Improved Modular Multiplication for Optimal Prime Fields*

Hwajeong Seo[1], Zhe Liu[2], Yasuyuki Nogami[3],
Jongseok Choi[1], and Howon Kim[1]**

[1] Pusan National University,
School of Computer Science and Engineering,
San-30, Jangjeon-Dong, Geumjeong-Gu, Busan 609–735, Republic of Korea
{hwajeong,jschoi85,howonkim}@pusan.ac.kr
[2] University of Luxembourg,
Laboratory of Algorithmics, Cryptology and Security (LACS),
6, rue R. Coudenhove-Kalergi, L–1359 Luxembourg-Kirchberg, Luxembourg
{zhe.liu}@uni.lu
[3] Okayama University,
Graduate School of Natural Science and Technology,
3-1-1, Tsushima-naka, Kita, Okayama, 700-8530, Japan
{yasuyuki.nogami}@okayama-u.ac.jp

Abstract. Optimal Prime Fields (OPFs) are considered to be one of the best choices for lightweight elliptic curve cryptography implementation on resource-constraint embedded processors. In this paper, we revisit efficient implementation of the modular arithmetic over the special prime fields, and present improved implementation of modular multiplication for OPFs, called Optimal Prime Field Coarsely Integrated Operand Caching (OPF-CIOC) method. OPF-CIOC method follows the general idea of (consecutive) operand caching technique, but has been carefully optimized and redesigned for Montgomery multiplication in an integrated fashion. We then evaluate the practical performance of proposed method on representative 8-bit AVR processor. Experimental results show that the proposed OPF-CIOC method outperforms the previous best known results in ACNS'14 by a factor of 5 %. Furthermore, our method is implemented in a regular way which helps to reduce the leakage of side-channel information.

Keywords: Montgomery Multiplication, Optimal Prime Fields, Embedded Processors, Public Key Cryptography, Operand Caching, Consecutive Operand Caching

* This work was supported by the Industrial Strategic Technology Development Program (This work was supported by the ICT R&D program of MSIP/IITP. [10043907 , Development of high performance IoT device and Open Platform with Intelligent Software]).

** Corresponding Author

1 Introduction

Public key cryptography applications including RSA [15], ECC [6] and pairing-based cryptography [16] are commonly used for secure and robust network services. These protocols highly rely on finite field operations. The main difference between real world number and finite field representation is that finite field computations should conduct reduction process once results go beyond the size of target field. Montgomery algorithm [14] is one of the efficient algorithms to perform the modular multiplication and squaring since it replaces expensive division operation with normal multiplication operations. Recently, a variant of Montgomery multiplication on OPFs was introduced in [3]. This method can be seen as a simplified version of Montgomery multiplication on OPFs, which is proposed to enhance the performance of Elliptic Curve Cryptography (ECC) on 8-bit AVR processors. One of the features of OPFs is the low hamming weight, which allows to remove or replace part of multiplication operations with several addition instructions when using Montgomery algorithm to perform modular multiplciation.

In this paper, we present a novel technique for implementing the Montgomery multiplication on OPFs. Instead of adopting the "traditional" multiplication techniques, e.g. operand scanning, product scanning and hybrid scanning multiplication, our work follows the state-of-the-art (consecutive) operand caching and has been finely redesigned for OPF-Montgomery algorithm. For practical performance evaluation, we implemented the proposed methods on 8-bit AVR processors and the performance enhancements are 5 % than previous best known results in ACNS'14 [13]. The remainder of this paper is organized as follows. In Section 2 and 3, we recap previous multi-precision multiplication methods and OPF-Montgomery algorithms. In Section 4, we present novel OPF-Montgomery multiplication. In Section 5, we describe the performance evaluation on 8-bit RISC microprocessors. Finally, Section 6 concludes the paper and shows the ideas for future work.

