1.AIM

The Number Theoretic Transform (NTT) provides efficient algorithms for cyclic convolutions. NTT has several advantages over Fast Fourier Transform (FFT) terms of error free and faster computation. The Discrete Fourier transforms (DFT) is generally computed using the most promising NTT variant Fermat number transforms (FNT). Considering the importance of this transform, several efforts have gone in to make it faster than ever. We chiefly discuss about a new modular reduction technique that is tailored for the special moduli required by the NTT. Based on this reduction, to speedup the NTT, faster, multi-purpose algorithm is discussed.

2. STATE OF ART

In 1972, Rader , proposed a scheme modulo Mersenne number primes defining Mersenne number transform .The p-th Mersenne numbers Mp's are defined to be the prime integers of type 2^p-1, p also being prime, so far there are only forty-one Mersenne primes found (the first few values of p are 2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, 2203.....). Mp=2^p-1 then it follows that: 2^p=1 mod Mp meaning that 2 is of order p in the finite ring of integers under operations of additions and multiplication modulo Mp. Therefore,2 could be an alpha to compute NTT!
However, the scheme suffered from some practical (implementation based) failures:

1)Prime length of sequences: With a as 2 the maximum length for convolution is p, a prime integer, while NTT follows the architecture of FFT that is efficient only when the length N is highly composite; with a as -2 the length can be doubled to 2p, but then even 2p is barely composite (the reasoning that -2 is a root of order 2p mod Mp follows from the fact that (2, p)=1, meaning 2 and p are mutually prime, 4 i.e., 2^2 is also root of order p, so -2 = 4^(1/2) will be a root of order 2p

2) Maximum length Nmax possible is Mp-1=2p-2 that will require a be a primitive root of order Nmax. This value of a will be represented by few bits unlike 2 and so will be its higher powers, leading to far more complexity in modular arithmetic especially in the case of multiplications. Seeing the above machine level limitations, Agarwal and Burrus suggested for Fermat Number Transform Fermat numbers denoted by Ft where t-th Fermat number is defined as 2^(2^t)+1, t an integer = 0, 1, 2, ... , (it may be noted that most of the the Fermat numbers are not primes; only the first five numbers F0 through F4 are primes, others so far searched are known to be composites Ft = 2^(2^t)+1, it follows that for primes, i.e., t <= 4, the order of 2 is 2^t+1 as, 2^(2^t) = -1 mod Ft leading to 2^((2^t).2)= 2^(2^t+1) = 1 mod Ft . Hence, with a as 2, Nmax would be 2^t+1. The concept of 2^0.5 was introduced to double the length to 2^t+2 with alphas that required only two bit representation. Equating b, the number of bits, with 2^t, 2 may be termed as a(2b) and 2^0.5 as a(4b). In the structure of Fermat Number Transform, multipliers can be effectively replaced by shifters as multiplication by some.

Actually the composite Fermat numbers F5 (b=32) and F6 (b=64) are attractive for the applications in signal processing. It was pointed out that while these numbers are composite and their factors are of the type K2^t+2 + 1,they can still be utilized for transforming with a(2b) and a(4b) with
A(4b) = 2^(b/4) (2^(b/2) - 1) and (a(4b))^2 = 2 mod Ft .Later on 2 dimensional techniques were employed for computing convolutions,Agarwal and Burrus proposed a two-dimensinal scheme for convolution of length N = LP being implemented as a two-dimensional cyclic convolution of length 2L by P that can be computed by using a two-dimensional FNT. Using this two dimensional scheme, the wordlength is proportional to the square root of the length of the sequences to be convolved which would give for a maximum length of 8b^2, rather than 4b.

3.RESEARCH WORK FOCUSSED

3.1 The Number Theoretic Transform (NTT)

The NTT is a specialized version of the discrete Fourier transform, in which the coefficient ring is taken to be a finite field (or ring) containing the right roots of unity. It can be viewed as an exact version of the complex DFT, avoiding roundoff errors for exact convolutions of integer sequences. While Gauss apparently used similar techniques, laying the ground work for modern FFT algorithms to compute the DFT, and therefore the NTT, is usually attributed to Cooley and Tukey's seminal paper .

