International Journal of Cyber-Security and Digital Forensics (IJCSDF) 1(1):15-19
The Society of Digital Information and Wireless Communications, 2012 (ISSN: 2305-0012)

# *Application of New Classes of Mersenne Primes for Fast Modular Reduction for Large-Integer Multiplication*

Suhas Sreehari, Huapeng Wu, and Majid Ahmadi
Department of Electrical and Computer Engineering
University of Windsor
Windsor, Canada
*{sreehar, hwu, ahmadi}*@uwindsor.ca

***Abstract*— This paper attempts to speed-up the modular reduction as an independent step of modular multiplication, which is the central operation in public-key cryptosystems. Based on the properties of Mersenne and Quasi-Mersenne primes, we have described four distinct sets of moduli which are responsible for converting the single-precision multiplication prevalent in many of today's techniques into an addition operation and a few simple shift operations. We propose a novel revision to the Modified Barrett algorithm presented in [3]. With the backing of the special moduli sets, the proposed algorithm is shown to outperform (speed-wise) the Modified Barrett algorithm by 80% for operands of length 700 bits, the least speed-up being around 70% for smaller operands, in the range of around 100 bits.**

***Keywords*–Large integer modular reduction; Mersenne primes; Quasi-Mersenne primes; Barrett-based reduction.**

## I. Introduction

Modular multiplication forms the core of modular exponentiation, which lies at the heart of cryptographical operations. Therefore, speeding up modular multiplication (especially of large integer operands) has been a much sought-after outcome for researchers in the area of information security. The given problem can be broken into two major problems in themselves: (a) large integer multiplication, and (b) modular reduction of large integers. Research has gone into tackling both these problems individually and jointly (in an interleaved fashion). The former approach is usually word-serial or parallel, and rarely bit-serial (since trading off area for time is not unusual in this field). However, the latter approach tends to be word-serial.

The focus of this paper is speeding up the reduction part of the problem, while the multiplication part is assumed to be completed in the fastest compatible way possible. The justification for separating modular reduction from multiplication comes from [4], and is revisited in the next section. The most important starting point for reduction-oriented research can be traced back to the Montgomery algorithm [1], in which trial division was circumvented for the first time. An alternative algorithm based on quotient estimation soon followed [2]. While the Montgomery algorithm has been hugely successful and popular, it has been plagued by residue-computation overhead, cost of the Extended Euclidean algorithm, and multiplications of the order of the modulus itself. The Barrett algorithm described in [2] has faced issues with two multiplications of the order of the modulus itself. Even though the *folding* proposed in [3] cuts down the operational cost to five multiplications of half the order of the modulus, there is still room for further reduction in complexity, and can be readily achieved through proper selection of the modulus. In this paper, we have defined four sets of moduli for which the folding developed in [3] can be more speedily realized. Apart from recommending the moduli sets, we make adaptations to the folding so as to reap the benefits of the special moduli. In our proposal as well as in [3], the first stage is a partial reduction, from which point onwards the burden of full reduction still lies on classic Barrett algorithm. Therefore, the level of reduction in the first stage of our algorithms is a direct indicator of the amount of time required by the Barrett algorithm to carry on the remaining reduction.

## II. Proposed Methodology

A broad-based approach to solving the modular multiplication problem defined in the previous section is to analyze all components of the problem separately, and then check if there is a fast algorithm to speed up each component. Achieving a fast modular multiplier may not be as simple as simply bringing together the sped-up components, which gives rise to concerns over algorithmic compatibility. Therefore, it is prudent to decide whether to adopt an integrated approach (that makes use of interleaved partial multiplication and partial reduction) or a serial approach (multiplication followed by reduction), before going into the specifics of the algorithms and the scope for improvements thereof.

A major reason to favor the serial approach is the freedom in choosing algorithms for each part, independent of the other. As long as it is made sure that the output of the multiplier section forms a compatible input to the reduction section, this independence is certainly a sought-after feature. However, another factor to be considered is the impact of this choice (of choosing the serial approach over the integrated) bears on the performance of both sections together. Cetin Kaya Koc, Tolga Acar, and Burton Kaliski Jr. have shown in [4] that the serial approach (which they refer to as the "Separated Operand Scanning" or simply SOS) is nearly as fast as the interleaved approach (which they refer to as the "Coarsely Integrated Operand Scanning" or just CIOS). This makes it rational to go with the serial/separated approach. In this paper, we assume that the multiplication has been performed in the fastest way possible, and we have the product ready to be reduced. This assumption makes the reduction completely parallel, rather than bit-serial or word-serial.

