Internship Overview

William Scarbro

September 1, 2020

1 Introduction

The mandate of this research group is to investigate optimizing Post-Quantum Cryptographic algorithms. After making a study of the candidates for the NIST Post-Quantum Cryptography standard we arrived at the Number Theoretic Transform (NTT) algorithm as an important subroutine to optimize. This is because many of the candidate standards (NTRU [1], BLISS [2], Ring-LWE [10]) perform multiplications in a polynomial ring, which is a principle application of the NTT algorithm. The NTT algorithm owes its efficiency to the application of the Fast Fourier Transform (FFT). There has already been quite a bit of research into how to speed up the NTT algorithm for applications in Post-Quantum Cryptography using the FFT algorithm [4, 7, 2, 9]. However, most of this research is focused on the radix-2 version of the FFT.

We believe we can increase the efficiency of the NTT algorithm by considering arbitrary radix applications of the FFT algorithm. A system called Spiral [3] has used this method to achieve high levels of efficiency for the Discrete Fourier Transform (DFT) which is a close cousin of the NTT algorithm. Spiral achieves faster running times for the DFT by considering the memory hierarchy, instruction set architecture, and parallelization possibilities of a target system to produce platform specific code.

The focus of my research has been to understand how to apply Spiral to NTT, while incorporating the optimizations of NTT made by [4, 7, 2, 9]. Mathematically, this means extending NTT optimizations from radix-2 to an arbitrary radix.

This paper is laid out as follows. Section 2 will introduce some mathematical preliminaries for readers not acquainted with the topic. Section 3 will apply these fundamentals to the Number Theoretic Transform and Section 4 will show how the Fast Fourier Transform speeds up this calculation. Section 5 deals with framing NTT within Spiral's MatSPL language. The only novel addition presented in this paper is in Section 6, where we show how to extend the ψ powers absorption method presented by Roy et al. [9] into radix-n. This extension is done within Spiral's MatSPL framework so it can be directly applied to a Spiral implementation of NTT. Section 7 presents such an implementation and discusses future work.

2 Polynomial Rings

Many proposed cryptosystems perform operations on elements in a polynomial ring. A polynomial ring is formed by extending a field by a variable. For example, we can start with the field $\mathbb{F}_p$ (where p is prime) and extend it by X to form $\mathbb{F}_p[X]$. This creates elements that look like

$$a_0 + a_1X + a_2X^2 + a_3X^3 + ...$$

Where each $a_i \in \mathbb{F}_p$.

We then define a quotient ring $\mathbb{F}_p[X]/(Y)$, where Y is an irreducible polynomial. Most of the cryptosystems use $Y = X^N + 1$ where N is a power of 2.

2.1 Convolutions [6]

The multiplication of two polynomials in $\mathbb{F}_p[X]$ can be viewed as a convolution between the vectors formed by their coefficients. The convolution of two vectors a and b of length N is written as

$$a \circledast b = c$$
$$c_i = \sum_{j=0}^{i} a_j b_{i-j} \tag{1}$$

Where i ranges from 0 to $2N - 1$

2.2 Wrapped Convolutions [6]

When we multiply polynomials in the quotient ring $\mathbb{F}_p/(Y)$ we must consider the equivalence created by $Y \equiv 0$. To facilitate computation, we will always apply this equivalence in order to reduce polynomials in an equivalence class to their representative with degree less than or equal to N. (This way they fit in the length N array we allocate for them).

A positive wrapped convolution is used when multiplying elements in the ring $\mathbb{F}_p/(X^N - 1)$ because the equivalence $X^N - 1 \equiv 0$ becomes $X^N \equiv 1$. As a result, the X^{N+i} powers reduce to X^i.

$$c_0 + ... + c_{N-1}X^{N-1} + c_N X^N + ... + c_{2N-1}X^{2N-1} \equiv$$
$$c_0 + ... + c_{N-1}X^{N-1} + c_N X^0 + ... + c_{2N-1}X^{N-1}$$

The positive wrapped c vector can be computed using

$$c_i = \sum_{j=0}^{i} a_j b_{i-j} + \sum_{j=i+1}^{N} a_j b_{n+i-j} \tag{2}$$

Notice we can also compute a positive wrapped convolution using equation (1) by extending j from 0 to N and evaluating the vertex indices modulo N. Equation (2) is doing this explicitly, without the modulo operation.