2 Multi-precision Multiplication

Multi-precision multiplication is a crucial operation for modular multiplication. In the past several decades, a large body of research has been attempted to speed up the performance of multiplication on 8-bit processors. The most basic technique is called operand scanning method, which consists of two parts, i.e. the inner and outer loops. In the inner loop, one register holds a digit of an operand and computes the partial product by multiplying all the digits of another operand. While in the outer loop, the index of operand increases by a word-size and then the inner loop is executed. An alternative method is called product scanning method, which computes all partial products in the same column by multiplication and addition [2]. Since each partial product in the column is computed and then accumulated, registers are not needed for intermediate results. The results are stored once, and the stored results are not reloaded since all computations have already been conducted. In CHES'04, the classical hybrid scanning method was proposed which combines both of the advantages of operand scanning and product scanning. Hybrid scanning method employs the product scanning as the outer loop and operand scanning method as the inner loop. This method reduces the number of `load` instructions by sharing the operands within one block [5]. In CHES'11, the operand caching (OC) method was introduced [7]. The method follows the product scanning method [2], but it divides the calculation into several row sections. By reordering the sequence of inner and outer row sections, the operands which have been loaded in working registers are reused for the next partial products. A few store instructions are added, but the number of required load instructions is reduced. However, a straightforward implementation of OC method has to reload operands whenever a row is changed, which generates unnecessary overheads. In order to avoid these shortcomings, an advanced version of operand caching named consecutive operand caching (COC) method was introduced at WISA'12 [17]. COC provides a connection point among rows that share the common operands for partial products.

3 Optimal Prime Field Montgomery Algorithm

Table 1. OPF prime for 160-, 192-, 224- and 256-bit [3]

160-bit: $52542 \times 2^{144} + 1$
0xCD3E000000000000000000000000000000000001
192-bit: $55218 \times 2^{176} + 1$
0xD7B200000000000000000000000000000000000000000001
224-bit: $50643 \times 2^{208} + 1$
0xC5D30000000000000000000000000000000000000000000000000001
256-bit: $37266 \times 2^{240} + 1$
0x919200000000000000000000000000000000000000000000000000000000001

The Montgomery algorithm was firstly proposed in 1985 [14]. Montgomery algorithm avoids division in modular multiplication and reduction by introducing simple shift operations. Given two integers A and B and the modulus M, in order to compute the product $P = A \cdot B \bmod M$ using Montgomery method, the first step is to convert the operands A and B into Montgomery domain, namely, $A' = A \cdot R \bmod M$ and $B' = B \cdot R \bmod M$. For efficiency, the Montgomery residue R is generally selected as a power of 2 and the constant $M' = -M^{-1} \bmod 2^r$ has to be pre-computed. Montgomery multiplication can be computed in the following three steps: (1)$P = A \cdot B$, (2) $Q = P \cdot M' \bmod 2^r$, (3) $Z = (P + Q \cdot M)/2^r$.

In 2006, a special family of prime fields, named Optimal Prime Fields (OPFs), was proposed by Großschädl in [3]. A typical y-bit OPF prime M can be represented as the form $M = U \cdot 2^k + V$. U and V are relatively small coefficients

