Parallel Algorithm for Multiplication on Elliptic Curves

Juan Manuel Garcia Garcia[1] and Rolando Menchaca Garcia[2]

[1] Department of Computer Systems
Instituto Tecnologico de Morelia
Morelia, Mexico
jmgarcia@sekureit.com

[2] Center for Computing Research
Instituto Politecnico Nacional
Mexico City, Mexico
menchaca@pollux.cic.ipn.mx

Abstract. Given a positive integer n and a point P on an elliptic curve E, the computation of nP, that is, the result of adding n times the point P to itself, called the *scalar multiplication*, is the central operation of elliptic curve cryptosystems. We present an algorithm that, using p processors, can compute nP in time $O(\log n + H(n)/p + \log p)$, where $H(n)$ is the Hamming weight of n. Furthermore, if this algorithm is applied to Koblitz curves, the running time can be reduced to $O(H(n)/p + \log p)$.

1 Introduction

Elliptic curve cryptosytems were first proposed by Koblitz [10] and Miller [15]. Their main attractive is their strongest security by key length. It is possible to construct elliptic curve cryptosystems over a smaller definition field than similar public-key cryptosystems based on the discrete logarithm problem, such as RSA [17] cryptosystems. Elliptic curve cryptosystems with a 160-bit key are believed to have the same security as, for example, the RSA with a 1024-bit key length.

Several fast implementation of elliptic curve cryptosystems has been reported [8, 7, 18, 4, 5]. The security of elliptic curve cryptosystems is based on the difficulty of the elliptic curve discrete logarithm problem, and then the main operation of those cryptosystems is the *exponentiation*, also known as *scalar multiplication*. For a general survey on exponentiation methods see [6]. In order to obtain fast elliptic curve exponentiation three factors has been considered: the field of definition [4][18], addition chains [4, 12, 16, 18] and optimal coordinate systems [3, 5]. Special elliptic curves with efficient arithmetics has also been considered [11] [20][21]. In this paper we propose the application of parallel processing to speed up the exponentiation on elliptic curves.

There already exists parallel algorithms for exponentiation of integers modulo a m-bit integer [2, 1, 13] and also for exponentiation on the fields $GF(2^m)$ [22] and $GF(q^m)$ [23]. We present in this paper a parallel algorithm to compute scalar multiplication on elliptic curves. This algorithm would be specially efficient for anomalous binary curves, also known as Koblitz curves. The application of our algorithm could lead to efficient implementations of elliptic curve cryptosystems on multiprocessor computers.

2 Binary Method

We have an elliptic curve E and a point $P \in E$. We want to compute nP, for some non negative integer n. Let $n = (b_{N-1}b_{N-2}b_{N-3}\cdots b_2b_1b_0)_2$ be the binary representation of n, where

$$N = \lfloor \log n \rfloor + 1. \tag{1}$$

Then

$$n = b_{N-1}2^{N-1} + b_{N-2}2^{N-2} + \cdots + b_2 2^2 + b_1 2 + b_0. \tag{2}$$

Using the Horner expansion, the last expression becomes

$$n = (\cdots((b_{N-1}2 + b_{N-2})2 + \cdots + b_1)2 + b_0). \tag{3}$$

From (3) we have

$$nP = 2(\cdots 2(2(2(b_{N-1}P) + b_{N-2}P) + \cdots + b_1P) + b_0P. \tag{4}$$

The last expression suggests a way to get nP. We can define the sucession of points $P_1, \cdots, P_N$ as follows. Let

$$\begin{aligned} P_1 &= P \\ P_i &= \begin{cases} 2P_{i-1} & \text{if } b_{N-i} = 0 \\ 2P_{i-1} + P & \text{if } b_{N-i} = 1 \end{cases} \end{aligned} \tag{5}$$

for $i = 2, \cdots, N$. We can easily observe that $P_N = nP$.

The method above outlined, to obtain nP by the computation of the terms $P_1, \cdots, P_N$, is known as the *binary algorithm* [9]. From (5) we can deduce that to obtain the term P_i we need one *doubling* and, if the corresponding bit of the binary representation of n is one, one adding. This computation is made $N - 1$ times, where $N = \lfloor \log n \rfloor + 1$. If $H(n)$ is the *binary Hamming weight* of n, that is, the number of bits set to one in the binary representation of n, then we need $H(n) - 1$ point addings besides the $N - 1$ doublings. Then we can state the following theorem.