A negative wrapped convolution corresponds to the ring $\mathbb{F}_p/(X^N + 1)$. In this ring $X^N \equiv -1$ so X^{N+i} becomes $-X^i$. To compute a negatively wrapped c vector we can use the definition

$$c_i = \sum_{j=0}^{i} a_j b_{i-j} - \sum_{j=i+1}^{N} a_j b_{n+i-j} \tag{3}$$

3 Number Theoretic Transform (NTT) [8]

The NTT algorithm is similar to the Discrete Fourier Transform with the difference that NTT is computed over a finite field rather than the complex numbers. NTT is defined as follows:

Let $N \in \mathbb{Z}$ and $a \in \mathbb{Z}^N$ be inputs.
Find $P \in Z$ s.t. P is prime and $N|P-1$ and $a_i < P \; \forall a_i$
Find ω_N, the Nth root of unity in $\mathbb{Z}_P$.

$$\begin{aligned} NTT : \mathbb{Z}_P^N &\mapsto \mathbb{Z}_P^N \\ a &\mapsto A \\ A_k &= \sum_{j=0}^{N-1} a_j \omega_N^{jk} \end{aligned} \tag{4}$$

Notice this is equivalent to evaluating a polynomial formed by the coefficients of the input vector at values $\omega^0, \omega^1, ..., \omega^{N-1}$.

3.1 Inverse

Application of the inverse NTT takes a vector from the transformed and finds the original vector. It requires the additional calculation of the inverse of N and ω in $\mathbb{F}_P$ and is defined as follows:

$$\begin{aligned} NTT^{-1} : Z_P^N &\mapsto Z_P^N \\ A &\mapsto a \\ a_k &= N^{-1} \sum_{j=0}^{N-1} A_j \omega_N^{-jk} \end{aligned} \tag{5}$$

3.2 Convolution

We can use NTT to calculate a convolution because multiplying two vectors in the transform space pointwise is equivalent to taking the convolution product of the original vectors. This can be shown as follows:

For $a, b \in \mathbb{Z}_P^N$

$$\begin{aligned} NTT(a \circledast b)_k &= \sum_{j=0}^{N-1} (a \circledast b)_j \omega_N^{jk} \\ [1] \ldots &= \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} a_i b_{j-i} \omega_N^{jk} \\ [2] \ldots &= \sum_{i=0}^{N-1} a_i \sum_{j=0}^{N-1} b_{j-i} \omega_N^{jk} \\ [3] \ldots &= \sum_{i=0}^{N-1} a_i \omega_N^{ik} \sum_{j=0}^{N-1} b_{j-i} \omega_N^{(j-i)k} \\ [4] \ldots &= \sum_{i=0}^{N-1} a_i \omega_N^{ik} NTT(b)_k \\ [5] \ldots &= NTT(a)_k NTT(b)_k \end{aligned} \tag{6}$$

Note the use of previous definitions. Specifically of equation (1) in step 1 and equation (4) in steps 4 and 5. All vertex indices are evaluated modulo N.

Using this property, we can calculate a convolution by applying NTT to both vectors, multiplying the transformed vectors component-wise, and then performing an inverse NTT on the result.

$$a \circledast b = NTT^{-1}(NTT(a) \cdot NTT(b)) \tag{7}$$

3.3 Negative Wrapped Convolution

To perform a negative wrapped convolution using the Number Theoretic Transform the input and output vectors must be scaled by powers of ψ, the 2Nth root of unity. This is done as follows:

$$\begin{aligned}
\hat{a} &= a_0 + \psi a_1 + ... + \psi^{N-1} a_N - 1 \\
\hat{b} &= b_0 + \psi b_1 + ... + \psi^{N-1} b_N - 1 \\
\hat{c} &= \hat{a} \circledast \hat{b} \\
c &= (1, \psi^{-1}, \psi^{-2}, ..., \psi^{1}) \cdot NTT^{-1}(NTT(\hat{a}) \cdot NTT(\hat{b}))
\end{aligned} \tag{8}$$

To understand why this works see [6] page 488.

4 Fast Fourier Transform

Using the above definitions of NTT requires $O(N^2)$ work, which is no better than the original convolution calculation. However, we can do less work if we apply the FFT algorithm. The Cooley-Tukey radix-2 [11] version of this algorithm divides the vector into two pieces of length $N/2$ and recursively calls the NTT algorithm on each piece. FFT uses the following result to divide NTT.

