SoK: Hardware Accelerated Modular Multiplication for ZKProofs

Erdinc Ozturk[1], Justin Drake[2], Sean Gulley[3], Simon Peffers[3], Kelly Olson[3]

Abstract
This talk focuses on hardware acceleration of modular multiplication and its application to the blockchain ecosystem. While there have been many exciting new developments in computer science and cryptography over the past decade, less effort has been spent on how to make these new techniques computationally practical. By increasing the performance of modular multiplication, novel cryptographic techniques like VDFs, SNARKs, and Accumulators can become feasible, enabling blockchain protocols to reduce their compute, storage, and networking requirements. This talk will discuss techniques for optimization across CPU, GPU, FPGA, and ASIC architectures, as well as both algorithmic and numerical representation techniques that enable improved performance. We will provide concrete data from recent optimization work on different architectures for RSA-based cryptography, and discuss the viability of this approach to improve blockchain security and scalability. The talk will conclude with a discussion of developments that have $<1\%$ the latency of a CPU system, and over 1000 times the throughput of a CPU core.

Background and Motivation

The goal of this talk is to discuss the methods and benefits of hardware acceleration for the blockchain industry. While it is true that in order to scale blockchain protocols computer science and algorithmic innovations are necessary, scaling blockchain protocols also consists of making trade-offs between compute, storage, and networking at the protocol and application layers. One simple example is the use of public key recovery in Bitcoin and Ethereum transactions. Rather than providing you public key within your transaction, it is possible to provide only a signed message from which the validator can 'recover' your public key. This approach reduces the networking requirements at the expense of increased computation for the validator.

Another approach to improve scaling is to move the compute and storage requirements away from the network/validator side to the client side. This can be done through the generation of cryptographic proofs that can be quickly verified. One of the 'proofs' most widely known in the blockchain industry is 'proof-of-work'. Proofs of work enable a miner to demonstrate that they have completed some expected amount of work manipulating the blockcheader through modifications to block header fields like nonce, merkle root, and timestamp. Once the proof is submitted to the network, nodes can quickly verify that the proof is valid at orders of magnitude less cost and time as was required to generate the proof. Proof of work is the strongest example of hardware acceleration in the generation of cryptographic proofs. The speed at which proofs for a given difficulty can be created with current hardware is several thousand times faster than a single CPU core at approximately a 250,000-fold reduction in energy cost. To date, over \$ 4 billion dollars has been spent on hardware due to the incentive

[1] Sabanci University
[2] Ethereum Foundation
[3] Supranational

structure of current blockchain protocols.

Indeed as we look more broadly at the internet ecosystem, we see the use of custom hardware to scale internet services and reduce the total cost of operations. Such hardware is targeted at use cases such as networking, graphics, transcoding, sound, and AI. More and more, large internet service providers are leveraging programmable and custom hardware such as FPGAs and ASICs for algorithms and cryptographic protocols like compression and transport-layer security (TLS). Even modern general purpose CPUs now have dedicated instructions and hardware accelerators for hashing (SHA) and encryption (AES).

We believe that hardware acceleration will be an essential component to enabling blockchains protocols to achieve the scalability required for practical use. Today blockchain protocols are limited by their 'on-chain' compute and storage capacities, as well as by their networking bandwidth. We believe that through the improvement of 'off-chain' hardware and client-side proof generation, we can drastically improve the performance of blockchain networks. This hardware acceleration can take two forms: 1) Better utilization of existing resources (e.g CPU, GPU) through improved software and algorithmic implementations and 2) The development of custom hardware (e.g. FPGA and ASIC) and novel algorithms suited for their hardware configuration and available resources. In this talk we will discuss the performance gains from each approach as applied to RSA group operations.