Theorem 1. *If P is a point on an elliptic curve E and n is a positive integer, to obtain nP by the binary method are required $\lfloor \log n \rfloor$ doublings and $H(n) - 1$ point addings.*

If we assume that the time taken to add two elliptic curve points is almost the same that the time taken to do a doubling, and we denote as t such time, we can conclude from theorem 1 the following:

Corollary 1. *The execution time of the binary algorithm is*

$$(\lfloor \log n \rfloor + H(n) - 1) \cdot t,$$

where t is the time taken by a single elliptic curve operation.

Some improvement to the binary method can be obtained by the use of redundant number systems. Morain and Olivos [16] observed that on elliptic curves the inverse of a point can be obtained without a cost. For curves $y^2 = x^3 + Ax + B$ over $GF(p)$ with $p > 3$, the inverse of (x, y) is $(x, -y)$ and, for $y^2 + xy = x^3 + Ax^2 + B$ over $GF(2^m)$, the inverse is $(x, x + y)$. Then, we can consider representations

$$n = \sum_{i=0}^{N-1} c_i 2^i \tag{6}$$

with $c_i \in \{-1, 0, 1\}$ for $i = 0, \ldots, N-1$. A *nonadjacent form (NAF)* is a representation with $c_i c_{i+1} = 0$ for $0 \leq i < N - 1$. Since the NAF in general has fewer nonzeros that the binary representation, then the advantage of using it is few number of additions. Morain and Olivos [16] showed that the expected number of nonzeros in a NAF is $N/3$.

3 The p^{th}-Order Binary Method

First, we divide the binary representation of n in $\lceil N/p \rceil$ blocks of p bits each one and then we split n into $s_0, s_1, \ldots, s_p$ such that the binary representation of each s_i is formed by $\lceil N/p \rceil$ blocks of p bits set to 0, except for the i-th bit, which has the same value that the i-th bit of the corresponding block of the binary representation of n. Thus, we can define s_i, for $i = 0, \ldots, p-1$, as follows:

$$\begin{aligned} s_0 &= b_0 + b_p 2^p + b_{2p} 2^{2p} + \cdots + b_{(\lceil \frac{N}{p} \rceil - 1)p} 2^{(\lceil \frac{N}{p} \rceil - 1)p} \\ s_1 &= b_1 2 + b_{p+1} 2^{p+1} + b_{2p+1} 2^{2p+1} + \cdots + b_{(\lceil \frac{N}{p} \rceil - 1)p+1} 2^{(\lceil \frac{N}{p} \rceil - 1)p+1} \\ &\vdots \\ s_{p-1} &= b_{p-1} 2^{p-1} + b_{2p-1} 2^{2p-1} + b_{3p-1} 2^{3p-1} + \cdots + b_{p \lceil \frac{N}{p} \rceil - 1} 2^{p \lceil \frac{N}{p} \rceil - 1}. \end{aligned} \tag{7}$$

That is

$$s_i = \sum_{j=0}^{\lceil \frac{N}{p} \rceil - 1} b_{jp+i} 2^{jp+i} \tag{8}$$

for $i = 0, 1, \ldots, p-1$, where we are considering some *padding bits* $b_k = 0$ for $N - 1 < k \leq p \lceil N/p \rceil - 1$. The integer set $\{s_0, s_1, \cdots, s_{p-1}\}$ can be easily obtained from n by XOR-ing it with the appropiate mask. It is clear from (2) and (7) that

$$n = s_0 + s_1 + \cdots + s_{p-1} \tag{9}$$

and then, we can compute nP as the sum of $s_0 P, s_1 P, \ldots, s_{p-1} P$.

Now, we can outline our p^{th}*-order binary method* in two phases as follows:

1. *Bits scattering:* Compute the set $\{s_0, s_1, \ldots, s_{p-1}\}$ as is defined in (7). Using p processors, compute in parallel the scalar multiplications

   $$s_0 P, s_1 P, \ldots, s_{p-1} P$$

   running the binary method on each processor.
2. *Associative fan-in:* Using $\lceil p/2 \rceil$ processors compute in parallel

   $$nP = s_0 P + s_1 P + \ldots + s_{p-1} P.$$

   To do this, first we separate the total sum as

   $$\sum_{i=0}^{p-1} s_i = \sum_{i=0}^{\lceil p/2 \rceil - 1} s_i + \sum_{i=\lceil p/2 \rceil}^{p-1} s_i,$$

   and then both halves can be computed in parallel. We proceed on each half the same way recursivelly until we only need to compute a single addition of two elliptic curve points. Clearly in the deepest level of the recursion we have to do $\lceil p/2 \rceil$ additions in parallel, and to obtain the total sum we have to make $\lceil \log p \rceil$ recursive steps.