### A. *The Moduli Sets*

It is to be noted that the modular operations being performed are in prime Galois fields, GF(p). The modulus (represented by M henceforth) in consideration would then be a prime number. The problem statement puts an additional constraint on the modulus in terms of the maximum length, restricting the prime modulus to $n$ bits, i.e., $0 < M < 2^n$, such that $\prod_i d_i(M) = M$, where $d_i(X)$ represents the $i^{th}$ positive integral divisor of the argument $X$.

The above representation is the most basic, broadest set of possible moduli, which is essential to the problem statement. However, going beyond the essentiality of this definition, researchers have continually opened up paths to speed up the reduction in GF(p). The most recent and significant example of constricting the permitted moduli sets can be seen in [4], wherein four sets of moduli are defined on the basis of speeding up stemming out of Mersenne properties – two each for Barrett-based reduction and Montgomery-based reduction. We present in this paper four moduli sets, different – both in representation and rationale – from those presented in [4]. Our proposed moduli sets and algorithm are based on strict and looser views of Mersenne numbers.

Let us start with the strict view of Mersenne numbers, which we shall refer to plainly as Mersenne. Consider a Mersenne number, $M_r$. By definition, $M_r = 2^r - 1$, where $r \in \mathbb{N}$. A widely known property of Mersenne numbers is that $M_r$ is prime for a prime value of r. For the sake of convenience, let us assign r an open value of $p$, which just indicates that a prime value is assigned to r. The notation we shall adopt in the rest of the paper is $M_r$ for a general Mersenne number (which may or may not be prime), and $M_p$ for a Mersenne prime.

At this stage, we will introduce the four sets of moduli we recommend for speeding up Barrett-based reduction, and follow up with a short discussion of the density function of each of these sets, in lieu of a lengthier treatment and illustration of the mathematics behind the choice of these sets, in view of brevity due to the space restrictions.

$$\text{Set 1: } M = 2^p - 1 = M_p^s \tag{1}$$

$$\text{Set 2: } M = d(M_c) = \mathrm{M_p^l} \tag{2}$$

$$\text{Set 3: } M = 2^k - 2^l \pm 1 = Q_p^s \tag{3}$$

$$\text{Set 4: } M = d(Q_c) = Q_p^l \tag{4}$$

where,
$M_p^s$ is a strict Mersenne prime.
$M_c$ is a Mersenne composite, i.e., $M_c \in \{M_r\} - \{M_p^s\}$.
$M_p^l$ is a loose Mersenne prime, which is any integral prime divisor of a Mersenne composite.
$Q_p^s$ is a strict Quasi-Mersenne prime.
$Q_c$ is a Quasi-Mersenne composite, i.e., $Q_c \in \{Q_r\} - \{Q_p^s\}$.
$Q_p^l$ is a loose Quasi-Mersenne prime, which is any integral prime divisor of a Quasi-Mersenne composite.

These four sets together constitute a fairly large number of prime numbers, thus ensuring there is sufficient choice in the design of the system. The following note on the density functions of each of the sets will serve as an intuitive aid in understanding the coverage of primes by these sets.

Let us start with a general assumption that the highest allowed value that a modulus can take is $V = 2^n - 1$, making V the highest possible Mersenne number attainable by the modulus. The number of Mersenne numbers up to V (including V) is $n$. Out of these $n$ Mersenne numbers, only $\pi(n)$ are prime [5]. Therefore we can report the cardinality of the strict Mersenne prime set as,

$$\#(M_p^s) = \pi(n) \tag{5}$$

Note that:
$$\pi(n) \sim Li(n) = \int_2^n \frac{dx}{\ln(x)} \tag{6}$$

Though (6) represents the generally adopted form of the logarithmic integral in prime counting, a faster convergence may be achieved [6] through:

$$Li(n) = \gamma + \ln\ln(n) + \sqrt{n}\sum_{i=0}^{\infty}\frac{(-1)^{i-1}(\ln n)^i}{i!\,2^{i-1}}\sum_{k=0}^{\lfloor (i-1)/2\rfloor}\frac{1}{2k+1} \tag{7}$$

where, $\gamma$ is the Euler-Mascheroni constant.