Novel Cryptography from RSA Groups
Over the past two years there have been a number of novel cryptographic techniques that are built on 'groups-of-unknown-order'. The two most discussed of these groups are RSA groups, where the factorization of the modulus is unknown, and class groups. Throughout this talk we will discuss acceleration of the underlying arithmetic for operations in RSA groups in particular. We will cover CPU, FPGA, and ASIC architectures and their relative performance. However before beginning, we would like to discuss three interesting applications that become practical through the acceleration of these group operations:

Verifiable Delay Functions (VDF) - VDFs provide a proof that a certain amount of sequential work has been completed. You can think about it as a sequential 'proof-of-work' that can not be parallelized. VDFs provide a succinct proof that a very large amount of sequential work has been completed and these proofs require less bandwidth than similar interactive protocols for sequential-proof-of-work. VDFs can be used to create unbiased, unpredictable, and unstoppable randomness that can be used in blockchain protocols for things like committee selection or application level randomness.
Accumulators - A cryptographic accumulator is a primitive that produces a succinct commitment to a set of elements, as well as provides short membership and non-membership proofs for individual elements in the set. Unlike Merkle trees, RSA accumulators enable a constant size membership proof which reduces bandwidth requirements of the network. However, this benefit likely comes at the cost of increased computation and storage client-side.
SNARKs - One of the most promising approaches to blockchain scalability is through SNARKs. Recently, a new polynomial commitment scheme that can be used in SNARKs

was released that can be built off of RSA groups. SNARKs enables a client to succinctly prove a statement is true (e.g. I am spending less coins than I own). They can be used to improve the scalability of blockchain protocols through transaction aggregation and they can add functionality such as privacy. SNARKs reduce storage and bandwidth requirements on the network at the cost of increased computation client-side.

Problem Statement: Group Operations

Now that we understand the potential of accelerating RSA operations, we'd like to briefly explain the underlying computations that needs to be accelerated to make the above techniques practical. The foundation of cryptosystems and cryptographic techniques today are predominantly built atop specific groups/curves and the operations on/over those groups. As a simple example, ECDSA uses elliptic curves and group operations such as point multiplication to enable digital signatures and their verification. In the techniques we are discussing today we are focusing on RSA groups, one of the most widely used systems today. RSA group operations today are predominantly made up of modular exponentiation of large integers. Arithmetically this often boils down to the multiplication and reduction of large integers. In some cases we want our optimization efforts to minimize the latency of a single operation (as is the case in VDFs), while in others we are focused on maximizing the throughput for a given architecture (as is the case with DARKs). Across this latency and throughput spectrum there are a variety of tradeoffs that can be made that will impact the cost, power, and performance of the resulting hardware.
With the opportunity and problem statement in mind, we will discuss algorithmic and hardware approaches to accelerating these operations.

Algorithms

One of the first places to look when accelerating cryptography is to evaluate various algorithmic approaches to the problem. RSA is a family of algorithms driven by low level primitives such as modular multiplication. To begin it is best to understand what group operations are consuming the most compute resources/time, and then investigate the low-level algorithms that are used to implement the primitives of higher-level cryptographic protocols. It is important to understand that every algorithm represents a unique set of tradeoffs between latency, throughput, power, and area. When selecting an algorithm it is imperative that you understand both the desired goal as well as the underlying hardware that is available for you to use. Algorithmic decisions can not be made in isolation and need to consider the target implementation.

While investigating the the RSA group operations, we discovered that modular multiplication and modular squaring were consuming the vast majority of the compute cycles. In order to accelerate the above mentioned cryptographic constructions, we began to evaluate various algorithms and representations that could be used to accelerate these operations. Some such algorithms include schoolbook multiplication, Montgomery multiplication, and Barrett reduction, while representations include polynomial form and residue numeral system. While we won't spend a lot of time in the talk discussing the various algorithmic approaches, we will provide a short summary of these algorithms as given at the 2019 Stanford VDF Day (https://www.youtube.com/watch?v=ITf4Wt2YgDE).

When it comes to representations, many people will understand that there are a variety of ways to represent a number. For example, the number '100' can be represented as 100, 99+1, 50+50, 25∗ 4, etc. In the talk we will briefly discuss representations such as Montgomery and polynomial and how some approaches take advantage of these representations to improve performance.