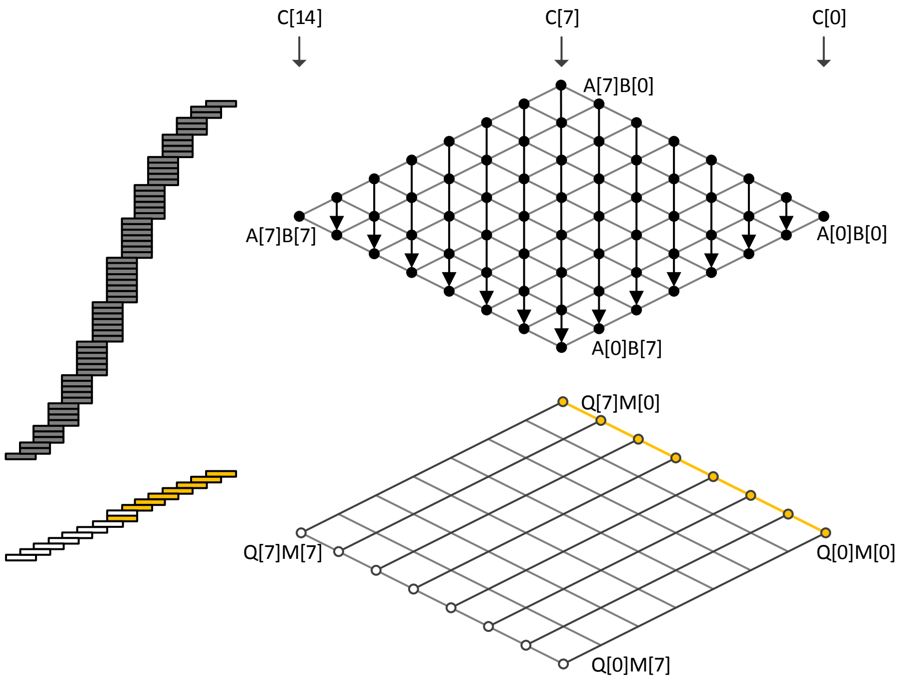

Fig. 1. Separated Product Scanning Method for Optimal Prime Fields

compared to 2^k, U is normally chosen as 8-, 16-bit which can be stored into one or two registers on 8-bit processor, V has several bits. Character k denotes $y - m \cdot w$ where m is a small integer and $m \cdot w$ is the size of U. The OPFs chosen in [3] set U as 16-bit long integer and V as 1 and is formalized in $M = U \cdot 2^{(y-16)} + 1$. Most of bits of OPF prime are 0 except a few bits in most and least significant words. Some examples of OPF are given in Table 1. Due to low hamming weight of optimal prime field, Montgomery multiplication is much simpler than ordinary counterparts. Recently, elliptic curve cryptography implementations over OPFs have been reported, for example, the work in [10] used OPF as the underlying field to evaluate GLV and Montgomery curves on 8-bit AVR processors. Their results show that OPF is efficient yet secure prime field which can be used for lightweight elliptic curve cryptography implementation.

Throughout the paper, we will use the following notations. Let A and B be two operands with a length of y-bit that are represented by multiple-word arrays. Each operand is written as follows: $A = (A[n-1], ..., A[2], A[1], A[0])$ and $B = (B[n-1], ..., B[2], B[1], B[0])$, whereby $n = \lceil y/w \rceil$, and w is the word size. The result of multiplication $C = A \cdot B$ is twice length of A, and represented by $C = (C[2n-1], ..., C[2], C[1], C[0])$. For clarity, we describe the method using a multiplication structure and rhombus forms. The multiplication structure describes order of partial products from top to bottom and each point in rhom-

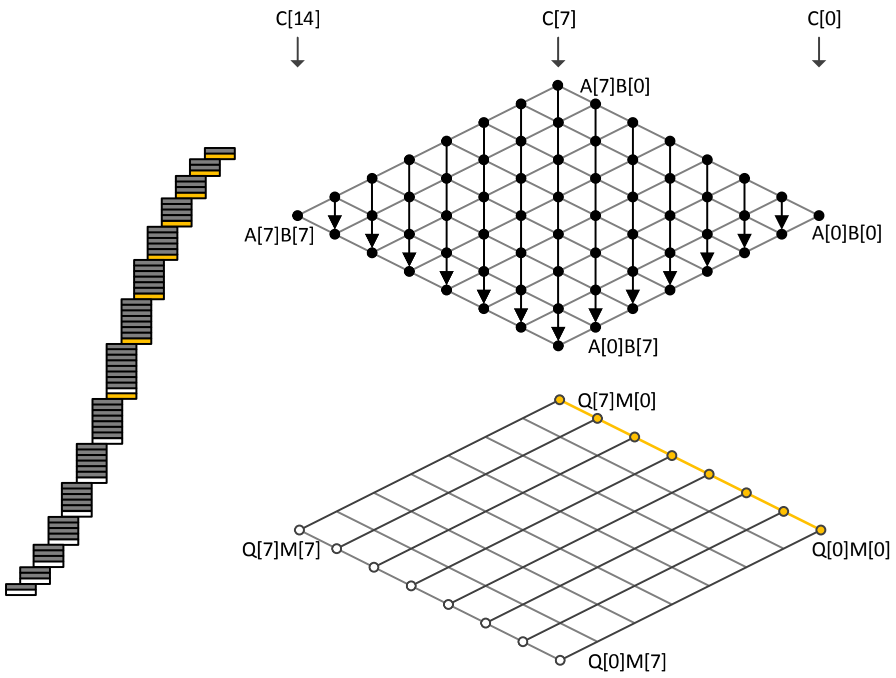

Fig. 2. Finely Integrated Product Scanning Method for Optimal Prime Fields

bus form represents a multiplication $A[i] \cdot B[j]$. The rightmost corner of the rhombus represents the lowest indices $(i,\ j = 0)$, whereas the leftmost represents corner the highest indices $(i,\ j = n-1)$. The lowermost side represents result indices $C[k]$, which ranges from the rightmost corner $(k = 0)$ to the leftmost corner $(k = 2n-1)$. To describe Montgomery multiplication, we introduce double rhombus forms. Upper rhombus represents multi-precision multiplication and under rhombus represents Montgomery reduction. The under rhombus form has two operands Q and M. Each operand follows same representation we used for multi-precision multiplication. For OPF-Montgomery reduction, we use two colored dots. The yellow dots describe the addition of Q to intermediate results because parameter M has one in the least significant bit which is computable with simple addition operation instead of partial products. In case of white dots, 16-bit partial products on $Q \cdot M$ are updated to intermediate results.