For k from 0 to $N-1$

$$\begin{aligned}
NTT(a)_k &= \sum_{j=0}^{N-1} a_j \omega^{jk} \\
&= \sum_{j=0}^{N/2-1} a_{2j} \omega^{2jk} + \sum_{j=0}^{N/2-1} a_{2j+1} \omega^{(2j+1)k} \\
&= \sum_{j=0}^{N/2-1} a_{2j} \omega^{2jk} + \omega^k \sum_{j=0}^{N/2-1} a_{2j+1} \omega^{(2j)k} \\
&= NTT(a^{even})_k + \omega^k NTT(a^{odd})_k
\end{aligned} \tag{9}$$

Notice the k value for each recursive NTT call is evaluated $mod(N/2)$, because this is the length of the resulting vectors. Alternatively, we can limit the value of k to between 0 and $N/2-1$ and reuse the calculated NTT values for both k and $N/2+k$.

$$\begin{aligned}
NTT(a)_k &= NTT(a^{even})_k + \omega^k NTT(a^{odd})_k \\
NTT(a)_{N/2+k} &= NTT(a^{even})_k + \omega^{N/2+k} NTT(a^{odd})_k
\end{aligned} \tag{10}$$

4.1 Time Results

If we assume N is a power of 2 we can apply FFT until we reach a constant sized base case (usually 2). This results in the following recurrence describing the work required at each level.

$$T(N) = 2T(N/2) + O(N)$$

At each level, there are two recursive calls made of size $N/2$ and $O(N)$ work is required to combine their results.

This recurrence produces the following time complexity using the master theorem [5].

$$T(N) = O(N\log(N))$$

5 Matrix Representations

Spiral [3] represents the application of the Fast Fourier Transform as a decomposition of one matrix, representing the original Discrete Fourier Transform, into four component matrices. This framework, the MAT-SPL language, provides the starting point for Spiral's optimization strategy. To apply Spiral to the Number Theoretic Transform, we must frame NTT in this language.

5.1 NTT

We can represent the NTT algorithm as a matrix vector product between the following matrix and the input vector.

$$NTT_k \in \mathbb{Z}^{N\times N} : a_ij = \omega^{i*j}$$

$$NTT_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega_4 & \omega_4^2 & \omega_4^3 \\ 1 & \omega_4^2 & \omega_4^4 & \omega_4^6 \\ 1 & \omega_4^3 & \omega_4^6 & \omega_4^9 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega_4 & -1 & -\omega_4 \\ 1 & -1 & 1 & -1 \\ 1 & -\omega_4 & -1 & \omega_4 \end{pmatrix}$$

5.2 FFT

The matrix described above also lets us represent the FFT algorithm as a multiplication of component matrices. These components multiply to create the original NTT matrix, and they are designed so that multiplying the input vector by each one in order follows the FFT algorithm.

Radix-m FFT (decimation in time, where $N/m = n$):

$$NTT_N = (NTT_m \otimes I_n^N)T_n^N(I_m^N \otimes NTT_n)L_m^N$$

Tensor Product: $(\otimes)$

$$A \otimes B = \begin{pmatrix} a_{0,0}B & a_{0,1}B \\ a_{1,0}B & a_{1,0}B \end{pmatrix}$$

$$I_2 \otimes NTT_2 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix}$$

$$NTT_2 \otimes I_2 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix}$$

Twiddle Matrix: (T_n^N)

$$A_{n,m} \in \mathbb{Z}^{n \times m} : a_i j = \omega_{nm}^{ij}$$

$$A_{3,2} = \begin{pmatrix} 1 & 1 \\ 1 & \omega_6 \\ 1 & \omega_6^2 \end{pmatrix}$$

$$T_n^N \in \mathbb{Z}^{N \times N} : T_n^N = Diag(Lin(A_{n,N/n}))$$

Where Lin(A) linearizes an nxm matrix into a vector of length $n * m$ in row major order. Diag(v) takes a vector of length k and creates a kxk matrix with the vector along its main diagonal.

$$T_3^6 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \omega_6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \omega_6^2 \end{pmatrix}$$

Permutation Matrix: (L_m^N)

$$f(i) = i/n + m * (i\%n)$$

$$L_m^N \in \mathbb{Z}^{N \times N} : l_{i,j} = \delta_{i,f(j)}$$

$$L_2^6 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

This is permutation is equivalent to the following process. Start with a vector of length $m * n$. Break it into m segments of size n and form a $m \times n$ matrix where each row is one of these segments. Take the transform of this matrix. Linearize the transform (again using row major order). The difference between the starting vector and the resulting vector represents the permutation resulting from multiplying L with a vertical vector.

