ASYMPTOTIC ARITHMETIC FOR POLYNOMIALS AND FORMAL POWER SERIES

ANDRZEJ CHMIELOWIEC

Abstract. This paper shows how to use fast Fourier transform (FFT) and chinese remainder theorem (CRT) to speed up polynomial and formal power series arithmetic. Method described in this article considers ability of use elements of finite fields instead of complex roots of unity. Presented algorithm is very fast and does not need floating point operations. Changing computation domain from complex field to finite field gives also possibility to apply chinese remainder theorem. This observation is crucial for further speed up and parallelisation of the algorithm.

1. Introduction

This article describes how to apply chinese remainder theorem to implement efficient arithmetic in formal power series with integer coefficients. Proposed method is well suited to compute Müller polynomials, which are necessary during elliptic curve point counting algorithm – SEA. Coefficients of those polynomials are determined using formal power series with very large precision. Application of asymptotically fast arithmetic is crucial if we want to count elliptic curve points and determine ECC (Elliptic Curve Cryptography) domains.

2. Roots of unity

We say that element $x \in K$ is a root of polynomial $A(X) \in K[X]$ iff $A(x) = 0$. In particular, every field element $x \in K$ which is root of polynomial

$$A(X) = X^n - 1$$

is called n-th root of unity. Moreover, if x is not a root of $X^d - 1$ for $d < n$ then it is called n-th primitive root of unity. The set H_n of all n-th roots of unity is a subgroup in $K^\times$. This fact is very easy to proof. It is enough to show that if $x, y \in H_n \subset K^\times$ are solutions of equation $X^n - 1 = 0$ then

$$\left(xy^{-1}\right)^n = 1.$$

This leads to conclusion that $xy^{-1} \in H_n$, which means that H_n is a group.

Example 1. Let $K = \mathbb{C}$ be a complex field. Polynomial $X^8 - 1$ has 8 complex roots which can be described as powers of $\omega_8 = e^{2\pi i\ /8}$. This leads to the following factorization scheme

$$\begin{array}{lllrcl}
 & & & (X-1) & = & (X-\omega_8^0) \\
 & & (X^2-1) & & & \\
 & & & (X+1) & = & (X-\omega_8^4) \\
 & (X^4-1) & & & & \\
 & & & (X-\omega_8^2) & = & (X-\omega_8^2) \\
 & & (X^2+1) & & & \\
 & & & (X+\omega_8^2) & = & (X-\omega_8^6) \\
(X^8-1) & & & & & \\
 & & & (X-\omega_8) & = & (X-\omega_8^1) \\
 & & (X^2-\omega_8^2) & & & \\
 & & & (X+\omega_8) & = & (X-\omega_8^5) \\
 & (X^4+1) & & & & \\
 & & & (X-\omega_8^3) & = & (X-\omega_8^3) \\
 & & (X^2+\omega_8^2) & & & \\
 & & & (X+\omega_8^3) & = & (X-\omega_8^7)
\end{array}$$

The sequence of polynomial factors is not random. It is known that for every polynomial of the form $X^{2^n}-1$ there exists a sequence of its factors such that

$$\prod_{j=2^k}^{2^k+2^l}(X-\omega_n^{i_j})$$

is binomial. This observation is crucial from the point of implementation of fast Fourier transform.

In the next part of the article we will consider only those roots of unity whose degree is a power of 2. From now on we assume that $n=2^m$ and polynomial X^n-1 has exactly n roots $\omega^0,\omega^1,\dots,\omega^{n-1}$.

Lemma 2. *If $\Phi_{0,k}=X-\omega^{l_k}$ and $l_k=\sum_{j=0}^{m-1}\left(\left\lfloor\frac{k}{2^j}\right\rfloor \mod 2\right)\cdot 2^{m-1-j}$ then*

$$\Phi_{j,k}=\Phi_{j-1,2k}\Phi_{j-1,2k+1}$$

are binomials with nonzero free coefficient and degree equal to 2^j.

