Parallel Scalar Multiplication on Elliptic Curves in Wireless Sensor Networks

Yanbo Shou[1], Herve Guyennet[1], and Mohamed Lehsaini[2]

[1] University of Franche-Comte, France
{yshou, herve.guyennet}@femto-st.fr
[2] University of Tlemcen, Algeria
m_lehsaini@mail.univ-tlemcen.dz

Abstract. In event-driven sensor networks, when a critical event occurs, sensors should transmit quickly and securely messages back to base station. We choose Elliptic Curve Cryptography to secure the network since it offers faster computation and good security using shorter keys than RSA. In order to minimize the computation time, we propose to distribute the computation of scalar multiplications on elliptic curves by involving neighbor nodes in this operation. The results of performance tests show that parallel computing certainly consumes much more resources, however it reduces considerably the computation time of scalar multiplications. This method is applicable in event-driven applications when execution time is the most critical factor.

Keywords: Wireless sensor networks, Elliptic curves, Scalar multiplication, Parallel computing

1 Introduction

A wireless sensor node is a small electronic device which consists of sensing, data processing and communicating components [1]. Such sensor nodes can be programmed to collect environmental data and to communicate with other nodes. A sensor node is often equipped of a low-cost low-power microcontroller which is not capable of doing complicated calculations. As the ability of a single node is very limited, a wireless sensor network is usually constituted of a large number of nodes which are interconnected to each other to form a large network. In most of cases a sensor node has to cooperate with other nodes to achieve a common goal.

Sensors are often deployed in hostile and inaccessible areas for human being. Today, we can find a wide spectrum of sensor networks applications such as environmental monitoring [2], industrial sensing [3], home automation [4], medical care [5] or military surveillance [6].

However sensors are vulnerable and subject of various attacks due to their lack of resources and the unreliability of wireless connection, in [7] we can find the presentation of almost all possible attacks in wireless sensor networks. In order

to secure wireless sensor networks, one of the most efficient solutions is to use cryptographic mechanisms [7,8]. Symmetric cryptographic algorithms are usually light weighted and can be efficiently implemented in hardware and software, but as we use the same key for data encryption and decryption, the key management becomes a challenging problem in wireless sensor networks, since the key can be exposed if a node is compromised. An other choice is asymmetric cryptography which is computationally more expensive but it's easier to manage the keys, a compromised node cannot provide clue to the private keys of non-compromised nodes [8]. Of course the security of sensor networks can still be threatened by physical attacks, but the protection against such attack is beyond the scope of this paper.

In this paper we choose the Elliptic Curve Cryptography (ECC) which is one of the most famous asymmetric cryptographic schemes. It has attracted considerable attention recently because of its shorter key length requirement comparing with the other widely used asymmetric cryptographic algorithm RSA. An elliptic curve cryptosystem using a 160-bits key can provide the same security level with a 1024-bit RSA key [9]. The security of elliptic curve cryptography mainly relies on the difficulty of discrete logarithm problem. All the points on a curve form a abelian group whose group law is the point addition which combines 2 points on the curve to get a third one. Based on the point addition, we may then perform point multiplication, also called scalar multiplication, for example $Q = kP$ where Q and P are 2 points on the curve and k is a positive integer. It's extremely difficult to compute the value of k given P and Q if k is big enough. Scalar multiplication is the most expensive operation on elliptic curves and it exists various solutions in the literature to optimize its performance [10], especially for embedded platforms which have very limited CPU power.

Parallelism is an other choice to improve the performance of scalar multiplication. Instead of performing calculations on a single sensor node, we try to split the computing task into smaller pieces which are then distributed to neighbor nodes and carried out simultaneously. In this paper we propose to use parallel computing to accelerate scalar multiplication which is the most complicated and time-consuming operation on elliptic curves. To the best of our knowledge, we are the first who have applied this technique in sensor networks. We find that the parallelization of scalar multiplication is efficient in execution time. The parallel overhead is relatively low comparing with the computation time, since the computing power of embedded microcontrollers are very weak.

We have also studied the memory usage et the energy consumption of parallel computing in wireless sensor networks. As more sensor nodes are involved in the computation, more resources will be consumed. However the objective of our solution is to accelerate the computation on elliptic curves in cases where execution time becomes the most critical factor.

The rest of the paper is organized as follows. Section 2 gives a presentation of basic concepts of elliptic curves, and in section 3 we present the related work of parallelization of scalar multiplication. Section 4 describes our parallel comput-

ing scheme for scalar multiplication in wireless sensor networks, and the results of performance test are given in section 5. Section 6 concludes the paper.

