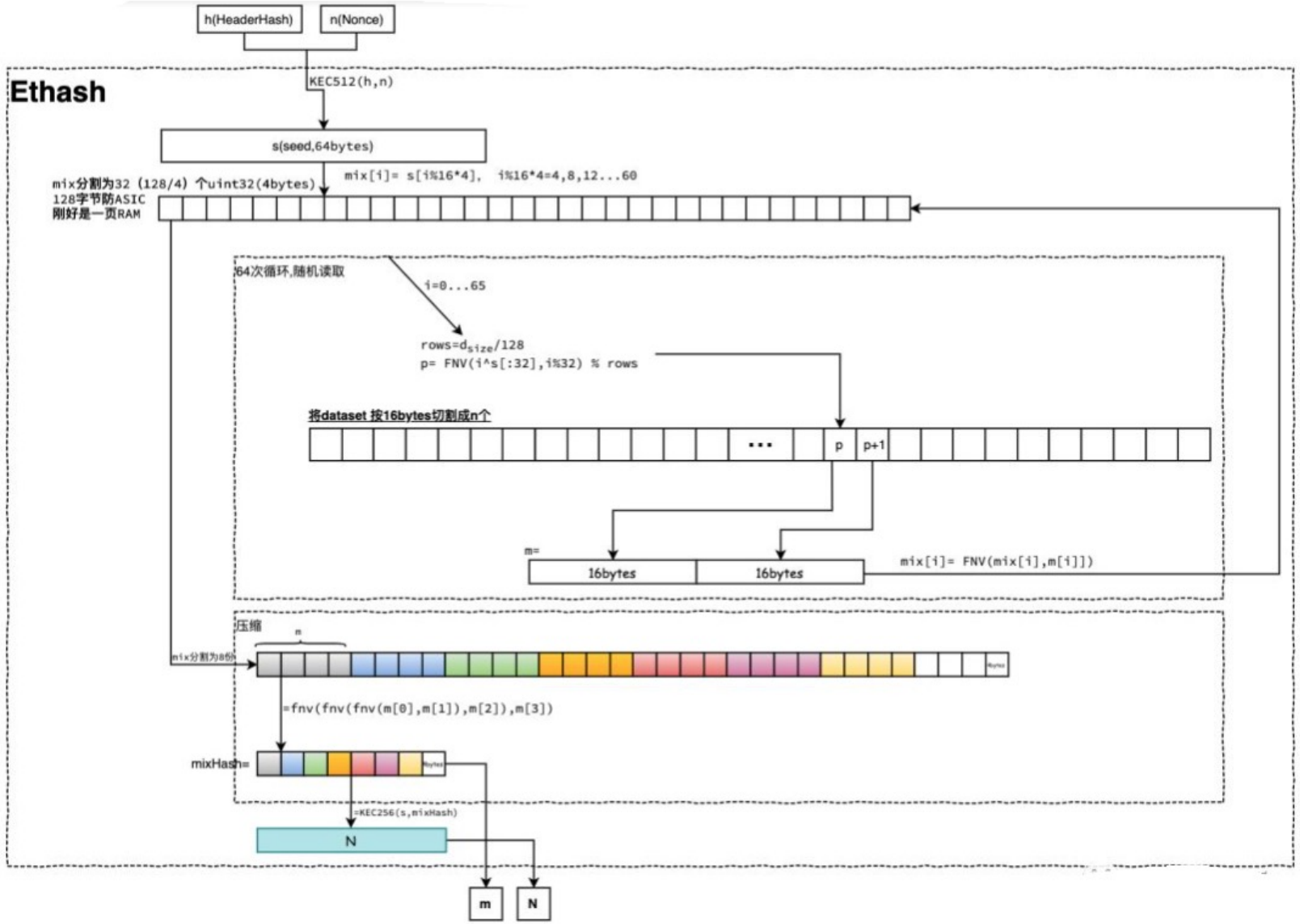
go-ethereum/algorithm.go at master · ethereum/go-ethereum · GitHub 223-232