Example

$$NTT_6 = (NTT_2 \otimes I_3^k) T_3^k (I_2^k \otimes NTT_3) L_2^k =$$

$$\begin{pmatrix} 1&0&0&1&0&0\\ 0&1&0&0&1&0\\ 0&0&1&0&0&1\\ 1&0&0&\omega_2&0&0\\ 0&1&0&0&\omega_2&0\\ 0&0&1&0&0&\omega_2 \end{pmatrix} \begin{pmatrix} 1&0&0&0&0&0\\ 0&1&0&0&0&0\\ 0&0&1&0&0&0\\ 0&0&0&\omega_6&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&\omega_6^2 \end{pmatrix} \begin{pmatrix} 1&1&1&0&0&0\\ 1&\omega_3&\omega_3^2&0&0&0\\ 1&\omega_3^2&\omega_3^4&0&0&0\\ 0&0&0&1&1&1\\ 0&0&0&1&\omega_3&\omega_3^2\\ 0&0&0&1&\omega_3^2&\omega_3^4 \end{pmatrix} \begin{pmatrix} 1&0&0&0&0&0\\ 0&0&1&0&0&0\\ 0&0&0&0&1&0\\ 0&1&0&0&0&0\\ 0&0&0&1&0&0\\ 0&0&0&0&0&1 \end{pmatrix}$$

6 Adapting Matrix Representation for Negative Wrapped Convolutions

Recall from equation (8) that to use NTT to perform a negative wrapped convolution we must first scale each vector by powers of ψ, the 2Nth root of unity. We can save $O(N)$ work in the negative wrapped transform by including the ψ powers into the precomputed constants of the transform. This is trivial when considering the single NTT_n matrix; simply scale each column by an appropriate ψ power. However, it becomes more complicated as we track this change through the application of an FFT. This was shown in Roy et al.[9] for a radix-2 Cooley-Tukey. The following represents a novel approach and extends their work for the radix-m Cooley-Tukey transform.

First, we define a new NTT^{ψ} transform matrix.

$$NTT_N^{\psi} = (\psi_N^j \omega_N^{ij})_{0 \leq < ij < N}$$

When performing the FFT decomposition of this matrix we must also change the twiddle factors in the T matrix. This can be done in the following way for a radix-m FFT.

$$A_{n,m} \in \mathbb{Z}^{n \times m} : a_{k,j} = \psi^{(1-n)j} \omega^{kj}$$
$$T_n^N = Diag(Lin(A_{n,m}))$$

A proof of this can be performed as follows. We will continue the convention that N is the size of the transform and $N = m * n$. ω is still the Nth root of unity and ψ is the 2Nth root, unless specified as otherwise. Recall, $\omega_m = \omega_N^n$.

First, we demonstrate how the radix-m transform is changed.

$$\begin{aligned} NTT^{\psi}(a)_k &= \sum_{j=0}^{N-1} \psi^j a_j \omega^{jk} \\ NTT^{\psi}(a)_k &= \sum_{j=0}^{m-1} \sum_{i=0}^{n-1} \psi^{in+j} a_{im+j} \omega^{(im+j)k} \\ NTT^{\psi}(a)_k &= \sum_{j=0}^{m-1} \psi^j \omega^{kj} \sum_{i=0}^{n-1} \psi^{im} a_{im+j} \omega^{imk} \\ NTT^{\psi}(a)_k &= \sum_{j=0}^{m-1} \psi^{j(1+2k)} NTT^{\psi}(a_{m,j})_k \end{aligned} \tag{11}$$

Where $a_{m,j}$ represents a vector of elements selected from a starting at j and striding by m.

We now know the recursive transforms must be multiplied by a $\psi^{j(1+2k)}$ factor. We can use this to define a matrix

$$F \in \mathbb{Z}^{N\times n} : f_{k,j} = \psi^{j(1+2k)}$$

.

There are two matrices which account for the factors attached to the recursive transformations, the left-most tensor product, and the twiddle matrix. We have already defined the tensor product by defining NTT^{ψ} and we will use this definition to discover the twiddle matrix. In the original NTT (without ψ powers) these two matrices are used to define the ω_N^{jk} powers using the separation

$$k = n\lfloor k/n \rfloor + k\%n$$

As a result, when viewed from the k, j indexing used to define F they are indexed by $(\lfloor k/n \rfloor, j)$ for the tensor product, and $(k\%n, j)$ for the twiddle matrix. This results in the following new definitions:

$$H \in \mathbb{Z}^{N\times m} : l_{k,j} = \omega_m^{\lfloor k/n \rfloor j} * \psi_m^j$$
$$G \in \mathbb{Z}^{N\times m} : t_{k,j} = \omega^{(k\%n)j}$$

