CS 220: Introduction to Parallel Computing

Proof-of-Work & Bitcoin

Lecture 19

Bitcoin: Mining for Heat

- https://qz.com/1117836/bitcoin-mining-heats-homes-for-free-in-siberia/

Today's Agenda

- Proof-of-work
- Bitcoin
- Project 3 tips

Today's Agenda

- **Proof-of-work**
- Bitcoin
- Project 3 tips

Proof-of-Work

- Proof-of-work (**POW**) systems help prevent DDoS attacks and other types of spamming
- Also useful in cryptocurrencies
- Give up some of your time (or computational power) to legitimize an action/object

Shell Money

- Sea shells were used for thousands of years as legal tender
- It takes time to collect shells, carve them, etc.
  - In some cases, the shells were woven into fabric/leather
  - The currency itself reflected the time it took to be made, and therefore determined its value
- Different groups used different shells/designs
  - Only carry value because we say so

CAPTCHA

- **C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part
- CAPTCHAs are basically proof-of-work systems for humans
- So in other words, POW is an annoying, time consuming task for your computer to do just in the interest of proving it's not spamming/DDoSing
  - Luckily computers don't get annoyed as easily as we do…

Pricing Functions

- POW systems use **pricing functions** to give the computer a workout
- A pricing function **f** has the following requirements:
  - **f** is moderately easy to compute
  - **f** has a consistent computational cost
  - given **x** and **y**, it is easy to determine if **y** = **f**(**x**)
    - (Much easier than computing f(x))

Dwork C., Naor M. *Pricing via Processing or Combatting Junk Mail*.

Hash Inversions (1/2)

- A common pricing function is having the computer perform **hash inversions**
  - What was the **input** that produced this hash code?
- A hash function takes data of **any size** and maps it to a **fixed size**:
  - "Hello world!" → `ed076287532e86365e841e92bfc50d8c`
  - "Blah blah blah blah blah blah" → `cd647e323da9f2fc2dac4800a37661f1`

Hash Inversions (2/2)

- So, a hash inversion is trying to get a specific output from the hash function
- Hash inversions are tough to compute (assuming a cryptographic hash function)
  - After all, they're designed to be **one way** functions
  - **But** any time we map an infinite set of inputs to a finite set of numbers (hash space), this is feasible

HashCash: Preventing Spam

- The core technology behind BitCoin was developed as an anti-spam technique
- You'll spend some time performing hash inversions to prove you're not a spammer
  - No big deal for us regular users, but spammers trying to send billions of messages will be slowed down!
- The result of your computation is included in an email header and verified on the receiving end

An Example (1/2)

- Let's send an email and prove we're not a spammer. Our mission is to find a hash with four leading zeros
- Start out with what we want to send:
  - "Hello World!"
- We also need to append a **nonce**
  - Number used only once
  - We increase this with each hash attempt
  - This will change our output hash each iteration

An Example (2/2)

- This approach allows us to eventually find our matching hash, but has a weakness
  - We can precompute the hashes and re-use them later
- We also need some type of identifier for this particular transaction
  - Maybe a centralized service hands out transaction IDs
  - We could use the current time, as long as we can assume clocks are reasonably synced up

Pseudocode – Pricing Function

```
while True:
    nonce = nonce + 1
    string = message + str(nonce)
    hash = sha1(string)
    if prefix(hash) == '0000':
        # Send message with hash
        break
```

Pseudocode - Verification

```
if sha1(msg.payload) == msg.hash:
    # Valid… Whew! That was tough!
    # (You could also verify the
    # transaction id or timestamp here)
```

Varying the Difficulty

- To change the difficulty, we'll just adjust the number of zeros we want
- **Note**: the difficulty won't increase linearly
- Approaches:
  - Perform a bitwise comparison rather than string (allows more precision)
  - Have the sender perform multiple inversions (maybe message1 + another nonce)

Why Hashcash Works

- Even heavy email users only send a few hundred emails per day
- Spammers want to send millions/billions
  - This is going to cost a lot of CPU time
- Additionally, sending an email with no header or an incorrect header will incur steep penalties
  - Too many incorrect headers? Ban the IP
- Best of all, we don't have to start paying for email

Why it Doesn't Work (1/2)

