Pamukkale Üniversitesi
Mühendislik Bilimleri Dergisi
Cilt 18, Sayı 2, 2012, Sayfa 95-103

Comparing Partial and Full Return Spectral Methods

Kısmi ve Tam Dönümlü Spektral Metotların Karşılaştırması

İhsan Haluk AKIN[a], Gökay SALDAMLI[b], Murat AYDOS[c]*

[a]Fatih Üniversitesi, Mühendislik Fakültesi, Bilgisayar Müh. Bölümü, E Blok, 34500, Büyükçekmece, İstanbul
[b]Boğaziçi Üniversitesi, Uygulamalı Bilimler Yüksekokulu, Yönetim Bilişim Sistemleri Bölümü, 34342, Bebek, İstanbul
[c]Pamukkale Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, 20070, Kınıklı, Denizli

Geliş Tarihi/Received : 21.06.2011, Kabul Tarihi/Accepted : 10.11.2011

ÖZET

Bu çalışmada, yakın zamanda sunulmuş spektral modüler aritmetik işlemlerinin aritmetik karmaşıklığı üzerindeki bir analiz adım adım değerlendirme yöntemi ile karşılaştırılmıştır. Bilgisayar aritmetiğinde spektral yöntemlerin standart kullanımı çarpma ve indirgeme adımlarının spektrum ve zaman uzayında birbirinden ayrı olarak gerçekleştirilmesi gerektiğini belirtmektedir. Bu tarz bir prosedür ise açıkça tam dönümlü (ileri ve geri yönde) DFT hesaplamalarına ihtiyaç duymaktadır. Öte yandan, bazı kısmı değerlerin işlem sırasında hesaplanması ile, yeni yöntemler indirgeme işlemi de dahil olmak üzere tüm verilerin tüm zamanlarda spektrumda tutulmasını gerektiren bir yaklaşımı benimsemişlerdir. Tüm bu yaklaşımların işlem süresi performanslarını karşılaştırdığımızda, tam dönümlü algoritmaların son zamanlarda önerilmiş yöntemlerden daha iyi performans gösterdiğini bu çalışmada göstermiş bulunmaktayız.

Anahtar Kelimeler: *Spektral modüler aritmetik, Modüler indirgeme, Modüler çarpma, Montgomery indirgeme.*

ABSTRACT

An analysis on the arithmetic complexity of recently proposed spectral modular arithmetic – in particular spectral modular multiplication- is presented through a step-by-step evaluation. Standart use of spectral methods in computer arithmetic instructs to utilize separated multiplication and reduction steps taking place in spectrum and time domains respectively. Such a procedure clearly needs full return (forward and backward) DFT calculations. On the other hand, by calculating some partial values on-the-fly, new methods adopt an approach that keeps the data in the spectrum at all times, including the reduction process. After comparing the timing performances of these approaches, it is concluded that full return algorithms perform better than the recently proposed methods.

Keywords: *Spectral modular arithmetic, Modular reduction, Modular multiplication, Montgomery reduction.*

1. INTRODUCTION

Spectral techniques for integer multiplications have been known for over a quarter of a century (Schönhage and Strassen, 1971). These methods are extremely efficient for applications using large size integer multiplications. The technique starts with transforming the encoded integers to the frequency domain (possibly via FFT), which is followed by a point multiplication in the spectrum. After this computation, an inverse transform and a decoding is applied to send the result back into the time domain as seen in Figure 1.

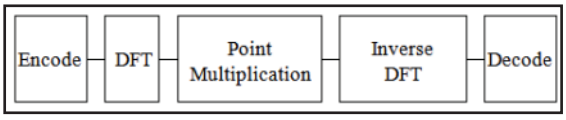

Figure 1. Shönhage and Strassen's Integer multiplication.