$$\gamma = \lim_{i\to\infty}\sum_{k=1}^{i}\left(\frac{1}{k} - \ln i\right) = 0.57721566\ldots \tag{8}$$

Next, let us estimate the number of loose Mersenne primes less than V+1. First, let us note the number of Mersenne composites less than V+1,

$$\#(M_c) = n - \pi(n) \quad (9)$$

In order to estimate the distinct, non-repeated prime divisors of the Mersenne composites, it would be useful to reduce the set of Mersenne composites to a set H of $h$ ($\leq \#(M_c)$) co-prime numbers by iterative application of the parallelized I-G Binary GCD algorithm, which is up to eight times faster than the traditional Euclidean approach [7].

$$\#(M_p^l) = \sum_{i=0}^{h-1} \omega(H_i) \mid \text{such that } d_j(H_i) \notin \{M_p^s\} \mid \prod_j d_j(H_i) = H_i, \forall\, i, j \quad (10)$$

where, $\omega(X)$ gives the number of distinct prime divisors of the argument $X$.

We now need to estimate the number of strict Quasi-Mersenne primes. This is trickier than the previous cases, mainly due to the unmanageable number of combinations and prime tests thereof. However, we can see that Proth primes (i.e., primes of the form: $k_1.2^{k_2} + 1$, where $k_1 < 2^{k_2}$) make up a considerable chunk of the $Q_p^s$ set. This can be easily visualized by plugging a difference of two powers of 2 into $k_1$. Fortunately, a Proth number can be rather easily checked if it is indeed a prime number, as shown in [8]. Then, we have the Solinas primes [9], of the form $2^a \pm 2^b \pm 1$. If $\#(P_p)$ indicates the number of Proth primes below V+1, and if $\#(S_p)$ indicates the number of Solinas primes below V+1, we have the lower and upper bounds on the number of strict Quasi-Mersenne primes below V+1.

$$\#(P_p) \leq \#(Q_p^s) \leq \#(S_p) \quad (11)$$

[10] contains a section on counting the Solinas primes.
On similar lines of quantifying the number of loose Mersenne primes, we can reduce the Quasi-Mersenne composites from $\#(Q_c) = n^2 - n$, to a set G of $g$ ($\leq \#(Q_c)$) co-prime numbers (again via iterative use of the parallelized I-G Binary GCD algorithm). Then, we simply pick the distinct prime factors of each element of G.

$$\#(Q_p^l) = \sum_{i=0}^{g-1} \omega(G_i) \mid d_j(G_i) \notin \{Q_p^s\};\ d_j(G_i) \notin \{M_p^l\};\ d_j(G_i) \notin \{M_p^s\} \mid \prod_j d_j(G_i) = G_i, \forall\, i, j \quad (12)$$

Let $\#(P_V)$ be the number of primes lesser than V+1, such that they belong to one of the four sets outlined in (1), (2), (3), and (4). Then, we have,

$$\#(P_V) = \#(M_p^s) + \#(M_p^l) + \#(Q_p^s) + \#(Q_p^l) \quad (13)$$

The physical interpretation of (13) is simply that we have $\#(P_V)$ number of choices to pick a prime number from as the modulus for the multiplication and the subsequent reduction. The forthcoming algorithms are framed with these four sets of moduli in mind, and are shown in the last section to be faster than the state-of-the-art algorithms.

