Fast, Parallel Algorithm for Multiplying Polynomials with Integer Coefficients

Andrzej Chmielowiec

***Abstract*—This paper aims to develop and analyze an effective parallel algorithm for multiplying polynomials and power series with integer coefficients. Such operations are of fundamental importance when generating parameters for public key cryptosystems, whereas their effective implementation translates directly into the speed of such algorithms in practical applications. The algorithm has been designed specifically to accelerate the process of generating modular polynomials, but due to its good numerical properties it may surely be used to solve other computational problems as well. The basic idea behind this new method was to adapt it to parallel computing. Nowadays, it is a very important property, as it allows to fully exploit the computing power offered by modern processors. The combination of the Chinese Remainder Theorem and the Fast Fourier Transform made it possible to develop a highly effective multiplication method. Under certain conditions, it is asymptotically faster than the algorithm based on Fast Fourier Transform when applied to multiply both: numbers and polynomials. Undoubtedly, this result is the major theoretical conclusion of this paper.**

***Index Terms*—parallel polynomial multiplication, parallel power series multiplication, FFT, CRT**

I. Introduction

In 1971 Schönhage and Strassen [15] proposed a new algorithm for large integer multiplication. Since that time, methods based on Fast Fourier Transform (FFT) have been continuously developed and upgraded. Now we have many of multiplication algorithms which are based on the FFT. They are used to multiply integers ([14], [3]) or power series ([16], [10], [11] [9]). Some of them are architecture independent and some are dedicated to a specific processor. The algorithms serve as black boxes which guarantee the asymptotic complexity of the methods using them. However, practical implementation often works in the case of such numbers for which it is ineffective to apply a fast multiplication method. The determination of modular polynomials is a good illustration of this problem. The latest methods for generating classic modular polynomials were developed by Charles, Lauter [1] and Enge [5]. Moreover, Müller [13] proposed another family of modular polynomials which may also be used in the process of counting points on an elliptic curve. The Müller's polynomials are characterized by a reduced number of non-zero coefficients and lower absolute values of coefficients, compared to classic modular polynomials. All the aforesaid authors give the computational complexity of algorithms used to determine modular polynomials based on the assumption that both polynomials and their coefficients are multiplied with the use of the Fast Fourier Transform. The complexity of such a multiplication algorithm is

$$O((n \log n)(k \log k)),$$

where n is the degree of the polynomial, and k is the number of bits of the largest factor. However, the application of an asymptotically fast algorithm to multiply numbers becomes effective only when the numbers are of considerable length. According to Garcias report [6], fast implementation of multiplication in GMP (GNU Multiple Precision Arithmetic Library) becomes as effective as classic multiplication algorithms only for numbers of at least $2^{17} = 131072$ bits. That is why it would be worth to develop a multiplication algorithm, which operates fast for polynomials with relatively low coefficients. In order to achieve that, we decided to use the Chinese Remainder Theorem (CRT). It is commonly used to accelerate the RSA algorithm by distributing computations. Inspired by this idea we extend the application of CRT to the case of polynomials with integer coefficients. In this article we propose a new method which can be used to implement efficient parallel arithmetic for the ring of polynomials with integer coefficients. This idea fits into the scheme proposed in the work [7].

The paper is organized as follows.

In Section II for completeness we briefly recall the general idea of Fast Fourier Transform. FFT may be implemented in many forms and a choice of proper implementation depends on the problem we want to solve and the processor we are using.

In Section III we show in detail how to use CRT to distribute polynomial arithmetic between many processors. Our new method is very simple both in concept and implementation. It does not need any communication between processors which is an additional advantage. This algorithm may use any implementation of FFT. Particularly it may be used with parallel FFT which reduces the total time of computation.

In Section IV we present numerical results of our implementation based on OpenMP parallel programming standard. We compare proposed method with algorithm based on FFT over large finite field.

To summarize, to multiply polynomials developer combines two independent techniques to achieve the best performance from a machine or processor:

1) distribute computations between smaller domains being polynomial rings (apply CRT),
2) optimize FFT operations within these smaller domains.