2 Basic Concepts of Elliptic Curve

Elliptic curve cryptography were proposed independently by Miller [11] and Koblitz [12] in the 80's. It has attracted researchers' attention in recent years due to its shorter key length requirement comparing with RSA, especially in the domain of embedded systems where devices have limited computing power.

In cryptography we work with the elliptic curves which are defined over a finite field F_q where $q = p^m$ and p is a prime number called the characteristic of F. If $m = 1$, then F is called a prime field, if $q = 2^m$, then F is a binary field.

In this paper we only work with the first case where $m = 1$ and $p \neq 2$ or 3, only for reason of easier explanation. Then a curve can be represented using the simplified Weierstrass equation (see formula 1).

$$y^2 = x^3 + ax + b \tag{1}$$

where the discriminant $\triangle$ of the curve

$$-16(4a^3 + 27b^2) \neq 0$$

All points on the curve, including the point at infinity, form an abelian group whose group law is the point addition. Suppose that $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ are 2 points on the curve and $P_3(x_3, y_3)$ is the sum of $P_1 + P_2$ which can be calculated using the formula 2.

$$\begin{cases} x_3 = s^2 - x_1 - x_2 \\ y_3 = s(x_1 - x_3) - y_1 \end{cases} \tag{2}$$

where

$$s = \begin{cases} (y_2 - y_1)/(x_2 - x_1) & \text{if } P_1 \neq P_2, \text{ addition} \\ (3x_1^2 + a)/2y_1 & \text{if } P_1 = P_2, \text{ doubling} \end{cases} \tag{3}$$

Geometrically P_3 is the reflection about x-axis of the intersection point of the curve with the line through P_1 and P_2.

We may also perform scalar multiplication $Q = kP$ which can be considered as a sequence of consecutive additions.

Q and P are 2 points on the curve and k is a positive integer. The point multiplication can be performed more efficiently than repeating point addition. The most basic method is using the Double and Add algorithm (see algorithm 1).

Suppose that integer k is represented in binary form $k = \sum_{i=0}^{l-1} k_i 2^i$ where l is the length of k. We keep reading k_i from the least significant bit to the most significant one. Every time when we read a bit, point P is doubled, and if k_i is a nonzero bit, $Q = Q + P$. The performance of the algorithm 1 mainly depends on the length of k and number of nonzero bits in its binary representation, the average number of operations needed is l point doublings and $\frac{l}{2}$ point additions.

Algorithm 1: Double and Add algorithm for point multiplication

Data: $k = (k_{l-1}, \ldots, k_1, k_0)_2$, $P \in E$
Result: $Q = kP$
$Q \leftarrow \infty$;
for i *from* 0 *to* $l-1$ **do**
  if $k_i = 1$ **then**
    $Q \leftarrow Q + P$;
  end
  $P \leftarrow 2P$;
end
return Q;

The performance of point multiplication can be significantly improved by choosing appropriate representation of k, coordinate system and size of finite field.

If $P = (x, y)$ is a point on an elliptic curve defined over a prime field F_p in which $-P = (x, -y)$, We can see that the subtraction of points on an elliptic curve is as efficient as addition [13], since $P_1 - P_2 = P_1 + (-P_2)$.

The first optimization possible is to use Non-Adjacent Form (NAF) method which uses a signed digit representation of k (see formula 4), and the average number of nonzero bits is reduced from $\frac{l}{2}$ to $\frac{l}{3}$ where l is the length of k.

$$k = \sum_{i=0}^{l-1} k_i 2^i \text{ where } k_i \in \{0, \pm 1\} \tag{4}$$

We read every digit k_i, if $k_i = 1$, then $Q = Q + P$, but if $k_i = -1$, $Q = Q - P = Q + (-P)$. If the length of k is l, the average number of operations is reduced to l point doubling and $\frac{l}{3}$ point addition (subtraction).

In formula 3 we see that both point addition and doubling need a field inversion which is more computationally expensive than field multiplication. An other optimization is to avoid the computation of field inversion by using projective coordinates.