Before we proof the above lemma, we explain correlation between root power and its position in the factorization sequence. In the lemma there is an expression

$$l_k=\sum_{j=0}^{m-1}\left(\left\lfloor\frac{k}{2^j}\right\rfloor \mod 2\right)\cdot 2^{m-1-j}$$

which looks very complicated. Fortunately its interpretation is extremely easy. If we write number k in the binary form with m digits (possibly with leading zeros) $k=\sum_{j=0}^{m-1}b_j2^j$, then $l_k=\sum_{j=0}^{m-1}b_{m-1-j}2^j$. It means that l_k is constructed from k by reversing order of bits in the representation.

Proof. From the recursion for $\Phi_{j,k}$ we have

$$\Phi_{j,k} = \prod_{i=2^jk}^{2^j(k+1)-1} \Phi_{0,i} = \prod_{i=2^jk}^{2^j(k+1)-1} \left(X - \omega^{l_i}\right).$$

Based on the remark made before the proof, we can conclude that if i runs all numbers from the set $\{2^jk+r : 0 \le r < 2^j\}$ then l_i runs all numbers from the set $\{2^{m-j}r+k' : 0 \le r < 2^j\}$. Where number k' is made from k by bit reversion in the binary representation and is equal to l_{2^jk}. Now we can write the following equation

$$\Phi_{j,k} = \prod_{r=0}^{2^j-1} \left(X - \omega^{2^{m-j}r+k'}\right).$$

Assuming that $\alpha = \omega^{k'}$ and $\beta = \omega^{2^{m-j}}$ we get simpler formula

$$\Phi_{j,k} = \prod_{r=0}^{2^j-1} (X - \alpha\beta^r) = \alpha^{2^j} \prod_{r=0}^{2^j-1} \left(\frac{X}{\alpha} - \beta^r\right).$$

But powers of β generate all roots of unity of degree 2^j. It means that product in above equation represents polynomial $(X/\alpha)^{2^j} - 1$ and final expression for $\Phi_{j,k}$ may be assembled to the form

$$\Phi_{j,k} = X^{2^j} - \alpha^{2^j} = X^{2^j} - \omega^{2^jk'}.$$

This ends the proof of the lemma. □

Example 3. We will show how the above lemma works in practice. Let $K = \mathbb{C}$ and roots of unity be powers of $\omega_8 = e^{2\pi i/8}$

$k = 0 = (0,0,0)_2$	$l_0 = (0,0,0)_2 = 0$
$k = 1 = (0,0,1)_2$	$l_1 = (1,0,0)_2 = 4$
$k = 2 = (0,1,0)_2$	$l_2 = (0,1,0)_2 = 2$
$k = 3 = (0,1,1)_2$	$l_3 = (1,1,0)_2 = 6$
$k = 4 = (1,0,0)_2$	$l_4 = (0,0,1)_2 = 1$
$k = 5 = (1,0,1)_2$	$l_5 = (1,0,1)_2 = 5$
$k = 6 = (1,1,0)_2$	$l_6 = (0,1,1)_2 = 3$
$k = 7 = (1,1,1)_2$	$l_7 = (1,1,1)_2 = 7$

Computed powers are the same as the ones from previous example.

3. Fast Fourier transform and polynomial multiplication

The main aim of this article is to show how FFT can be used to speed up arithmetic in the rings of polynomials and formal power series. We will explain how the change of polynomial representation leads to very efficient multiplication algorithms.

3.1. **Discrete Fourier transform.** In this part we will show how to change polynomial representation efficiently. Our aim is to represent polynomial as a set of values in the roots of unity. During our consideration we will assume that $n = 2^m$ and polynomial $\Phi_n(X) = X^n - 1 \in K[X]$ has exactly n roots, that are denoted by $\omega^0, \omega^1, \ldots, \omega^{n-1}$. The following lemma gives basic result which is going to be very useful in the next part of the article