Manuscript received March 18, 2012. This paper has been supported by Polish National Science Centre grant N N516478340.

Andrzej Chmielowiec, Institute of Fundamental Technological Research, Polish Academy of Sciences, Pawinskiego 5B, 02-106 Warszawa, Poland, e-mail: achmielo@ippt.gov.pl, andrzej.chmielowiec@cmmsigma.eu.

ISBN: 978-988-19252-1-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

The whole idea is illustrated on the following scheme.

$$\begin{array}{ccccccc} & & \mathbb{F}_{p_1}[X] & \xrightarrow{FFT} & \mathbb{F}_{p_1}[X] & & \\ & & \vdots & & \vdots & & \\ \mathbb{Z}[X] & \xrightarrow{CRT} & \mathbb{F}_{p_i}[X] & \xrightarrow{FFT} & \mathbb{F}_{p_i}[X] & \xrightarrow{CRT^{-1}} & \mathbb{Z}[X] \\ & & \vdots & & \vdots & & \\ & & \mathbb{F}_{p_k}[X] & \xrightarrow{FFT} & \mathbb{F}_{p_k}[X] & & \end{array}$$

It means that multiplications in $\mathbb{Z}[X]$ can be distributed between k independent rings $\mathbb{F}_{p_i}[X]$ and each such multiplication can be done independently in parallel.

II. Fast Fourier Transform and its implementations

A Fast Fourier Transform (FFT) is an efficient algorithm to compute the Discrete Fourier Transform. The basic idea of DFT is to represent polynomials as sequences of values rather than sequences of coefficients. Computing DTF of n values using the definition takes $O(n^2)$ arithmetic operations, while FFT can compute the same result in only $O(n \log n)$ operations. This is the reason why Fast Fourier Transform plays a very important role in efficient computations and it is considered in many publications. Some of them give a general description of FFT [12], [4], [2], [8] others contain details about very fast implementations [10], [11], [9], [17]. In our numerical experiments in the last section a classic algorithm of FFT has been used. However for practical purposes we suggest application of the *cache-friendly truncated FFT* recently developed [9]. This new FFT method reduces the computational cost and is optimized against modern processor architecture.

III. Using Chinese Remainder Theorem to distribute computations between many processors

In the rest of this paper we will assume that only n first positions of power series are significant and n is a power of 2. In other words we will work with power series with precision n (reduced modulo X^n). We will also assume that the largest absolute value of series coefficients is less than B. Multiplication of two power series with limited precision is in fact the same as multiplication of two polynomials. Next we have to find family of finite fields $\mathbb{F}_{p_i}$ in which computations will be done. To do this product $\prod_{i=1}^{k} p_i$ should be large enough to eliminate modular reduction during the multiplication process.

Definition 1: Let $f(X) = f_{n-1}X^{n-1} + \cdots + f_1 X + f_0 \in \mathbb{Z}[X]$ and $M \in \mathbb{Z}$. We define $f(X) \bmod M$ as follows

$$f(X) \bmod M = (f_{n-1} \bmod M)X^{n-1} + \cdots + (f_0 \bmod M),$$

where

$$f_i \bmod M \in \left\{ \left\lfloor \frac{-M+1}{2} \right\rfloor, \ldots, -1, 0, 1, \ldots, \left\lfloor \frac{M-1}{2} \right\rfloor \right\}.$$

□

Lemma 1: Let $f(X) = f_{n-1}X^{n-1} + \cdots + f_1 X + f_0$, $g(X) = g_{n-1}X^{n-1} + \cdots + g_1 X + g_0$ be polynomials with integer coefficients such that $|f_i| < B$ and $|g_i| < B$. If integer M satisfies the following condition

$$2nB^2 < M$$

then $f(X)g(X) \bmod M = f(X)g(X)$.