In Jacobian coordinates, a projective point $(X, Y, Z), Z \neq 0$ is equivalent to affine point $(X/Z^2, Y/Z^3)$, and an elliptic curve is represented by the new equation

$$Y^2 = X^3 + aXZ^4 + bZ^6 \tag{5}$$

We can then obtain the formula for point doubling in Jacobian coordinates by substituting x by $\frac{X}{Z^2}$ and y by $\frac{Y}{Z^3}$ in formula 2,

$$\begin{cases} X_3 = (3X_1^2 + aZ_1^4)^2 - 8X_1Y_1^2 \\ Y_3 = (3X_1^2 + aZ_1^4)(4X_1Y_1^2 - X_3) - 8Y_1^4 \\ Z_3 = 2Y_1Z_1 \end{cases} \tag{6}$$

and the formula for point addition in Jacobian coordinates can be derived in the same manner.

There are obviously more multiplications using Jacobian coordinates, but during a point multiplication we only need to perform field inversion once to convert the final result back to affine coordinates. The comparison of operations counts for both coordinate systems are given in table 1[13].

Table 1. Operation counts for point doubling and addition. A=Affine, J=Jacobian, M=Multiplication, S=Squaring, I=Inversion.

Doubling		Addition	
$2A \to A$	1I,2M,2S	$A + A \to A$	1I, 2M, 1S
$2J \to J$	4M, 4S	$J + J \to J$	12M, 4S

We may also improve the performance of point multiplication by choosing specific finite field. For example the modular reduction in prime field can be very fast if we use recommended NIST primes, like $p_{192} = 2^{192} - 2^{64} - 1$[14].

3 Parallelization of Scalar Multiplication

As previously presented, parallel computing can be used to accelerate computation and balancing workload. A task is divided into smaller ones which are then carried out simultaneously by different processors. The parallel computing of scalar multiplication is a hot research topic in cryptography and various solutions have been proposed in literature, but a lot of them are hardware implementations using multi-core [15] or FPGA [16] architecture.

The paper [17] presents a fast exponentiation method using precomputed table. For example we want to calculate g^n where g is a element of Z/pZ and n is a positive integer of length l. We can represent n in base h as follows

$$n = \sum_{i=O}^{l-1} a_i x_i, \quad 0 \leq a_i \leq h-1 \text{ and } 0 \leq i < l \tag{7}$$

We precompute and store $g^{x_0}, g^{x_1}, \ldots, g^{x_{l-1}}$in a table, and then g^n can be computed easily using formula 8.

$$g^n = \prod_{d=1}^{h-1} c_d^d, \quad c_d = \prod_{i:a_i=d} g^{x_i} \tag{8}$$

The computation of g^n is consisted of three main steps:

1. Determine the representation of $n = \sum_{i=O}^{l-1} a_i x_i$ in base h.
2. Compute $c_d = \prod_{i:a_i=d} g^{x_i}$.
3. Compute $g^n = \prod_{d=1}^{h-1} c_d^d$.

This method is based on the precomputation, and it can also be applied to point multiplication by precomputing points $2^1P, 2^2P \ldots 2^{l-1}P$.

Most time is spent in the second and third steps, and both of them can be parallelized. For example, if we have $h-1$ processors, then we can parallelize the second step and each processor calculates its c_d separately. In step 3, each processor can calculate a c_d^d for one d.

An other method based on point precomputation is proposed in [18]. Suppose that we want to calculate $Q = kP$ where Q and P are 2 points represented in Jacobian coordinates and k is a positive integer of 160 bits.

We prepare a precomputed table which consists of 62 points.

$$\begin{aligned} A[s] &= \textstyle\sum_{j=0}^{4} a_{s,j} 2^{32j} G \\ B[s] &= \textstyle\sum_{j=0}^{4} a_{s,j} 2^{16+32j} G \end{aligned} \tag{9}$$

where $1 \leq s \leq 31$ and $a_{s,0}, \ldots, a_{s,4}$ is a binary representation of $s = \sum_{j=0}^{4} a_{s,j} 2^j$. Then the algorithm to compute kP is as follows:

Algorithm 2: Elliptic curve exponentiation based on precomputation

Data: $k = \sum_{i=0}^{l-1} k_i 2^i$, P
Result: kP
for $0 \leq j \leq 15$ **do**
  $u_j = \sum_{i=0}^{4} k_{32i+j} 2^i$;
  $v_j = \sum_{i=0}^{4} k_{32i+16+j} 2^i$;
end
$A[0] = \infty$;
$B[0] = \infty$;
$T = \infty$;
for i *from* 15 *to* 0 **do**
  $T \leftarrow 2T$;
  $T \leftarrow T + A[u_i] + B[v_i]$;
end
return T;

As this method is also based on precomputation, once the precomputed table is prepared, the exponentiation loop can be performed separately by different processors.