We can observe that if $p = 1$ then our algorithm reduces to the ordinary binary algorithm. In what follows, we analyze the running time of the p^{th}-order binary method.

Theorem 2. *The running time of the p^{th}-order binary algorithm, on the best case, is*

$$\lfloor \log n \rfloor + \lceil H(n)/p \rceil + \lceil \log p \rceil - 1$$

and, on the worst case,

$$\lfloor \log n \rfloor + H(n) - 1.$$

Proof. First, we must observe that the integer set $\{s_0, s_1, \cdots, s_{p-1}\}$ can be obtained from n in a negligible time. In the phase one of the algorithm, each processor executes the binary algorithm to compute $s_i P$, where i is the processor's index. From corollary 1, it requires $\lfloor \log s_i \rfloor + H(s_i) - 1$ steps to do this computation. Now, since $a_{N-1} = 1$, one of the processors have to compute $N-1$ doublings and if we suppose that in order to start the phase two all the processors must finish its computation, then the running time of the phase one will be

$$T_1 = N + H^* - 2, \tag{10}$$

where

$$H^* = \max_{0 \le i \le p-1} H(s_i). \tag{11}$$

Since we have that

$$H(n) = \sum_{i=0}^{p-1} H(s_i) \tag{12}$$

then

$$\lceil H(n)/p \rceil \le H^* \le H(n). \tag{13}$$

In the best case, all the bits set to one in the binary representation of n are equally scattered among the integers $\{s_i\}$, so the load is perfectly balanced on all the processors, and then we have

$$H^* = \left\lceil \frac{H(n)}{p} \right\rceil. \tag{14}$$

And from (14), (10) and (1) we have that, on the best case, the time of the phase one is given by

$$T_1 = \lfloor \log n \rfloor + \lceil H(n)/p \rceil - 1. \tag{15}$$

In the phase two of our algorithm, the sum $s_0 P + \cdots + s_{p-1} P$ is computed using the associative fan-in algorithm in

$$T_2 = \lceil \log p \rceil \tag{16}$$

steps, and then the total running time for the best case will be

$$T = T_1 + T_2 = \lfloor \log n \rfloor + \lceil H(n)/p \rceil + \lceil \log p \rceil - 1. \tag{17}$$

In the worst case $H^* = H(n)$, which implies by the definition of H^* (11) that exists some index k, $0 \le k \le p-1$, such that $H(s_k) = H(n)$ and, from (12), $H(s_i) = 0$ for $i \ne k$ and then $s_i = 0$ for $i \ne k$. From (9) we have that $n = s_k$ and then $s_k P = nP$. Then we have that nP is computed on the phase one by the processor k. Therefore the running time will be the same as that stated on the corollary 1.

Some improvement can be obtained if we use the NAF as input to our algorithm instead of the binary representation of n. Since the NAF has a fewer number of nonzeros, then we can obtain the same running times with fewer processors. But before this we can see that a bottleneck in our algorithm is caused by the number of doublings needed on the first phase, represented by the linear term on N in (10), which is independent from the number p of processors.

In the deduction of the running time of our algorithm, was supossed that the time needed to do a doubling or a point addition is almost the same, but in general it depends of the coordinate system in which an elliptic curve is represented. The most well known coordinate systems are the affine and projective coordinates [19]. Another two coordinate systems, the Jacobian coordinates and Chudnovsky Jacobian coordinates have been proposed in [3]. The efficiency of Jacobian coordinates for elliptic curve exponentiation was discussed in [4]. In [5] has been proposed a modified Jacobian coordinate system, which gives faster doublings than affine, projective, Jacobian and Chudnovsky Jacobian coordinates. Then we can use this modified Jacobian coordinate system to partially overcome the bottleneck of doublings in our algorithm.

A more fruitful approach could be the use of special elliptic curves in which doublings are unnecessary at all. This will be discussed at the next section.

