bitcoin

Mining

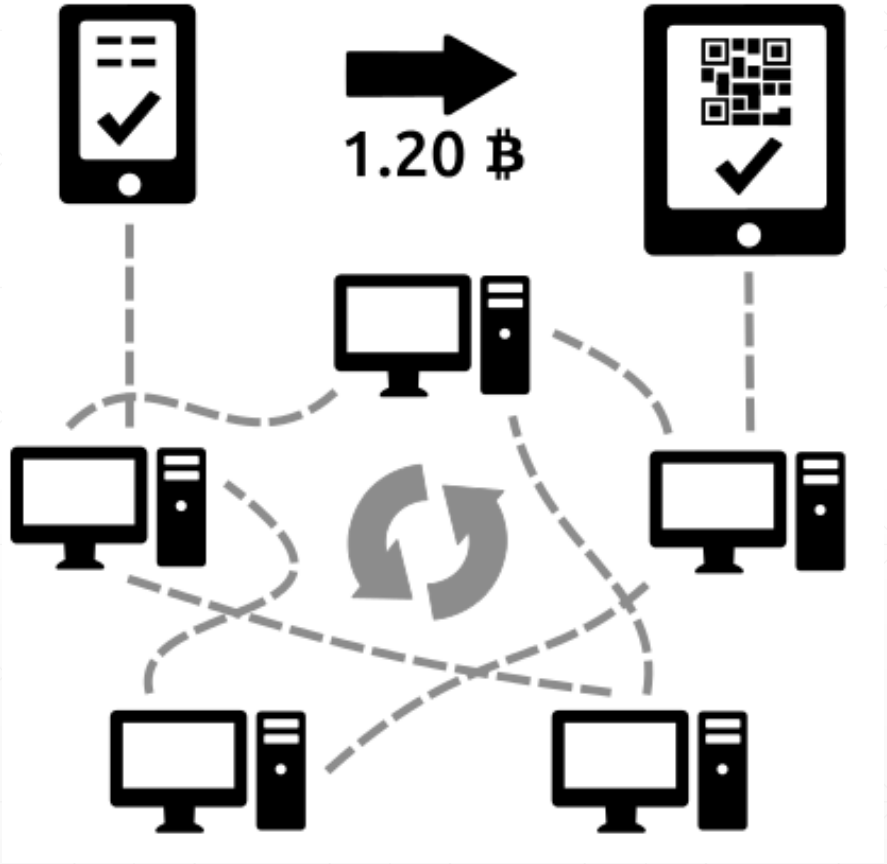
Team 9 Sebastian Helmer, Philipp Kahn 08.02.2016

Was erwartet Euch?

1. Bitcoins – eine Übersicht
2. Bitcoin Mining - Pseudocode
3. Demolauf unseres Programms
4. Codevorstellung
5. Speedup
6. Herausforderungen und Wissenserwerb

Bitcoin - ein dezentrales Netzwerk

- Kein zentraler Server
- Peer 2 Peer
- Broadcasting untereinander
- Jeder Teilnehmer kennt alle Transaktionen des Gesamtnetzwerkes
- Gleiche Verfügungsgewalten
- Anonym