* Yazışılan yazar/Corresponding author. E-posta adresi/E-mail address : maydos@pau.edu.tr (M. Aydos)

After the RSA proposal (Rivest et al., 1978), modular arithmetic-in particular modular reduction-attracts more and more interest. Perhaps, a method (Montgomery, 1985) described by P. Montgomery in 1985 is the most notable presentation among several other methods. Montgomery reduction carries numbers into n-residues, in which modular multiplication is more effective if consecutive multiplications are performed.

Saldamli proposed a new method for integer modular reduction (Saldamli, 2005; Saldamli and Koc, 2007). This method performs reduction on spectral domain rather than the time domain. In fact the method is an adaption of the redundant Montgomery algorithm to the spectral domain. Based on this reduction, he further proposed spectral modular multiplication, and spectral modular exponentiation. However, in their work, the authors did not conduct a true comparision with the existing literature. In this study, our main objective is to give a regirous comparision between the usual redundant Montgomery algorithm and proposed methods.

Going back again to the history: RSA altered the history of cryptography by bringing up the public key cryptography notion. Later in late 80s, Koblitz and Miller independently introduced the elliptic curve cryptography-ECC, (Miller, 1986; Koblitz, 1987). Because of its efficiency, short key lengths and mature mathematics ECC is recently adopted by the U.S. Government as the basic technology for key agreement and digital signature standard (NIST, 2009).

The security of the ECC depends on the well known discrete logarithm problem. To setup the system one has to compute exponentiations in the elliptic curve group, requiring several calculations (especially multiplications) with in a finite field. As the ECC over binary and prime fields are standardized (IEEE, 1999; ANSI, 2001), one can argue that the practical (i.e. implementation) aspects of these systems are fairly mature. On the other hand, the arithmetic in medium size characteristics extension fields (i.e. GF (p^k) for some positive integer k and a prime p such that $0 < p < 2^{128}$) is still a very active research topic. Recently, some researchers proposed and evaluated the spectral modular reduction over the medium size characteristics fields (Baktir et al., 2007; Baktir, 2008). Moreover, he successfully applied the method to ECC.

In this study, we compare the performance of the standard modular FFT multiplication and spectral modular multiplication. To be more specific, FFT multiplication combined with the redundant Montgomery reduction and recently proposed spectral algorithms given by Saldamli and Baktir et al. (Saldamli, 2005; Baktir et al., 2007). We believe, developers in particular cryptographic engineers would benefit the outcomes of this work as it would give them a fair foreseeing before doing their design work. The presentation of our study is organized as follows.

In the next section, after giving the preliminary definitions, we state the standard modular FFT multiplication as a combination of Schönhage and Strassen's integer multiplication algorithm (SaSIMA) and redundant Montgomery reduction. Our spectral modular multiplication presentation follows the notation and terminology given in (Saldamli, 2005).

In Section 3, we present our evaluation results for the prime fields showing that SaSIMA performs much better than the spectral modular multiplication. In Sections 4 and 5, we turn our attention to multiplication over the medium size characteristics fields; present an adaption of SaSIMA to GF (p^k) and report a similar result when it is compared with the algorithm proposed in (Baktir et al., 2007). Finally, we conclude our work in the last section.

2. SPECTRAL MODULAR REDUCTION

We briefly give the basic terminology needed for the presentation of the spectral modular operations.

Definition 1 Let $x(t) = x_0 + x_1 t + \cdots + x_{d-1} t^{d-1} \in Z[t]$. If $x(b) = a$ for some $a \in Z$ than we say $x(t)$ is a polynomial representation of a with respect to base b.

2. 1. Discrete Fourier Transform (DFT)

The definition and properties of DFT in a finite field setting is slightly different from the

common use of this transform in engineering. In order to suppress on this distinction we start with a formal definition of DFT over finite fields.