Let n being a power of 2 and q a prime with $q \equiv 1 \pmod{2n}$, let $a = (a[0],\ldots, a[n-1]) \varepsilon Z^n_q$, , and let ω be a primitive n-th root of unity in Zq, which means that $\omega^n \equiv 1 \pmod{q}$. The forward transformation ~a = NTT(a) is defined as $\sim a[i] = \sum_{j=0}^{n-1} a[j]\omega^{ij} \bmod q$ for i =0,1,…, n - 1. The inverse transformation is given by b = INTT(~a), where $b[i] = n^{-1}\sum_{j=0}^{n-1} \sim a[j]\omega^{-ij} \bmod q$ for i =and we have INTT(NTT(a)) = a.

3.2. Fermat number transforms

In this section one of the NTT variant Fermat number transforms are explained. FNT is the most promising variant of NTT and found several applications [9-10]. Here the modulus chosen is called as Fermat number $Ft = 2^b + 1$, where $b = 2^t$, t=0,1,2,…

Only F0 to F4 are prime and next numbers are composite. The first few Fermat numbers are:

$F_0 = 3, F_1 = 5, F_2 = 17, F_3 = 257, F_4 = 65537$ and so on.

Since Fermat numbers up to F_4 are all prime, FNT for any length $N = 2^m$ where $m \le b$ is possible.

3.3 NTTs FOR FASTER CONVOLUTION

In this section we consider convolution of two sequences as an example of NTT application. Convolution is heart of any transform based application. In the previous year's several methods have been shown to implement convolution that vary in terms of computations required, the amount of storage needed and round off error effects. Reducing the number of multiplications in convolution is often necessary as the multiplication operation is time taking and complex.

3.3.1 *Methodology*

DFT is employed to find convolution as DFT[h*x] = DFT[h] x DFT[x] and this implies that convolution can be found by $Y(n) = \text{IDFT}\{\text{DFT}[h] \times \text{DFT}[x]\}$

The disadvantage of this method is significant round off error, complex domain operations and considerable number of multiplications.

Number theoretic transforms are shown to possess property of CCP and have the similar structure of DFT.

A general transform has following expression with DFT structure $X(k) = \sum_{j=0}^{n-1} x[j]\omega^{jk}$

The NTT of a signal x (n) and its inverse INTT are defined as follows:

NTT: $X(k) = \sum_{j=0}^{n-1} x[j]\omega^{j/k} \bmod M$, k=0,1,2…n-1

INTT: $x(j) = (n^{-1} \sum_{k=0}^{n-1} X[k]\omega^{jk}) \bmod M$, j=0,1,2…n-1

In Fourier transform the $e^{-2x/n}$ is a complex number and transform operation is carried out in complex domain even if sequences are real. In NTT $e^{-2x/n}$ term is replaced by an integer ω of order N. When two sequences x[i] and h[i] are convolved using NTT , their output Y[i] is congruent to convolution of x and h mod M . By suitable choice of n, modulus M and value it is possible to do the operation without any multiplications but using only shifts and additions. One such NTT is FNT as explained earlier. As the modulo arithmetic doesn't use approximations these transforms do not have round off errors.

3.4 Comparison with FFT

NTT of a signal is carried out in the integer domain whereas DFT/FFT in complex domain.NTT coefficients do not have any physical meaning as the operations carried out modulo arithmetic. Because of modular arithmetic all the outputs are integers in the range 0 to M-1 where M is modulus, and no approximations

take place. Hence NTTs do not have any round off errors as in FFT/DFT based algorithms. The special case of NTT, Fermat number transform uses ω=2 or powers of 2 and hence all multiplications can be converted into shifts and additions. This is much simpler than complex domain multiplications; hence FNT is faster than FFT. If two sequences x[i] and h[i] have a1 and a2 bit representations, and transform length n is then convolution output Y[i] has maximum ($a1 + a2 + \log_2 n$) bit representation. In FFT every data sample has real and imaginary part, it requires two words one for real and one for imaginary. Hence storage and hardware requirements for both NTT and FFT remain almost same.