In [19] an other method is proposed for performing parallel scalar multiplication $Q = kP$ with 2 processors using a shared memory.

The first processor initially reads P and then keeps scanning k_i and computing point doubling. It writes 2^iP into the buffer whenever a non-zero k_i is detected. The second processor reads 2^iP from the buffer and performs additions. The computation is terminated when there is no more 2^iP in the buffer.

An other method for performing parallel computing of fast exponentiation is presented in [20]. Suppose that we want to calculate g^R where R is a positive

integer of length n. We divide R into h blocks R_i of length $a = \lceil \frac{n}{h} \rceil$, then each R_i is still divided into v smaller blocks $R_{i,j}$ of length $b = \lceil \frac{a}{v} \rceil$ (formula 10).

$$\begin{array}{ll} R &= R_{h-1} \dots R_1 R_0 \quad = \sum_{i=0}^{h-1} R_i 2^{ia} \\ R_i &= R_{i,v-1} \dots R_{i,1} R_{i,0} = \sum_{j=0}^{v-1} R_{i,j} 2^{jb} \end{array} \tag{10}$$

Let $g_0 = g$ and define $g_i = g_{i-1}^{2^a} = g^{2^{ia}}$ for $0 < i < h$. Using formula 10, we can express g^R as

$$g^R = \prod_{i=0}^{h-1} g_i^{R_i} = \prod_{j=0}^{v-1} \prod_{i=0}^{h-1} (g_i^{2^{jb}})^{R_{i,j}} \tag{11}$$

If the binary representation of R_i is $R_i = e_{i,a-1} e_{i,a-2} \dots e_{i,0}$ and $R_{i,j} = e_{i,jb+b-1} e_{i,jb+b-2} \dots e_{i,jb+1} e_{i,jb}$, then the formulas 11 can be rewritten as follows

$$g^R = \prod_{k=0}^{b-1} (\prod_{j=0}^{v-1} \prod_{i=0}^{h-1} g_i^{2^{jb} e_{i,jb+k}})^{2^k} \tag{12}$$

If we precompute and store the following values for all $1 \leq i < 2^h$ and $0 \leq j < v$.

$$\begin{array}{l} G[0][i] = g_{h-1}^{e_{h-1}} g_{h-2}^{e_{h-2}} \cdots g_0^{e_0} \\ G[j][i] = (G[j-1][i])^{2^b} = (G[0][i])^{2^{jb}} \end{array} \tag{13}$$

The formula 12 can be still rewritten as

$$g^R = \prod_{k=0}^{b-1} (\prod_{j=0}^{v-1} G[j][I_{j,k}])^{2^k} \tag{14}$$

where $I_{j,k} = e_{h-1,bj+k} \dots e_{1,bj+k} e_{0,bj+k} (0 \leq j < b)$. Then the computation of g^R can be parallelized using precomputed $G[j][i]$.

We can see that all those parallelization schemes are based on the point precomputation and scalar decomposition. The configuration of elliptic curves, like field type, curve form and coordinate system, doesn't have any impact on the result of calculation. Thus the optimization methods presented in section 2 can be used with the parallelization schemes to improve their performance.

4 Parallel Scalar Multiplication in Wireless Sensor Networks

In this section we present our method of parallel computing for accelerating scalar multiplication on elliptic curves in wireless sensor networks. A sensor network is composed of a great number of low-cost and low-power sensor nodes which are always deployed in hostile territories and then work in unattended

mode. These sensors don't have enough power to perform complicated calculations, but in some disaster monitoring applications, sensors might have only a few seconds to send message back to base station before being destructed.

Our method is designed for event-driven applications in which sensors remain idle during most of the time to preserve energy, but when a sensor detects a critical event, the message should be sent back to base station as fast as possible, even at the expense of consuming more energy. We suppose that in a cluster-based sensor network, sensors use a symmetric cryptographic primitive, like Trivium [21,22], to secure internal cluster communications. For inter-cluster communication, we use the elliptic curve cryptography since it provides increased security and digital signature.

Our method is mainly based on the idea of [20] since it offers a efficient scalar decomposition and it doesn't require shared memory [19]. The goal is to reduce the execution time by asking neighbour nodes to perform computations together. We suppose that the elliptic curve is preloaded before node deployment, and the generator point G doesn't change during the entire lifetime of the network. When a node wants to perform a multiplication $Q = kG$, the node looks for available neighbours in the same cluster and asks them to participate in computation. The node which leads the computation is the *master node* and the other ones are called *slave nodes*.

