Approximate Bitcoin Mining

Matthew Vilim, Henry Duwe, Rakesh Kumar

Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign

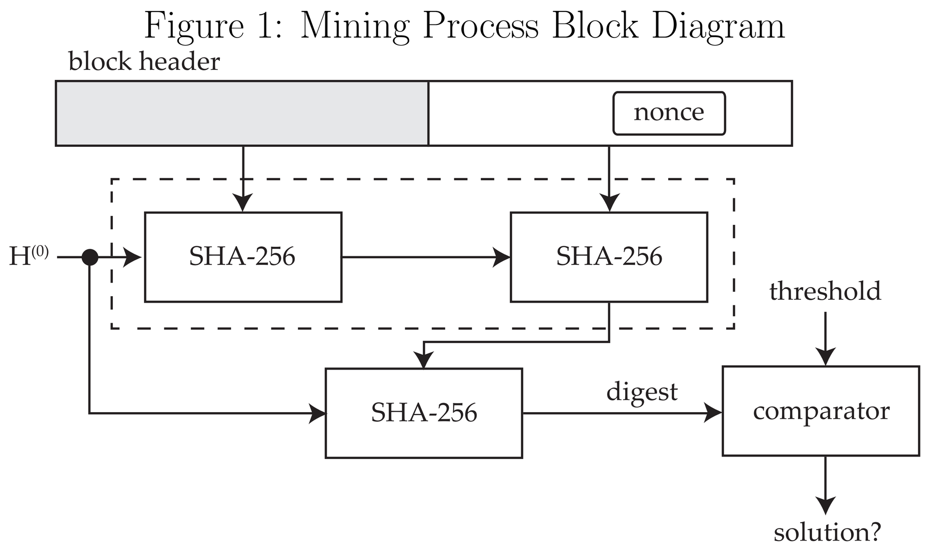
Abstract

BITCOIN is the most popular cryptocurrency today. A bedrock of the Bitcoin framework is *mining*, a computation intensive process that is used to verify Bitcoin transactions for profit. We observe that mining is inherently error tolerant due to its embarrassingly parallel and probabilistic nature. We exploit this inherent tolerance to inaccuracy by proposing approximate mining circuits that trade off reliability with area and delay. These circuits can then be operated at Better Than Worst-Case (BTWC) to enable further gains. Our results show that *approximation* has the potential to increase mining profits by 30%.

Mining Background

THE Bitcoin mining process is summarized in Figure **??**. Mining consists of searching for a cryptographic *nonce* value within a block such that the hash of the block falls within a certain range.

Figure 1: Mining Process Block Diagram



THE mining algorithm is shown in Algorithm **??**. In short, mining is a search for the nonce value that results in a double SHA-256 hash *digest* (Algorithm **??**) value less than a given *threshold*. The nonce is a 32-bit field within a 1024-bit *block header*. In order to verify transactions at a steady rate, this threshold varies over time as a function of difficulty $D(t)$. Difficulty is adjusted by the network regularly such that a solution is expected to be found approximately every 10 minutes, regardless of the network's collective hash rate.

Algorithm 1 Mining Process

```
nonce ← 0
while nonce < 2^32 do
    threshold ← ((2^16 − 1) ≪ 208)/D(t)
    digest ← SHA-256(SHA-256(header))
    if digest < threshold then
        return nonce
    else
        nonce ← nonce + 1
    end if
end while
```

SHA-256 Datapath Overview

FOR our studies, we selected as baseline the SHA-256 ASIC design outlined by Dadda et al. A summary of SHA-256 is provided in Algorithm **??**. The hashing core in this design is implemented as two parallel pipelines, the Compressor (Line **??** of Algorithm **??**) and the Expander (Line **??** of Algorithm **??**) shown in Figure **??**.

- The message M is divided into N 512-bit blocks $M^{(0)}, M^{(1)}, \ldots, M^{(N-1)}$. Each of these blocks is further subdivided into 16 32-bit words $M_0^{(i)}, M_1^{(i)}, \ldots, M_{15}^{(i)}$.
- The intermediate hash value $H^{(i)}$ is composed of 8 32-bit words $H_0^{(i)}, H_1^{(i)}, \ldots, H_7^{(i)}$.
- $Ch(x,y,z) \equiv (x \wedge y) \oplus (\neg x \wedge z)$
- $Maj(x,y,z) \equiv (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$
- $\Sigma_0(x) \equiv x \ggg 2 \oplus x \ggg 13 \oplus x \ggg 22$
- $\Sigma_1(x) \equiv x \ggg 6 \oplus x \ggg 11 \oplus x \ggg 25$
- $\sigma_0(x) \equiv x \ggg 7 \oplus x \ggg 18 \oplus x \gg 3$
- $\sigma_1(x) \equiv x \ggg 17 \oplus x \ggg 19 \oplus x \gg 10$