In general, the Montgomery multiplication can be implemented in a separated or integrated fashion according to the computation order. The separated mode performs the reduction process after the entire multiplication as shown in Figure 1. For example, Separated Product Scanning (SPS) [9, 11] firstly conducts multiplication in product-scanning, and then performs the Montgomery reduction. The distinctive strength of SPS is that it requires less registers, since three intermediate registers are sufficient. For this reason, this method is considered to be a good choice when it comes to resource constrained devices where the platform has limited number of registers. The alternative method, Separated Operand Scanning (SOS), calculates the products in an operand scanning way and then reduces the results separately [8]. However, this is not recommended on embedded processors since the OS method needs more memory-access instruction in order to get the intermediate results.

In case of integrated mode, the multiplication and reduction are performed in an interleaved way. This can avoid a number of memory accesses for intermediate results, but it requires many registers to retain a number of parameters including operands, modulus and intermediate results. The Coarsely Integrated Operand Scanning (CIOS) method improves previous SOS method by integrating the multiplication and reduction steps. Instead of computing the multiplication processes separately, multiplication and reduction steps are alternated in every loop. With this technique, we can update intermediate results more efficiently. In CIOS, two inner loops are computed separately and this causes inefficient computation processes. The alternative method, Finely Integrated Operand Scanning (FIOS) integrates the two inner loops of multiplication and reduction and compute the one inner loop. However, operand scanning method is not good choice due to high requirements of registers to retain operands. In order to further improve performance, many works have studied Finely Integrated Product Scanning (FIPS) described in Figure 2 [4, 19, 1, 13]. The method conducts product scanning multiplication and reduction in integrated form. This method pursues two main benefits. On one hand, the number of required registers is relatively lower than OS because PS does not need many registers for intermediate results. On the other hand, the method does not re-load/store intermediate results so the number of memory access is significantly reduced. However, PS method is not the fastest multiplication method so far. In CHES'11 and WISA'12, OC and COC multiplication methods are released. It shows that there is some space to improve Montgomery multiplication by adopting the OC and COC multiplication methods. In this paper, we challenge to this point and present novel OPF-Montgomery multiplication with OC and COC methods.

4 Optimal Prime Field-Coarsely Integrated Operand Caching (OPF-CIOC)

In this section, we present novel Coarsely Integrated Operand Caching for OPFs (OPF-CIOC). We selected the fastest multiplication methods including operand caching and consecutive operand caching methods for multiplication part. In order to further reduce the number of intermediate results `load` and `store` instructions, we chose the integrated mode. The OC and COC multiplication methods show high performance in ordinary multi-precision multiplication but they consume a number of registers to retain operands. To combine the multiplications on resource constrained devices, we divided multiplication into two parts. First part is only computing multi-precision multiplication by size of $n - m$ where n and m represent size of operand and inverse of modulus (M'). The first