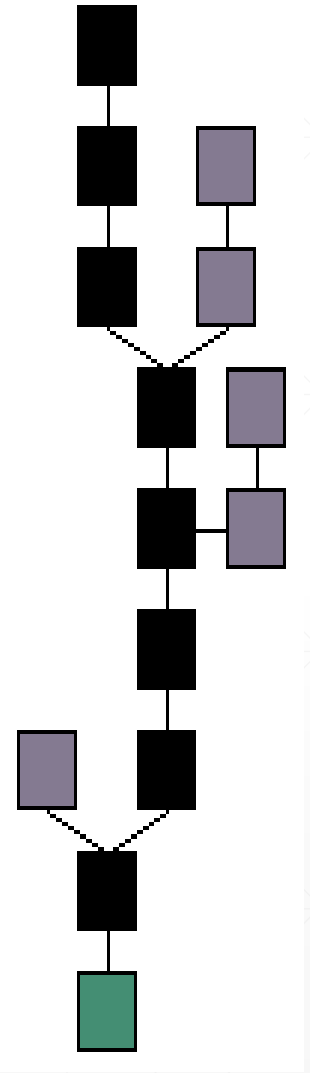


Bitcoin – Block Chain

- Blocks enthalten Transaktionen
- „Buchführungssystem“ des Bitcoin Netzwerkes

  - von allen Nodes des Systems geteilt
  - Volle Kopie beinhaltet jede stattgefundene Transaktion !
  - Wegweiser → Hashwert des vorhergehenden Blockes (bis Genesis trackbar)
  - Problem: Forks, Orphans
  - Lösung: Block Maturation

  → längste Chain als Valid anerkannt



Aufbau eines Blockes

Wichtigste:

Transactions

Blockheader

Field	Description	Size
Magic no	value always 0xD9B4BEF9	4 bytes
Blocksize	number of bytes following up to end of block	4 bytes
Blockheader	consists of 6 items	80 bytes
Transaction counter	positive integer VI = VarInt	1 - 9 bytes
transactions	the (non empty) list of transactions	<Transaction counter>-many transactions

Field	Purpose	Updated when...	Size (Bytes)
Version	Block version number	You upgrade the software and it specifies a new version	4
hashPrevBlock	256-bit hash of the previous block header	A new block comes in	32
hashMerkleRoot	256-bit hash based on all of the transactions in the block	A transaction is accepted	32
Time	Current timestamp as seconds since 1970-01-01T00:00 UTC	Every few seconds	4
Bits	Current target in compact format	The difficulty is adjusted	4
Nonce	32-bit number (starts at 0)	A hash is tried (increments)	4

Wichtigste: *Target, Hash-BlockHeader*

Bitcoin – Mining „Pseudocode“

➔ Finde eine Nonce, für die der hash des Blockheaders genügend klein ist

1. get Blockheader
2. for (nonce = 0; nonce < 0xffffffff; nonce++):
3. set nonce in Blockheader
4. if (sha256(sha256(Blockheader)) <= difficulty)
5. found nonce! You won 25bc ☺
6. update time in Blockheader
7. goto 2

Bitcoin – Mining Verbesserungspotential

- Iterierung über Noncen in einer „for“-Schleife

  ➔ Parallelisierungspotential mittels OpenMP / MPI

- Verwendung schneller Sha-Funktion zum hashen

  ➔ Vektorisierungspotential (an einigen Stellen, Abhängigkeiten!)

  Alignment der Daten (Blieb ohne Effekt)

Bitcoin – Mining „Real Code“

```
unsigned char data[80], hash[32];
size_t datasz;
unsigned int dataid;
uint32_t nonce;
blktemplate_t *tmpl;
SHA256 ctx = SHA256();

// Hier beginnt das eigentliche Mining
cout << "Press any Key to start mining" << endl;
WAIT_FOR_KEY;
cout << "Started mining ..." << endl;
do {
        // Hole aktuellen Block vom BC-Netzwerk
        tmpl = getblocktemplate(argc == 2);

        // Bilde Blockheader, speichern in data[80] 80-Byte
        datasz = blkmk_get_data(tmpl, data, sizeof(data), time(NULL), NULL, &dataid);
        assert(datasz >= 76 && datasz <= sizeof(data));

        // Schleifenabbruch - Variable, falls Nonce gefunden
        bool abort = false;

        cout << "iterating nonces ..." << endl;

        // Verteile Nonce-Intervalle auf die Cores (0xffffffff / #Cores)
        #pragma omp parallel for private(hash, ctx) firstprivate(data) shared(abort)
        for (nonce = 0; nonce < 0xffffffff; ++nonce)
        {
                ...
                ...
                ...

        }

        cout << "Tried all Nonces. Getting new Template" << endl;
} while (blkmk_time_left(tmpl, time(NULL)) && blkmk_work_left(tmpl));
cout << "the end";
blktmpl_free(tmpl);
```

```
if(!abort) {
        // Setze aktuelle Nonce an richtige Stelle im Dataarray
        *(uint32_t*)(&data[76]) = nonce;

        // Berechne doppel-Hash auf Datenarray
        ctx.init();
        ctx.update( (unsigned char*)data, 80);
        ctx.final((unsigned char*)hash);
        ctx.init();
        ctx.update( (unsigned char*)hash, 32);
        ctx.final((unsigned char*)hash);

        // Vergleich, ob Hashwert die Difficulty erfüllt
        if (!*(uint32_t*)(&hash[28])) {
                abort = true;
                cout << "found nonce " << nonce << " with Hash: " << flush;
                print_hash(hash);

                // Antwortgenerierung
                json_t *resp;
                nonce = ntohl(nonce);
                resp = blkmk_submit_jansson(tmpl, data, dataid, nonce);
                assert(resp);
                send_json(resp);
        }
 }
```