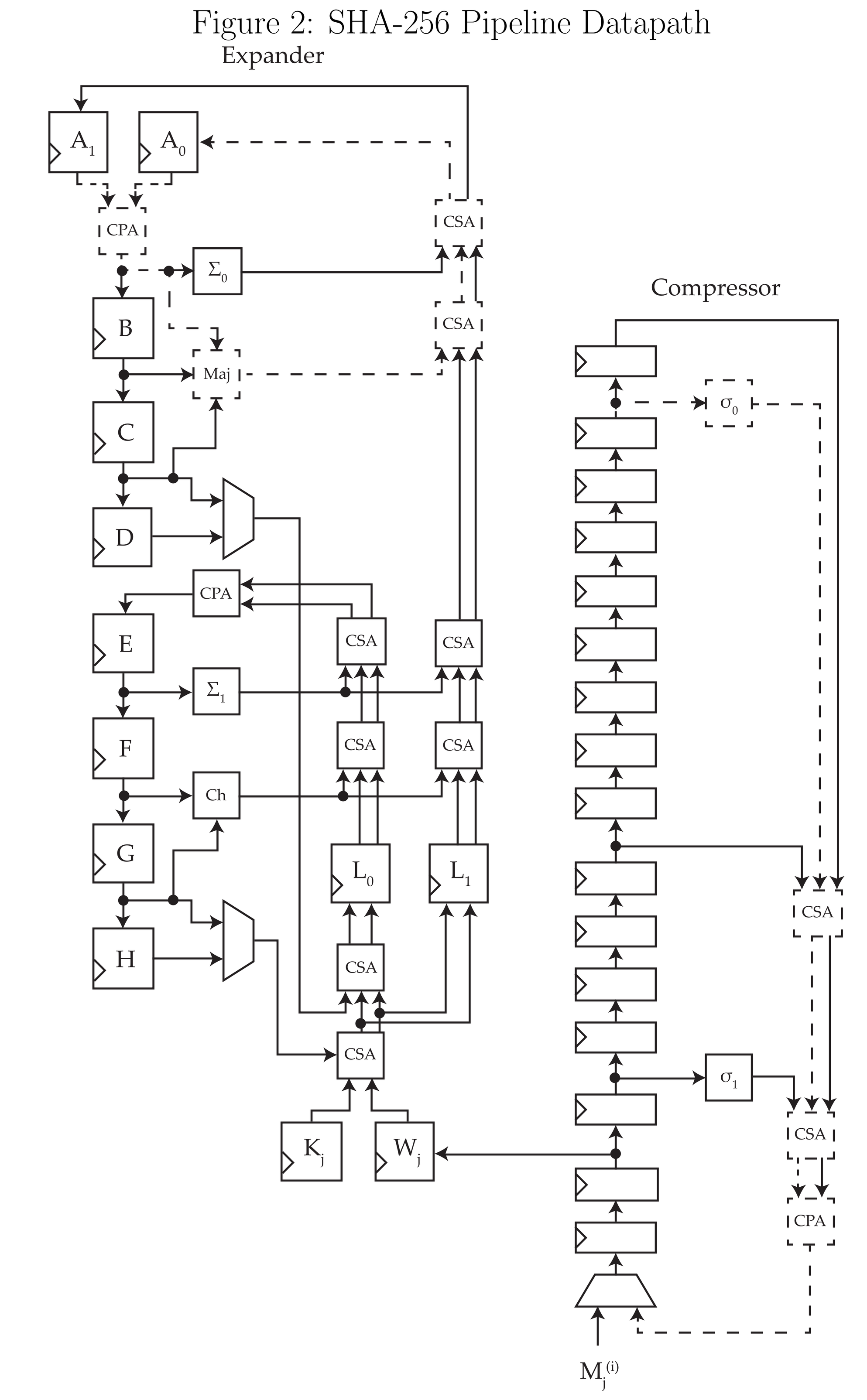
Algorithm 2 SHA-256

```
function SHA-256(M)
    for i from 0 to N − 1 do
        for j from 0 to 15 do
            W_j = M_j^(i)
        end for
        for j from 16 to 63 do
            W_j = σ1(W_{j−2}) + W_{j−7} + σ0(W_{j−15}) + W_{j−16}
        end for

        for j from 0 to 63 do
            t0 ← h + Σ1(e) + Ch(e, f, g) + K_j + W_j
            t1 ← Σ0(a) + Maj(a, b, c)
            h ← g; g ← f
            f ← e; e ← d + t1
            d ← c; c ← b
            b ← a; a ← t1 + t2

            H_0^(i) ← H_0^(i−1) + a; H_1^(i) ← H_1^(i−1) + b
            H_2^(i) ← H_2^(i−1) + c; H_3^(i) ← H_3^(i−1) + d
            H_4^(i) ← H_4^(i−1) + e; H_5^(i) ← H_5^(i−1) + f
            H_6^(i) ← H_6^(i−1) + g; H_7^(i) ← H_7^(i−1) + h
        end for
    end for
    return H^(N−1)
end function
```

