

Efficient Implementation of Elliptic Curve Cryptography (ECC) on VLIW-Micro-Architecture Media Processor

Yu Hu, Qing Li and C.-C. Jay Kuo

*Integrated Media Systems Center and Department of Electrical Engineering
University of Southern California, Los Angeles, CA 90089-2564
E-mails: yhu0@usc.edu, {qingli,cckuo}@sipi.usc.edu*

Abstract

Elliptic curve cryptography (ECC) serves as an excellent candidate for secure embedded multimedia applications due to its small key size and high security protection. With performance profiling, several major bottlenecks of the ECC implementation are identified, and some suitable integer multiplication schemes over the VLIW and SIMD architecture are proposed. In particular, FIR-based multiplication using the special FIR instruction provided by Trimedia TM1300 was studied. The performance improvement of proposed schemes is reported and discussed.

1. Introduction

Embedded multimedia systems with mobility have become a focal point in the digital communication era. The need of privacy and data integrity protection demands a small-size and resource-limited embedded system yet with multimedia and security functionalities. Elliptic curve cryptography (ECC) is an excellent candidate for secure embedded multimedia applications due to its small key size and high security protection.

The implementation and optimization of ECC algorithms can be categorized into three classes. The first class tries to finetune the parameters of elliptic curves to improve the performance such as the field selection [1]. The second class of research focused on optimizing arithmetic operations, *e.g.* the multiplication in $GF(p^k)$ for ECC [2]. The third class considers hardware platform-dependent implementations, where the ECC performance is improved based on features of the specific platform. For example, Guajardo *et al.* [3] presented the ECC implementation on TI MSP430x33x controller and Aydos *et al.* [4] speeded up ECC on the ARM processor. In our opinion, it is important to study the ECC implementation and optimization based on features of a hardware platform. Thus, this is the approach taken in this work.

Despite that the VLIW and SIMD architectures are popular in embedded media processors and that there

exists a great demand on secure embedded multimedia applications, there is little work on the implementation and optimization of ECC on the VLIW and SIMD architecture. The goal of this work is to obtain an efficient ECC implementation and to find a way to enhance the ECC performance under the VLIW and SIMD architecture.

To achieve this goal, we proceed as follows. First, the workload of ECC over Trimedia TM1300, a brand media processor, is briefly analyzed to figure out the bottleneck of ECC over the VLIW and SIMD platform. Then, we choose the multiplication of two big numbers as the first step to improve the ECC performance due to its high percentage in terms of the cycle count, *i.e.* execution time. Several multiplication schemes are compared, and the FIR-based multiplication is chosen for the SIMD architecture. Experimental results show that our implementation can speed up ECC by a factor between 1.5 to 2 times depending on the key size. The main approach in speeding up ECC under the VLIW and SIMD architecture is concluded.

The rest of the paper is organized as follows. First, the background of multimedia processors and the ECC implementation are given in Section 2. Performance profiles are analyzed in Section 3. Section 4 provides some optimized multiplication of big numbers. Finally, experiment results and concluding remarks are given in Section 5.

2. ECC on Embedded Media Processors

2.1 Architectures of Embedded Media Processors

Most embedded multimedia processors employ the Single Instruction Multiple Data (SIMD) and the Very Long Instruction Word (VLIW) micro-architecture. The SIMD architecture can exploit the inherent *Data Level Parallelism (DLP)* in multimedia applications. The processors with the VLIW architecture issue and complement multiple instructions in one cycle. Take our test bed, TM1300, as an example. It can issue 5 instructions simultaneously. The advantage of VLIW is accomplished using the *Instruction Level Parallelism (ILP)*. To fully utilize this feature, it often requires

some optimization from either the compiler or the programmer.

2.2 ECC Implementation

In recent years, ECC has received more attention due to its smaller key size yet with a similar degree of security protection in comparison with the traditional public key cryptographic algorithm such as RSA. For example, ECC with a key size of 160 bits can have the same security level as RSA with 1024 bits. The ECC protocol is the conventional public key cryptographic algorithm based on elliptic curves. For example, the elliptic curve Diffie-Hellman (ECDH) scheme is a variant of the Diffie-Hellman cryptographic algorithm, and the elliptic curve digital signature algorithm (ECDSA) is the analogue of DSA. Among them, the most frequently employed benchmark in experiments is ECDSA.