Bitcoin – Mining „Sha Funktion“ - Vektorisierungspotential

Auszug aus SHA256 Algorithmus

```
31      for (i = 0; i < (int) block_nb; i++) {
32          sub_block = message + (i << 6);
33          for (j = 0; j < 16; j++) {
34              SHA2_PACK32(&sub_block[j << 2], &w[j]);
35          }
36          for (j = 16; j < 64; j++) {
37              w[j] =  SHA256_F4(w[j -  2]) + w[j -  7] + SHA256_F3(w[j - 15]) + w[j - 16];
38          }
39          for (j = 0; j < 8; j++) {
40              wv[j] = m_h[j];
41          }
42          for (j = 0; j < 64; j++) {
43              t1 = wv[7] + SHA256_F2(wv[4]) + SHA2_CH(wv[4], wv[5], wv[6])
44                  + sha256_k[j] + w[j];
45              t2 = SHA256_F1(wv[0]) + SHA2_MAJ(wv[0], wv[1], wv[2]);
46              wv[7] = wv[6];
47              wv[6] = wv[5];
48              wv[5] = wv[4];
49              wv[4] = wv[3] + t1;
50              wv[3] = wv[2];
51              wv[2] = wv[1];
52              wv[1] = wv[0];
53              wv[0] = t1 + t2;
54          }
55          for (j = 0; j < 8; j++) {
56              m_h[j] += wv[j];
57          }
58      }
59  }
```

Vektorisierung erfolgreich

w[j] = … w[j – 2] … ;