Lemma 4. *Let $x \in K$ and $A, B, C, R \in K[X]$. If $A \mod B = R$ and $B \mod C = 0$, then*

$$A(x) = A \mod (X - x) \qquad \text{and} \qquad A \mod C = R \mod C.$$

Proof. To proof first part we assume that $A(X) = \sum_{j=0}^{n-1} a_j X^j$. The following relation

$$X^k = (X^{k-1} + xX^{k-2} + \cdots + x^{k-2}X + x^{k-1})(X - x) + x^k,$$

shows that $X^k \mod (X - x) = x^k$. But operation $\mod$ is natural homomorphism of ring $K[X]$, thus we have

$$\begin{aligned} A \mod (X - x) &= \left(\sum_{j=0}^{n-1} a_j X^j \right) \mod (X - x) \\ &= \sum_{j=0}^{n-1} a_j \left(X^j \mod (X - x) \right) \\ &= \sum_{j=0}^{n-1} a_j x^j = A(x), \end{aligned}$$

This ends the proof of the first part of lemma. From the second relation we know that if $A \mod B = R$, then exists polynomial $D \in K[X]$ such that $A = D \cdot B + R$. If we add condition that $B \mod C = 0$ then we conclude

$$\begin{aligned} A \mod C &= (D \cdot B + R) \mod C \\ &= (D \mod C)(B \mod C) + (R \mod C) \\ &= R \mod C. \end{aligned}$$

And this is the end of the proof. □

Lemmas 2 and 4 give very fast algorithm of finding polynomial values in the roots of unity.

Theorem 5. ***(Discrete Fourier transform)*** *Let $A \in K[X]$ be polynomial of degree less than $n = 2^m$. We also assume that $\Phi_{0,k} = X - \omega^{l_k}$ for*

$$l_k = \sum_{j=0}^{m-1} \left(\left\lfloor \frac{k}{2^j} \right\rfloor \mod 2 \right) \cdot 2^{m-1-j}$$

and

$$\Phi_{j,k} = \Phi_{j-1,2k} \Phi_{j-1,2k+1}.$$

If the sequence $A_{j,k}$ is defined as follows

$$A_{j,k} = A_{j+1,\lfloor k/2 \rfloor} \mod \Phi_{j,k} \qquad \textit{and} \qquad A_{m,0} = A,$$

then all its elements can be computed using $m \cdot n$ multiplications in K and $A_{0,k} = A(\omega^{l_k})$.

Proof. First we show that $A_{0,k} = A(\omega^{l_k})$. To do this we consider sequence of operations that leads to $A_{0,k}$.

$$\begin{aligned}
A_{0,k} &= \left(A_{1,\lfloor k/2 \rfloor} \mod \Phi_{0,k}\right) \\
&= \left(A_{2,\lfloor k/2^2 \rfloor} \mod \Phi_{1,\lfloor k/2 \rfloor}\right) \mod \Phi_{0,k} \\
&\vdots \\
&= \left(\left(\ldots\left(A_{m,\lfloor k/2^m \rfloor} \mod \Phi_{m,\lfloor k/2^{m-1} \rfloor}\right)\ldots\right) \mod \Phi_{1,\lfloor k/2 \rfloor}\right) \mod \Phi_{0,k}
\end{aligned}$$

But $k < n = 2^m$ and $A_{m,0} = A$. This leads us to the relation

$$A_{0,k} = \left(\left(\ldots\left(A \mod \Phi_{m,\lfloor k/2^{m-1} \rfloor}\right)\ldots\right) \mod \Phi_{1,\lfloor k/2 \rfloor}\right) \mod \Phi_{0,k}.$$

Analysing formula for $\Phi_{j,k}$ we conclude that $\Phi_{j,k} \mid \Phi_{j+1,\lfloor k/2 \rfloor}$ and thus $\Phi_{0,k} \mid \Phi_{1,\lfloor k/2 \rfloor} \mid \cdots \mid \Phi_{m,\lfloor k/2^{m-1} \rfloor}$. Based on lemma 4 we have

$$A_{0,k} = A \mod \Phi_{0,k} = A(\omega^{l_k}).$$

To count multiplications necessary for sequence $A_{j,k}$ determination we will use results from lemma 2. Observe that polynomial $A_{j,k}$ is determined by finding reduction of $A_{j+1,\lfloor k/2 \rfloor}$ modulo binomial $\Phi_{j,k} = X^{2^j} - \alpha^{2^j}$. This process needs 2^j multiplications in K

$$\begin{aligned}
A_{j+1,\lfloor k/2 \rfloor} \mod \Phi_{j,k} &= \left(\sum_{i=0}^{2^{j+1}-1} a_i X^i\right) \mod \left(X^{2^j} - \alpha^{2^j}\right) \\
&= \sum_{i=0}^{2^j-1} a_i X^i + \alpha^{2^j} \sum_{i=0}^{2^j-1} a_{2^j+i} X^i \\
&= \sum_{i=0}^{2^j-1} \left(a_i + \alpha^{2^j} \cdot a_{2^j+i}\right) X^i.
\end{aligned}$$