Hardware Implementation

CPU Implementation

The first step of hardware optimization is to perform a simple implementation of the operation that you would like to accelerate. For the purposes of this talk we will be focusing on improving the latency and throughput of a modular squaring operation. To begin we implemented a 2048-bit square and reduce in Python and achieved a time of ∼ **5000 nanoseconds (ns)**. Python is often a good place to begin when implementing a new algorithm as it teaches us the 'complexity' of the algorithm. However, since large integer modular squaring is already commonly done, we quickly moved on to more performant libraries. One library that is particularly suited to this type of problem is GMP. GMP is a well optimized library for a variety of arithmetic operations. Through the use of GMP we were able to achieve ∼ 1200ns per operation. In the talk we will discuss how GMP is able to get improved performance through the use of special CPU features such as SIMD and carry chain flags (e.g. mulx/adcx/adox). We will briefly discuss the various SIMD options that are widely available or soon-to-be-available (AVX2 and AVX512) as well as their benefits and drawbacks. Finally, we will demonstrate how it is often possible to accelerate your specific function through hand written assembly, where we were able to achieve a modular squaring in ∼ **1000ns**. This section will conclude with a graph of latency and throughput for the various CPU implementations.

GPU Implementation

The next part of the talk will focus on GPU implementations. GPUs offer increased parallelism over CPUs while still operating at relatively high frequencies (e.g. 1GHz+). GPUs can be programmed in languages like OpenCL or CUDA, or compilers can be used that convert C to these GPU languages. For the purposes of our acceleration, which was predominantly latency focused, GPUs were not a good hardware candidate. While GPUs are highly parallel, affordable, and easy to purchase, we will discuss some of their drawbacks as well such as the round-trip latency to move off the CPU, as well their lower frequencies. This section of the talk will conclude with some naive latency and throughput numbers on a modern GPU.

FPGA Overview

In this part of the talk we will discuss what an FPGA is, and also discuss its hardware characteristics such as size, resources, frequency, and cost. We will provide a comparison of FPGA vs. CPU resources and discuss circumstances where an FPGA may be the appropriate hardware. We will discuss why different algorithms may be more suited for an FPGA based off of the available resources, lookup tables, and the amount of work per clock that can be done. We will also discuss the drawbacks of FPGAs including their cost, maximum frequency, and complexity of programming.

FPGA Implementation

In this part of the talk we will discuss the novel algorithm that was developed to achieve a low latency modular squaring for VDFs. We will discuss why the algorithm is appropriate given the goal (low latency) and the resources of the FPGA. The algorithm we will be discussing is the 'Ozturk' design, developed by the co-author of this paper and available here: https://eprint.iacr.org/2019/826. We will briefly discuss the polynomial representation and the use of LUTs used in the design that delivered a 20x speedup **($\sim$ 70ns per modular squaring)** over our CPU implementation. The implementation of this design was done in System Verilog and designed to run in the AWS FPGA Cloud (F1). We will provide information about the resource utilization of the algorithm (DSPs, BRAMs/URAMs, LUTs) and the frequency it is able to achieve. We will also discuss how this design was used to solve a 20-year-old outstanding cryptographic puzzle (https://www.wired.com/story/a-programmer-solved-a-20-year-old-forgotten-crypto-puzzle/) by Ron Rivest in only two months. Finally we will discuss the FPGA design competitions that we have held since the initial design where the speed of the design was improved a further two fold.

Implementation Code

Code will be made available at https://github.com/supranational/

Future Plans - ASIC Implementation and Manufacturing

This part of the talk will discuss our future plans for hardware acceleration of these RSA operations. We will briefly discuss what an ASIC is, and why it has better performance than an FPGA (e.g. resources, frequency, cost, etc.). After this we will discuss the determinants of ASIC performance such as algorithm, implementation, process technology, and further optimizations like custom cells. We will then provide performance estimates for a modern ASIC based off synthesis and place-and-route of the design in EDA tools **($\sim$ 5ns)**. We will conclude the presentation with a summary table that shows the performance results and estimates across CPU, GPU, FPGA, and ASIC. These results will include data such as area, frequency, power, latency, throughput, and cost as applicable. The conclusion of the talk will briefly discuss the intended next steps for our hardware acceleration work. We will provide information about where our implementations can be found as all of this work is intended to be open source. We will also briefly mention the other areas that interested contributors can get involved as we move towards production (e.g. firmware, industrial design, software, etc.)