Let Ψ be a primitive 2n-th root of unity in Z_q such that $\Psi^2 = \omega$, and let a =(a[0],..., a[n -1]), b = (b[0],..., b[n -1]) belonging to Z^n_q be two vectors. Also, define ^a =(a[0],Ψ a[1]..., Ψ^{n-1}a[n-1]) and ^b = (b[0],Ψ b[1]..., Ψ^{n-1}b[n-1]). The negative wrapped convolution of a and b is defined as c = (1, Ψ^{-1}, Ψ^{-2},..., $\Psi^{-(n-1)}$)) o INTT(NTT(^a) o NTT(^b)), where o denotes component-wise multiplication. This operation satisfies c = a . b in R_q, and thus it allows us to compute a full polynomial multiplication that implicitly includes the reduction modulo X^n+1, without increasing the length of the inputs.

Some additional optimizations are available to the NTT-based polynomial multiplication. Previous works explain how to merge multiplications by the powers of ω with the powers of Ψ and Ψ^{-1} inside the NTT. Consequently, important savings can be achieved by precomputing and storing in memory the values related to these parameters. In particular, Roy showed how to merge the powers of Ψ with the powers of ω in the forward transformation. This merging did not pose any difficulty in the case of the well-known decimation-in-time NTT, which is based on the Cooley-Tukey butterfly . Similarly, it was showed how to merge the powers of Ψ^{-1} with the powers of w in the inverse transformation based on Gentleman-Sande butterfly which is decimation-in-frequency NTT .

Several works in the literature have applied a relatively expensive reordering or bit-reversal step before or after the NTT computation. From here, we denote by NTT := $\text{NTT}_{\text{CT},\psi\ \text{rev}}$ an algorithm that computes the forward transformation based on the Cooley-Tukey buttery that absorbs the powers of in bit-reversed ordering. This function receives the inputs in standard ordering and produces a result in bit-reversed ordering. Similarly, we denote by INTT := $\text{INTT}_{\text{GS},\psi^{-1}\ \text{rev}}$ an algorithm computing the inverse transformation based on the Gentleman-Sande buttery that absorbs the powers of Ψ^{-1} in the bit reversed ordering. This function receives the inputs in bit-reversed ordering and produces an output in standard ordering. Following is, the combination of these two functions eliminates any need for a bit-reversal step. Optimized algorithms for the forward and inverse NTT are presented in Algorithms 1 and 2, respectively.

Algorithm 1 Function NTT based on the Cooley-Tukey (CT) butterfly.

Input: A vector $a = (a[0], a[1], ..., a[n-1]) \in \mathbb{Z}_q^n$ in standard ordering, where q is a prime such that $q \equiv 1 \bmod 2n$ and n is a power of two, and a precomputed table $\Psi_{rev} \in \mathbb{Z}_q^n$ storing powers of ψ in bit-reversed order.
Output: $a \leftarrow \text{NTT}(a)$ in bit-reversed ordering.

```
1: t = n
2: for (m = 1; m < n; m = 2m) do
3:     t = t/2
4:     for (i = 0; i < m; i++) do
5:         j1 = 2 · i · t
6:         j2 = j1 + t − 1
7:         S = Ψ_rev[m + i]
8:         for (j = j1; j ≤ j2; j++) do
9:             U = a[j]
10:            V = a[j + t] · S
11:            a[j] = U + V mod q
12:            a[j + t] = U − V mod q
13:        end for
14:    end for
15: end for
16: return a
```

Algorithm 2 Function INTT based on the Gentleman-Sande (GS) butterfly.

