Goldilocks NTT Trick

Tomer Solberg
tomer@ingonyama.com

Abstract

Polynomial arithmetic is at the heart of modern Zero Knowledge Proving (ZKP) systems. The Number Theoretic Transform (NTT) is a crucial tool in facilitating efficient computational complexity over large polynomials encountered in ZKPs. NTTs are dominated by the number of field multiplications.

In this short note we focus on the specific case of NTT in the 64 bit Goldilocks field. Following the observation that 2^{48} is a root of unity, expressing the first few roots of unity in terms of powers of 2, allows modular multiplications to be replaced with simple bit-shift operations, resulting in a significant cost saving for NTT.
Update: This trick is well known in the FHE (Fully Homomorphic Encryption community) [1, 2, 3, 4]

1 Introduction

NTT is a core math primitive found in many of today's deployed ZKPs. For example, in STARK arithmetization, the trace produced by the prover is interpolated into a suitable univariate polynomial form for building a polynomial Interactive Oracle Proof (IOP). The transformation between the coefficients form and evaluation form of the polynomial is done efficiently via an NTT.

In any ZKP, as the size of the vector becomes larger $n > 2^{20}$ the quasi-linear behavior of the NTT is expected to dominate [5]. Noting this potential problem as the sizes of circuits grow, the ZKP industry has taken various directions to solve this computational bottleneck.

One direction is to avoid NTTs entirely: for instance, the approach of Hyperplonk [6], which proposes multivariate interpolations on a Boolean hypercube instead of univariate interpolations on roots of unity. While this direction is interesting, it replaces the NTT with another computation that dominates its sumcheck protocol [7] involving MultiLinear Extension (MLE).

An alternative direction involves special primes which possess unique algebraic properties. One such example is the Goldilocks prime introduced by Mike Hamburg [8] based on the prime modulus:

$$q = \phi^2 - \phi - 1 \tag{1}$$

This prime was originally introduced in the context of elliptic curve Ed446-Goldilocks with $\phi = 2^{224}$, since $224 = 32 \cdot 7 = 28 \cdot 8 = 56 \cdot 4$ and thus it supports fast arithmetic in radix 2^{28} or radix 2^{32} suited for 32 bit machines and radix 2^{56} suited for 64 bit machines. The main advantage of the Goldilocks prime is an ultra-fast Karatsuba multiplication. If we

represent field elements as $a + b\phi$ for a, b of size 32 bit, we get:

$$\begin{aligned}(a + b\phi) \cdot (c + d\phi) &= ac + (ad + bc)\phi + bd\phi^2 \\ &= (ac + bd) + ((a + b)(c + d) - ac)\phi \mod q \end{aligned} \tag{2}$$

The final step of the multiplication is taking the modulo q reduction of the result. This is also extremely computationally cheap in the context of Goldilocks primes, given that $\phi = 2^k$ is some power of 2. Note the following

$$\begin{aligned} 2^{2k} &\equiv 2^k - 1 \mod q \\ 2^{3k} &\equiv -1 \mod q \end{aligned} \tag{3}$$

where the second property follows from the factorization

$$\left(2^{3k} + 1\right) = \left(2^k + 1\right)\left(2^{2k} - 2^k + 1\right) \equiv 0 \mod q. \tag{4}$$

Then, if a number x which is represented by 4k bits (say, a product of two $2k$ bit numbers from the field), we can easily reduce it modulo q, by splitting it into the $2k$ least significant bits x_{LSB}, the k intermediary significant bits x_{ISB}, and the k most significant bits x_{MSB}, and reduce it in the following manner

$$\begin{aligned} x &\equiv x_{LSB} + 2^{2k}x_{ISB} + 2^{3k}x_{MSB} \mod q \\ &\equiv x_{LSB} + (2^k - 1)x_{ISB} - x_{MSB} \mod q \end{aligned} \tag{5}$$

The final step is to add or subtract q if an underflow or overflow occurs. Therefore, the complexity of the modular multiplication in the Goldilocks case is equivalent to the standard Karatsuba integer multiplication, plus a small number of extra additions.

For 64 bits, the Goldilocks prime [9]

$$p = 2^{64} - 2^{32} + 1 \tag{6}$$

has seen extensive usage in the ZKP industry already. Several existing ZKP systems such as Polygon ZKEVM [10], Polygon Miden VM [11], and Plonky2 [12] have STARK-based prover components that exploit efficient computations modulo p.

ethSTARK [13] uses a slightly different prime $p' = 2^{61} + 20 \cdot 2^{32} + 1$ and the NTT dominates about 80% of the prover time. If this prime were replaced with the Goldilocks prime, the net performance of ethSTARK would see a significant performance boost, due to size and efficient modular reduction with the Goldilocks field. While much of the tricks discussed above are known in the industry already, in this article we discuss another less known fact (in the ZK community) which makes higher radix NTTs in the Goldilocks field faster than ever before.