Proof: If $f(X)g(X) = h(X) = h_{2n-2}X^{2n-2} + \cdots + h_1 X + h_0$ then

$$\begin{aligned} h(X) &= \left(\sum_{i=0}^{n-1} f_i X^i\right)\left(\sum_{j=0}^{n-1} g_j X^j\right) \\ &= \sum_{i=0}^{n-1}\sum_{j=0}^{i} f_j g_{i-j} X^i + \\ &\quad \sum_{i=1}^{n-1}\sum_{j=0}^{n-1-i} f_{i+j} g_{n-1-j} X^{n-1+i} \\ &= \sum_{i=0}^{n-1} X^i \sum_{j=0}^{i} f_j g_{i-j} + \\ &\quad \sum_{i=1}^{n-1} X^{n-1+i} \sum_{j=0}^{n-1-i} f_{i+j} g_{n-1-j} \end{aligned}$$

Based on the assumption that $|f_i| < B$ and $|g_i| < B$ we have

1) for all i from 0 to $n-1$ we have

$$\begin{aligned} |h_i| &= \left|\sum_{j=0}^{i} f_j g_{i-j}\right| \le \sum_{j=0}^{i} |f_j||g_{i-j}| \\ &< \sum_{j=0}^{i} B^2 = (i+1)B^2, \end{aligned}$$

2) for all i from 1 to $n-1$ we have

$$\begin{aligned} |h_{n-1+i}| &= \left|\sum_{j=0}^{n-1-i} f_{i+j} g_{n-1-j}\right| \\ &\le \sum_{j=0}^{n-1-i} |f_{i+j}||g_{n-1-j}| \\ &< \sum_{j=0}^{n-1-i} B^2 = (n-i)B^2. \end{aligned}$$

It means that $|h_i| < nB^2$ for all i from 0 to $2n-2$. If $M > 2nB^2$ then all coefficients (represented like in Definition 1) of $f(X), g(X)$ and $h(X)$ can be represented in residue system modulo M without reduction. This leads to the formula $f(X)g(X) \bmod M = f(X)g(X)$ and ends proof. ■

Theorem 1: Let $f(X) = f_{n-1}X^{n-1} + \cdots + f_1 X + f_0$, $g(X) = g_{n-1}X^{n-1} + \cdots + g_1 X + g_0$ be polynomials with integer coefficients such that $|f_i| < B$ and $|g_i| < B$. If prime numbers p_i satisfy the following conditions:

(1) $p_i \neq p_j$,
(2) $M = \prod_{i=1}^{k} p_i$,
(3) $2nB^2 < \prod p_i$ = M,
(4) $p_i = 2^{m+1}r_i + 1$ for some $2^{m+1} \ge 2n$ and $r_i \in \mathbb{Z}$,

ISBN: 978-988-19252-1-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

then

$$\begin{aligned} f(X)g(X) &= f(X)g(X) \bmod M \\ &= (f(X) \bmod M)(g(X) \bmod M) \bmod M \end{aligned}$$

and fields $\mathbb{F}_{p_i}$ can be used to parallel multiplication of polynomials f and g with FFT method.

Proof: Since operation $\bmod M$ is a natural homomorphism of $\mathbb{Z}$ then we have

$$(f(X) \bmod M)(g(X) \bmod M) \bmod M =$$

$$f(X)g(X) \bmod M$$

Based on Lemmma 1 we achieve the second equality

$$f(X)g(X) \bmod M = f(X)g(X).$$

It means that multiplication of $g(X), f(X) \in \mathbb{Z}[X]$ gives the same result as multiplication of $g(X) \bmod M, f(X) \bmod M \in (\mathbb{Z}/M\mathbb{Z})[X]$ if elements of ring $\mathbb{Z}/M\mathbb{Z}$ are represented by $\{-\frac{M-1}{2}, \ldots, -1, 0, 1, \ldots, \frac{M-1}{2}\}$. But M is a product of different primes p_i and the Chinese Remainder Theorem implies the following isomorphism:

$$\mathbb{Z}/M\mathbb{Z} \simeq \mathbb{F}_{p_1} \times \cdots \times \mathbb{F}_{p_k}.$$

