Project proposal for Cpr E 583

Blake Vermeer, Piriya Kris Hall, and Rohit Zambre

***Abstract*—The abstract goes here.**

***Index Terms*—IEEEtran, journal, LaTeX, paper, template.**

I. Introduction

CRYPTOCURRENCIES have been around for some time, though it did not enter popular thought until the creation of Bitcoin, with the paper outlining Bitcoin being published in 2008. Since then many new and interesting cryptocurrencies have been created following Bitcoin's example. Cryptocurrencies such as Litecoin, Dogecoin, Vertcoin, and Primecoin, are a subset of what are called alt-coins, or alternative coins, so called for being an alternative to Bitcoin. In regards to this project proposal, the alt-coin Primecoin is of interest. In order to explain the interest given to Primecoin, it is imperative that the reader understand how cryptocurrencies work, at least at the conceptual level.

A. High level explanation of cryptocurrencies

Cryptocurrencies are a form of digital currency, that is similar to physical currency. Unlike physical currency, however, cryptocurrencies use programs, called wallets, as a means for individuals to transfer cryptocurrencies to each other. Each wallet has an address that is used for the sending and receiving of funds. Another thing that these wallets have is what is called the block chain. The block chain holds information of all the transactions using the specified cryptocurrency. For example, a wallet for Bitcoin will hold an address that can only receive Bitcoins, and it will contain the Bitcoin block chain. Similarly, a wallet for Primecoin or Vertcoin would have addresses specific to each coin, and contain their respective block chains.

Suppose an individual A wants to transfer an arbitrary amount of currency to another user B of the same coin, then A would generate a message that includes their wallet address, B's wallet address, the amount to send, and a few extra pieces of information. B checks the block chain to see if A has that amount to spend, if A does, then B would accept the transaction. A then appends this message to the end of the block chain. Once the message is appended, the updated block chain is broadcasted from A's wallet through the peer-to-peer network of wallets. Once the updated block chain is broadcasted, the network then attempts to verify the new transaction through what is called mining.

The details of the process of mining is potentially different between coins, but one aspect of mining is present in all cryptocurrencies and that is called proof-of-work. Proof-of-work is a way for the network to guarantee that the verification of transactions are not haphazard and that there was some effort put into the verification. Generally, the proof of work is an algorithm chosen by the developers of the coin with the property that the algorithm is hard or computationally intensive to perform, but very easy to verify.

This project is interested in the proof of work aspect of Primecoin. Where most cryptocurrencies define a specific algorithm, Primecoin defines its proof-of-work to be a result. Primecoin defines its proof of work to be a sequence of prime numbers, more specifically, Cunningham chains of the first or second kinds, and Bi-twin chains.[4] Due to such a proof-of-work, and its definition, there has been very little success at hardware acceleration of such a proof-of-work. The seemingly lack of success at hardware acceleration of such a proof-of-work is due in part to the lack of a defined algorithm to accelerate, along with the fact that verification of prime numbers is quite a hard problem in itself.

With such a problem in mind, the goal of this project is to attempt to perform hardware acceleration on the process of the generation of prime number candidates and the validation of these candidates in the hopes of generating a Cunningham chain of the first kind.

II. Core Problems

After researching the topic of prime number generation and verification, we have identified three main core problems that need to be solved:

A. Generation of Cunningham Chains

A Cunningham chain, named after the mathematician A.J.C. Cunningham and sometimes called a chain of nearly doubled primes, is a certain sequence of prime numbers such that the following property is true:

$$a_{i+1} = (2^i)a_i \pm (2^i - 1)$$

Where a_i is a prime number seed. Furthermore, Cunningham chains can be categorized as being of the first or second kind. Cunningham chains of the first kind are described as follows:

$$a_{i+1} = (2^i)a_i + (2^i - 1)$$

Again, a_i is the prime number seed for the chain. Cunningham chains of the second kind are described as follows:

$$a_{i+1} = (2^i)a_i - (2^i - 1)$$

Cunningham chains of the first kind generate what are called Sophie Germain primes, which is defined as:

A prime number p is a Sophie Germain prime if 2p + 1 is also prime.

The value 2p + 1 that is generated from a Sophie Germain prime is called a safe prime. A Cunningham chain is called **complete** if the chain can no longer be extended. Using a Cunningham chain of the first kind, candidates can be quickly generated with very little computational overhead. This is due to the fact that each a_{i+1} is effectively $2a_i + 1$, or $2a_i - 1$.

B. Generation of Prime Numbers

The problem of generating prime numbers ultimately boils down to determining if a given number has any factors apart from 1 and the number itself. Over the years, multiple factorisation algorithms have been developed to tackle this problem. Alternatively, sieve techniques don't require complex mathematics and are used to find prime numbers within a certain range. Since an immense amount of parallelization can be achieved with a seive-based technique, they are ideal for hardware implementation.