Definition 2 Let ω be a primitive d-th root of unity in Z_q and, let x(t) and X (t) be polynomials of degree d – 1 having entries in Z_q. The DFT map over Z_q is an invertible set map sending x(t) to X (t) given by the following equation;

$$X_i = DF\ T_d^{\omega}(x(t)) := \sum_{j=0}^{d-1} x_j \omega^{ij} \bmod q \qquad (1)$$

With the inverse,

$$x_i = IDF\ T_d^{\omega}(x(t)) := d^{-1} \sum_{j=0}^{d-1} X_j \omega^{-ij} \bmod q$$

for i, j = 0, 1, . . . , d – 1. We say x(t) and X (t) are transform pairs, x(t) is called a time polynomial and sometimes X (t) is named as the spectrum of x(t).

Remark 1 In the literature, DFT over a finite ring spectrum is also known as the Number Theoretical Transform (NTT). Moreover, if q has some special form such as a Mersenne or a Fermat number, the transform named after this form; Mersenne Number Transform (MNT) or Fermat Number Transform (FNT).

Note that, unlike the DFT over the complex numbers, the existence of DFT over finite rings is not trivial. In fact, Pollard mentions that the existence of primitive root d-th of unity and the inverse of d do not guarantee the existence of a DFT over a ring (Pollard, 1976). He adds that a DFT exists in ring R if and only if each quotient field R/M (where M is maximal ideal) possesses a primitive root of unity.

To simplify our discussions, throughout this text we take q as a Mersenne prime and the principal root of unity as ω = –2 without loss of generality. According to Saldamli and Baktir et al., such a preference reflects the best performance for DFT computations, spectral multiplications and eductions among other choices (Saldamli, 2005; Baktir et al., 2007).

2. 2. SaSIMA Combined with Redundant Montgomery Reduction

To be consistent, we adopt the previous section's notation and state the standard modular FFT multiplication as a combination of SaSIMA and redundant Montgomery reduction.

Let r, b, u ∈ Z, $b = 2^u$ and n_i (t) be the polynomial representation of an integer multiple of modulus n such that the zeroth coefficient of n_i (t) satisfies $(n_i)_0 = 2^{i-1}$ for i = 1, 2, . . . , u (observe that n(t) = n_1(t)).

Now, we write a β multiple of n(t) as

$$\beta.n(t) = \sum_{i=1}^{u} \beta_i . n_i(t) \qquad (2)$$

Where b > β ∈ N and β_i represents the binary digits of β.

Algorithm 1 Spectral multiplication with time reduction

Suppose that there exist a d-point DFT map for some principal root of unity ω in Z_q, and X (t) and Y (t) are transform pairs of x(t) and y(t) respectively where x(b) = x and y(b) = y for some x, y < n. Let $N_T = \{n_1(t), n_2(t), \ldots, n_u(t)\}$ be the set of special polynomials as described above;

Input : X (t), Y (t) and a basis set NT
Output: Z (t) = DF T (z(t)) where $z \equiv xy2^{-db} \bmod n$ and z(b) = z,

1. : Z (t) := X (t) ◎ Y (t)
2. : z(t) := I DF T_t^w (Z (t))
3. : α := 0
4. : for i = 0 to d − 1
5. : β := −(z_0 + α) mod b
6. : α := (z_0 + α + β)/b
7. : z(t) := $z(t) + \prod_{j=1}^{u} \beta_j . n_j(t)$
8. : z(t) := z(t)/t
9. : end for
10. : z_0 (t) := z_0 (t) + α
11. : Z (t) := DF T (z(t))
12. : return Z (t)

Observe that Alg. 1 requires a full return computation (i.e. Step 2) right after the

component-wise multiplication. Moreover, Steps 4 through 9 perform the reduction in time domain implementing the so called redundant Montgomery reduction.