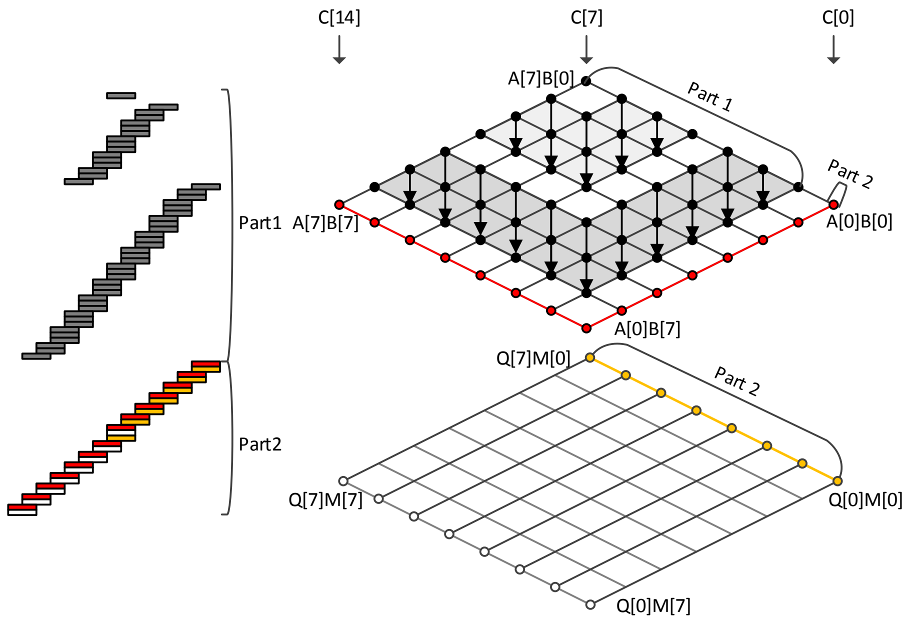

Fig. 3. Coarsely Integrated Operand Caching Method for Optimal Prime Fields

part is computed with multiplication methods including OC and COC. The both methods have very similar performance, so we should carefully select the proper method depending on operand parameters. The detailed costs are drawn in Table 2. The result shows that COC method is only slightly faster than OC in case of OPF-256-bit so for the others OC methods are better choice than COC. Secondly, remaining multiplication part is integrated with reduction computations. The size of remaining part is size of inverse of modulus (M'). In the paper, we pick OPF-CIOC in Figure 3 to describe our method, but this is simply applied to OPF-CI(C)OC as described in Figure 4.

The proposed OPF-CIOC method combines the OC multiplication together with OPF reduction process for Montgomery multiplication as described in Figure 3. The multiplication is divided into two parts. Part 1 computes ordinary multiplications on $A[i] \cdot B[j]$ where $1 \leq i \leq 7$ and $0 \leq j \leq 6$. The number of row is computed by following the equation $\lceil (n-m)/c \rceil$ where n, m and c are size of operand, inverse of modulus and caching registers. Part 2 is integrating remaining partial products and Montgomery reduction. For the first step, partial products on $A[i] \cdot B[j]$ where $0 \leq i, j \leq 7$ are computed and then operand $Q[k]$ where $0 \leq k \leq 7$ is generated. After then Montgomery reduction described in yellow and white dots are computed with multiplying $M[i]$ by $Q[j]$ where $0 \leq i, j \leq 7$. In particular, the yellow dots are simple addition operations and white dots are 16-bit multiplication operations so it has lower overheads than

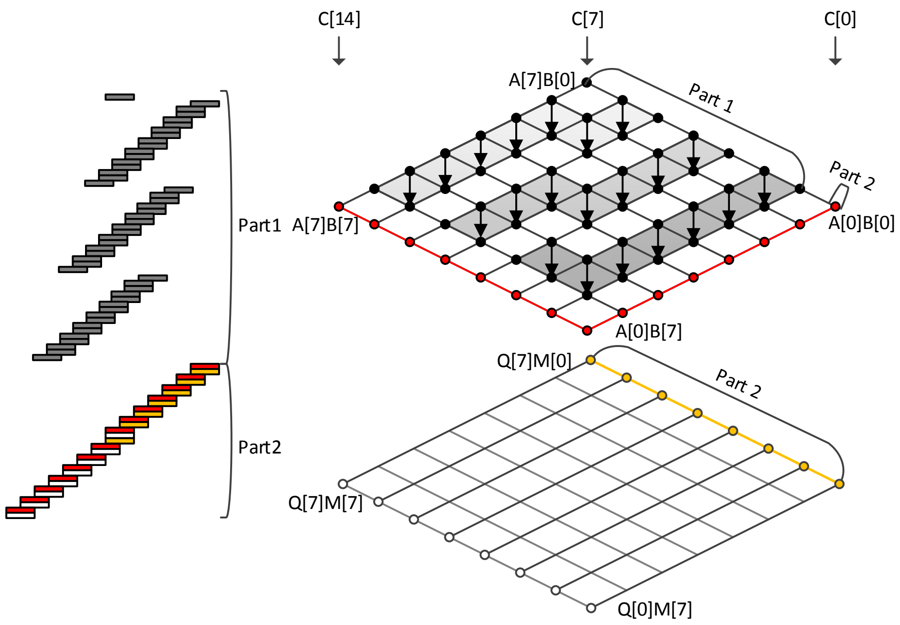

Fig. 4. Coarsely Integrated Consecutive Operand Caching Method for Optimal Prime Fields

ordinary Montgomery algorithm. Finally the results are updated to intermediate result and this process is iterated till the end of operands.