Vektorisierung erfolgreich

Abhängigkeiten von t1 und t2

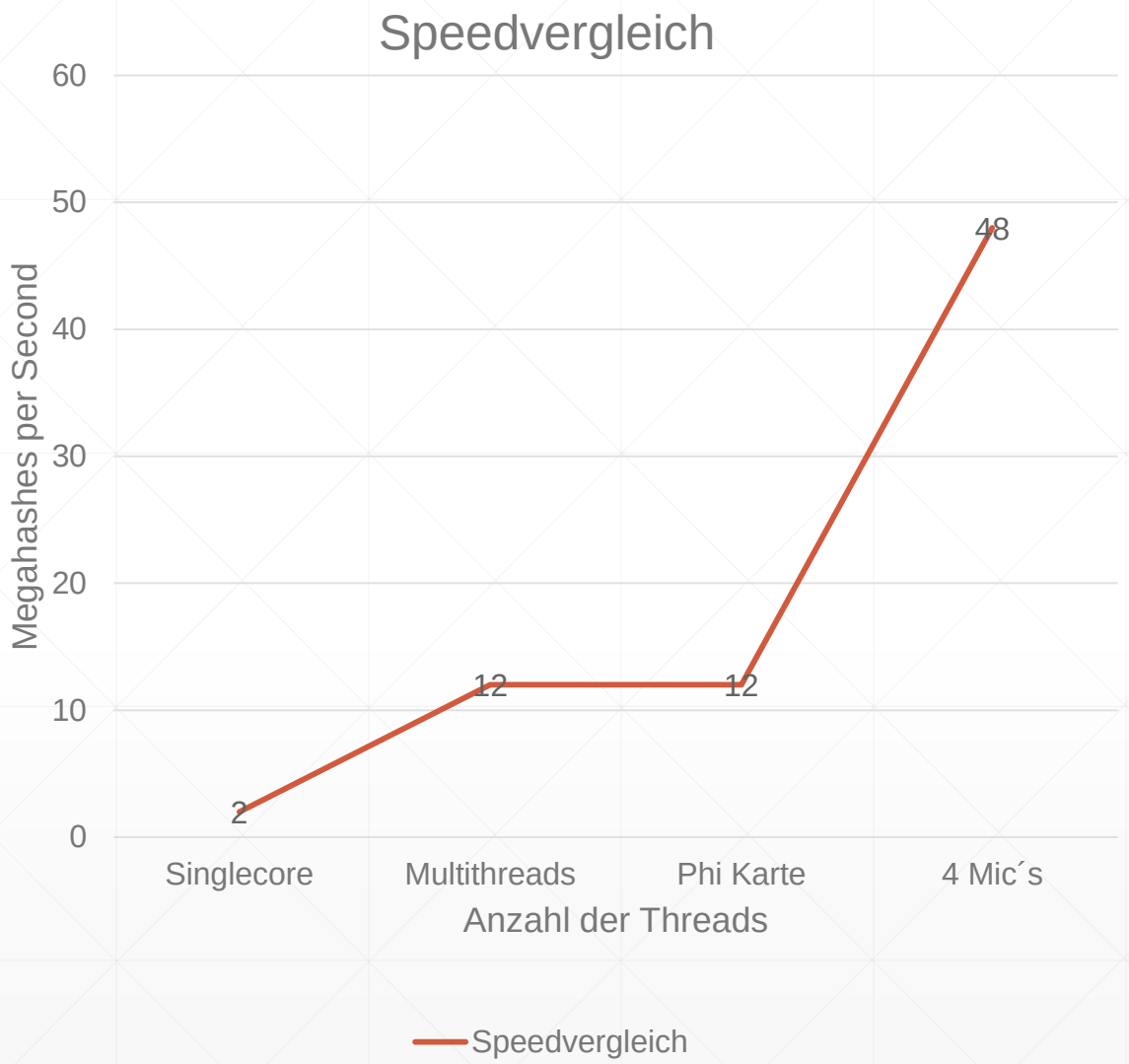
Vektorisierung erfolgreich

Speedup - Betrachtung

- Parallelisierung mittels OpenMP
- Vektorisierung (in Sha-Funktion)
- Verteilung mittels MPI auf den Mic´s (Nur Sha Funktion)

- Ergebnis:
  - Speedup von 2 Megahashes/Second auf 12 Meghashes/Second

➔ Speedup von 6 multithreaded,



Herausforderungen und Lerneffekt

- Herausforderungen
  - **Bitcoin Funktionsweise und Konventionen verstehen**
    - Daten aus Netzwerk anfordern und verarbeiten
    - Target-Wert auslesen und verarbeiten
    - Datenstrukturen verstehen
  - Hash-Funktion optimieren

- Lerneffekt
  - Hardwarenahes Programmieren in C++
  - Funktionsweise von Bitcoin
  - Methoden zur Laufzeitbeschleunigung von Algorithmen

Quellen

- Internet-Ressourcen
  - https://bitcoin.org/de/
  - https://en.bitcoin.it/wiki/
  - http://www.yogh.io/#mine:last

Vielen Dank für Eure Aufmerksamkeit