2. 3. Modified Spectral Modular Product (MSMP)

On the other hand, MSMP describes a partial return algorithm originally described by Saldamli (Saldamli, 2005). Let r, b, u ∈ Z, b = 2^u and $n_i(t)$ be the polynomial representation of an integer multiple of n such that the zeroth coefficient of n_i (t) satisfies $(n_i)_0 = 2^{i-1}$ for i = 1, 2, . . . , u (note that n(t) = n_1 (t)). We can now write β · N (t) as

$$\beta.N(t) = \sum_{i=1}^{u} \beta_i.N_i(t) \quad (3)$$

Where β_i is a binary digit of β and $N_i(t) = DFT_d^{\omega}$ (ni(t)) for i = 1, 2, . . . , u. Note that β < b and $\beta_i = 0$ for i > u.

Algorithm 2 MSMP algorithm

Suppose that there exist a d-point DFT map for some principal root of unity ω in Z_q, and X (t) and Y (t) are transform pairs of x(t) and y(t) respectively where x(b) = x and y(b) = y for some x, y < n and b > 0. Let $N_F = \{N_1(t), N_2(t), \ldots, N_u(t)\}$ be the set of special polynomials as described above;

Input: X (t), Y (t) and a basis set N_F
Output: Z (t) = DF T (z(t)) where $z \equiv xy2^{-db} \bmod n$ and z(b) = z,

1. : Z (t) := X (t) ◎ Y (t)
2. : α := 0
3. : for i = 0 to d – 1
4. : $z_0 := d^{-1} \cdot (Z_0 + Z_1 + \ldots + Z_{d-1}) \bmod q$
5. : $\beta := -(z_0 + \alpha) \bmod b$
6. : $\alpha := (z_0 + \alpha + \beta)/b$
7. : $Z(t) := Z(t) + \prod_{j=1}^{u} \beta_j.N_j(t) \bmod q$
8. : Z (t) := Z (t) − (z_0 + β)(t) mod q
9. : Z (t) := Z (t) ◎ Γ(t) mod q
10. : end for
11. : Y (t) := Y (t) + α(t)
12. : return Z (t)

3. COMPARING SaSIMA AND MSMP

Notice that both of the algorithms perform their multiplication in the spectral domain. However, they employ different reduction process. To be more informative; the reduction in Alg. 1 takes place in time whereas Alg. 2 computes the modular reduction in spectral domain. Therefore, we particularly probe this difference to compare the arithmetic and ASIC performances of these algorithms through a step-by-step evaluation.

3. 1. Arithmetic Performance

In order to perform the Montgomery reduction, the least significant word of the partial sum has to be known in advance at each iteration. Therefore, Alg. 2 requires partial returns to time domain to determine the least significant words. With the help of these partial returns, reduction calculations are performed in spectral domain.

On the other hand, Alg. 1 performs the Montgomery reduction in time domain. Naturally, such a reduction needs a full return of the multiplication result to the time domain. Once the reduction is completed using redundant Montgomery method, a forward DFT transform is applied to grasp the spectral coefficients.

At first glance, the partial return of the Alg. 2 seems advantageous over Alg. 1 requiring full forward and backward DFTs. However, if the arithmetic requirements of both algorithms are evaluated step-by-step (i.e. given in Tables 1 & 2) and further summed up in Table 3, it is easily seen that full return algorithm needs less additions and hence behaves better than the partial return one.

Another comparison concern is the memory requirements of both algorithms. As both algorithms enjoy the performance gain comes with the high radix Montgomery reduction, one has to pre-compute and store the basis sets. Observe that Alg. 1 and Alg. 2 require u and q sized words respectively for such allocations. If the relation 2u<q is considered (see "Saldamli, 2005" for the exact ratio), one sees that Alg. 1 is advantageous over Alg. 2.

Table 1. Step by step arithmetic requirements of Alg. 1.

Step 1	contains d multiplications.
Step 2	contains d multiplications. contains d(d−) additions and d(d−1) rotations and d constant multiplications
Step 3	none.
Step 4	none (loop d times).
Step 5	contains 1 addition.
Step 6	contains 1 addition (step 5 and 6 uses a temporary variable to avoid one addition). contains (u − 1)d + d additions, which
Step 7	makes ud. contains only memory mappings.
Step 8	none (end loop d times).
Step 9	contains 1 addition.
Step 10	contains d(d − 1) additions and d(d − 1)
Step 11	rotations.
Step 12	none.