This means that determination of single element $A_{j,k}$ needs $\deg\left(\Phi_{j,k}\right)$ multiplications in K. For every recursion level there is a need to make n multiplications because $\prod_k \Phi_{j,k} = \Phi_n = X^n - 1$. There are m recursion levels and to find all $A_{j,k}$ we have to make $m \cdot n$ multiplications. □

The following example illustrates discrete Fourier transform in practice.

Example 6. Let $K = \mathbb{F}_{17}$. All elements of this field are 16-th roots of unity. In this example we only use roots of degree 4: $\omega^0 = 1, \omega^1 = 13, \omega^2 = 16, \omega^3 = 4$. From previous considerations we get the following hierarchy of $\Phi_{j,k}$ polynomials.

$$\begin{array}{ccc} & & \Phi_{0,0} = X - 1 \\ & \Phi_{1,0} = X^2 - 1 & \\ & & \Phi_{0,1} = X - 16 \\ \Phi_{2,0} = X^4 - 1 & & \\ & & \Phi_{0,2} = X - 13 \\ & \Phi_{1,1} = X^2 - 16 & \\ & & \Phi_{0,3} = X - 4 \end{array}$$

Suppose that we want to find values of polynomial $A(X) = X^3 + 2X^2 + 3X + 4$ in points $\omega^0, \dots, \omega^3$. Making use of theorem 1 we get:

$$\begin{array}{rcl} A_{2,0} & = & X^3 + 2X^2 + 3X + 4 \\ \\ A_{1,0} & = & A_{2,0} \mod \Phi_{1,0} = (X^3 + 2X^2 + 3X + 4) \mod (X^2 - 1) \\ & = & 4X + 6 \\ A_{1,1} & = & A_{2,0} \mod \Phi_{1,1} = (X^3 + 2X^2 + 3X + 4) \mod (X^2 - 16) \\ & = & 2X + 2 \\ \\ A_{0,0} & = & A_{1,0} \mod \Phi_{0,0} = (4X + 6) \mod (X - 1) \\ & = & 10 = A(1) \\ A_{0,1} & = & A_{1,0} \mod \Phi_{0,1} = (4X + 6) \mod (X - 16) \\ & = & 2 = A(16) \\ A_{0,2} & = & A_{1,1} \mod \Phi_{0,2} = (2X + 2) \mod (X - 13) \\ & = & 11 = A(13) \\ A_{0,3} & = & A_{1,1} \mod \Phi_{0,2} = (2X + 2) \mod (X - 4) \\ & = & 10 = A(4) \end{array}$$

3.2. **Inverse discrete Fourier transform.** To complete our considerations we have to show how to determine inverse Fourier transform and how to describe polynomial by its coefficients.

Theorem 7. ***(Inverse discrete Fourier transform)*** *Let* $\Phi_{0,k} = X - \omega^{l_k}$,

$$l_k = \sum_{j=0}^{m-1} \left(\left\lfloor \frac{k}{2^j} \right\rfloor \mod 2 \right) \cdot 2^{m-1-j}$$

and

$$\Phi_{j,k} = \Phi_{j-1,2k} \Phi_{j-1,2k+1}.$$

Denote by $A \in K[X]$ *polynomial with degree less than* $n = 2^m$ *and assume that values of* $A(\omega^0), A(\omega^1), \dots, A(\omega^{n-1})$ *are known. If sequence* $A_{j,k}$ *is defined as follows*

$$A_{j,k} = A_{j+1,\lfloor k/2 \rfloor} \mod \Phi_{j,k} \qquad \text{and} \qquad A_{0,k} = A(\omega^{l_k}),$$

then all its elements can be computed using $2 \cdot m \cdot n$ *multiplications in* K *and* $A_{m,0} = A$.