It is clear that the above isomorphism can be extended to isomorphism of polynomial rings, more precisely we have:

$$(\mathbb{Z}/M\mathbb{Z})[X] \simeq \mathbb{F}_{p_1}[X] \times \cdots \times \mathbb{F}_{p_k}[X].$$

It means that multiplications in $(\mathbb{Z}/M\mathbb{Z})[X]$ can be distributed between k independent rings $\mathbb{F}_{p_i}[X]$ and each such multiplication can be done independently in parallel. Moreover all prime numbers $p_i = 2^{m+1}r_i + 1$ were chosen in the way to be well suited for FFT because each field $\mathbb{F}_{p_i}$ contains primitive root of unity of degree 2^{m+1}. ■

Suppose now that we have k prime numbers p_i that have the same bit length and satisfy the conditions described in Theorem 1. We have the following theorem:

Theorem 2: If $\lfloor \log_2(p_i) \rfloor = \lfloor \log_2(p_j) \rfloor$ and formal power series have precision n, then the multiplication algorithm described in Theorem 1 consists of

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

multiplications in $\mathbb{F}_{p_i}$. Where c_1, c_2 are some constants.

Proof: Since $\lfloor \log_2(p_i) \rfloor = \lfloor \log_2(p_j) \rfloor$ for each i, j, then we can assume that the cost of multiplication in every $\mathbb{F}_{p_i}$ is the same. Single FFT multiplication consists of three basic steps:

1) Reduction modulo every chosen prime requires $c_1 k^2 n$ multiplications in $\mathbb{F}_{p_i}$. Each coefficient can be reduced modulo p_i using $c_1 k$ multiplications in $\mathbb{F}_{p_i}$. We have n coefficients and k small primes. It means that the total cost of this step is equal to $c_1 k \cdot n \cdot k = c_1 k^2 n$.
2) We perform the FFT multiplication for all $i \in \{1, \ldots, k\}$:
   a) Fourier transform of two power series with n coefficients requiring $2n\log(n)$ multiplications in $\mathbb{F}_{p_i}$,
   b) scalar multiplication of two vectors with $2n$ coefficients with requires $2n$ multiplications in $\mathbb{F}_{p_i}$,
   c) inverse Fourier transform of the vector to the power series with $2n$ coefficients requiring $n\log(n)$ multiplications $\mathbb{F}_{p_i}$.
3) Application of the Chinese Remainder Theorem to get back final coefficients which requires $c_2 k^2 n$ multiplications in $\mathbb{F}_{p_i}$. Each solution of the system $x \equiv a_i \bmod p_i$ can be reconstructed using $c_2 k^2$ multiplications in $\mathbb{F}_{p_i}$. Since we have to reconstruct n coefficients, the total cost is equal to $c_2 k^2 \cdot n = c_2 k^2 n$.

Thus the multiplication algorithm described in Theorem 1 consists of

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

multiplications in $\mathbb{F}_{p_i}$. ■

Finally, lets see how the new algorithm compares with the method using the Fast Fourier Transform for multiplying both: polynomials and coefficients. If we assume that numbers p_i are comprised within a single register of the processor, then the complexity of the algorithm which multiplies the polynomial and its coefficients using FFT is

$$O((n\log n)(k\log k)).$$

The complexity of our algorithm is equal to

$$O(kn\log n + k^2 n).$$

If we assume that $k = O(n)$, it is clear that the algorithm based totally on FFT is much faster. Its complexity is equal to $O(n^2 \log^2 n)$, whereas our algorithm works in time $O(n^3)$. But what happens when the polynomial coefficients are reduced? Lets assume that $k = O(\log n)$. Under this assumption, the complexity of the algorithm based totally on FFT is $O(n\log^2 n \log\log n)$, whereas the asymptotic complexity of our method is $O(n\log^2 n)$. Although the difference is not significant, we definitely managed to achieve our goal which was to develop an effective algorithm for multiplying polynomials with coefficients of an order much lower than the degree.