3. ECC Profiling on Media Processors

We are interested in the ECC workload profiling on the VLIW and SIMD architecture of media processors so as to identify the most time-consuming part. Following the NIST FIPS 186-2 recommendation, ECDSA over prime fields is adopted as the benchmark in our experiment. Experimental data indicate that two function modules, “redc” and “multiply”, occupy a large portion of instruction counts for 5 prime fields. The percentages of these two functions with respect to the total instruction count with different key sizes are listed in Table 1.

Table 1: Occupancy percentages of two main functions in the prime field

Key Size		redc (%)	multiply (%)	Total (%)
192	Signature	22.23	18.30	40.53
	Verification	21.51	17.72	39.23
224	Signature	23.70	19.46	43.16
	Verification	22.92	18.92	41.84
256	Signature	25.30	20.80	46.10
	Verification	24.37	20.15	44.52
384	Signature	27.66	22.84	50.50
	Verification	26.83	22.36	49.19
521	Signature	29.64	24.44	54.08
	Verification	28.87	24.07	52.94

For the VLIW micro-architecture, the instruction issue rate is an important metric for program efficiency. The higher the issue rate, the more efficient the program. The issue rates for two functions above and the overall program are given in Table 2. Note that only the issue rates of the 192-bit key size for the prime field are given in Table 2, since the instruction issue rate does not vary too much (less than 10%) as the key

size increases for both the signature and the verification stages.

Table 2. ECC instruction issue rates

	redc	multiply	total/average
Instruction issues	1.54	1.59	1.21

Considering the capacity of 5 issue slots provided by TM1300, the ECC implementation given above is obviously inefficient and the throughput is quite low. As we show later, exploiting ILP and splitting data dependency are key points to solve this problem. The efficient implementation of multiplication, one of two time-consuming functions on the SIMD and VLIW structured media processor, is examined in the next section.

4. Efficient Multiplication on Embedded Media Processor

It is desirable to find an efficient implementation of the multiplication of two large numbers for ECC due to the high percentage of multiplication operations in terms of execution time. Moreover, we are interested in the impact of the VLIW and SIMD architecture on ECC. Furthermore, we would like to find a way to enhance the performance of ECC under this architecture.

The multiplication of two big integers can be treated as the multiplication of two polynomials [4]. For two large numbers x and y , we set the product of x and y to be z . First, let us write x , y and z as

$$\begin{aligned} x &= x_0 + x_1 g^1 + x_2 g^2 + \dots + x_n g^n, \\ y &= y_0 + y_1 g^1 + y_2 g^2 + \dots + y_n g^n, \\ z &= z_0 + z_1 g^1 + z_2 g^2 + \dots + z_{2n} g^{2n}, \end{aligned} \quad (1)$$

where g is the digital base. Thus, the number and value of coefficients will be different with a different g . The subscripts distinguish different coefficients with the corresponding digit. Hence, x_0 and x_n present respectively the least significant digit and the most significant digit of x .

$$\begin{aligned} z_0 &= x_0 \cdot y_0 \\ &\vdots \\ z_n &= x_0 \cdot y_n + x_1 \cdot y_{n-1} + \dots + x_n \cdot y_0 \\ &\vdots \\ z_{2n} &= x_n \cdot y_n \end{aligned} \quad (2)$$

Basically, x , y and z are stored in three arrays and the elements of the array are set as a 32-bit integer. In other words, g in the above equations is chosen to be 2^{32} . Embedded media processors generally take more cycles to execute the multiplication of two 32-bit registers than the general purpose processor. For instance, TM1300 cannot store the 64-bit product into two 32-bit registers. Instead, the multiplication is done by a two-step multiplication: using ‘mul’ to compute

the less significant 32-bit result, and using ‘mulm’ to compute the more significant 32-bit result. Consequently, we should carefully implement the multiplication of two large integers on the embedded media processors. Several implementation schemes for multiplication are discussed below.

There are two general implementation schemes called the rectangular fashion and the diagonal fashion multiplication according to the data element access sequence [5]. The pseudo-codes for the rectangular fashion implementation and the unrolled diagonal fashion implementations are given in Figure 1.

