Unified framework of low complexity algorithms for Fast Modular Reduction

Dhananjay S. Phatak and Tom Goff
Computer Science and Electrical Engineering Department
University of Maryland, Baltimore County, Baltimore, MD 21250
phatak@umbc.edu
UMBC CSEE Technical Report No. TR-CS-05-08

EXTENDED ABSTRACT

For modular reduction w.r.t. a fixed modulus P we, show the following results :

(1) We show a straight remaindering method that yields the true remainder in $\Theta(2.5N\lg N)$ effort, which is equivalent to 2.5 wrap-around (cyclic/negacyclic) convolutions-with-one-operand-fixed. The main features of this method are
(a) it produces the true remainder ($X \mod P$) in $\Theta(2.5N\lg N)$ effort (in contrast, Montgomery's reduction step produces $X \cdot R^{-1} \mod P$).
(b) most importat : it works for any modulus P, irrespective of whether relatively co-prime with certain radicii.

This method covers all cases where Montgomery's algorithm is not applicable (either because
(i) the modulus P is not relatively co-prime w.r.t the radicii of interest, or
(ii) because the number of repeated-related modular reductions required is small: for example performing a single modular multiplication $A \times B \mod P$. Here, the forward-and reverse transformations required by Montgomery's method make it too slow)
(iii) By casting straight remaindering in a form similar to Montgomery's method we have effectively extended the framework in [**?**] to straight remaindering.

(2) For Montgomery-reduction: we show a new variation wherein, the order of cyclic and negacyclic convolutions can be changed (so that the total effort remains $\Theta(2N\lg N)$). This seemingly trivial change of order has fundamental implications because it allows the choice of $R = R_+ = 2^n + 1$ thereby substantially increasing the number of cases where Montgomery's method is applicable because of 3 reasons:
(i) $2^n + 1$ has lot fewer facctors than $2^n - 1$ so that it is more likely to be coprime w.r.t. modulii P.
(ii) $R_+ = 2^n + 1$ and $R_- = 2^n - 1$ are always relatively coprime which allows for the dynamic selection of one of the two as the R.
(iii) For most efficient FFT computations the word-length n is a power of 2. So $R_+ = 2^{2^n} + 1$ which are the well known Fermat numbers. This has its own advantages further explained in the manuscript.

(3) We extend Montgomery's method to include modulli P that are not relatively co-prime w.r.t. R but are relatively co-prime w.r.t. R_{gi} (which is the generalized R^{-1} The modular inverse of R does not exist). This further enhances the applicability of the proposed implementations of Montgomery's methods.

(4) The focus is on large wordlengths where multiplication is implemented via complex floating point FFTs However, the algorithms are cast in terms of wrap-around (cyclic/negacyclic) convolutions. Hence, the methods will yield a speedup even for small word-lengts (beyond a few machine words, certainly 512 bits and above). Fundamentally, as long as the effort required to perform a wrap-around convolution is different from the effort required to perform a full-linear convolution, the proposed methods will reflect the gain.

(The algorithms and the analytical results in the paper, have been extensively simulated in Maple and Verified). fast modular reduction, large wordlength, elliptic-curve, cryptography, FFT multiply, number theoretic transforms, linear convolution, cyclic convolution, principle of separation

1 Introduction and Background

1.0.1 Notation

Word length of operands in bits = n.
$R_n = 2^n$, $R_+ = 2^n + 1$ and $R_- = 2^n - 1$
n-bit numbers are represented as N digit numbers where each digit is a radix-β digit, so that length of the transforms is $N = 2^n / \lg\beta$. For transform length N to be a power of 2, β is typically selected from the set $\{2^4, 2^8, 2^{16}\}$. 64-bit architectures imply that for efficient hardware support, β is limited to 2^{16} and transform length to N to about 2^{17}
Cyclic convolution of A and B is denoted as $\langle A \otimes_C B \rangle$
Negacyclic convolution is denoted as $\langle A \otimes_N B \rangle$
A wrap-around convolution (which can be cyclic or negacyclic) is denoted as $\langle A \otimes_W B \rangle$.
Linear convolution is the full product $A \times B$ and needs no other notation.

