24.12.17 Eligibility Check Benchmark

Xun Zhang Wuyun Siqin Bingsheng Zhang
Zhejiang University, CHN
22221024@zju.edu.cn 3210101763@zju.edu.cn bingsheng@zju.edu.cn

December 2024

1 Circuit Implementation

1.1 Mapping and Merkle Tree

We implemented the optimization in the paper:

Fortunately, we don't actually need to evaluate ϕ in the proof: we can replace stake in the tree with $\phi(stake)$ and proceed with the comparison directly

So we replace the *stake* by $\phi(stake)$ in our merkle tree construction.

And we use the mapping function in the Mithril open source code:

```rust
/// Dense mapping function indexed by the index to be evaluated.
/// We hash the signature to produce a 64 bytes integer.
/// The return value of this function refers to
/// `ev = H("map" || msg || index || σ) <- MSP.Eval(msg,index,σ)` given in paper.
pub fn eval(&self, msg: &[u8], index: Index) -> [u8; 64] {
    let hasher = Blake2b512::new()
        .chain_update(b"map")
        .chain_update(msg)
        .chain_update(index.to_le_bytes())
        .chain_update(self.to_bytes())
        .finalize();

    let mut output = [0u8; 64];
    output.copy_from_slice(hasher.as_slice());

    output
}
```

The corresponding description in the paper as following:

For the concatenation proof system PSC in Section 5.2 we use a random oracle $H : \{0,1\}^* \to \mathbb{Z}_p$ for the mapping as: $M^{\sigma}_{mag,\mathsf{index}}(\sigma) = H(\text{"}map\text{"}||msg||\mathsf{index}||\sigma)$

We instantiate the hash function as Poseidon function, which is same as the merkle tree.

1.2 Comparison

For the comparison between $\phi(stake)$ and mapping result, we use a compare gate in halo2-lib.

Note that in our circuit, the value of $\phi(stake)$ is a big integer in the prime filed, and we compare it with a hash ouput(also a big integer). This process is different from the Mithril code because they do not actually compute a $\phi(stake)$.

Below is our code implementation:

```
let less: AssignedValue<F> = big_less_than::assign(
    base_chip.range(),
    ctx,
    a: ev.clone(),
    b: phi_stake.clone(),
    base_chip.limb_bits,
    limb_base: base_chip.limb_bases[1],
);

// println!("less:{:?}", less.value());

let equal: AssignedValue<F> = big_is_equal::assign(
    base_chip.gate(),
    ctx,
    a: ev,
    b: phi_stake,
);

// println!("equal:{:?}", equal.value());

let result: AssignedValue<F> = base_chip.gate().or(ctx, a: less, b: equal);
```

Due to the complex implementation mechanism of halo2-lib's integer backend, this comparison circuit may have lower efficiency.

2 Benchmark

We benchmarked the performance of eligibility check circuit, including mapping function and comparison function. This is just a linear time task, see reults below:

num	**128**	**256**	**512**	**1024**	**2048**
Proving/s	12.8556	16.3610	22.8908	36.4483	63.0147
proof/bit	1344	1920	2848	4800	8832
verifying/ms	7.2702	7.5913	6.9618	8.8133	10.3663

Table 1: eligibility check benchmark