Proof. From lemma 2 we know that binomials $\Phi_{j,k}$ have form $X^{2^j} - \alpha^{2^j}$, where α may be different for every $\Phi_{j,k}$. Since $\Phi_{j,k} = \Phi_{j-1,2k}\Phi_{j-1,2k+1}$ is a product of polynomials with the same degree, then

$$\Phi_{j-1,2k} = X^{2^{j-1}} - \alpha^{2^{j-1}} \qquad \text{and} \qquad \Phi_{j-1,2k+1} = X^{2^{j-1}} + \alpha^{2^{j-1}}.$$

Determination of sequence $A_{j,k}$ is impossible with formula given in the theorem, because we have only values of $A_{0,k}$. We need another condition which would allow to determine sequence in reverse direction. The following relation gives necessary formula

$$A_{j,k} = \frac{1}{2}\left(A_{j-1,2k} + A_{j-1,2k+1}\right) + \frac{X^{2^{j-1}}}{2\alpha^{2^{j-1}}}\left(A_{j-1,2k} - A_{j-1,2k+1}\right).$$

To prove the above equation it is enough to show that $A_{j-1,2k} = A_{j,k} \mod \Phi_{j-1,2k}$ and $A_{j-1,2k+1} = A_{j,k} \mod \Phi_{j-1,2k+1}$. But it is obvious since

$$A_{j,k} \mod \Phi_{j-1,2k} =$$
$$\left(\frac{1}{2}\left(A_{j-1,2k} + A_{j-1,2k+1}\right) + \frac{X^{2^{j-1}}}{2\alpha^{2^{j-1}}}\left(A_{j-1,2k} - A_{j-1,2k+1}\right)\right) \mod \left(X^{2^{j-1}} - \alpha^{2^{j-1}}\right) =$$
$$\frac{1}{2}\left(A_{j-1,2k} + A_{j-1,2k+1}\right) + \frac{1}{2}\left(A_{j-1,2k} - A_{j-1,2k+1}\right) = A_{j-1,2k}$$

$$A_{j,k} \mod \Phi_{j-1,2k+1} =$$
$$\left(\frac{1}{2}\left(A_{j-1,2k} + A_{j-1,2k+1}\right) + \frac{X^{2^{j-1}}}{2\alpha^{2^{j-1}}}\left(A_{j-1,2k} - A_{j-1,2k+1}\right)\right) \mod \left(X^{2^{j-1}} + \alpha^{2^{j-1}}\right) =$$
$$\frac{1}{2}\left(A_{j-1,2k} + A_{j-1,2k+1}\right) - \frac{1}{2}\left(A_{j-1,2k} - A_{j-1,2k+1}\right) = A_{j-1,2k+1}.$$

One can see that reverse formula needs 2 times more multiplications. Thus total number of multiplications for inverse discrete Fourier transform is equal to $2\cdot m\cdot n$. □

Fast Fourier transform allows to construct efficient method for polynomial multiplication. Idea of this algorithm is very simple and can be described in three steps.

1. Transformation of polynomials $A, B \in K[X]$.
2. Scalar multiplication of point values.
3. Inverse transformation of multiplication result.

4. Asymptotically fast arithmetic in the ring of formal power series

In this section we will describe fast methods for multiplication and division in the ring of formal power series with integer coefficients. In our considerations we assume that only n first positions of power series are significant and n is a power of 2. This means in fact that it is arithmetic with precision n. Multiplication of two power series with limited precision looks the same as multiplication of two polynomials. What we have to do is finding finite field $\mathbb{F}_p$ in which computations would be done. Prime p should be large enough to eliminate modular reduction during multiplication process. It means that p must satisfy two following conditions:

(1) $4R^2 \cdot \lfloor \log_2 n + 1 \rfloor < p$ – there is no modular reduction in the result of computations,
(2) $p = 2^{m+1}r + 1$ for some $2^{m+1} \geq 2n$ – there are roots of unity well suited for discrete Fourier transform.