At first the master node splits the integer k into n blocks B_i of $b = \lceil \frac{l}{n} \rceil$ bits according to the number of available neighbours.

$$B_i = \sum_{j=ib}^{ib+b-1} a_j 2^j \tag{15}$$

As the generator point G is chosen a priori and it doesn't change, the precomputation of points $G_i = 2^{ib}G$ is then possible, and calculation of kG

$$Q = kG = \sum_{i=0}^{l-1} a_i 2^i G \tag{16}$$

can be divided into n independent parts

$$\begin{aligned} Q_0 &= B_0 G \\ Q_1 &= B_1 2^b G \\ &\cdots \\ Q_{n-1} &= B_{n-1} 2^{b(n-1)} G \end{aligned} \tag{17}$$

Then $Q = Q_0 + Q_1 + \ldots + Q_{n-1}$ and each Q_i can be computed independently using the basic Double and Add algorithm.

Before task distribution, the master node copies one of the n blocks into its local memory, for example the block B_0, and then places the remaining blocks B_i where $0 < i < n$ into a task distribution message, and task assignment informations are also sent with B_i.

For example, k is a positive integer of 160 bits and 4 nodes participate in computation of kG. Thus $n = 4$, $b = 40$ and all nodes have G_i for $0 \leq i \leq 3$ precomputed and stored in theirs memories, like in figure 1. The master splits k into 4 blocks of 40 bits. It keeps the block B_0 in its local memory and places the remaining 3 blocks into a task distribution message. Then it attaches task assignment informations to the message, encrypts it using the cluster's cryptographic primitive and broadcasts it to slave nodes.

$G[0]$	$G[1]$	$G[2]$	$G[3]$
G	$2^{40}G$	$2^{80}G$	$2^{120}G$

Fig. 1. Array containing precomputed points

An example of task assignment is given in figure 2. ID_i are the IDs of slave nodes. When the slave nodes receive the message, they decrypt it and calculate respectively $B_1G[1]$, $B_2G[2]$ and $B_3G[3]$.

0	1	2	3	4	5
B_1	B_2	B_3	ID_1	ID_2	ID_3

Fig. 2. Structure of the task distribution message

The entire procedure is driven by a simple protocol (see diagram 3). When a critical event takes place, like forest fire, volcanic eruption or earthquake. There's a chance that several nodes detect this event at almost the same time. All nodes which detect the event should turn on the radio and get ready for parallel computing. The protocol is described by the following steps :

1. A node detects a critical event and it turns on its radio. Then it waits for call of parallel computing from other nodes for a random period.
2. If it doesn't receive any call, then it becomes the Master node, and it broadcasts a call for parallel computing.
3. If it receives a call from an other node, then it becomes a Slave node and it should reply to show its availability.
4. After received all replies from slave nodes, the master node broadcasts the task distribution message and then waits for results.
5. Slaves receive the task distribution message and perform theirs computations.
6. Slaves send theirs results back to their master and get back to initial state.
7. The master node combines the received results to get the final result and sends it back to base station.

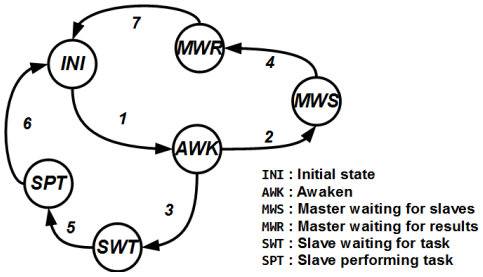

Fig. 3. Protocol of proposed parallel computing scheme

As this method is based on precomputation too, the maximum number of points to precompute and store depends on the requirement of the application and the memory size of sensor nodes. More points are precomputed, more nodes can participate in parallel computing, but more energy will be consumed.

5 Experimental Studies

To test the performance of our method, we have implemented it in nesC on Crossbow's Telosb motes [23] which are then deployed randomly in a zone of $10m \times 10m$. We ask them to keep repeating scalar multiplications $Q = kG$ on a preloaded elliptic curve defined over $NIST_{192}$ prime field [14] for both affine and Jacobian coordinates, G is the generator point of the curve and the scalar k is an integer of 160 bits using NAF representation.

The execution times in milliseconds with and without parallel computing are given in table 2. It's hard to compare the absolute values of the results to other implementations due to the variety of techniques and test scenarios. Here we only interest in the performance gain produced by parallel computing.