1) The sieve of Eratosthenes: The sieve of Eratosthenes is an ancient and simple algorithm to find all prime numbers up to any given limit. The following describes the algorithm:

2) Related work: Using dynamic reconfigurable logic M. Saleeba and R. Pose from Monash University in Australia implemented a sieve-based algorithm to generate prime numbers using dynamically reconfigurable logic. This project was primarily implemented on the DRAMA research tool to demonstrate the tool's powerful features including runtime reconfigurability. In the algorithm of the implementation, prime numbers are essentially generated by checking each successive integer for divisibility by each prime number less than equal to its square root. Any integer with no primes is added to the list of primes. A re-sizeable logic array is used to append units that check divisibility of the number-in-test by prime factors.

The algorithm consists of two stages: candidate generation and candidate verification. In the candidate generation stage, algorithm counts upwards through the integers and tests the primality of a new integer on each clock cycle. At a given instance, primality is checked against prime factors that have been discovered by the algorithm thus far. With the detection of every new prime factor, a new unit that tests for divisibility by the new prime factor is added to the logic array during run time. The prime factors in the logic array are termed as "low-value primes." When a prime number candidate is generated, it enters the candidate verfication stage in which the candidate is tested against groups of "high-value primes." The hardware for each of the stages has been described in the paper.

For RSA encryption/decryption C. Torng and Y. Lee from Cornell University implemented a prime nunmber generator and RSA encrypter/decrypter using the Altera DE2 FPGA. The prime number generation was based on the Miller-Rabin primality test to search for prime numbers. They fed in pseudo-randomly generated odd numbers to the test to "generate" prime numbers. We have described the Miller-Rabin test in the following section.

C. Verification of Generated Prime Numbers

Since the prime number generators work on the principal of generating numbers that has a reasonably high probability of being prime, the numbers must be verified to insure that they are actually prime. This is a very resource intensive problem since it requires testing if the potential prime number number is evenly divisible by a set of numbers. Since it is prohibitively expensive to test if large prime numbers are definitively prime, probabilistic methods have been developed that will determine if a number is prime with a high probability. One of these probability methods of determining if a number is prime is called the Rabin-Miller primality test.

1) Rabin-Miller Primality Test: The Rabin-Miller primality test is a probabilistic method of determining if a number is prime. The algorithm is based around the idea of testing if a number in question is divisible by a subset of small primes. If it is not divisible by any of the small primes in the set then it is a prime number with a probability that is determined by the number of small primes in the set. The larger the set of small prime numbers is allows for the primality of the number in question to be determined with greater confidence. Since the probability of correctly identifying a prime number is directly related to the number of small primes tested again the test number, the accuracy of the Rabin-Miller primality test can be easily customized. [1]

Now to dive into the theory of the Rabin-Miller primality test. The Rabin-Miller theory is based on the contrapositive of Fermat's little theorem which states that for a prime number n:

$$a^{n-1} \equiv 1 \pmod{n}$$

Therefore, the Rabin-Miller primality test can show that a number is not positive of for a number n if we can find a number a such that:

$$a^d \not\equiv 1 \pmod{n}$$

and

$$a^{2^r d} \not\equiv -1 \pmod{n}$$

for all $0 \leq r \leq s - 1$, then n is not prime. [3]

The Rabin-Miller Primality test is a good test to implement on hardware since the test can be parallelized by testing many small primes again the number in question concurrently and the math operations involved would be fairly expense in terms of cycles if implemented in software because of its sequential nature. Since we are attempting to implement this design in hardware let's first examine how to do the modulus operation in hardware.

One way to implement a modulus operation in hardware is to use the Montgomery algorithm to perform a modulo-multiply.

III. Conclusion

The conclusion goes here.

References

[1] Cheung, R., Brown, A., Luk, W., Cheung, P.: A scalable hardware architecture for prime number validation. In: IEEE Int. Conf. on Field-Programmable Technology, pp. 177-184 (2004)

[2] A. Daly and W. Mamane. Efficient architectures for implementing montgomery modular multiplication and RSA modular exponentiation on reconfigurable logic. In *Tenth ACM International Symposium on Field-Programmable Gate Arrays* pages 40-49, February 2002.

[3] Miller-Rabin Primality Test. On: *Wikipedia*, 27 Oct. 2014. <http://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test>

[4] S. King. (2007, July 7). Primecoin: Cryptocurrency with Prime Number Proof-of-Work(*1st ed.*)[Online] Available:http://primecoin.io/bin/primecoin-paper.pdf

[5]

[6]

[7]

[8]

[9]

[10]