Input: A vector $a = (a[0], a[1], ..., a[n-1]) \in \mathbb{Z}_q^n$ in bit-reversed ordering, where q is a prime such that $q \equiv 1 \bmod 2n$ and n is a power of two, and a precomputed table $\Psi_{rev}^{-1} \in \mathbb{Z}_q^n$ storing powers of ψ^{-1} in bit-reversed order.
Output: $a \leftarrow$ INTT(a) in standard ordering.

```
1: t = 1
2: for (m = n; m > 1; m = m/2) do
3:     j1 = 0
4:     h = m/2
5:     for (i = 0; i < h; i++) do
6:         j2 = j1 + t − 1
7:         S = Ψ_rev^{-1}[h + i]
8:         for (j = j1; j ≤ j2; j++) do
9:             U = a[j]
10:            V = a[j + t]
11:            a[j] = U + V mod q
12:            a[j + t] = (U − V) · S mod q
13:        end for
14:        j1 = j1 + 2t
15:    end for
16:    t = 2t
17: end for
18: for (j = 0; j < n; j++) do
19:    a[j] = a[j] · n^{-1} mod q
20: end for
21: return a
```

3.5 Modular Reduction and Speeding up the NTT

Most FFT algorithms to compute the NTT over a finite field or ring need certain roots of unity. In the specific setting discussed in the previous section, one needs primitive 2n-th roots of unity to exist modulo q, which imposes a congruence condition on q, namely q ≡1 (mod 2n). Modular reduction. In this section, we introduce a new modular reduction method for moduli of this special shape. We note that it works similarly for any modulus of the form $k.2^m \pm l$, where k and l are small positive integers such that k ≥ 3 and l ≥1. However, for ease of exposition and to focus on the case most relevant in the context of the NTT, we only treat the case $q = k.2^m + 1$.

Let $0 \le a, b < q$ be two integers modulo q and let C = a.b be their integer product. Then $0 \le C < q^2 = k^2 2^{2m} + k2^{m+1} + 1$. The goal is to reduce C modulo q using the special shape of q, namely using the fact that $k2^m \equiv 1 \pmod q$. Write $C = C0 + 2^m C1$, where $0 \le C0 < 2^m$. Then $0 \le C1 = (C - C0)/2^m < k^2 2^m + 2k + 1/2^m = kq + k + 1/2^m$. We have that $kC \equiv kC0 - C1 \pmod q$, and given the above bounds for C0 and C1, it follows that the integer kC0 - C1 has absolute value bounded by $|kC0-C1| < (k+1/2^m)q$. As k is a small integer, the value kC0 - C1 can be brought into the range [0; q) by adding or subtracting a small multiple of q. The maximal value for C is $(q-1) = k^2 2^{2m}$, in which case C0 = 0 and $C1 = k^2 2^m = k(q-1)$, meaning that (k-1)q must be added to kC0-C1 to fully reduce the result. In the modified NTT described below, however, this final reduction into [0, q) is not performed throughout the computation, but rather only at the very end of the algorithm.
For implementation of this logic, function K-RED is defined as follow

```
function K-RED(C)
    C0 ←C mod 2^m
    C1 ←C/2^m
    return kC0 - C1
end function
```

The function K-RED can take any integer C as input. It then returns an integer D such that $D \equiv kC \pmod q$ and $|D| < q + |C|=2m$. Although this function alone does not properly reduce the value C modulo q, it should still be called a reduction because it brings D close to the desired range.

In the context of a specific, longer computation, and depending on the parameter n and the target platform, additional reductions might need to be applied to a limited number of intermediate values, for which overflow may occur. In this case, as an optimization, two successive reductions can be merged as follows. Let the input operand C be decomposed as $C = C0 + C1 .2^m + C2 .2^{2m}$ with $0 \le C0,C1 < 2^m$. Then we can reduce C via the following function K-RED-2x.

function K-RED-2x(C)
    $C0 \leftarrow C \bmod 2m$
    $C1 \leftarrow C/2^m \bmod 2^m$
    $C2 \leftarrow C/2^{2m}$
return k^2 C0 - kC1 + C2
end function

3.5.1 Speeding up the NTT

In the context of the NTT algorithm, a redundant representation of integers modulo q by allowing them to grow up to 32 bits is used and, when necessary, the reduction function K-RED is applied to reduce the sizes of coefficients. For the sake of illustration, consider Algorithm 1. The main idea is to apply the function K-RED only after multiplications, i.e., one reduction per iteration in the inner loop, letting intermediate coefficient values grow such that the final coefficient values become congruent to K.a[.] mod q for a fixed factor K. This factor can then be used at the end of the NTT-based polynomial multiplication to correct the result to the desired value.