In Figure 5, we give a detailed example of OPF-CIOC in 160-bit operand and 8-bit word sizes but this is readily extended to COC multiplication and other operand and word sizes. The main body is divided into Part 1 and 2. Part 1 conducts multiplication and Part 2 computes remaining multiplications together with reduction process.

Part 1 The size of operand and inverse of modulus is 160-bit ($n = 20$) and 32-bit ($m = 4$). The length of Part 1 is 128-bit ($160 - 32$, $n - m$) and the number of row is 2 ($\lceil 16/10 \rceil$, $\lceil (n - m)/c \rceil$) where caching registers are 10. These two rows compute partial products on $A[i] \cdot B[j]$ where $4 \leq i \leq 19$ and $0 \leq j \leq 16$ by following operand caching method.

Part 2 The reduction and remaining partial products are integrated in Part 2. The partial products ($A[0-3] \cdot B[0-3]$ named block 1 is computed and then parameter $Q[0-3]$ is obtained by multiplying intermediate results ($C[0-3]$) by inverse of modulus (M'). In block 2, the partial products ($Q[0-3] \cdot M[0]$) are added to intermediate results ($C[0-3]$). Following this order, remaining blocks from 3 to 9 are computed. In block 10, most significant word is multiplied before generate parameter $Q[18-19]$. After then in block 11, yellow dots including

Table 2. Comparison of operand caching and consecutive operand caching in terms of number of memory access

Algorithms	`load`	`store`	`total`
OC 128-bit for OPF 160-bit	50	44	94
OC 160-bit for OPF 192-bit	70	60	130
OC 192-bit for OPF 224-bit	116	84	200
OC 224-bit for OPF 256-bit	152	108	260
COC 128-bit for OPF 160-bit	50	44	94
COC 160-bit for OPF 192-bit	70	60	130
COC 192-bit for OPF 224-bit	118	90	208
COC 224-bit for OPF 256-bit	146	110	256

$M[0] \cdot Q[16-19]$ are added to intermediate results. From block 12 to 21, the same process of multiplication and reduction are iterated.

Final subtraction without conditional statements in OPF Final subtraction is conducted when final results go beyond size of target prime field. The final subtraction of Montgomery multiplication is computable with conditional branch by checking the carry bit, which is vulnerable to side-channel attacks since the attacker can catch the leakage information based on this conditional statement [18]. Our work follows the idea of constant-time Montgomery multiplication which has been presented in [11, 10].

5 Result

This section discusses the computation complexity of the proposed OPF-Montgomery multiplications in terms of memory access and real implementations on 8-bit AVR processor.

Memory Access The number of memory access should be concerned because the operations are extremely expensive on embedded processors. OPF-CIOC consists of two main bodies. The Part 1 conducts multiplication on operand by $n-m$. We adopted (consecutive) operand caching method where the number of load and store instructions is $2(n-m)^2/c$ and $(n-m)^2/c+(n-m)$, respectively. Part 2 conducts integrated multiplication and reduction which needs to load intermediate results by $2 \cdot (n-m)$ and operands A and B by $2n$ for remaining multiplications. For reduction, we load modulus (M) by 16-bit. In case of store instruction, final results are stored by $2n$ times because after reduction process, length of results are reduced from $4n$ to $2n$. Finally total costs of load and store for OPF-CI(C)OC are $2(n-m)^2/c + 4n - 2m$ and $(n-m)^2/c + 3n - m$, respectively.