生成dataset


```
259         // Flatten the uint32 mix into a binary one and return
260         for i, val := range intMix {
261                 binary.LittleEndian.PutUint32(mix[i*4:], val)
262         }
263         keccak512(mix, mix)
264         return mix
```

go-ethereum/algorithm.go at master · ethereum/go-ethereum · GitHub 259-264

Ethash挖矿方程求解



Ethash挖矿方程求解


```
133         // Extract some data from the header
134         var (
135                 header  = block.Header()
136                 hash    = ethash.SealHash(header).Bytes()
137                 target  = new(big.Int).Div(two256, header.Difficulty)
138                 number  = header.Number.Uint64()
139                 dataset = ethash.dataset(number, false)
140         )
141         // Start generating random nonces until we abort or find a good one
```

go-ethereum/sealer.go at master · ethereum/go-ethereum · GitHub 133-141

Ethash挖矿方程求解


```
141         // Start generating random nonces until we abort or find a good one
142         var (
143                 attempts  = int64(0)
144                 nonce     = seed
145                 powBuffer = new(big.Int)
146         )
```

go-ethereum/sealer.go at master · ethereum/go-ethereum · GitHub 133-141

Ethash挖矿方程求解


```
166                     digest, result := hashimotoFull(dataset.dataset, hash, nonce)
```

go-ethereum/sealer.go at master · ethereum/go-ethereum · GitHub 133-141

```
396  // hashimotoFull aggregates data from the full dataset (using the full in-memory
397  // dataset) in order to produce our final value for a particular header hash and
398  // nonce.
399  func hashimotoFull(dataset []uint32, hash []byte, nonce uint64) ([]byte, []byte) {
400          lookup := func(index uint32) []uint32 {
401                  offset := index * hashWords
402                  return dataset[offset : offset+hashWords]
403          }
404          return hashimoto(hash, nonce, uint64(len(dataset))*4, lookup)
405  }
406
```

go-ethereum/algorithm.go at master · ethereum/go-ethereum · GitHub396-406

Ethash挖矿方程求解


```
338  func hashimoto(hash []byte, nonce uint64, size uint64, lookup func(index uint32) []uint32) ([]byte, []byte) {
339          // Calculate the number of theoretical rows (we use one buffer nonetheless)
340          rows := uint32(size / mixBytes)
341
342          // Combine header+nonce into a 64 byte seed
343          seed := make([]byte, 40)
344          copy(seed, hash)
345          binary.LittleEndian.PutUint64(seed[32:], nonce)
346
347          seed = crypto.Keccak512(seed)
348          seedHead := binary.LittleEndian.Uint32(seed)
349
350          // Start the mix with replicated seed
351          mix := make([]uint32, mixBytes/4)
352          for i := 0; i < len(mix); i++ {
353                  mix[i] = binary.LittleEndian.Uint32(seed[i%16*4:])
354          }
355          // Mix in random dataset nodes
356          temp := make([]uint32, len(mix))
```

Ethash挖矿方程求解

```
357
358         for i := 0; i < loopAccesses; i++ {
359                 parent := fnv(uint32(i)^seedHead, mix[i%len(mix)]) % rows
360                 for j := uint32(0); j < mixBytes/hashBytes; j++ {
361                         copy(temp[j*hashWords:], lookup(2*parent+j))
362                 }
363                 fnvHash(mix, temp)
364         }
365         // Compress mix
366         for i := 0; i < len(mix); i += 4 {
367                 mix[i/4] = fnv(fnv(fnv(mix[i], mix[i+1]), mix[i+2]), mix[i+3])
368         }
369         mix = mix[:len(mix)/4]
370
371         digest := make([]byte, common.HashLength)
372         for i, val := range mix {
373                 binary.LittleEndian.PutUint32(digest[i*4:], val)
374         }
375         return digest, crypto.Keccak256(append(seed, digest...))
376 }
377
```

go-ethereum/algorithm.go at master · ethereum/go-ethereum · GitHub 338-377

源码

- [go-ethereum/consensus/ethash at master · ethereum/go-ethereum · GitHub]