1 Introduction

Our main optimizations of PoSW Marlin includes the Following:

1 The Concurrent Implementation

2 GPU fft optimizations

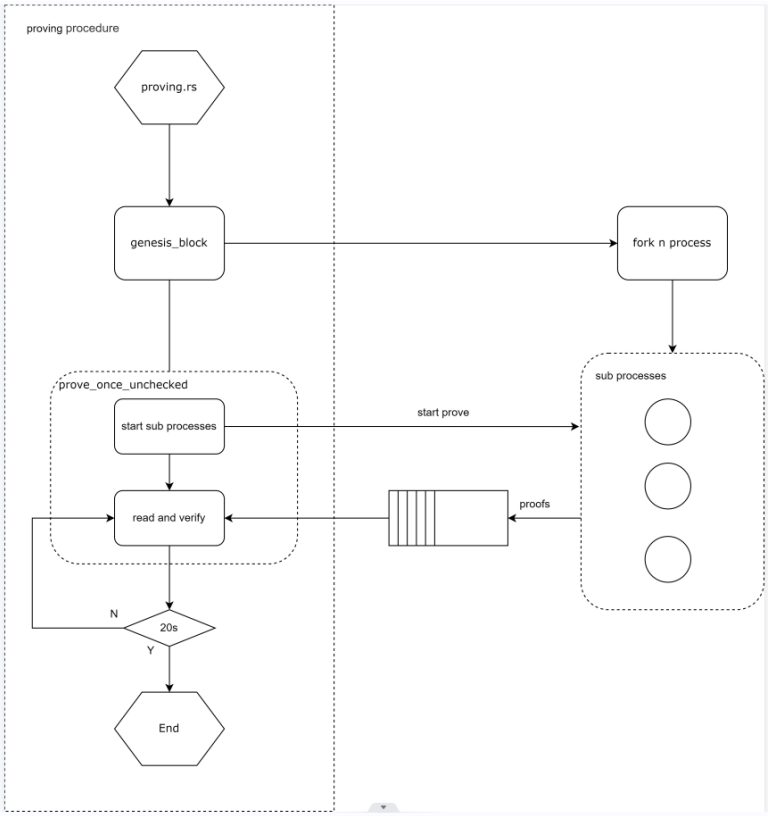
3 GPU msm optimizations

4 AHP Optimizations

5 CPU Optimzations

2 The Concurrent Implementation

In order to make use of cpu and gpu hardware resources as much as possible, we adapt **the combination of multi-process and multi-thread** to carry out the concurrent computation of PoSW proofs. To ensure the uniformity of the zprize test hardness code, we move our changes into the original prove function to distribute prove requests among multiple processes.



3 GPU fft Optimizations

We implement the GPU code of fft in ./algorithms/src/fft, the algorithm is based on the butterfly implementation of fft (ifft,cosetfft, cosetifft). We have several optimazations compared to the original implementation.

- We use **Precomputations** to store omegas of the roots of unit to decrease the computations of the fft kernel function
- We use **Buffer pool** to greatly decrease the frequencies of creating and destroying buffers in GPU
- We use the loop combination(first and last) to **decrease the syncthread() times** in fft kernel function, which is very effective
- We implement to **compute multiple polynomials of the same degree simultaneously**,

we also implement the corresponding kernel functions, and greatly improve the performance by using more threads

4 GPU msm optimizations

We implement sppark code of msm in ./extern/sppark with the following optimizations

- We only need to run the init function **ars_multi_scalar_mult_init function** once per GPU
- We adjust the parameters **WITS and NTHREADS** to get the best perfermance
- We **cache the msm bases in GPU** to decrease the data transfer from CPU to GPU

5 AHP optimizations

The main optimizations of AHP Rounds functions are:

- we **reorginize and expand the round functions**(First round,second round, Third round), so that we can **compute the ffts of multiple polynomials simultaneously**
- we **reorganize and expand the second round and third round functions**, where different kinds of ffts are combined into a single gpu function to **decrease the CPU and CPU data trasnfer**.

6 CPU Optimizations

We optimize the follwoing CPU functions:

- init_prover: We **cache the base data** to decrease computations of the three kinds of scalar_multiplication functions
- divide_by_vanishing_polynomials: We use **the special property** of the vanishing polynomial to speed the computations
- constuct_linear_combinations: We use **parallel** to speed the function
- Calculate_t: We use **special numbers** to speed the computations
- batch_eval_unnormalized_bivariate_lagrange_poly_with_diff_inputs_over_domain: we use **fft precomputations** to simplify the computations