4 The p^{th}-order τ-ary method

The *binary anomalous curves*, proposed by Koblitz [11], have been used to obtain efficient implementations of elliptic curve cryptosystems [14][20] [21]. These curves are

$$E_1 \ : \ y^2 + xy = x^3 + x^2 + 1 \tag{18}$$

and

$$E_2 \ : \ y^2 + xy = x^3 + 1 \tag{19}$$

over $GF(2^m)$. For these it can be easily verified the property that if the point (x, y) is in the curve, also is the point (x^2, y^2). Then we can define their Frobenius automorphisms as $\varphi(x, y) = (x^2, y^2)$, which corresponds to multiplication by $\tau = (1 + \sqrt{-7})/2$ on E_1 and by $-\bar{\tau} = (-1 + \sqrt{-7})/2$ on E_2. Using normal basis on $GF(2^m)$, the multiplication by τ requires only two cyclic shifts and then it can be done in a negligible time.

Since τ is an element of norm 2 in the Euclidean domain $\mathbb{Z}[\tau]$ any integer n has a representation as

$$n = \sum_{i=0}^{\infty} c_i \tau^i \tag{20}$$

for $c_i \in \{0, 1\}$. For any $n \in \mathbb{Z}[\tau]$ a representation (20) is called a NAF if $c_i \in \{0, \pm 1\}$ and $c_i c_{i+1} = 0$ for all $i \geq 0$. Solinas [20] showed the existence of a NAF of minimal weigth by giving an algorithm that computes the NAF directly. If $\tau | n$, then $c_0 = 0$. Otherwise, τ^2 divides either $n + 1$ or $n - 1$ and the NAF ends in $(0, -1)$ or $(0, 1)$, respectively. This process is repeated on n/τ, $(n + 1)/\tau^2$ or $(n - 1)/\tau^2$.

Because $\varphi^m(x, y) = (x^{2^m}, y^{2^m}) = (x, y)$, any two representations which agree modulo $\tau^m - 1$ will yield the same endomorphism on the curve. Based on this property, Meier and Staffelbach [14] showed the following:

Theorem 3. *Every $n \in \mathbb{Z}[\tau]$ has a representation*

$$n \equiv \sum_{i=0}^{m-1} c_i \tau^i \pmod{\tau^m - 1}, \tag{21}$$

with $c_i \in \{0, \pm 1\}$.

The binary method can be extended in a natural way to a τ-ary method based on the representation of n given by (21). Now we can define a parallel p^{th}-order τ-ary method.

Using (21) we can define the set $\{s_0, \cdots, s_{p-1}\} \subseteq \mathbb{Z}[\tau]$, where

$$s_i = \sum_{j=0}^{\lceil \frac{m}{p} \rceil - 1} c_{jp+i} \tau^{jp+i} \tag{22}$$

for $i = 0, 1, \ldots, p - 1$.

The *p^{th}-order τ-ary method* consists in the following two stages:

1. *Bits scattering:* Using p processors compute in parallel

$$s_0 P, s_1 P, \cdots, s_{p-1} P$$

running the τ-ary algorithm on each processor.
2. *Associative fan-in:* Using $\lceil p/2 \rceil$ processors compute

$$\sum_{k=0}^{p-1} s_k P$$

with the associative fan-in algorithm, in $\lceil \log p \rceil$ steps.

If we define $H(n)$ as the number of nonzeros in the representation given by (21), we can state the following theorem:

Theorem 4. *The running time of the p^{th}-order τ-ary method, on the best case, is*

$$\lceil H(n)/p \rceil + \lceil \log p \rceil - 1$$

and

$$H(n) - 1$$

on the worst case.

Proof. The proof of this theorem is the same as that of theorem 2 except that the time for the phase one will be

$$T_1 = H^* - 1 \tag{23}$$

where H^* is defined as in (11).

Meier and Staffelbach [14] conjecture, based on experimental evidence, that on average half of the c_i will be nonzero. So we can expect that when $p \approx m/2$ the running time will be closer to $\log H(n)$. Further improvements can be made to the τ-ary representation to reduce the number of nonzeros [21], so a small number of processors can bring similar speed-up in the p^{th}-order τ-ary method.

5 Conclusions

We have discussed in this paper a extension of the binary algorithm, which by the use of parallel processing can speed up the computation of scalar multiplications on elliptic curves. We have also discussed some ways to improve this algorithm. We have showed how it can be specially efficient for Koblitz curves.