- Back in 1992 when Hashcash was invented, we didn't have such a huge variety of computing hardware
  - Smartphones, tablets, refrigerators, etc.
  - This makes coming up with the right difficulty for the challenge… difficult.
- The power of computing hardware isn't distributed uniformly across the Earth
- Hash inversions are amenable to parallelism and custom hardware

Why it Doesn't Work (2/2)

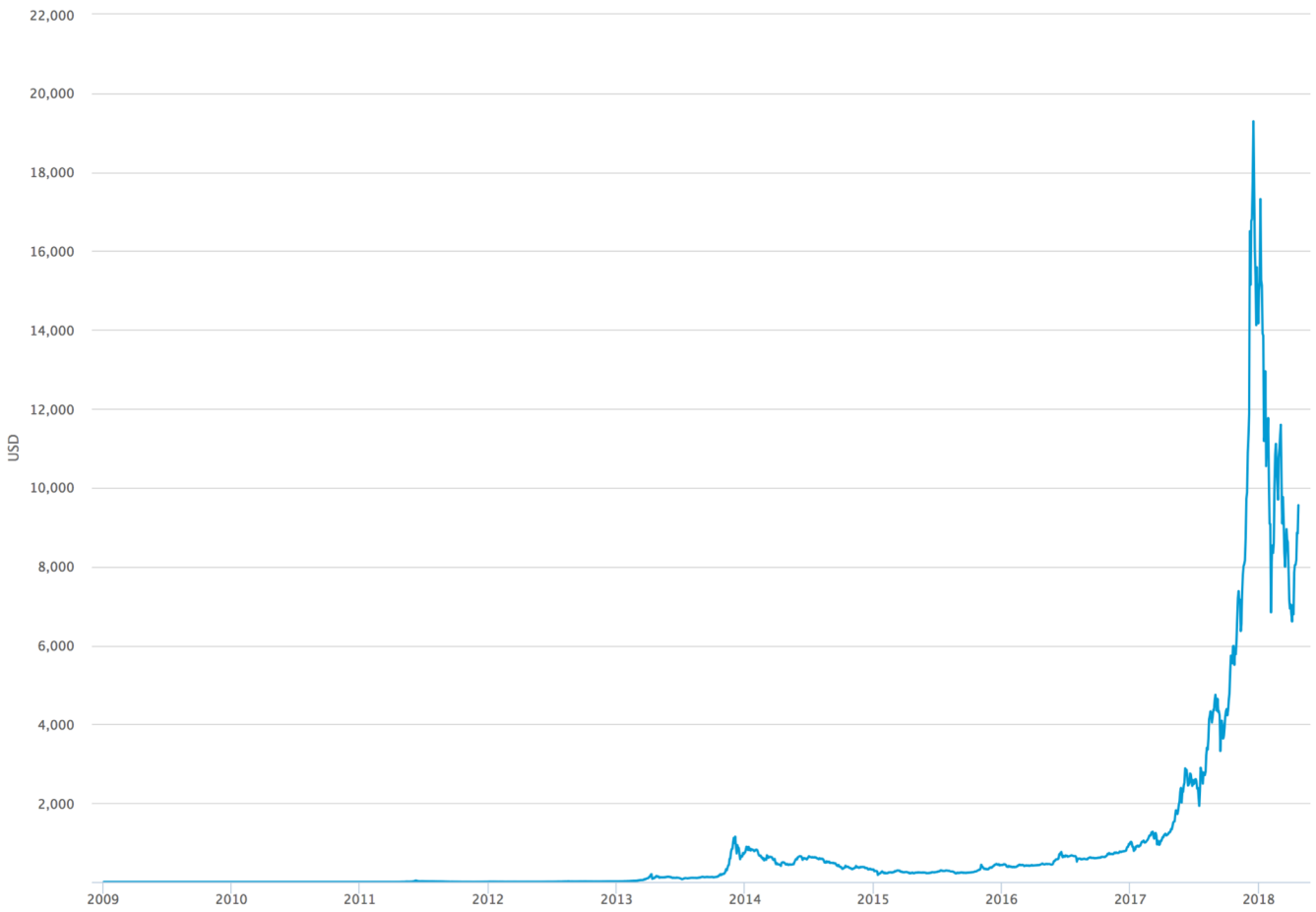
- Spammers could adopt similar hardware to that of Bitcoin miners
  - GPUs, ASICs
  - Depends on cost vs. benefit
  - Related: cloud instances. Computing is **so** cheap!
- Since email is decentralized, you can't force everyone to use this new standard
  - Would actually be easier nowadays (get Google and Microsoft on board, and you're just about done)