Figure 2: SHA-256 Pipeline Datapath



Approximate Mining

A miner's instantaneous profit $p(t,f)$ at time t and frequency f is a function of the mining yield $Y(t)$ (USD/GHash), hash rate $H(f)$ (GHash/s), power consumption $P(f)$ (kW), and cost of electricity $C_e(t)$ (USD/kWh).

$$p(t,f) = H(f) \cdot Y(t) - P(f) \cdot \frac{C_e(t)}{60 \cdot 60} \tag{1}$$

IN the presence of approximation, the effective hash rate changes. A fraction $E(f)$ (error rate) of the computed hashes will be incorrect, and a normalized reduction in area $\hat{A}$ may occur.

$$\tilde{H}(f) = \frac{1 - E(f)}{\hat{A}} \cdot H(f) \tag{2}$$

THERE are $64 \cdot 3 = 192$ additions in a single round of SHA-256, each with error rate E_{CPA}. The error rate of a single round in the hashing core, therefore, is:

$$E_f = 1 - (1 - E_{CPA})^{192} \tag{3}$$

THE error rate at each operating point is found through simulation. Each simulated SHA-256 round has error rate $E_i(f)$, the sum of functional and operational error rates. Bitcoin requires two rounds for each nonce iteration; hence, we can extrapolate to calculate cumulative error rate $E(f)$, assuming the hash inputs and outputs to be uniform random variables.

$$E_i(f) = E_f + E_o(f) \qquad E(f) = 1 - [1 - E_i(f)]^2 \tag{4}$$

Results

TABLE **??** lists the adders' delay and area. Each adder was inserted into the hashing core pipelines in the CPA slots indicated in Figure **??**. The resulting hashing core area and delay are provided in Table **??**. Approximate variants are highlighted in gray. Figure **??** shows the error rate-frequency characteristic $E_i(f)$ of each hashing core for various adders after simulating a full round of SHA-256. The resulting frequency-profit relation is shown in Figure **??**.

Table 1: Adder Comparison

Adder	delay · area (ns · µm²)	P (mW)	E_{CPA}
RCA	7116	0.170	NA
CLA	4834	0.889	NA
$GDA_{(1,4)}$	3558	0.950	1.90×10^{-5}
KSA_{32}	3631	0.867	NA
KSA_{16}	2862	0.814	4.60×10^{-5}
KSA_8	2102	0.715	2.26×10^{-2}

Table 2: Hashing Core Comparison for Different Adder Choices

Adder	delay · area (ns · µm²)	P (mW)	E_f
RCA	210,0597	7.19	NA
CLA	123,865	12.1	NA
$GDA_{(1,4)}$	108,207	13.8	7.27×10^{-3}
KSA_{32}	90,769	17.33	NA
KSA_{16}	82,744	19.0	8.79×10^{-2}
KSA_8	73,152	20.4	1.00