3. 2. ASIC Performance Evaluation

Since spectral methods exploit massive parallelism, ASIC architectures are utmost suitable for their employments. In this respect, precise ASIC analysis for both algorithms have to be given for a healthy comparison. As the complexity of the multiplication for both algorithms is same, we exclude its cost from our analysis. Alg. 1 consists of three stages, namely; iDFT, reduction steps and DFT. Among those three, DFT and iDFT can be calculated with the same FFT hardware, preferably with a butterfly network taking logarithmic time with respect to the operand size.

If Alg. 1 is considered, it has a single stage, consists of reduction steps and partial return embeddings.

Table 2. Step by step arithmetic requirements of Alg. 2.

Step 1	contains d multiplications.
Step 2	none.
Step 3	none (loop d times).
Step 4	contains d − 1 additions and 1 constant multiplication.
Step 5	contains 1 addition.
Step 6	contains 1 addition (step 5 and 6 uses a temporary variable to avoid one addition).
Step 7	contains (u − 1)d + d additions, which makes ud.
Step 8	contains d + 1 additions.
Step 9	contains d rotations.
Step 10	none (loop d times).
Step 11	contains d additions.
Step 12	none.

This stage loops d times and calculates a single reduction step in one clock cycle as seen in Figure 2.

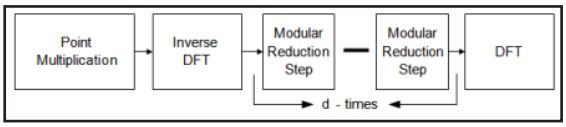

Figure 2. Steps of Alg. 1.

On the other hand, the loop of Alg. 2 contains a partial return, which calculates the value z0. As mentioned before this single word computation takes the same logarithmic time as the full iDFT calculation. Since this partial return is computed at every iteration of the loop as it can be seen in Figure 3, the rest of the remaining steps in both algorithms have similar complexities.

Moreover, if redundancy and bound control, and smaller sized precomputation (see "Saldamli, 2005") are considered Alg. 1 again outperforms. If presented formally; let reduction and iDFT times are denoted by T_{red} and T_{iDFT} respectively, then

$$T_{Alg.1} = d \cdot T_{red} + T_{DFT} + T_{iDFT}$$

and

$$T_{Alg.2} = d \cdot (T_{red} + T_{iDFT})$$

Here, note that $T_{DFT} < T_{iDFT}$ because of the constant multiplication.

Table 3. Arithmetic performance of Alg. 1 & Alg. 2.

	Alg. 1	Alg. 2
Multiplication	d	d
Constant Multiplication	d	d
Addition	$(u+2)\cdot d^2+1$	$(u+3)\cdot d^2+2d$
Shift and Rotate	2d(d-1)	d.d
Stored Memory (bits)	u.b.d	q.b.d

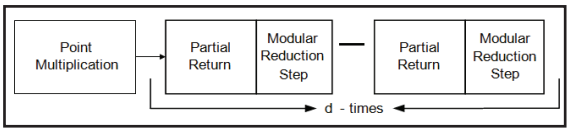

Figure 3. Steps of Alg. 2.

Above analysis demonstrates a fair comparison of both algorithms. In fact, one can equipped Alg. 1 with more features that one can not do that with Alg. 2. For instance; better parameters on the encoding and decoding can be chosen while transforming to the non-redundant form. With these parameters, Montgomery reduction requires less values to store and can be calculated faster as described by some references (Tenca and Koc, 1999; Todorov et al., 2001; Bunimov and Schimmler, 2003).

As a last remark, we remind that in our analysis we reference the worst case DFT and iDFT computa- tion. The analysis of fast Fourier transform algorithms are beyond the scope of this text. However; in real world applications one should benefit the fruits of this mature methods. We refer the reader to textbook presentations for such discussions (Nussbaumer, 1982; Blahut, 1985).