Table 2. Execution time (ms) and gain of proposed parallel computing scheme

Nb of nodes	Affine	Gain	Jacobian	Gain
1	2307.27	–	1003.55	–
2	1189.96	48.43%	549.71	45.22%
3	861.48	62.66%	424.60	57.69%
4	665.68	71.15%	342.51	65.87%
5	583.29	74.72%	309.93	69.12%
6	581.32	74.80%	311.87	68.92%

We can see that the gain increases when more nodes participate in the calculation. Figure 4 shows that the execution time decreases gradually with increase in the number of nodes. Suppose that the execution time using p nodes is T_p,

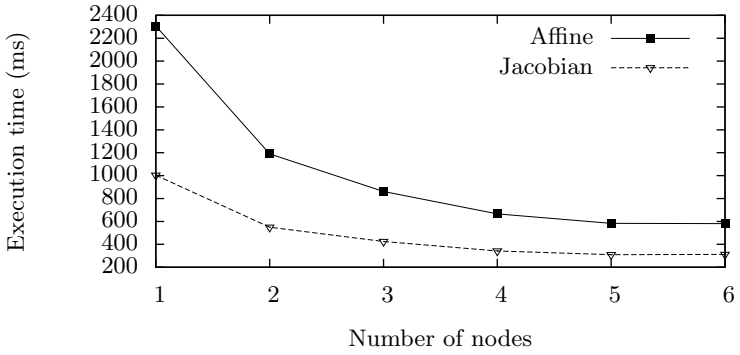

Fig. 4. Execution times using the proposed parallel computing scheme

we evaluate the performance of our method by calculating its speedup $S_p = \frac{T_1}{T_p}$ and the results are given in table 3 and represented graphically in figure 5.

Table 3. Speedup of the proposed parallel computing scheme

Nb of nodes	1	2	3	4	5	6
Affine	1.00	1.94	2.68	3.47	3.96	3.97
Jacobian	1.00	1.83	2.36	2.93	3.24	3.22

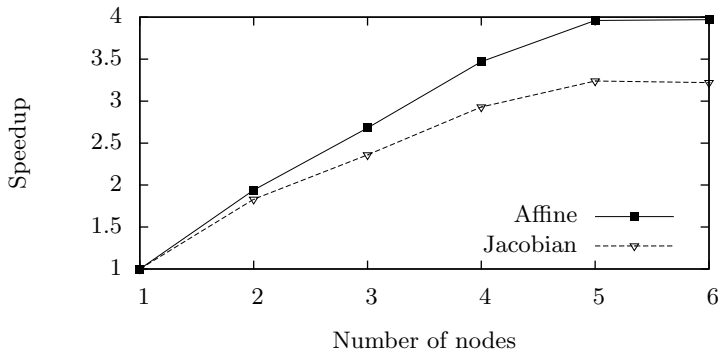

Fig. 5. Speedup $S_p = \frac{T_1}{T_p}$ of the proposed parallel computing scheme

There is a considerable drop in speedup when we use more than 5 nodes, since the network needs more time to schedule radio communications and the master node requires more time to combine the received points and get the final result. The parallel overhead is shown in table 4 and in figure 6. We can see

that when we use more than 5 nodes, there is a significant increase in overhead. Thus according to the result of our experiment, the number of nodes should be limited to less than 5.

Table 4. Overhead (ms) of the proposed parallel computing scheme

Nb of nodes	1	2	3	4	5	6
Average overhead	0.00	36.33	92.39	88.86	121.84	196.78

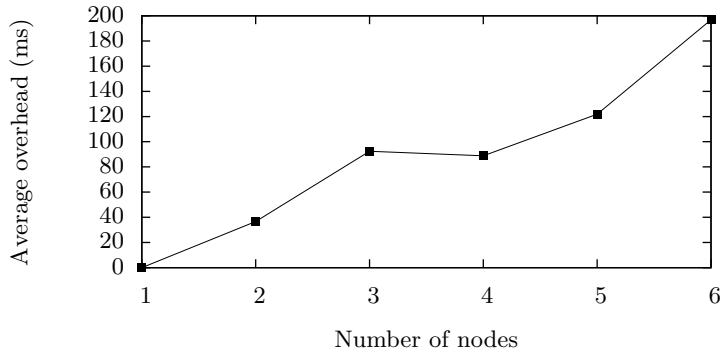

Fig. 6. Parallel overhead (ms) of the proposed method

Suppose that the elliptic curve is defined in a field of 192 bits [14], so a point in affine coordinate (x, y) can be represented by 2 integers of 192 bits, then it can also be converted to Jacobian coordinates $(x, y) \rightarrow (x, y, 1)$. The memory size needed to store precomputed points are given in table 5.