Changing complex number field to finite field can be regarded as nonsense movement. But this operation gives ability to remove expensive floating point operations and substitute them for fast integer arithmetic. This leads to faster algorithms because floating point multiplication is about 30 times slower than integer multiplication on Intel Core 2 processors. So, there are two main reasons for applying this kind of field:

(1) increase in speed of algorithm,
(2) elimination of error control during floating point operations.

Of course there is a question if this method is really faster? In the case of complex field we have real floating point multiplications, but single multiplication in finite field consists of integer multiplication and modular reduction. The problem of modular reduction can be solved in two different ways. First idea is to find such prime number for which reduction can be done using only few additions and subtractions. There are many such primes, for example $2^{224} - 2^{96} + 1 = 2^{96}(2^{128} - 1) + 1$ and $2^{512} - 2^{32} + 1 = 2^{32}(2^{480} - 1) + 1$. The second idea is based on Montgomery multiplication algorithm [2]. This method can be applied for odd modules and it does not need integer division to perform modular reduction. Special procedure gives ability to compute reduction with only two multiplications. Application of classic modular reduction is also possible. This approach does not change asymptotic complexity of entire algorithm, but is hard to implement in efficient way [4]. This is the reason why we recommend using field defined by prime with special form and extremely fast modular reduction.

Now we calculate computational complexity of presented method. As we mentioned earlier, power series multiplication consists of: Fourier transform, scalar multiplication and inverse Fourier transform. In all those operations the main cost is related with multiplication in field $\mathbb{F}_p$. Thus we estimate computational complexity by the number of necessary field multiplications.

(1) Fourier transform of two power series with n coefficients – $2n\log(n)$ multiplications in $\mathbb{F}_p$.
(2) Scalar multiplication of two vectors with $2n$ coefficients – $2n$ multiplications in $\mathbb{F}_p$.
(3) Inverse Fourier transform of the result to the power series with $2n$ coefficients – $n\log(n)$ multiplications $\mathbb{F}_p$.

We can see that cost of formal power series asymptotic multiplication is about $n(2 + 3\log(n))$ multiplications in $\mathbb{F}_p$. But when it is reasonable to use asymptotic method instead of classic algorithm? To answer this question, first we have to compare classic integer multiplication with $\mathbb{F}_p$ multiplications. We will consider the following two cases:

(1) Multiplication in $\mathbb{F}_p$ for special form of the prime p, which is about 6 times more expensive than integer coefficient multiplication. In this cast FFT is

better than classic if

$$6 \cdot n(2 + 3\log(n)) < n^2.$$

It means that if $n > 140$ then asymptotic method is faster than classic one.

(2) Multiplication in $\mathbb{F}_p$ with Montgomery representation, which is about 12 times more expensive than integer coefficient multiplication. In this cast FFT is better than classic if

$$12 \cdot n(2 + 3\log(n)) < n^2.$$

It means that if $n > 324$ then asymptotic method is faster than classic one.

To decrease complexity of the above algorithm we will use chinese remainder theorem. This approach gives ability to swap single multiplication in $\mathbb{F}_p$ by many multiplications in much smaller fields $\mathbb{F}_{p_i}$. To achieve our goal we first have to find prime numbers p_i which will be well suited for this method. Quick look at the conditions for p gives the following requirements:

(1) $4R^2 \cdot \lfloor \log_2 n + 1 \rfloor < \prod p_i$ – there is no modular reduction in the result of computations,
(2) $p_i = 2^{m+1} r_i + 1$ for some $2^{m+1} \geq 2n$ – there are roots of unity well suited for discrete Fourier transform.