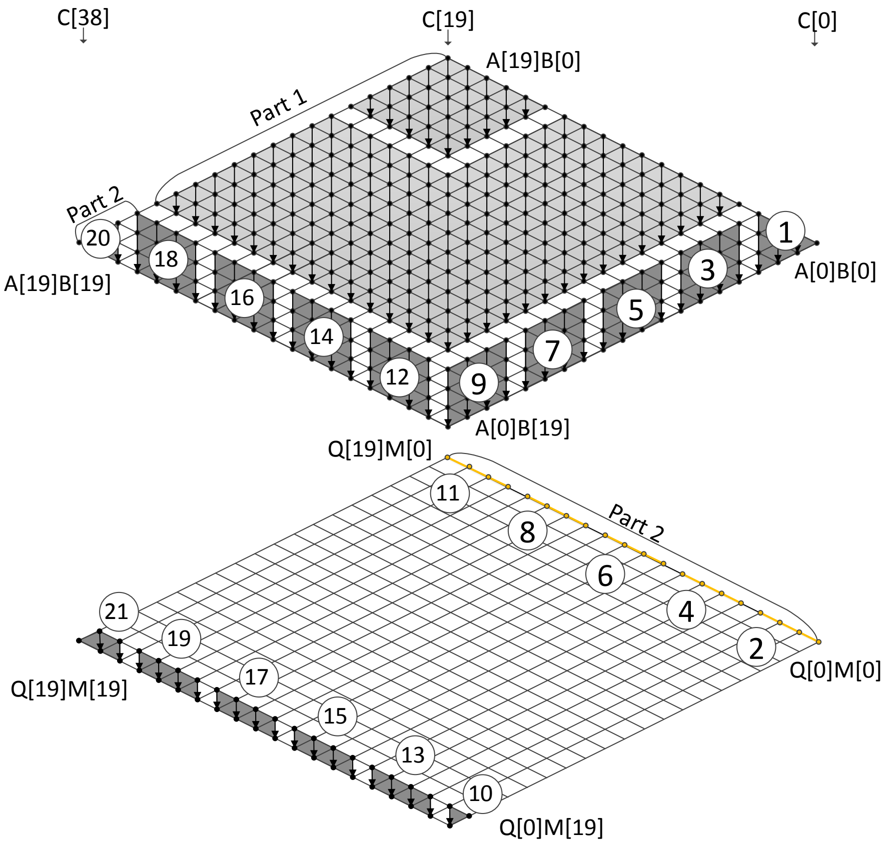

Fig. 5. Coarsely Integrated Operand Caching for Optimal Prime Field in 160-bit

Evaluation on 8-bit Platform ATmega128 On an AVR platform, each `mul`, `load` and `store` instruction consumes 2 clock cycles, while other arithmetic and logical operations need 1 clock cycle. Target board runs at a frequency of 7.3728 MHz and program is evaluated using AVR studio 6.0.

In Table 3, performance evaluations of OPF-Montgomery algorithm are described. In ACNS'14, works by [13] adopted FIPS because this method is readily integrating the reduction and multiplication and requiring small number of registers. They achieved 3237, 4500, 5971 and 7650 clock cycles for 160-, 192-, 224- and 256-bit OPF-Montgomery multiplications. For comparison, the proposed OPF-CIOC method achieves 3116, 4288, 5650 and 7258 clock cycles. The performance gains are from 3.7 $\sim$ 5 %. The reason to draw high performance enhancement is we adopt the most advanced multiplication for OPF-Montgomery multiplication and finely integrated multiplication and reduction processes. To do this, we divided OC and COC methods into two parts and then integrated second part of multiplication with reduction process. This approach challenges

Table 3. Execution time (in clock cycles) of OPF-Montgomery multiplication for different operand lengths on the ATmega128.

Algorithms	160	192	224	256
OPF-FIPS [4]	5239	7070	n/a	n/a
OPF-FIPS [1]	3588	n/a	n/a	n/a
OPF-FIPS [19]	3542	4851	6545	8091
OPF-FIPS [13, 10]	3237	4500	5971	7650
This work (OPF-CIOC)	3116	4288	5650	7258
This work (OPF-CICOC)	3116	4288	5668	7250

to current OPF-FIPS and breaks the speed records. Furthermore, our method is implemented in a constant-time way and therefore resists against simple power analysis attack.

6 Conclusion and Future Work

In this paper, we presented novel Optimal Prime Field Montgomery multiplication over embedded microprocessors. For high performance enhancements, we integrated previous best known multiplications with OPF reduction process. This is first trial to combine OC and COC with OPF Montgomery reduction. The design is highly exploiting limited number of registers together with high performance gains. In order to measure power of proposed method, we implemented the method on representative 8-bit RISC processor. Finally, we achieved remarkable performance enhancements by 5% than previous best known results.

A future work based on this work is to evaluate the proposed technique for the modular squaring over OPFs. Furthermore, it would also be interesting to apply the proposed method to enhance the performance of elliptic curve cryptography over OPFs on resource constrained devices, similar as the work [12], which employed COC methods to push the speed limit of NIST curve on 8-bit AVR processors.