Where H and G represent the extension of the NTT_m^{ψ} matrix and (pre-linearization) twiddle matrix respectively into $\mathbb{Z}^{N\times m}$. We now have the equality

$$F_{j,k} = H_{j,k} * G_{j,k} \tag{12}$$

Let ζ represent some function of j and k to modify the twiddle matrix. G becomes

$$G \in \mathbb{Z}^{N\times m} : t_{k,j} = \zeta\omega_N^{(k\%n)j}$$

Equality (12) becomes

$$\begin{aligned} \psi^{j(1+2k)} &= \omega_m^{\lfloor k/n \rfloor j} * \psi^{nj} * \zeta\omega^{(k\%n)j} \\ \psi^{j(1+2k)} &= \psi^{2n\lfloor k/n \rfloor j + nj} * \zeta\psi^{2(k\%n)j} \end{aligned} \tag{13}$$

It is clear ζ must take the form ψ^{ρ}

$$\begin{aligned} j(1+2k) &= 2n\lfloor k/n \rfloor j + nj + \rho + 2(k\%n)j \\ 1+2k &= 2n\lfloor k/n \rfloor + n + \rho/j + 2(k\%n) \\ 1+2k &= 2n\lfloor k/n \rfloor + n + \rho/j + 2(k - n\lfloor k/n \rfloor) \\ 1 &= n + \rho/j \\ \rho &= (1-n)j \\ \zeta &= \psi^{(1-n)j} \end{aligned} \tag{14}$$

7 Implementation in Spiral

Armed with our description of the Number Theoretic Transform in MatSPL we can use Spiral to generate optimized NTT code. See 'ntt.gi' for the Gap3 code required to define NTT in Spiral. Notice the code produced is lacking a key feature, the operations are not performed in a modulus. This means the resulting polynomial coefficients will not be reduced to their least representative in $\mathbb{Z}/p$. Cryptographic algorithms will expect these values to be reduced, so this is a necessary feature of an NTT implementation. Precisely how and when to perform these reductions is an ongoing study and [4, 2, 7] provide several different solutions.

The ψ powers absorption method in Section 6 was also done in the MatSPL framework so we can incorporate it into our optimized NTT code. See 'ntt_nw.gi' for this implementation. Notice in equation (8) that after taking the inverse NTT, the resulting vector is again scaled by powers of ψ. Poppelmann et al. [2] show how to include these values in the inverse transform. This feature should be included in the Spiral generated code for any NTT implementation that supports negative wrapped convolutions.

References

[1] Hoffstein J., Pipher J., Silverman J.H. (1998) NTRU: A ring-based public key cryptosystem. In: Buhler J.P. (eds) Algorithmic Number Theory. ANTS 1998. Lecture Notes in Computer Science, vol 1423. Springer, Berlin, Heidelberg.

[2] Pöppelmann T., Oder T., Güneysu T. (2015) High-Performance Ideal Lattice-Based Cryptography on 8-Bit ATxmega Microcontrollers. In: Lauter K., Rodríguez-Henríquez F. (eds) Progress in Cryptology – LATINCRYPT 2015. LATINCRYPT 2015. Lecture Notes in Computer Science, vol 9230. Springer, Cham.

[3] F. Franchetti et al., SPIRAL: Extreme Performance Portability, in Proceedings of the IEEE, vol. 106, no. 11, pp. 1935-1968, Nov. 2018, doi: 10.1109/JPROC.2018.2873289.

[4] D.Harvey. Faster arithmetic for number theoretic transforms. 2014.

[5] T. Cormen et al. Introduction to Algorithms. MIT Press, 2009.

[6] Rodney R. Howell. Chapter 15: The fast fourier transform. 2009. http://people.cs.ksu.edu/ rhowell/algorithms-text/text/chapter-15.pdf.

[7] Patrick Longa and Michael Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In Sara Foresti and Giuseppe Persiano, editors, Cryptology and Network Security, pages 124–139, Cham, 2016. Springer International Publishing.

[8] Nayuki. Number-theoretic transform (integer dft). 2017. https://www.nayuki.io/page/number-theoretic-transform-integer-dft.

[9] Sujoy Sinha Roy, Frederik Vercauteren, Nele Mentens, Donald Donglong Chen, and Ingrid Verbauwhede. Compact ring-lwe cryptoprocessor. In Lejla Batina and Matthew Robshaw, editors, Cryptographic Hardware and Embedded Systems – CHES 2014, pages 371–391, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[10] O. Regev V. Lyubashevky, C. Piekert. On ideal lattices andlearning with errors over rings. 2013. https://eprint.iacr.org/2012/230.pdf.

[11] W. Tukey W. Cooley. An algorithm for the machine calculations of complex fourier series. 1964. https://www.ams.org/journals/mcom/1965-19-090/S0025-5718-1965-0178586-1/S0025-5718-1965-0178586-1.pdf.