The application of this algorithm could lead to efficient implementations of elliptic curves cryptosystems on multiprocessor architectures.

References

1. M. Adleman and K. Kompella, Using smoothness to achieve parallelism, *Proceedings of the 20th ACM Symposium on the Theory of Computing*, pp. 528-538, 1988.
2. P.W. Beame, S.A. Cook and H.J. Hoover, Log depth circuits for division and related problems, *SIAM Journal on Computing*, vol. 15, pp. 994-1003, 1986.
3. D.V. Chudnovsky and G.V. Chudnovsky, Sequences of numbers generated by addition in formal groups and new primality and factorization tests, *Advances in Applied Mathematics*, vol. 7, pp. 385-434, 1986.
4. H. Cohen, A. Miyaji and T. Ono, Efficient elliptic curve exponentiation, *Advances in Cryptology - Proceedings of ICICS'97*, LNCS 1334, pp. 282-290, 1997.
5. H. Cohen, A. Miyaji and T. Ono, Efficient elliptic curve exponentiation using mixed coordinates, *Advances in Cryptology - ASIACRYPT'98*, LNCS 1514, pp. 51-65, 1998.
6. D.M. Gordon, A survey of fast exponentiation methods, *Journal of Algorithms*, vol. 27, pp. 129-146, 1998.
7. J. Guajardo and C. Paar, Efficient algorithms for elliptic curve cryptosystems, *Advances in Cryptology - CRYPTO'97*, LNCS 1294, pp. 342-356, 1997.
8. G. Harper, A. Menezes and S. Vanstone, Public-key cryptosystems with very small key lengths, *Advances in Cryptology - EUROCRYPT'92*, LNCS 658, pp. 163-173, 1993.
9. D.E. Knuth. *The Art in Computer Programming. Vol 2: Seminumerical Algorithms.* Addison-Wesley, 2nd. Ed., 1981.
10. N. Koblitz, Elliptic curve cryptosystems, *Mathematics of Computation*, vol. 48, pp. 203-209, 1987.
11. N. Koblitz, CM-curves with good cryptographic properties, *Advances in Cryptology - CRYPTO'91*, LNCS 576, pp. 279-287, 1992.
12. K. Koyama and T. Tsuruoka, Speeding up elliptic cryptosystems by using a signed binary window method, *Advances in Cryptology - CRYPTO'92*, LNCS 740, pp. 345-357, 1993.
13. C.H. Lim and P.J. Lee, More flexible exponentiation with precomputation, *Advances in Cryptology - CRYPTO'94*, LNCS 389, pp. 95-107, 1994.
14. W. Meier and O. Staffelbach, Efficient multiplication on certain nonsupersingular elliptic curves, *Advances in Cryptology - CRYPTO'92*, vol. 740, pp. 333-344, 1993.
15. V. Miller, Uses of elliptic curves in cryptography, *Advances in Cryptology - CRYPTO'85*,LNCS 218, pp.417-426, 1986.
16. F. Morain and J. Olivos. Speeding up the computations on an elliptic curve using addition-substraction chains. *Inform. Theor. Appl.*, 24, 531-543, 1990.
17. R. Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public key cryptosystems, *Communications of the ACM*, vol. 21, no. 2, pp. 120-126, 1978.
18. R. Schroeppel, H. Orman, S. O'Malley and O. Spatscheck, Fast key exchange with elliptic curve systems, *Advances in Cryptology - CRYPTO'95*, LNCS 963, pp. 43-56, 1995.
19. J.H. Silverman, *The Arithmetic of Elliptic Curves*, GTM 106, Springer Verlag, 1986.
20. J. Solinas, An improved algorithm for arithmetic on a family of elliptic curves, *Advances of Cryptology - CRYPTO'97*, LNCS 1294, pp. 357-371, 1997.
21. J. Solinas, Efficient arithmetic on Koblitz curves, *Design, Codes and Cryptography*, vol. 19, pp. 195-249, 2000.

22. D.R. Stinson, Some observations on parallel algorithms for fast exponentiation in $GF(2^n)$, *SIAM Journal on Computing*, vol. 19, pp. 711-717, 1990.
23. J. von zur Gathen, Efficient and optimal exponentiation in finite fields, *Computational Complexity*, pp. 360-394, 1991.