Corollary 1: If $k = O(\log n)$, the complexity of the proposed algorithm is lower than the complexity of the multiplication algorithm based on FFT only, and equals to

$$O(n\log^2 n),$$

whereas the complexity of the FFT-based algorithm is

$$O(n\log^2 n \log\log n).$$

■

However, in practice we managed to achieve much more than this. The numerical experiments showed that the new algorithm brings obvious benefits already in the case of polynomial coefficients consisting of several hundred bits. It means that its application is effective already for small values of k and n.

IV. Results of practical implementation for 32-bit processors

The implementation of the fast algorithm for multiplying polynomials has been prepared for 32-bit architecture with the use of OpenMP interface. The obtained time results

ISBN: 978-988-19252-1-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

turned out exceptionally good. They confirmed that in practice, the combination of the Fast Fourier Transform with the Chinese Remainder Theorem considerably accelerates computations. Tables I and II present the performance times of the algorithm for multiplying polynomials of the same degree with coefficients ranging from $[0, 2^{256})$ and $[0, 2^{512})$.

Our implementation is dedicated for x86 processors and use 32-bit integer arithmetic. The numerical experiments were done on Intel Core 2 processor (2.4 GHz) and confirmed that the simultaneous application of CRT and FFT is very efficient. To the end of this section we will assume that: $p_{544} = 2^{544} - 2^{32} + 1$, $p_{1088} = 2^{1088} - 2^{416} + 2^{256} + 1$ and $2^{31} < p_i < 2^{32}$. We compare our FFT-CRT based implementation with multiplication algorithm based on FFT over fields $\mathbb{F}_{p_{544}}$ and $\mathbb{F}_{p_{1088}}$. For the sake of completeness of the presented measurements, the performance time of the classic algorithm for multiplying polynomials was also given. The comparison of the performance time of the classic algorithm with the multicore implementation based on the Fourier Transform and the Chinese Remainder Theorem is presented in tables III and IV.

We use OpenMP standard to implement parallel version of proposed algorithm. In Tables I and II fraction T_2/T_3 gives us information about how many of our 4 cores are on average used by single multiplication. We can see that algorithm based on FFT and CRT uses between 80% to 90% computational power. It is very good result for arithmetic algorithm.

TABLE I
MULTIPLICATION OF TWO POLYNOMIALS OF DEGREE $n/2-1$ WITH COEFFICIENTS LESS THAN 2^{256}

Poly. degree	FFT $\mathbb{F}_{p_{544}}$ (1 core)	FFT-CRT $\bigotimes_{i=1}^{18}\mathbb{F}_{p_i}$ (1 core)	FFT-CRT $\bigotimes_{i=1}^{18}\mathbb{F}_{p_i}$ (4 cores)		
$n/2-1$	T_1	T_2	T_3	T_1/T_2	T_2/T_3
511	0.0423 s	0.0183 s	0.0052 s	2.3	3.5
1023	0.0930 s	0.0383 s	0.0111 s	2.4	3.4
2047	0.2020 s	0.0803 s	0.0259 s	2.5	3.1
4095	0.4360 s	0.1705 s	0.0481 s	2.6	3.5
8191	0.9370 s	0.3575 s	0.1012 s	2.6	3.5
16383	2.0100 s	0.7444 s	0.2161 s	2.7	3.4
32767	4.2700 s	1.5491 s	0.4283 s	2.8	3.6
65535	9.0700 s	3.2168 s	0.9339 s	2.8	3.4
131071	19.1700 s	6.6716 s	1.8919 s	2.9	3.5

V. SUMMARY

We present analysis of a new algorithm for multiplying polynomials and power series. It has been designed so as to exploit fully the computing power offered by modern multicore processors. Thanks to using the Chinese Remainder Theorem, it is possible to easily allocate tasks between the available threads. Moreover, under the adopted approach there is no need to synchronize the computations and to ensure communication between individual threads, which is an additional asset. For that reason the algorithm can be easily implemented with the use of a parallel programming standard OpenMP. The ratio T_2/T_3 in tables