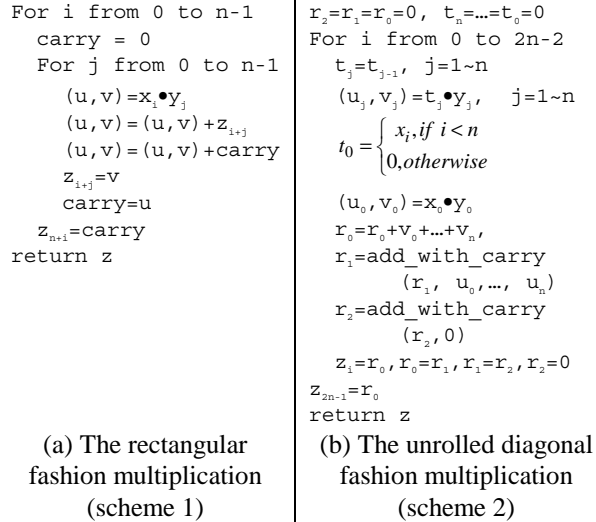
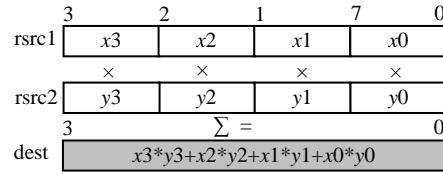


Figure 1. Two implementations of multiplication.

The advantage of the rectangular fashion multiplication is the fixed length for the loops. However, its drawback is the data dependency among each loop, *i.e.* the computation in each step needs the result of ‘carry’ computed in the previous step. The rectangular method takes the inherent character of the dependency that cannot be solved with loop unrolling. We unroll the inner loop of the original rectangular method, named as scheme 4, to show the impact of data dependency. Due to the limited space, the pseudo-code for scheme 4 is not given here.

The original diagonal fashion implementation computes all products in the inner loop that contribute to the same degree of the digital base g such that the calculation in each loop is independent. The drawback of the diagonal method is that the length of the diagonals varies. In our experiment, we unroll the inner loop and compensate variables with zero to fit the fixed length for the loop, shown as scheme 2 in Figure 1(b). In each loop, elements for array x are shifted among variable series t . Except for shifting, there is little data dependency. The dependency of the shift operation can be hidden by paralleled instructions. The condition for setting t_0 can be easily done using the guard option

provided by most media processors. It is confirmed by experimental results that scheme 2 has the best performance.



(a) Illustration for the FIR instruction

$t_0=t_1=...=t_n=carry=0$;

For i from 0 to $2n-1$ do

$$x = \begin{cases} x_i, & \text{if } i < n \\ 0, & \text{if } i \geq n \end{cases}$$

Left-shift 8 bits from x to t_n, t_{n-1}, \dots, t_0

Do FIR for all the pairs of t_m and y_n

Set r_0 as the sum of all FIR results

Compute r_1 with the same steps as above (left-shift, FIR, sum)

Compute r_2 by the same progress

Compute r_3 by the same progress

$z_i = r_0 + (r_1 << 8) + (r_2 << 16) + (r_3 << 24) + carry$

$carry = add_with_carry((r_2 >> 16), (r_3 >> 8))$

return z

(b) FIR based multiplication (scheme 3)

Figure 2. Multiplication implementation using FIR instructions.

Most embedded media processors provide the FIR instruction for fast computation in media applications. For example, in TM1300, instruction ‘ufir8uu’ is used to compute the unsigned sum of the product of unsigned bytes, which is illustrated in Figure 2(a). Apparently, the FIR instruction can be applied to the multiplication of two big numbers. We apply the FIR instruction to every four elements in each step as shown in Figure 2(b) and sum all results of FIR together to get the result for the corresponding coefficients of z . Note that the data elements operated in FIR instructions are in the unit of one byte, *i.e.* 8-bit. In other words, g in equation (1) is equal to 2^8 , which is different from 2^{32} in the general implementation.

Note that all variables in the pseudo codes above have the 32-bit length, including x , y , z and r . $r_0 \sim r_3$ store the consecutive four sums of the FIR results, *i.e.* the four coefficients of the polynomial with g equal to 28. Moreover, left-shifting 8 bits for computing each r is due to that the available FIR instruction only covers four bytes. Actually, this scheme has already unrolled the loop for 4 times that is determined by the load/store unroll factor. Thus, the access over arrays of x , y and z can be integer-aligned. The drawback of this algorithm is that the additional shift operation and the pack operation are needed as the overhead. However, the shift and the pack operations are generally treated as fast operations and these two operations can be fed into

other slots except the FIR instructions for the VLIW architecture of media processors. Note that Comba's suggestion of full unrolling was not adopted in schemes 2, 3, and 4, since the cache and memory resources in the embedded system are limited and the large code size would degrade the performance.