Figure 3: Frequency/Error Rate Trade-off for Cores

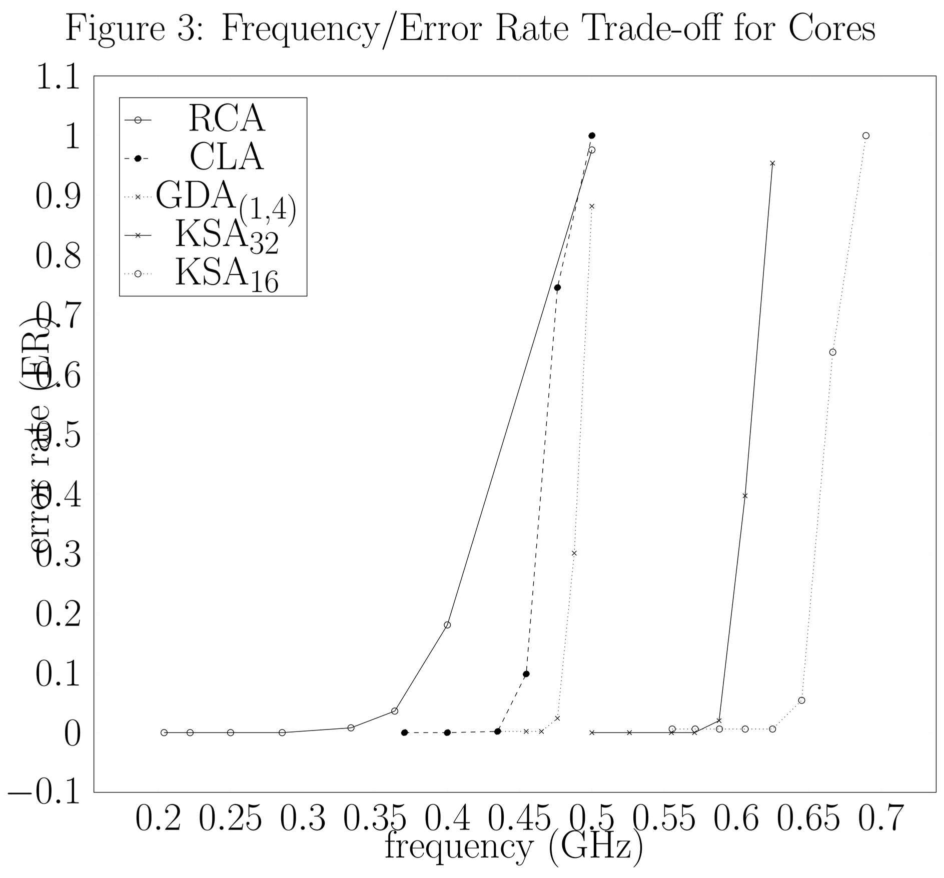
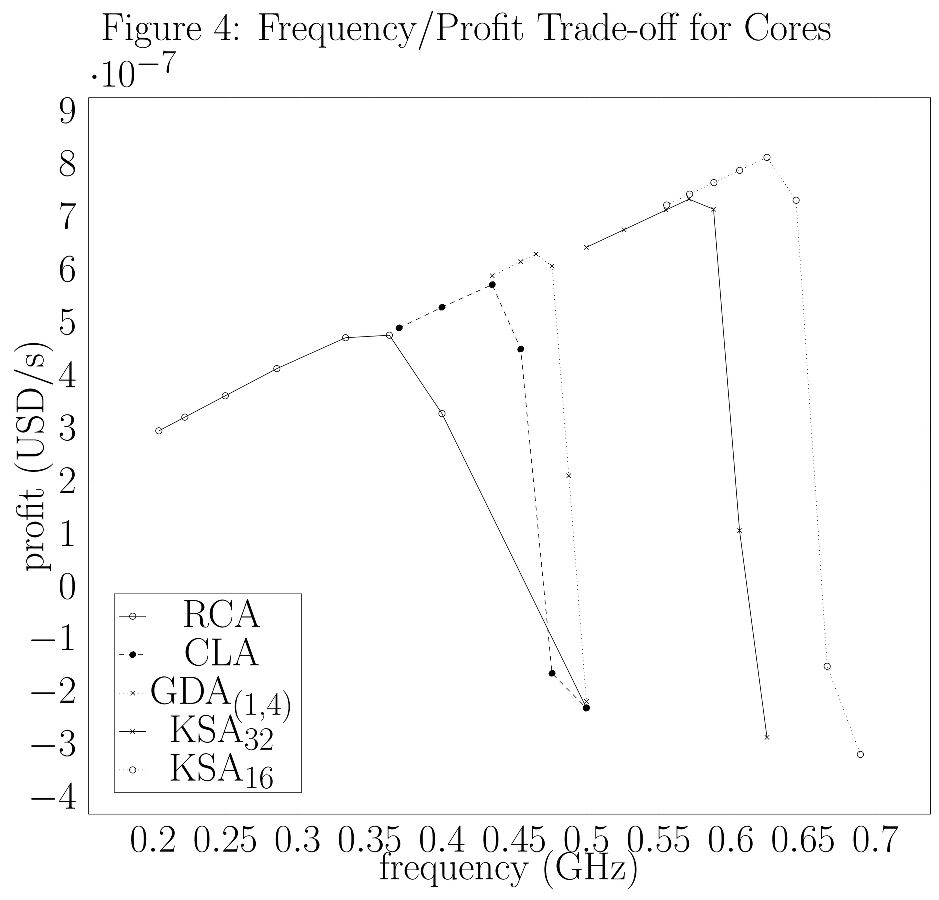


Figure 4: Frequency/Profit Trade-off for Cores



Conclusions

MINING is a particularly good candidate for approximation because hashes are computed independently and in parallel, mitigating the effect of errors, and a built-in verification system detects any false positives. Furthermore, we have identified adders as beneficial choices for approximation in hashing cores in a mining ASIC. However, not all approximate adders yield increases in profit. Profits are maximized by adders that minimize delay at the expense of area, and approximate adders should be chosen accordingly. Moreover, profits may be improved by operating the hashing cores at Better Than Worst-Case (BTWC) operating points, past their nominal frequencies.