Update: We were informed by Michiel Van Beirendonck that @CosicBe used this trick to get the 3rd place in the Zprize event. The repository is not open sourced yet.

For the rest of this note and w.l.o.g, let us focus on the 64-bit Goldilocks field mentioned above. Namely, $\mathbb{F}_p$ with $p = 2^{64} - 2^{32} + 1$.

2 Goldilocks Imaginary Unit and Its Roots

A useful property of $\mathbb{F}_p$ is that the imaginary unit $j = \sqrt{-1}$ is a power of 2. This is true for any field $\mathbb{F}_q$ with $q = 2^{2k} - 2^k + 1$ where k is even, where

$$2^{3k} \equiv -1 \mod q \tag{7}$$

implies $j = \sqrt{-1} = 2^{3k/2}$, and specifically for $\mathbb{F}_p$ we get $j = 2^{48}$. If $k/2$ can be divided further by factors of 2, other Roots of Unity (RoU) can also be expressed in powers of 2. Defining the N-th root of unity of the Goldilocks field $\mathbb{F}_p$ as ω_N, we can write

$$\begin{aligned} \omega_N^{N/4} &= j = 2^{48} \\ \omega_N^{N/8} &= 2^{24} \\ \omega_N^{N/16} &= 2^{12} \\ &\vdots \\ \omega_N^{N/64} &= 2^3. \end{aligned} \tag{8}$$

Expressing RoUs in terms of powers of two is useful in the context of NTT's, as multiplications can simply be replaced with bit shifts of the samples, followed by reduction modulo p, which in the Goldilocks case as we have seen is a matter of simple additions.

3 Efficient NTT with Goldilocks' Imaginary Unit

NTT computation is generally performed using the Cooley-Tukey (CT) algorithm, which is a recursive algorithm that typically uses basic building blocks referred to as radix-2^k blocks. Sticking to NTT lengths that are powers of 2 for simplicity, say $N = 2^\ell$, allows using recursive stages of radix-2^k blocks, assuming k divides ℓ. In this case, the number of stages is k/ℓ and each stage contains $2^{k-\ell}$ such radix-2^k blocks.

Without getting into exact details, the calculation of an NTT of size N requires multiplications by powers of the N-th RoU and additions. In the CT algorithm, this translates to using powers of the k-th RoU inside the radix-2^k blocks and powers of the N-th RoU outside the blocks. The multiplications outside the blocks are also known as multiplications by twiddle factors. The total number of multiplications required per each radix-2^k block with its associated twiddles is $k2^{k-1}$ of which $2^k - 1$ are the twiddle factors. Roughly speaking, the above can be used to calculate the ratio of twiddle factors for the overall length N NTT.

For example, a radix-4 block contains internally a single multiplication by $\omega_N^{N/4} = j$. This is a useful fact for computing FFTs, as multiplication by j of a complex number is computationally trivial. Similarly, using a radix-4 architecture for computing NTT over the Goldilocks field can save 1/4 of the multiplications, by replacing the internal multiplication with a shift and addition, as shown previously. This situation improves further by considering higher radices. Radix-8 blocks contain a total of 12 multiplications, of which 5 are internal multiplications by powers of $\omega_N^{N/8} = 2^{24}$. Replacing these reduces the number of multiplications by a factor of 7/12. See Diagram 1 for a visualization of a radix-8 NTT block. This can be further utilized all the way up to radix-64 as is shown in Table 1.