5. Experimental Results and Conclusion

Table 3 gives the performance comparison of 192-bit key size among various multiplication schemes discussed in Section 4. The unrolled diagonal fashion multiplication (*i.e.* scheme 2) and the FIR-based multiplication (*i.e.* scheme 3) are better than schemes 1 and 4. Scheme 2 achieved the best performance since the efficiency of the instruction multiplication is higher than that of the FIR instruction. To complete the multiplication of two 32-bit integers, we have to either call the multiplication instruction two times or call the FIR instruction four times. Even though the FIR instruction can save the cycle on the "addition" instruction, it is not most time-consuming.

Table 3. Comparison of multiplication implementation schemes.

	Instruction count	Issue rate
Scheme 1	6,465,452	2.25
Scheme 2	4,161,626	3.70
Scheme 3	5,344,969	3.96
Scheme 4	5,985,629	1.42

It is also remarkable to see that the issue rate for the unrolled rectangular fashion multiplication is even less than that of the original rectangular method. This is due to data dependency among loop iterations, which confirm our analysis above.

Table 4. The performance improve of scheme 1 and scheme 2 on different key size

	Instruction Count		Performance Improvement	Issue Rate
	scheme 1	scheme 2		
P-192	4,285,330	2,955,084	1.45	2.04
P-224	6,465,452	4,161,626	1.55	3.70
P-256	9,575,976	4,928,144	1.94	4.24
P-384	31,007,503	14,613,947	2.12	4.46
P-521	81,281,140	37,891,371	2.14	4.81

Table 4 shows the performance improvement of scheme 2 over scheme 1 with different key sizes. As shown in the table, the issue rate for the proposed multiplication algorithm (scheme 2) is much higher than the general multiplication (scheme 1). This is due to the fact that scheme 2 is four times unrolled, and various types of instructions, including FIR, pack, shift, load/store, addition, are developed to fit the five slots of TM1300. Furthermore, the issue rate is increasing as the key size arises. As discussed above, more registers

will be used to represent the variable series t so that more instructions will reside in each loop. As a result, we can get a higher issue rate at a longer key size. Besides, the performance improvement gets better as the key size increases, *e.g.* from 12.6% at the 192-bit key size to 32.6% at the 521-bit key size. This is due to the increasing issue rate as the key size increases. For comparison, the issue rate for scheme 1 is almost the same for different key sizes as shown in Table 2.

We can draw the following conclusions for conducted research in the current work. The SIMD architecture is not as much helpful to the cryptographic algorithms as to media processing applications, since the typical data type in media applications is the 8-bit or the 16-bit integer, *i.e.* byte or short. In contrast, large integers are utilized in ECC, since most of the operations are applied to larger numbers. Thus, it is desirable to demand that the length of the operands of the instruction be as long as possible. This explains why the FIR-based multiplication is slower than the unrolled diagonal fashion multiplication. Loop unrolling is desirable for the crypto algorithm to exploit ILP under the VLIW architecture. Once the loop is unrolled, the compiler can find enough instructions to fit the slots for performance improvement. The performance will be further enhanced if efficient multiplication and addition instructions on 32-bit operands are supported in the embedded media processors. For example, the result of multiplication of two 32-bit integers can be stored in two 32-bit registers rather than calling multiplication instruction twice.

References

- [1] D.V. Bailey, "Optimal extension fields for fast arithmetic in public-key algorithms", *In Advances in Cryptology-Crypto '98*, volume 1462, Springer-Verlag, 1998, pp. 472-485.
- [2] J.-C. Bajard, L. Imbert, C. Negre and T. Plantard, "Efficient multiplication in $GF(p^k)$ for Elliptic Curve Cryptography", Proceedings of the 16th IEEE Symposium on Computer Arithmetic (ARITH'03), Spain, 2003, pp.181-187
- [3] J. Guajardo, R. Blümel, U. Krieger, C. Paar, "Efficient implementation of elliptic curve cryptosystems on the TI MSP430x33x family of microcontrollers", The International Workshop on Practice and Theory in Public Key Cryptography (PKC'01), Korea, 2001, pp. 365-382
- [4] M. Aydos, T. Yantk, Ç.K. Koç, "An high-speed ECC-based wireless authentication protocol on an ARM microprocessor", the 16th Annual Computer Security Applications Conference (ACSAC'00), Louisiana, 2000, pp. 401-418
- [5] Comba, P.G., "Exponentiation cryptosystems on the IBM PC", *IBM Systems Journal*, Vol. 29, No. 4 1990, pp. 526-538.