Trabalho I de Computação Concorrente: Simulação de Mineração de Bitcoin

Gabriel da Fonseca Ottoboni Pinho - DRE 119043838
Rodrigo Delpreti de Siqueira - DRE 119022353

25/04/2021

1 Descrição do problema

1.1 Sobre a Bitcoin

Bitcoin é uma criptomoeda criada por Satoshi Nakamoto em 2009, tendo como objetivo ser uma moeda digital descentralizada. Uma parte essencial da Bitcoin é uma tecnologia chamada *Blockchain*, que consiste em armazenar todas as transações que já ocorreram em blocos interligados. Cada transação é criptograficamente assinada por seu emissor e então enviada ao resto da rede, que verifica a assinatura e a existência dos fundos.

Um ponto importante é que cada transação é verificada por todos, de modo que nenhum integrante precisa confiar em nenhum outro, cada um sendo capaz de verificar a verdade independentemente. Os integrantes da rede que coletam as transações e as colocam em blocos são os mineradores e o sistema de consenso que determina qual minerador terá o direito de criar o bloco se chama *Proof of Work* (PoW).

A ideia da PoW é que o minerador que conseguir a resposta de um desafio primeiro tem o direito de criar o bloco. Esse desafio consiste em achar um número *nonce*, tal que quando o *hash* SHA-256 do *header* do bloco (que contém a *nonce*) seja menor que um número chamado de dificuldade. Pelo fato do SHA-256 ser uma função *hash* criptográfica, a única forma de achar a *nonce* correta é chutando. Em outras palavas, não é possível achar um x tal que $\text{SHA256}(x) = y$. Por outro lado, tendo um x, é fácil verificar se $\text{SHA256}(x) = y$. Essa propriedade é importante, pois, dessa forma, todos os integrantes da rede podem verificar facilmente se a *nonce* encontrada é uma solução válida de fato.

1.2 Sobre o SHA-256

O SHA-256 é uma função *hash* criptográfica, que gera uma sequência de 256 bits pseudo-aleatória para uma entrada qualquer. O cálculo dessa função é computacionalmente custoso, e é o *work* na PoW da Bitcoin. O cálculo da função consiste em 3 passos principais:

1. Pré-processamento
2. Criação do array de mensagens
3. Compressão

Durante o pré-processamento, a entrada é será divida em pedaços chamados *chunks* de 512 bits cada. Depois disso, para cada *chunk*, um array de 64 elementos de 4 bytes é preenchido com base no conteúdo do *chunk* atual. Por fim, 8 variáveis são inicializadas com valores pré-determinados que serão modificados em cada *round* do loop de compressão. O valor final do *hash* será a concatenação das 8 variáveis.

2 Projeto da solução concorrente

O objetivo do nosso trabalho é implementar o SHA-256 e usá-lo para simular a mineração da Bitcoin. Em outras palavras, temos que receber como entrada um número d representando a dificuldade e o número n de threads a serem utilizados. A saída será uma *nonce* que resolveria um bloco com essa dificuldade e o tempo levado para calculá-la. A dificuldade será o número de vezes seguidas que o byte `0x00` aparece no início do *hash* calculado. Como foi explicado na sessão anterior, não é exatamente assim que a dificuldade é definida na implementação da Bitcoin, mas foi decidido simplificar essa etapa.

A simulação funcionará da seguinte forma:

1. 76 bytes aleatórios são gerados. Esses bytes representarão todos os campos do *header* do bloco, exceto a *nonce*.

2. Cada thread calculará o *hash* da concatenação desses mesmos 76 bytes com mais 4 bytes, a *nonce*. Temos assim um total de 80 bytes, que é o tamanho real do *header* de um bloco de Bitcoin.

3. Se SHA256(*header*) começar com pelo menos d bytes `0x00` seguidos, conseguimos achar uma *nonce* correta, fim. Senão, repetimos as etapas 2 e 3 com uma *nonce* diferente.

É importante notar que cada tentativa é completamente independente da outra, ou seja, podemos facilmente acrescentar mais threads, com cada uma testando diferentes valores para a *nonce* até que a resposta correta seja encontrada. A estratégia utilizada foi testar valores consecutivos para a *nonce*, e distribuir esses números entre as n threads de n em n. Quando alguma das threads achar a resposta, uma variável global *flag* é setada e as outras threads terminam. A thread que encontrou a resposta retorna seu valor para a thread principal, que imprime os resultados na tela.

3 Casos de teste

A principal função do programa que precisa ser testada é `sha256`, que recebe um sequência de bytes e retorna uma string com o valor hexadecimal do *hash* SHA-256 da entrada. Cada teste chama essa função com valores de entrada pré-determinados e compara o *hash* retornado com um valor correto pré-calculado.

4 Avaliação de desempenho

5 Discussão

Referências

[Gul] Sean Gulley. *Intel® SHA Extensions*. URL: `https://software.intel.com/content/www/us/en/develop/articles/intel-sha-extensions.html`. (acessado em 20/04/2021).

[Nak] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. URL: `https://bitcoin.org/bitcoin.pdf`. (acessado em 22/04/2021).

[Wag] Lane Wagner. *How SHA-256 Works Step-By-Step*. URL: `https://qvault.io/cryptography/how-sha-2-works-step-by-step-sha-256/`. (acessado em 19/04/2021).

[Wik] Bitcoin Wiki. *Bitcoin Protocol Documentation*. URL: `https://en.bitcoin.it/wiki/Protocol_documentation#Block_Headers`. (acessado em 22/04/2021).