## B. *The Algorithm*

```
Input: $A, B, M$
Output: $\underset{\cdot}{R}$
Pre-computations:
    1. $N = A.B$
    2. $F = 2^k \mid F \bmod M = 2^x, \text{for the smallest value of } x\ ; k, x \in \mathbb{Z}.$
    3. $M' = F \bmod M = 2^x$

Step 1: $R_1 = N[k-1:0]$
Step 2: $R_2 = N[2n-1:k].M'$
Step 3: $\underset{\cdot}{R} = R_1 + R_2$
Step 4: Return $\underset{\cdot}{R}$.
```

Defining variables used in the algorithm:

$A$: The multiplicand, $2^{n-1} \leq A < 2^n$.
$B$: The multiplier, $2^{n-1} \leq B < 2^n$.
$N$: The product to be reduced, $2^{2n-1} \leq N < 2^{2n}$.
$M$: The modulus, $N < 2^n$.
$\underset{\cdot}{R}$: The partially reduced result.
$k$: The smallest integer value such that $x$ reaches its minimum value.
$\alpha[i:h]$: The portion of any variable $\alpha$, between and including its $i$th and $h$th bits.

Brief analysis of the algorithm:

The multiplication is put under pre-computation since the focus of the algorithm is the modular reduction which follows the multiplication.

Step 1 computes the remainder of the division of the product and the power of 2 (F), which is just a case of masking the higher order bits, while choosing only the lower k bits.

Step 2 could either fold once more, or make the adjustment to reduce the result with the actual modulus. In case of a second fold (which is applicable only to moduli of sets 3 and 4), the unity adjustment occurs later in the step.

Step 3 is basically the summation of the results of the first two steps, which gives the partial reduction. It is this result which is fed into the classical Barrett algorithm.

The cost involved during the run-time of the algorithm is a maximum of three shift operations, bit-masking, and two $s$-bit addition operations (where $s$ is at most 20% longer than the modulus length).

Average complexity of the algorithm can be worked out to be $\Theta(\lceil n/2 \rceil)$.

## III. Results and Conclusion

Before presenting the results, let us qualitatively see why the algorithm has the potential to produce faster modular reduction. It is clear that reducing any large number with a power of 2 as the intermediate modulus is a simple case of bit masking, and is a negligible hardware effort. Reducing the partially reduced result further with the actual modulus costs a single-precision multiplication [3]. However, if the actual modulus is smaller than the intermediate modulus by unity (as defined by moduli set 1), the need for the multiplication vanishes, and is replaced by shift and addition/subtraction operations. It should be noted that the actual modulus need not just be unity short of the intermediate modulus; it may alternatively be an integral prime divisor of unity less than the intermediate modulus, thus leading to the moduli set 2. The authors of [3] also propose one more level of partial reduction, called a double-fold, which may bring further time reduction. If the double-fold were to be applied to the algorithm described in this paper, it would naturally necessitate the usage of moduli sets 3 and 4.

The algorithm has been extensively tested for products that range from 20 bits to 100 bits. The general rule followed in testing the algorithm is that the length of the modulus is less than or equal to half the product length in bits. The testing has been carried out on Altera Quartus II, the device family being Stratix III. 700 operand/moduli-pairs have been tested upon – 100 operands of width 100 bits, 100 of width 300 bits, and so on till 700-bit operands. The mix of moduli has been made as heterogeneous as possible, with roughly equal representation from all four sets. For the purpose of comparison, we have chosen the Modified Barrett algorithm (which appears in the Fig. 1 as "Hasenplaugh") of [3], since it is faster than Montgomery and classical Barrett algorithms, and the algorithm of [4] (which appears in the Fig. 1 as "Knezevic (Belgium)").

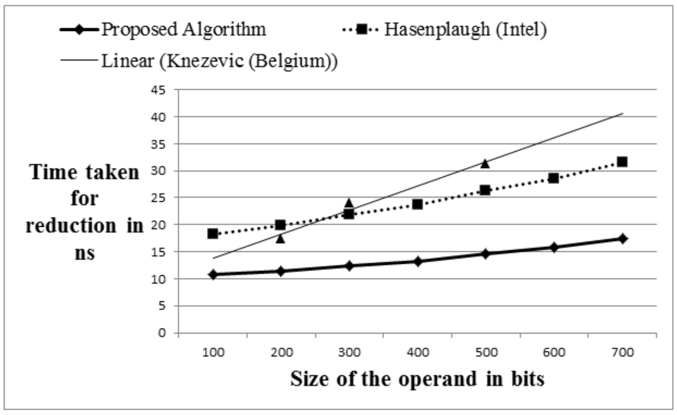

Figure 1. Time delay comparison of the proposed algorithm against Hasenplaugh et al.'s Modified Barrett algorithm, and Knezevic et al.'s algorithm.