4. SPECTRAL MODULAR ARITHMETIC FOR FINITE FIELD EXTENSIONS

In this section, we turn our attention to the arithmetic in the extension fields and revisit two methods of multiplication including an adaption of Schönhage and Strassen's algorithm and the algorithm of Baktir et al. (Schönhage and Strassen, 1971; Baktir et al., 2007).

Abstractly, a finite field consists of a finite set of objects together with two binary operations (addition and multiplication) that can be performed on pairs of field elements. These binary operations must satisfy certain compatibility properties. There is a finite field containing q field elements if and only if q is a power of a prime number, and in fact for each such q there is precisely one finite field denoted by GF (q). When q is prime the finite field is called a prime field whereas if $q = p^k$ for a prime p and k>1, the finite field GF (p^k) is called an extension field. The number p is named as the characteristic of the finite field and in case of p = 2, the extension field is called a binary extension field.

The extension field GF (p^k) can be represented by the set of polynomials with polynomial addition and multiplication modulo an irreducible polynomial f (t) over GF (p) having degree k. The degree of the polynomial f (t) is also referenced as the degree of the extension. In fact, the defining polynomial f (t) characterizes the structure of the mathematical object consist of the polynomial congruent classes. Since for every prime power q there exists a unique finite field, the structure of the finite field does not depend on the choice of the defining polynomial as long as it is an irreducible having degree k.

Being a polynomial ring, the arithmetic in extension fields is the familiar modular polynomial arithmetic. Since the characteristic is p, addition is performed by adding polynomials modulo p whereas multiplication involves a polynomial multiplication and a reduction with respect to the defining irreducible polynomial f (t).

We assume that the parameter p and f (t) can arbitrarily be chosen without concerning about the security of a cryptosystem defined over the extension field. Certainly, our first choice for p would be a Mersenne prime enjoying the one's complement arithmetic. Similarly we would tend to choose f (t) as a low hamming weight polynomial such as a binomial or a trinomial. Moreover, we would insist on fixing the coefficients of f (t) to powers of two, so that multiplications on the coefficients enjoys shifts instead of full multiplications.

Obviously, the above extension field selection exploits the spectral algorithms built over it. If this is furnished with the selection of DFT parameter ω as a power of 2, one would utilize the best performing spectral algorithm setup. For instance, in a study, such a selection is presented by choosing $f(t) = t^k - 2$, $\omega = -2$ and p a Mersenne prime such as $2^{13} - 1$ or $2^{19} - 1$ (Baktir et al., 2007).

Notice that the choice of the field generation polynomial as $t^k - 2$ makes the reduction simpler. Indeed, instead of sequential Montgomery

reduction taking $t^k = 2$ and simple adding at once has better approach in time domain. The next algorithm presented under this consideration. Therefore; it does not includes a Montgomery reduction step.

Algorithm 3 Standard DFT modular multiplication for GF (p^k).

Suppose that there exist a d-point DFT map for some principal root of unity ω in Zq, and X (t) and Y (t) are transform pairs of x(t) and y(t) respectively

Input: $d > 2k - 1$, $f(t) = t^k - \omega$, X (t), Y (t),
Output: Z (t) where iDF T (Z (t)) ≡ x(t) · y(t) ∈ GF (p^k)

1: Z (t) := X (t) ⊚ Y (t)
2: z(t) := I DF T_t^w (Z (t))
3: for j = 0 to k – 2 do
4: zj (t) := zj (t) + ωz_{j+k}
5: end for
6: Z (t) := DF T (z(t))
7: return Z

DFT Modular Algorithm, which is defined by (Baktır, 2008) performs Montgomery reduction in spectral domain. Let $f_N(t) = f(t)/f(0)$ be normalized field generating polynomial and $F_{Ni} = f_i / f(0)$.