References

1. D. Chu, J. Großschädl, Z. Liu, V. Müller, and Y. Zhang. Twisted edwards-form elliptic curve cryptography for 8-bit avr-based sensor nodes. In *Proceedings of the first ACM workshop on Asia public-key cryptography*, pages 39–44. ACM, 2013.
2. P. G. Comba. Exponentiation cryptosystems on the ibm pc. *IBM systems journal*, 29(4):526–538, 1990.
3. J. Großschädl. Tinysa: A security architecture for wireless sensor networks. In *Proceedings of the 2006 ACM CoNEXT conference*, page 55. ACM, 2006.
4. J. Großschädl, M. Hudler, M. Koschuch, M. Krüger, and A. Szekely. Smart elliptic curve cryptography for smart dust. In *Quality, Reliability, Security and Robustness in Heterogeneous Networks*, pages 623–634. Springer, 2012.

5. N. Gura, A. Patel, A. S. Wander, H. Eberle, and S. Chang Shantz. Comparing elliptic curve cryptography and RSA on 8-bit CPUs. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems — CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 119–132. Springer Verlag, 2004.
6. D. Hankerson, S. Vanstone, and A. J. Menezes. *Guide to elliptic curve cryptography*. Springer, 2004.
7. M. Hutter and E. Wenger. Fast multi-precision multiplication for public-key cryptography on embedded microprocessors. In *Cryptographic Hardware and Embedded Systems–CHES 2011*, pages 459–474. Springer, 2011.
8. Ç. K. Koç, T. Acar, and B. S. Kaliski Jr. Analyzing and comparing montgomery multiplication algorithms. *Micro, IEEE*, 16(3):26–33, 1996.
9. Z. Liu, J. Großschädl, and I. Kizhvatov. Efficient and side-channel resistant RSA implementation for 8-bit AVR microcontrollers. In *Proceedings of the 1st International Workshop on the Security of the Internet of Things (SECIOT 2010)*, 2010.
10. Z. Liu, J. Großschädl, and D. S.Wong. Low-weight primes for lightweight elliptic curve cryptography on 8-bit avr processors. In D. Lin, S. Xu, and M. Yung, editors, *The 9th China International Conference on Information Security and Cryptology—INSCRYPT 2013*, Lecture Notes in Computer Science. Springer Verlag, 2013.
11. Z. Liu and J. Großschäl. New speed records for montgomery modular multiplication on 8-bit AVR microcontrollers. In D. Pointcheval and D. Vergnaud, editors, *Progress in Cryptology - 7th International Conference on Cryptology in Africa—AFRICACRYPT 2014*, volume 8469 of *Lecture Notes of the Institute for Computer Science*, pages 215–234. Springer Verlag, 2014.
12. Z. Liu, H. Seo, J. Großschädl, and H. Kim. Efficient implementation of NIST-compliant elliptic curve cryptography for sensor nodes. In S. Qing, J. Zhou, and D. Liu, editors, *The 15th International Conference on Information & Communications Security — ICICS 2013*, volume 8233 of *Lecture Notes in Computer Science*, pages 302–317. Springer Verlag, 2013.
13. Z. Liu, E. Wenger, and J. Groszschädl. Mote-ecc: Energy-scalable elliptic curve cryptography for wireless sensor networks. In *Applied Cryptography and Network Security-12th International Conference, ACNS 2014, Lausanne, Switzerland, June 10-13, 2014. Proceedings*. Springer Verlag, 2014.
14. P. L. Montgomery. Modular multiplication without trial division. *Mathematics of computation*, 44(170):519–521, 1985.
15. R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
16. M. Scott. Implementing cryptographic pairings. *Lecture Notes in Computer Science*, 4575:177, 2007.
17. H. Seo and H. Kim. Multi-precision multiplication for public-key cryptography on embedded microprocessors. In *Information Security Applications*, pages 55–67. Springer, 2012.
18. C. D. Walter and S. Thompson. Distinguishing exponent digits by observing modular subtractions. In *Topics in Cryptology CT RSA 2001*, pages 192–207. Springer, 2001.
19. Y. Zhang and J. Grossschadl. Efficient prime-field arithmetic for elliptic curve cryptography on wireless sensor nodes. In *Computer Science and Network Technology (ICCSNT), 2011 International Conference on*, volume 1, pages 459–466. IEEE, 2011.