Next, lets take an example by analysing the details of the method for n ε {256, 512, 1024} for the prime q = 12289. We limit the analysis to platforms with native 32 (or higher)- bit multipliers.

The case q = 12289. The modified NTT algorithms using K-RED and K-RED-2x are shown in Algorithm 3 and Algorithm 4 for the modulus q = 12289, which in practice is used with n = 512 or 1024 . In Steps 7 of Algorithm 3 and Step 7 of Algorithm 4, the pre computed values scaled are by k^{-1}, precomputed tables for $\Psi_{rev,k^{-1}}[.]=k^{-1}.\Psi_{rev}[.]$ and $\Psi^{-1}{}_{rev,k^{-1}}[.]=k^{-1}.\Psi^{-1}{}_{rev}[.]$ are used . From hereon, let's denote these modified algorithms by $NTT^K := NTT^K_{CT,\psi(rev,k^{-1})}$ and $INTT^K := INTT^K_{GS,\psi(rev,k^{-1})^{-1}}$ respectively.

Given two input vectors a and b, let c = INTT(NTT(a) o NTT(b)) be computed using Algorithms 1 and 2. It is easy to see that the resulting coefficients after applying Algorithms 3 and 4, i.e., after computing $INTT^K(NTT^K(a) \text{ o } NTT^K(b))$, are congruent to K . c[.] modulo q for a certain fixed integer $K = k^s$ and an integer s. In Line 7 of Algorithm 3 the value S carries a factor k^{-1} which then cancels with the factor k introduced by K-RED in Step 15. Only additional reductions such as those in Steps 12 and 13 increase the power of k in the final result.

At the end of the computation, the final results can be converted back to the standard representation by multiplying with the inverse of the factor K. Moreover, this conversion can be obtained for free if the computation is merged with the scaling by n^{-1} during the inverse transformation, that is, if scaling is performed by multiplying the resulting vector with the value $n^{-1} .K^{-1}$. However, a better additional speedup is obtained by merging the second entry of the table $\Psi_{rev,k}{}^{-1}$ with the fixed value $n^{-1} . K^{-1}$ in the steps 25-28, this creates elimination of additional n/2 multiplications and modular reductions. This is shown in Steps 25-28 of Algorithm 4.

Algorithm 3 Modified function $\texttt{NTT}^K$ using K-RED and K-RED-2x for reduction modulo $q = 12289$ (32 or 64-bit platform).

Input: A vector $a = (a[0], a[1], ..., a[n-1]) \in \mathbb{Z}_q^n$ in standard ordering, where $n \in \{256, 512, 1024\}$, and a precomputed table $\Psi_{rev,k^{-1}} \in \mathbb{Z}_q^n$ of scaled powers of ψ in bit-reversed order.
Output: $a \leftarrow \texttt{NTT}^K(a)$ in bit-reversed ordering.

$t = n$
for $(m = 1;\ m < n;\ m = 2m)$ **do**
  $t = t/2$
  for $(i = 0;\ i < m;\ i{+}{+})$ **do**
    $j_1 = 2 \cdot i \cdot t$
    $j_2 = j_1 + t - 1$
    $S = \Psi_{rev,k^{-1}}[m + i]$
    for $(j = j_1;\ j \le j_2;\ j{+}{+})$ **do**
      $U = a[j]$
      $V = a[j + t] \cdot S$
      if $m = 128$ **then**
        $U = \texttt{K-RED}(U)$
        $V = \texttt{K-RED-2x}(V)$
      else
        $V = \texttt{K-RED}(V)$
      end if
      $a[j] = U + V$
      $a[j + t] = U - V$
    end for
  end for
end for
return a

Algorithm 4 Modified function $\texttt{INTT}^K$ using K-RED and K-RED-2x for reduction modulo $q = 12289$ (32 or 64-bit platform).