Algorithm 4 DFT modular multiplication for GF (p^k) Suppose that there exist a d-point DFT map for some principal root of unity ω in Z_q, and X (t),Y (t), A(t) and f_N (t) are transform pairs of x(t),y(t),a(t) and f_N (t) respectively, where a(t) = t

Input: $d > 2k - 1$, N (t) , X (t), Y (t),
Output: Z where iDF T (Z) ≡ x(t) · y(t) · $x^{(k-1)}$ ∈GF (p^k)

1: for i = 0 to d – 1 do
2: $Z_i := X_i \cdot Y_i$
3: end for
4: for j = 0 to k – 2 do
5: S := 0
6: for i = 0 to d – 1 do
7: $S := S + Z_i$
8: end for
9: S := –S/d
10: for i = 0 to d – 1 do
11: $C_i := (Z_i + F_{Ni} \cdot S) \cdot A_i^{-1}$
12: end for
13: end for
14: return (Z)

5. COMPARISON OF ALGORITHMS 3 AND 4

5. 1. Arithmetic Performance

In fact, the discussion in Section 3 while comparing Algorithms 1 and 2 is clearly valid in here also. Notice that likewise in Alg. 2 there exist a partial return in Alg. 4. However, this time not all of inverse DFT is calculated by Alg. 2 over the loop, since we have $d > 2k - 1$. More precisely $d = 2k$, see (Baktır, 2008).

The selection of $t^k - 2$ does not only improve performance of the Al. 4 but also of the Alg. 3. With this selection as we mentioned above there is no need for a sequential reduction or Montgomery process. That is the reason why Alg. 3 has no reduction loops. Tables 4 & 5 tabulated the arith- metic requirements of both algorithms. Table 6 simply summarizes these two tables in addition to memory requirements for necessary pre-computations. From this analysis, we conclude that Alg. 4 has better arithmetic performance over Alg. 3.

Table 4. Step by step arithmetic requirements of Alg. 3.

Step 1	contains 2k multiplications.
Step 2	contains $2k^2$ additions and $2k^2 - 2k$ rotations and 2k − 1 constant multiplications.
Step 3	none (loop k times).
Step 4	contains 1 constant multiplications and 1 addition.
Step 5	none (end loop k times).
Step 6	contains $k^2 + k$ additions and k^2 rotations.
Step 7	none.

5. 2. ASIC Performance Evaluation

The ideas of Section 3.2 discussing the ASIC performance evaluation can be applied in here also. In the light of these ideas, the simple reduction of the Alg. 3 gives much better performance.

Putting these in a more formal setting gives the following analysis. Suppose that T_{sRed} and Tred are the time of the reductions of Algorithms 3 and 4, respectively. Let $T_{DF\ T}$ and $T_{iDF\ T}$ be the times of DFT and inverse DFT to be performed, respectively, then

$$T_{Alg.3} = T_{DFT} + T_{sRed} + T_{iDFT},$$

and

$$T_{Alg.4} = m\ .\ (T_{iDFT} + T_{Red})$$

Clearly, the above analysis shows the superiority of the Alg. 3 over Alg. 4.

Table 5. Step by step arithmetic requirements of Alg. 4.

Step 1	none (loop d times = 2k times).
Step 2	1 multiplication.
Step 3	none (end loop d times = 2k times).
Step 4	loop (k-1 times).
Step 5	none.
Step 6	none (loop d times = 2k times).
Step 7	1 addition.
Step 8	none (end loop d times = 2k times).
Step 9	1 constant multiplication.
Step 10	none (loop d times = 2k times).
Step 11	1 additions, 1 rotates.
Step 12	none (end loop d times = 2k times).
Step 13	none (end loop k-1 times)
Step 14	none.

6. CONCLUSIONS

In this study, we compare partial and full return modular multiplication algorithms proposed for ring of integers and finite field extensions. Our comparison is based on a step-by-step evaluation of their arithmetic operations and ASIC performance.