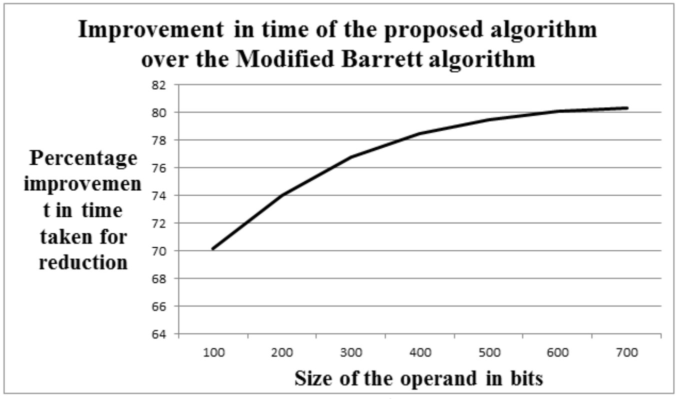

Figure 2. Percentage improvement in time of the proposed algorithm over the Modified Barrett algorithm.

These results presented in figures 1 and 2 are over 45% better (faster) than the original implementation presented in [11] – mainly due to speed-optimized FPGA implementation and a much larger test scheme, thereby representing real-life results.

TABLE 1: Time delay comparison (measured in ns)

| Bits | Modified Barrett [3] | Knezevic [4] | Proposed |
|---|---|---|---|
| **100** | 18.33 | - | 10.77 |
| **200** | 19.82 | 17.54 (128 bits) | 11.39 |
| **300** | 21.80 | 24.05 (256 bits) | 12.33 |
| **400** | 23.60 | - | 13.22 |
| **500** | 26.28 | 31.3 (512 bits) | 14.64 |
| **600** | 28.58 | - | 15.87 |
| **700** | 31.62 | - | 17.54 |

Knezevic algorithm has be tested for operands of power-of-2 bit lengths, so that the words can be of power-of-2 bit lengths.

It has been made evident in this paper that by defining moduli sets based on Mersenne, Quasi-Mersenne, and divisor primes thereof, and by updating and tuning the Modified-Barrett algorithm presented in [3], we achieve better speed (as seen by an average of more than 80% decrease in time requirements) for operands 700 bits long.

## References

[1] P. Montgomery, "Modular Multiplication Without Trial Division," *Math. Computation, Vol. 44, pp. 519–521*, 1985.

[2] P. Barrett, "Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor," *Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 311–323. Springer, Heidelberg*, 1987.

[3] W. Hasenplaugh, G. Gaubatz, and V. Gopal, "Fast Integer Reduction," *18th IEEE Symposium on Computer Arithmetic (ARITH '07), pp. 225–229*, 2007.

[4] M. Knezevic, F. Vercauteren, and I. Verbauwhede, "Faster Interleaved Modular Multiplication Based on Barrett and Montgomery Reduction Methods," *IEEE Transactions on Computers, Vol. 59, Issue 12, pp. 1715–1721*, December 2010.

[5] D. Shanks, "Solved and Unsolved Problems in Number Theory, 4th ed.," *New York: Chelsea, p. 15*, 1993.

[6] B. C. Berndt, "*Ramanujan's Notebooks, Part IV.*," New York: Springer-Verlag, pp. 126-131, 1994.

[7] T. Jebelean, "Comparing Several GCD Algorithms," Proceedings of the 11th Symposium on Computer Arithmetic, pp. 18 –185, 1993.

[8] E. W. Weisstein, "Proth's Theorem." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/ProthsTheorem.html

[9] J. A. Solinas, "Generalized Mersenne Numbers," *Technical Report CORR 99-39, Centre for Applied Cryptographic Research, University of Waterloo,* 1999. http://cacr.uwaterloo.ca/techreports/1999/corr99-39.ps

[10] J. J. Angel and G. M- Luna, "Solinas primes of small weight for fixed sizes," *Cryptology ePrint archive,* February 2010.

[11] S. Sreehari, H. Wu, and M. Ahmadi, "Fast modular reduction for large-integer multiplication for cryptosystem application," *2nd International Conference on Digital Information and Communication Technologies and its Applications,* May 2012.