Today's Agenda

- Proof-of-work
- **Bitcoin**
- Project 3 tips

Bitcoin Value: Last Semester

Bitcoin Value: Now



As of Now

- 1 BTC = 8911.89 USD
  - Gone up $3000 since last semester!
- 215,980 transactions per day
- ~17m bitcoins in circulation
- See: http://blockchain.info

Blockchain

- The Bitcoin **blockchain** is a decentralized database of Bitcoin transactions
- Each block in the chain includes the hash of the previous block
- Starts with the **genesis block**
- When a transaction occurs, it is added to the current block and will be verified by miners

Blocks

- A **block** is a list of transactions with some metadata
- Magic number (4 bytes) = 0xD9B4BEF9
- Block size (4 bytes)
- Block header
- Transaction counter
- Transaction data

Block Headers

- Version
- Hash of the previous block
  - This makes tampering with the chain difficult
- Current hash of the transactions in the block
- Timestamp (last update)
- Difficulty
- Nonce

Mining Bitcoin

- Bitcoin uses the Hashcash algorithm for a different purpose: **mining** coins
- "Mining" means verifying a block of transactions
  - Finding the nonce (aka **solution**)
- Miners, who are the basis of transaction verification, are paid in new bitcoins and transaction fees
  - The reward of new bitcoins is halved every 210,000 blocks (~4 years)
  - Monetary supply limited to 21m bitcoins

Verification

- In bitcoin, the **difficulty** of the challenge is varied to keep the network chugging along
- Once all 21m bitcoins are created, miners will be rewarded for verification via transaction fees only
- What is the cost vs. benefit of mining these coins?
  - Electricity vs. the size of the reward
- Lots of companies now build power-efficient hardware specifically for mining

Pooled Mining

- As difficulty goes up, the chances of a single miner verifying a block goes down
- To combat this, **pools** of miners formed
- Pools divide up the work (nonces) among participants
  - Rewarded with a **share** of new bitcoins based on how much work was done
  - Less wasted effort, but less reward

"Moral" Issues

- We are consuming massive amounts of fossil fuels to produce fake money
  - Production is only hard because we make it so
- Mining hardware gets bought up and then discarded once we move to harder hash inversions
- Some *Useful Proof-of-Work* systems try to do beneficial work
  - Finding prime numbers (**Primecoin**)
  - Protein folding (**Curecoin**)

Today's Agenda

- Proof-of-work
- Bitcoin
- **Project 3 tips**

Project 3

- In this assignment, we'll look at the **mining** aspect of bitcoin
- Given a block of data (in our case, just a string), we'll find the nonce that satisfies a given difficulty
  - Number of zeros at the front of the hash string
- We'll implement a parallel mining approach with the producer-consumer paradigm

Producer-Consumer

- We'll distinguish between two types of threads in this assignment:
  - Producer – main thread
  - Consumers – worker threads
- The main thread generates **tasks**
  - Arrays of nonces to append to the block data & hash
  - Each array of tasks will be placed in a staging area (called `task_pointer`)
  - The size of these arrays is something you will experiment with
- The consumers pick up a task and perform the hash inversions, checking the number of zeros at the start of the hash

What You'll Implement

- The mining function is given to you already, as well as a working sha1 hash function
- You will:
  1. Implement variable difficulty (the user can specify how many zeros they want to find)
  2. Use condition variables instead of simply busy waiting for tasks
  3. When a solution is found, tell other threads to stop
  4. Print out timing info / statistics

Setting Specific Bits

- Let's say I asked you to set the 3rd bit in a bit field
- How would you accomplish this?
- bit_field = bit_field | (0x1 << 3)

Setting Difficulty

- We can extend this approach to adjust the difficulty of our bitcoin miner
- We'll just need to find out how many bits we need to to set to 1
- Then, loop until all the bits are set