Input: A vector $a = (a[0], a[1], ..., a[n-1]) \in \mathbb{Z}_q^n$ in bit-reversed ordering, where $n \in \{256, 512, 1024\}$, a precomputed table $\Psi^{-1}_{rev,k^{-1}} \in \mathbb{Z}_q^n$ of scaled powers of ψ^{-1} in bit-reversed order, and constants $n_K^{-1} = n^{-1} \cdot k^{-11}$, $\Psi_K^{-1} = n^{-1} \cdot k^{-10} \cdot \Psi^{-1}_{rev,k^{-1}}[1] \in \mathbb{Z}_q$, where $k = 3$.
Output: $a \leftarrow \texttt{INTT}^K(a)$ in standard ordering.

$t = 1$
for $(m = n;\ m > 2;\ m = m/2)$ **do**
  $j_1 = 0$
  $h = m/2$
  for $(i = 0;\ i < h;\ i{+}{+})$ **do**
    $j_2 = j_1 + t - 1$
    $S = \Psi^{-1}_{rev,k^{-1}}[h + i]$
    for $(j = j_1;\ j \le j_2;\ j{+}{+})$ **do**
      $U = a[j]$
      $V = a[j + t]$
      $a[j] = U + V$
      $a[j + t] = (U - V) \cdot S$
      if $m = 32$ **then**
        $a[j] = \texttt{K-RED}(a[j])$
        $a[j + t] = \texttt{K-RED-2x}(a[j + t])$
      else
        $a[j + t] = \texttt{K-RED}(a[j + t])$
      end if
    end for
    $j_1 = j_1 + 2t$
  end for
  $t = 2t$
end for
for $(j = 0;\ j < t;\ j{+}{+})$ **do**
  $U = a[j]$
  $V = a[j + t]$
  $a[j] = \texttt{K-RED}((U + V) \cdot n_K^{-1})$
  $a[j + t] = \texttt{K-RED}((U - V) \cdot \Psi_K^{-1})$
end for
return a

4.CONCLUSION

In this paper the properties of new transforms, NTTs and its variants are studied. NTTs have attractive features as the modulo arithmetic is carried out on integers and all representations are in integers. Because of this they do not have round off errors as DFT. They need very less or no multiplications on employing suitable NTT module (n and ω) parameters as in the case of FNT. This paper implements convolution using FNT with a new modular reduction which gives faster result compared to convolution using FFT or the standard NTT algorithms. The new modular reduction algorithm allows reductions to be carried out only after multiplications. Since the new NTT and INTT algorithm with modular reduction is based on Cooley-Tukey and Gentleman-Sande butterfly operation techniques respectively so additional implementation for bit reversal order is not required for overall computation of multiplication of polynomials or large numbers which gives additional speedups .So as a result we can say that NTTs (more specifically FNTs and improved NTTs with the discussed modular reduction)can be used for applications like faster multiplication of large integers and large degree polynomials.

5.REFERENCES

[1]. Patrick Longa and Michael Naehrig :Speeding up the Number Theoretic Transform for Faster Ideal Lattice-Based Cryptography, Microsoft Research, USA

[2]. *M Bhattacharya, **R Creutzburg, *J Astola (2004): Some historical notes on number theoretic transform, * Institute of Signal Processing,Tampere University of Technology, ** Fachhochschule Brandenburg University of Applied Sciences

*[3]. *Salila Hegde, **RohiniNagapadma Number Theoretic Transforms for Fast Digital Computation, Department of ECE,NIE Institute of Technology*, NationalInstitute of Engineering***

*[4]. *Wan-Chi Siu, **M.Phil., and ***A.G. Constantinides: On the computation of discrete Fourier transform using Fermat number transform, AP(HK)*, (C.Eng., M.I.E.R.E., Mem.I.E.E.E.**),(B.Sc.(Eng.), Ph.D., C.Eng., M.I.E.E., Sen.Mem.I.E.E.E.***)*

[5]. J. W. Cooley and J.W. Tukey. An algorithm for the machine calculation of complex

Fourier series. Mathematics of Computation,

[6]. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein : Introduction to Algorithms, Second Edition, McGraw-Hill Book Company

[7]. https://cp-algorithms.com/algebra/fft.html

SUBMITTED BY:
BHARGAB GAUTOM(180123008)
HARSH YADAV(180123015)

SUBMITTED TO:
PROF. K.V. KRISHNA
DEPARTMENT OF MATHEMATICS,IITG