TABLE II
MULTIPLICATION OF TWO POLYNOMIALS OF DEGREE $n/2-1$ WITH COEFFICIENTS LESS THAN 2^{512}

Poly. degree	FFT $\mathbb{F}_{p_{1088}}$ (1 core)	FFT-CRT $\bigotimes_{i=1}^{36}\mathbb{F}_{p_i}$ (1 core)	FFT-CRT $\bigotimes_{i=1}^{36}\mathbb{F}_{p_i}$ (4 cores)		
$n/2-1$	T_1	T_2	T_3	T_1/T_2	T_2/T_3
511	0.1598 s	0.0511 s	0.0136 s	3.1	3.7
1023	0.3500 s	0.1055 s	0.0280 s	3.3	3.8
2047	0.7600 s	0.2203 s	0.0608 s	3.4	3.6
4095	1.6420 s	0.4562 s	0.1210 s	3.6	3.8
8191	3.5310 s	0.9430 s	0.2527 s	3.7	3.7
16383	7.5500 s	1.9412 s	0.5254 s	3.9	3.7
32767	16.0900 s	3.9944 s	1.0960 s	4.0	3.6
65535	34.1300 s	8.2184 s	2.1926 s	4.1	3.7
131071	72.2100 s	16.9245 s	4.5895 s	4.3	3.7

TABLE III
COMPARISON OF CLASSIC POLYNOMIAL MULTIPLICATION WITH PROPOSED ALGORITHM FOR TWO POLYNOMIALS OF DEGREE $n/2-1$ AND COEFFICIENTS LESS THAN 2^{256}

Poly. degree	Classic polynomial multiplication (1 core)	FFT-CRT $\bigotimes_{i=1}^{18}\mathbb{F}_{p_i}$ (4 cores)	
$n/2-1$	T_1	T_2	T_1/T_2
511	0.2018 s	0.0052 s	39
1023	0.8074 s	0.0111 s	73
2047	3.2296 s	0.0259 s	125
4095	13.0862 s	0.0481 s	272
8191	52.3449 s	0.1012 s	517
16383	206.6953 s	0.2161 s	956
32767	826.7812 s	0.4283 s	1930
65535	3350.0744 s	0.9339 s	3587
131071	13400.2979 s	1.8919 s	7083

I and II shows how many processors out of the four ones available were used on average during a single multiplication. The measurements show that the algorithm uses from 80% to 90% of the available computing power. In the case of an arithmetic algorithm, this should be considered a very good result. Therefore, we may conclude that the goal which consisted in designing a parallel algorithm for multiplying polynomials has been achieved.

As far as the theoretical results of the paper are concerned, the analysis conducted in Section III and the Corollary 1 being its essence, are of key importance. If we assume that the degree of the polynomial is n and the accuracy of its coefficients is k, then the asymptotic complexity of the proposed algorithm is

$$O(kn \log n + k^2 n).$$

Owing to the two essential components of the asymptotic function, it is impossible to determine explicitly whether the new solution is better or worse than the method based on FFT only. It is due to the fact that if we use the Fast Fourier Transform to multiply both the polynomial and its coefficients, the complexity is equal to

$$O((n \log n)(k \log k)).$$

Therefore, one can see that if $k = O(n)$, the proposed algorithm performs worse than the method based on FFT only. However, if $k = O(\log n)$, the complexity of the new

ISBN: 978-988-19252-1-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

TABLE IV
COMPARISON OF CLASSIC POLYNOMIAL MULTIPLICATION WITH PROPOSED ALGORITHM FOR TWO POLYNOMIALS OF DEGREE $n/2-1$ AND COEFFICIENTS LESS THAN 2^{512}

Poly. degree	Classic polynomial multiplication (1 core)	FFT-CRT $\bigotimes_{i=1}^{36}\mathbb{F}_{p_i}$ (4 cores)	
$n/2-1$	T_1	T_2	T_1/T_2
511	0.7759 s	0.0136 s	57
1023	3.1247 s	0.0280 s	112
2047	12.3731 s	0.0608 s	203
4095	50.1638 s	0.1210 s	415
8191	197.9711 s	0.2527 s	783
16383	799.9376 s	0.5254 s	1522
32767	3167.5383 s	1.0960 s	2890
65535	12670.1535 s	2.1926 s	5778
131071	50508.8154 s	4.5895 s	11013

algorithm is lower. The computational complexity ratio is $O(\log\log n)$ to the advantage of the method presented in the paper. This reasoning allows us to conclude that the algorithm based on CRT and FFT should be used when the number of coefficients of a polynomial exceeds greatly their accuracy. This is often the case when computations use long polynomials or power series with a modular reduction of coefficients.