We show that this purpose is much more efficient than method proposed at the beginning of this section. To get maximum performance there is a need to choose such primes p_i which can be stored in single processor register. It is easy to achieve with 32-bit processors and we do not discuss this aspect wider. Suppose now that we have k prime numbers p_i that have the same bit length and meet conditions described above. New multiplication algorithm for formal power series can be expressed as follows:

(1) Coefficient reduction modulo every chosen prime – $c_1 k^2 n$ multiplications in $\mathbb{F}_{p_i}$.
(2) For all $i \in \{1, \ldots, k\}$ we make FFT multiplication:
    (a) Fourier transform of two power series with n coefficients – $2n\log(n)$ multiplications in $\mathbb{F}_{p_i}$,
    (b) scalar multiplication of two vectors with $2n$ coefficients – $2n$ multiplications in $\mathbb{F}_{p_i}$,
    (c) inverse Fourier transform of the result to the power series with $2n$ coefficients – $n\log(n)$ multiplications $\mathbb{F}_{p_i}$.
(3) Use chinese remainder theorem to get back result coefficients – $c_2 k^2 n$ multiplications in $\mathbb{F}_{p_i}$.

Since numbers p_i have the same bit size, then we assume that multiplication cost in every field $\mathbb{F}_{p_i}$ is equal. This means that the total complexity of new algorithm is equal to

$$c_1 k^2 n + kn(2 + 3\log(n)) + c_2 k^2 n$$

multiplications in $\mathbb{F}_{p_i}$. To compare this result with the previous version of the algorithm we have to know that single multiplication in field $\mathbb{F}_p$ is equivalent to k^2

multiplications in $\mathbb{F}_{p_i}$. Thus complexity of new method can be also expressed as

$$\frac{n(2+3\log(n))}{k} + n(c_1+c_2)$$

multiplications in $\mathbb{F}_p$. One can see that application of chinese remainder theorem leads us to algorithm which is asymptotically k times faster then the original. Of course constants c_1 and c_2 play a crucial role in the practical applications. Thus number of coefficients for which new algorithm is really faster have to be determined during numerical experiments. We cannot give values of c_1 and c_2 in direct form, because they are strongly related to processor architecture and algorithms used in points 1 and 3.

Division of formal power series needs finding an inverse. Using classical method to compute inverse is unacceptable, because we lose all performance given by FFT multiplication algorithm. Fortunately there exists a very simple method which allows to compute invert power series – simple Newton iteration method to invert elements of p-adic rings [1]. This algorithm is fast and uses only multiplications, additions and subtractions. To invert formal power series with n coefficients we need about $\log n$ iterations. If we have series

$$A = \sum_{j=0}^{n-1} a_j X^j,$$

for which first coefficient is invertible, then the following procedure computes inverted series with precision n:

```
1.    $m \leftarrow 0$;
2.    $B \leftarrow \frac{1}{a_0}$;
3.    while $2^m < n$ do
3.1.      $B \leftarrow 2B - B^2 \sum_{j=0}^{2^m} a_j X^j$;
3.2.      $m \leftarrow m + 1$;
4.    return $B$;
```

One can see that every iteration of Newton method doubles precision of the result.

5. Summary

The second version of the algorithm for power series multiplication has two main advantages:

(1) smaller numerical complexity,
(2) possibility to make all computations parallel.

Chinese remainder theorem gives possibility to split computations during many processors or even computers. All parts of the algorithm responsible for operations in $\mathbb{F}_{p_i}$ fields can be distributed and run parallel on different machines.

Proposed method is well suited to compute Müller polynomials, that are necessary during elliptic curve point counting algorithm – SEA. Coefficients of those polynomials are determined using formal power series with very large precision. Application of asymptotically fast arithmetic is crucial if we want to count elliptic curve points and determine ECC (Elliptic Curve Cryptography) domains.

We should also mention the possibility of FFT parallelisation [5]. It is quite easy to do because there is no correlation between coordinates of transformed vector. Especially efficient architecture we can get using SIMD instructions (Single Instruction Multiple Data). This kind of instructions is dedicated to make the same operation on many arguments and this is the case in FFT.

References

[1] Fernando Q. Gouvêa, *p-adic Numbers*, Springer-Verlag, 1993.
[2] Alfred J. Menezes, Paul C. Van Oorschot, Scott A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.
[3] Thomas H Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Introduction to Algorithms*, MIT Press, 2003.
[4] Donald E. Knuth, *Art of Computer Programming*, Addison-Wesley Professiona, 1998.
[5] Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar, *Introduction to Parallel Computing*, Addison Wesley, 2003.

E-mail address: andrzej.chmielowiec@cmmsigma.eu