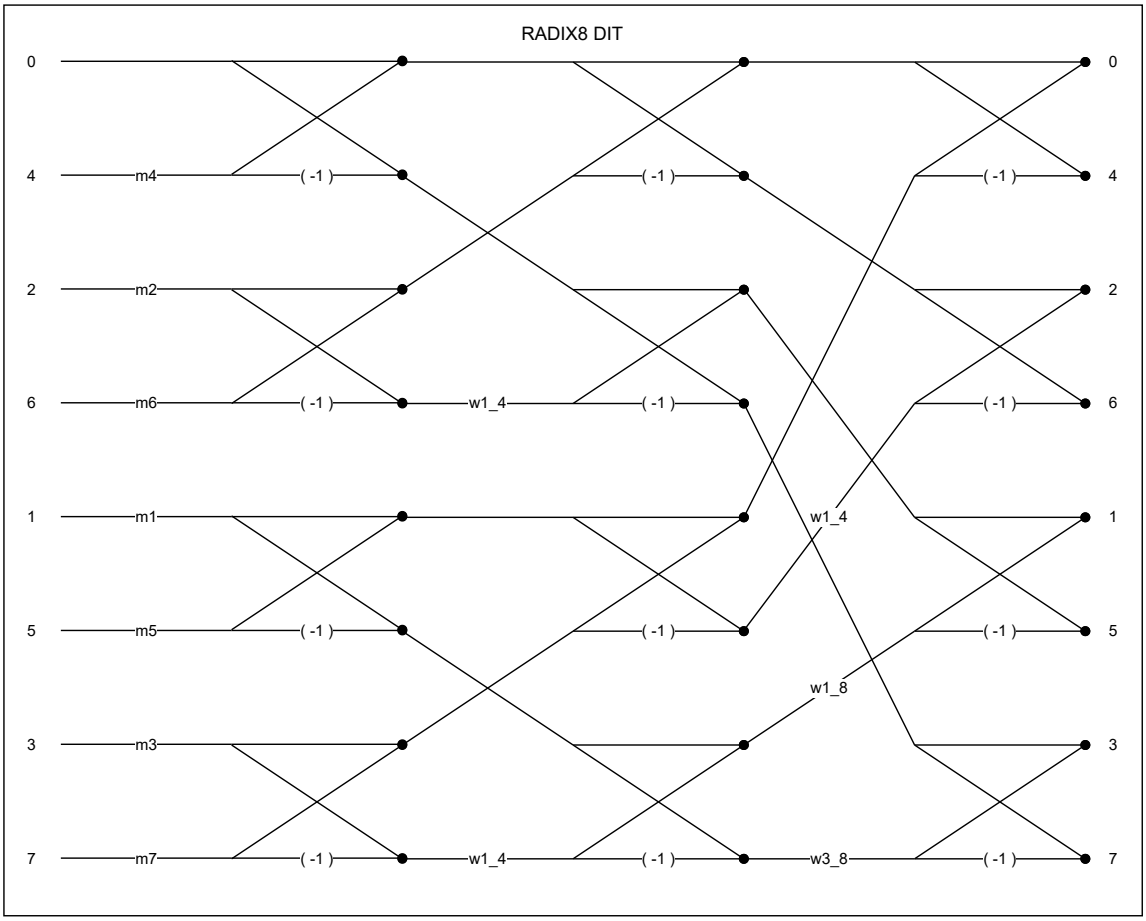

Figure 1: Radix-8 NTT Block

4 Acknowledgements

The author would like to thank Yuval Domb, Karthik Inbasekar and Omer Shlomovits for useful discussions. We would also like to thank Michiel Van Beirendonck for updating us regarding the usage of the NTT trick in Zprize.

References

[1] Wei Wang and Xinming Huang. Fpga implementation of a large-number multiplier for fully homomorphic encryption. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2589–2592, 2013.

[2] nuFHE. Polynomial multiplication. `https://nufhe.readthedocs.io/en/latest/implementation_details.html`.

[3] Niall Emmart and Charles Weems. High precision integer multiplication with a gpu. In *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pages 1781–1787, 2011.

Radix	Twiddle factors	Internal multiplications	Improvement factor
2	1	0	1
4	3	1	3/4
8	7	5	7/12
16	15	17	15/32
32	31	49	31/80
64	63	129	21/64

Table 1: Multiplication reduction factor of different NTT architectures

[4] nuFHE. nucypher. https://github.com/nucypher/nufhe.

[5] Dan Boneh. Zkp workshop 2022. https://youtu.be/6psLQv5Hf_I.

[6] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. Cryptology ePrint Archive, Paper 2022/1355, 2022. https://eprint.iacr.org/2022/1355.

[7] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, oct 1992.

[8] Mike Hamburg. Ed448-goldilocks, a new elliptic curve. Cryptology ePrint Archive, Paper 2015/625, 2015. https://eprint.iacr.org/2015/625.

[9] Thomas Pornin. Ecgfp5 : a sepcialized elliptic curve. https://github.com/pornin/ecgfp5/blob/main/doc/ecgfp5.pdf/.

[10] Polygon-Hermez. Polygon zkevm. https://github.com/0xpolygonhermez.

[11] Polygon-Miden VM. polygon miden. https://wiki.polygon.technology/docs/miden/intro/overview/.

[12] Polygon-Zero. Plonky2. https://github.com/mir-protocol/plonky2.

[13] StarkWare. ethstark documentation. Cryptology ePrint Archive, Paper 2021/582, 2021. https://eprint.iacr.org/2021/582.