Table 5. Memory needed (Byte) to store precomputed points

Nb of nodes	1	2	3	4	5	6
Memory needed	0	48	96	144	192	240

Telosb mote has 48KB of Flash memory [23] and Micaz has 128KB [24], which is obviously sufficient to store those precomputed points.

As previously presented, during the parallel computing more nodes are involved in the computation, so more energy is consumed since they need to communicate between them. In order to estimate the energy consumption of our method, we have run simulations with Avrora [25]. It gives only theoretical simulated results, but they're precise enough to compare the energy consumption according to the number of nodes participating in the computation (see table 6).

Table 6. Energy consumption (Joule) of proposed method. TEC: Total energy consumption.

Nb of nodes	1	2	3	4	5	6
TEC	0.889	2.125	3.156	4.192	5.225	6.256

In figure 7, we can see that when computation is done on a single node, it consumes very little energy. However when parallel computing is used, as slave nodes have to receive tasks from the master node and return the results back to it, they consume much more energy.

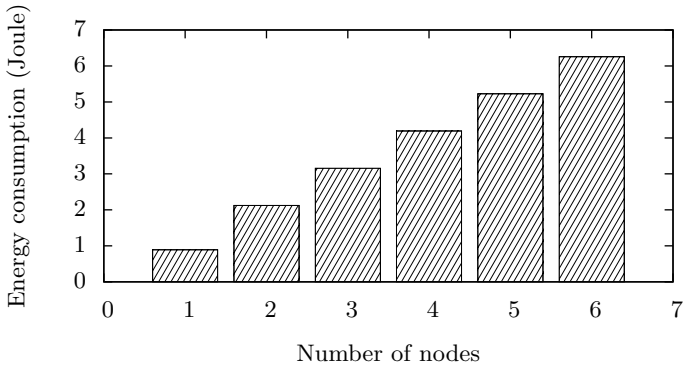

Fig. 7. Total energy consumption (Joule)

When we parallelize the calculation between 6 nodes, our method provides a gain of around 70.0% in execution time. We have also tried to encrypt and decrypt the task distribution message using the symmetric cryptographic primitive Trivium, the result shows that the calculation can be done in less than 1 millisecond, thus its impact on the result of the performance test can be safely neglected.

However as previously presented, in sensor networks sensor nodes work in unattended manner and communicate with each other using wireless connection which might be unstable due to radio interferences and low battery level. In such cases the master node should be able to detect the faults. For example we may divide the computation into 2 tasks which are then carried out by 3 nodes. The master node compare the results returned by 2 slaves, if they're different, it means that 1 of the 2 results is wrong. We may also apply trust and reputation assessment techniques in our system [26]. If a node always returns erroneous results, it will be excluded from the parallel computing. There are still other strategies, but the presentation of fault tolerance techniques is not the objective of this paper.

6 Conclusion

In this paper we use parallel computing technologies to accelerate the scalar multiplication on elliptic curves. We have tested our method using up to 6 Telosb motes, and the results show that we obtain a gain of about 70.0% in execution time. However we propose that the number of nodes should be limited to less than 5 due to parallel overhead. As the method is based on precomputation, nodes need to store precomputed points locally.

The only drawback is the energy consumption since nodes have to communicate with each other for task distribution and result retrieval. Thus parallel computing should not be used as the default computation mode in wireless sensor networks, it can only be used in cases where execution time is the most critical factor, like in disaster monitoring and military applications.

In our future work, we will try to reduce the energy consumption by minimizing radio transmissions. As nodes communicate with each other using unreliable wireless communication, fault tolerance will also be taken into account.