Our arithmetic performance calculations shows that although Alg. 1 requires full return to time domain, it is better choice over Alg. 2 for integer modular multiplication. When multiplication over medium size characteristic fields is taken into account, Alg. 4 is better choice over Alg. 3. Due to the zero memory requirements, Alg. 3 may become a suitable choice over Alg. 4 for some processing environments.

Table 6. Arithmetic performance of Alg. 3 & Alg. 4.

	Alg. 3	Alg. 4
Multiplication	2k	2k
Constant Multiplication	2k-1	k-1
Addition	$3k^2+2k-1$	$4k^2-4k$
Shift and Rotate	$3k^2-k+1$	$2k^2-2k$
Stored Memory (bits)	none	2kp

If the ASIC performance comparison is considered Algorithms 1&3 do not have better performance. Interestingly, although Alg. 4 has better arithmetic performance over Alg. 3, its ASIC performance is worse than its rival.

As a final remark, we conclude that all algorithms are evaluated have inputs with frequency coefficients and complete the result in spectral domain. However, when ASIC implementations are considered there must be some DFT implementations which would give further stress on these deployments.

7. REFERENCES

ANSI, 2001. X9.62-2001. Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography, Draft Version.

Baktir, S. 2008. Frequency Domain Finite Field Arithmetic for Elliptic Curve Cryptography, Ph.D. Thesis, Electrical and Computer Engineering Department, Worcester Polytechnic Institute, Worcester, MA, USA, April.

Baktir, S., Kumar, S., Paar, C. and Sunar, B. 2007. A State-of-the-Art Elliptic Curve Cryptographic Processor Operating in the Frequency Domain. Mobile Networks and Applications (MONET). 12 (4), 259–270.

Blahut, R. E. 1985. Fast Algorithms for Digital Signal Processing, Addison-Wesley Publishing Company.

Bunimov, V. and Schimmler, M. 2003. Area and Time Efficient Modular Multiplication of Large Integers. ASAP'03.

IEEE, 1999. P1363: Standard Specifications for Public-Key Cryptography, November 12, Draft Version.

Koblitz, N. 1987. Elliptic Curve Cryptosystems. Mathematics of Computation. (48), 201–209.

Miller, V. 1986. Use of Elliptic Curves Cryptography. Advances in Cryptology; Proc. Crypto'85, LNCS 218, Springer-Verlag. 417–426.

Montgomery, P. L. 1985. Modular Multiplication without Trial Division. Mathematics of Computation. 44 (170), 519–521.

NIST, 2009. Fips Pub. 186-3: Digital Signature Standard (DSS), June.

Nussbaumer, H. J. 1982. Fast Fourier Transform and Convolution Algorithms, Springer, Berlin, Germany.

Pollard, J. M. 1976. Implementation of Number Theoretic Transform. Electronics Letters. 12 (15), 378–379.

Rivest, R. L., Shamir, A. and Adleman, L. 1978. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM. 21 (2), 120–126.

Saldamlı, G. 2005. Spectral Modular Arithmetic, Ph.D. Thesis. Department of Electrical and Computer Engineering, Oregon State University.

Saldamli, G. and Koc¸ C. K. 2007. Spectral Modular Exponentiation. ARITH'07: Proceedings of the 18th IEEE Symposium on Computer Arithmetic. 123–132.

Schönhage, A. and Strassen, V. 1971. Schnelle Multiplikation Grosser Zahlen. Computing. (7), 281–292.

Tenca A. F. and Koc, C. K. 1999. A word-Based Algorithm and Scalable Architecture for Montgomery Multiplication. Lecture Notes in Computer Science, Springer-Verlag. (1717), 94–108.

Todorov, G., Tenca A. F. and Koc, C. K. 2001. High-Radix Design of a Scalable Modular Multiplier. Lecture Notes in Computer Science, Springer-Verlag. (1717), 189–206.