The results of numerical tests presented in Section IV show that the proposed method has numerous practical applications. In this section the algorithm has been intentionally compared with the implementation using the classic algorithm for multiplying coefficients in large bodies $\mathbb{F}_p$. It results from the fact that in the case of numbers p consisting of 500 or 1000 bits, multiplication based on the Fourier Transform is completely ineffective. The measurement results came as a great surprise, as it turned out (Tables I and II) that the proposed algorithm is several times faster even when its application is not parallel. Furthermore, the new algorithm compares favorably with the classic algorithm for multiplying polynomials with complexity equal to $O(n^2)$ (Tables III and IV). Therefore, there is no doubt that the presented algorithm performs exceptionally well when applied in practice.

REFERENCES

[1] D. Charles, K. Lauter, "Computing modular polynomials", *Journal of Computational Mathematics*, pp. 195–204, 2005.
[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, "Introduction to Algorithms", MIT Press, 2003.
[3] R. Crandall, B. Fagin, "Discrete weighted transforms and large integer arithmetic", *Mathematics of Computation*, vol. 62, pp. 305–324, 1994.
[4] R. Crandall, C. Pomerance, "Prime Numbers – a computational perspective", Springer-Verlag, 2001.
[5] A. Enge, "Computing modular polynomials in quasi-linear time", *Math. Comp.*, vol. 78, pp. 1809–1824, 2009.
[6] L. Garcia, "Can Schönhage multiplication speed up the RSA encryption or decryption?", *University of Technology, Darmstadt*, 2005.
[7] S. Gorlatch, "Programming with divide-and-conquer skeletons: A case study of FFT", *Journal of Supercomputing*, vol. 12, pp. 85–97, 1998.
[8] A. Grama, A. Gupta, G. Karypis, V. Kumar, "Introduction to Parallel Computing", Addison Wesley, 2003.
[9] D. Harvey, "A cache-friendly truncated FFT", *Theoretical Computer Science*, vol. 410, pp. 2649–2658, 2009.
[10] J. van der Hoeven, "The truncated Fourier transform and applications", in: ISSAC 2004, ACM, pp. 290–296, 2004.
[11] J. van der Hoeven, "Notes on the truncated Fourier transform", unpublished, retrieved from http://www.math.u-psud.fr/˜vdhoeven/, 2005.
[12] D. E. Knuth, "Art of Computer Programming", Addison-Wesley Professiona, 1998.
[13] V. Müller, "Ein Algorithmus zur Bestimmung der Punktzahlen elliptischer Kurven über endlichen Körpern der Charakteristik grösser drei", *Ph.D. Thesis, Universität des Saarlandes*, 1995.
[14] H. J. Nussbaumer, "Fast polynomial transform algorithms for digital convolution", *IEEE Trans. Acoust. Speech Signal Process.*, vol. 28 (2) pp. 205–215, 1980.
[15] A. Schönhage, V. Strassen, "Schnelle Multiplikation grosser Zahlen", *Computing*, vol. 7, pp. 281–292, 1971.
[16] A. Schönchage, "Asymptotically fast algorithms for the numerical multiplication and division of polynomials with complex coefficients", *Lecture Notes in Computer Science*, vol. 144, pp. 3–15, 1982.
[17] D. Takahashi, Y. Kanada, "High-performance radix-2, 3 and 5 parallel 1-D complex FFT algorithms for distributed-memory parallel computers", *Journal of Supercomputing*, vol. 15, pp. 207–228, 2000.

ISBN: 978-988-19252-1-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)