References

1. Akyildiz, I., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: a survey. Computer networks **38**(4) (2002) 393–422
2. Werner-Allen, G., Lorincz, K., Ruiz, M., Marcillo, O., Johnson, J., Lees, J., Welsh, M.: Deploying a wireless sensor network on an active volcano. Internet Computing, IEEE **10**(2) (2006) 18–25
3. Kim, S., Pakzad, S., Culler, D., Demmel, J., Fenves, G., Glaser, S., Turon, M.: Wireless sensor networks for structural health monitoring. In: Proceedings of the 4th international conference on Embedded networked sensor systems, ACM (2006) 427–428
4. Baker, C., Armijo, K., Belka, S., Benhabib, M., Bhargava, V., Burkhart, N., Der Minassians, A., Dervisoglu, G., Gutnik, L., Haick, M., et al.: Wireless sensor networks for home health care. In: Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on. Volume 2., IEEE (2007) 832–837
5. Welsh, M., Moulton, S., Fulford-Jones, T., Malan, D.: Codeblue: An ad hoc sensor network infrastructure for emergency medical care. In: International Workshop on Wearable and Implantable Body Sensor Networks, April, London, UK. (2004)
6. Gosnell, T., Hall, J., Jam, C., Knapp, D., Koenig, Z., Luke, S., Pohl, B., Schach von Wittenau, A., Wolford, J.: Gamma-ray identification of nuclear weapon materials. Technical report, Lawrence Livermore National Lab., Livermore, CA (US) (1997)
7. Walters, J., Liang, Z., Shi, W., Chaudhary, V.: Wireless sensor network security: A survey. Security in distributed, grid, mobile, and pervasive computing **1** (2007) 367
8. Zhou, Y., Fang, Y., Zhang, Y.: Securing wireless sensor networks: a survey. Communications Surveys & Tutorials, IEEE **10**(3) (2008) 6–28
9. Gura, N., Patel, A., Wander, A., Eberle, H., Shantz, S.: Comparing elliptic curve cryptography and rsa on 8-bit cpus. In: Cryptographic hardware and embedded systems–CHES 2004: 6th international workshop, Cambridge, MA, USA, August 11-13, 2004: proceedings. Volume 6., Springer-Verlag New York Inc (2004) 119

10. Gordon, D.: A survey of fast exponentiation methods. Journal of algorithms **27**(1) (1998) 129–146
11. Miller, V.: Use of elliptic curves in cryptography. In: Advances in Cryptology - CRYPTO'85 Proceedings, Springer (1986) 417–426
12. Koblitz, N.: Elliptic curve cryptosystems. Mathematics of computation **48**(177) (1987) 203–209
13. Hankerson, D., Vanstone, S., Menezes, A.: Guide to elliptic curve cryptography. Springer-Verlag New York Inc (2004)
14. Brown, M., Hankerson, D., López, J., Menezes, A.: Software implementation of the nist elliptic curves over prime fields. Topics in Cryptology - CT-RSA 2001 (2001) 250–265
15. Panda, B., Khilar, P.: Fpga based implementation of parallel ecc processor. In: Proceedings of the 2011 International Conference on Communication, Computing & Security, ACM (2011) 453–456
16. Purnaprajna, M., Puttmann, C., Porrmann, M.: Power aware reconfigurable multiprocessor for elliptic curve cryptography. In: Design, Automation and Test in Europe, 2008. DATE'08, IEEE (2008) 1462–1467
17. Brickell, E.F., Gordon, D.M., Mccurley, K.S., Wilson, D.B.: Fast exponentiation with precomputation: Algorithms and lower bounds. Technical report, in Proc. of EUROCRYPT'92 (1995)
18. Miyaji, A., Ono, T., Cohen, H.: Efficient elliptic curve exponentiation. In: Information and communications security: first international conference, ICIS [ie ICICS]'97, Beijing, China, November 11-14, 1997: proceedings. Volume 1334., Springer Verlag (1997) 282
19. Ansari, B., Wu, H.: Parallel scalar multiplication for elliptic curve cryptosystems. In: Communications, Circuits and Systems, 2005. Proceedings. 2005 International Conference on. Volume 1., IEEE (2005) 71–73
20. Lim, C., Lee, P.: More flexible exponentiation with precomputation. In: Advances in Cryptology - CRYPTO'94, Springer (1994) 95–107
21. De Canniere, C., Preneel, B.: Trivium specifications. estream. ECRYPT Stream Cipher Project, Report **30** (2005) 2005
22. Raddum, H.: Cryptanalytic results on trivium. eSTREAM, ECRYPT Stream Cipher Project, Report **39** (2006) 2006
23. MEMSIC: Telosb mote platform datasheet. `http://www.memsic.com/products/wireless-sensor-networks/wireless-modules.html` (11 2011)
24. MEMSIC: Mica2/micaz mote platform datasheet. `http://www.memsic.com/products/wireless-sensor-networks/wireless-modules.html` (6 2011)
25. Titzer, B., Lee, D., Palsberg, J.: Avrora: Scalable sensor network simulation with precise timing. In: Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on, IEEE (2005) 477–482
26. Chen, H., Wu, H., Zhou, X., Gao, C.: Reputation-based trust in wireless sensor networks. In: Multimedia and Ubiquitous Engineering, 2007. MUE'07. International Conference on, IEEE (2007) 603–607