1.1 Problem Definition

1.1.1 Modular Reduction

The modular-reduction problem deals with generating the remainder R when a given dividend X is divided by a modulus P:

$$\begin{aligned} X &= Q \cdot P + S \quad \text{where} \quad 0 < X < P^2,\ 0 \leq R < P \\ S &= X\%P \quad \text{following C syntax} \end{aligned} \tag{1}$$

Modulus P is assumed to be n bits long:

$$2^{n-1} < P < 2^n \tag{2}$$

i.e., in the unsigned (magnitude only) form, the nth bit of P (which has a weight of 2^{n-1}) is 1 and at least one more of the remaining bits of P is 1.

It is convenient to split X into an upper half $\mathcal{U}(X)$ (i.e., higher order n bits of X) and a lower half $\mathcal{L}(x)$ (i.e., lower order n bits of X):

$$\begin{aligned} X &= \mathcal{U}(X) \cdot 2^n + \mathcal{L}(X) = X_u \cdot 2^n + X_l \quad \text{where} \\ \mathcal{U}(Z) &= \lfloor \frac{Z}{2^n} \rfloor \quad \text{and} \\ \mathcal{L}(Z) &= Z\%2^n \end{aligned} \tag{3}$$

1.1.2 Modular Exponentiation

Here the problem is to evaluate $X^Y\%P$. If $X\%P = R_0$, repeated squaring and reducing yields $X^2\%P,\ X^4\%P,\ X^{16}\%P,\ \cdots X^{2^k}\%P$.
The pseudo-code to find the result is:

```
Let Y = (y_{k-1} y_{n-2} ··· y_0) in binary.
initialize:  R_pow = X%P,  R = 1
for (i = 0; i < k; i++){
     if (y_i == 1)  R = (R · R_pow)%P
     if (i < k-1)  R_pow = (R_pow · R_pow)%P
}
```

1.2 Linear versus Wrap-around convolutions

Since numbers are polynomials evaluated at the-radix, results that hold for polynomials hold for integers as well (the converse is not true).

Let $A(x)$ and $B(x)$ be two polynomials of degree $(n-1)$, defined by their corresponding vectors of coefficients $\overline{A}$ and $\overline{B}$ of length n. Then their product $C(x)$ is a polynomial of degree $(2n-2)$. While this needs a vector of length $(2n-1)$, the corresponding integer multiplication requires $2n$ digits. So $C(x)$ is represented by vector $\overline{C}$ of length $2n$ whose upper and lower halves are $\overline{U}$ and $\overline{L}$. Then a cyclic polynomial convolution is defined as the product modulo (x^n-1):

$$\begin{aligned} C(x)\%(x^n-1) &= (U(x)x^n+L(x))\%(x^n-1) \\ &= [(U(x)\%(x^n-1)\cdot(x^n\%(x^n-1)) \\ &\quad +L(x)\%(x^n-1)]\%(x^n-1) \\ &= U(x)+L(x) \end{aligned} \tag{4}$$

which is the sum of upper halves $U(x)$ and $L(x)$ of full (linear) convolution $C(x)$.

Likewise, a negacyclic convolution is defined as a product modulo (x^n+1).

$$\begin{aligned} C(x)\%(x^n+1) &= (U(x)x^n+L(x))\%(x^n+1) \\ &= L(x)-U(x) \end{aligned} \tag{5}$$

1.2.1 Wrap-around convolutions take less effort

[1] For transform based methods, cyclic/negacyclic convolutions require half the work of a linear convolution. Assume the bit-length n is a power of 2 (otherwise the transform based methods are a little less efficient). Transform based methods select a radix $\beta = 2^k$ so that an n bit number is represented as an N digit number where $N = \frac{n}{k}$.
(a) To perform a linear convolution of two numberes A and B of length N digits (represented by vectors $\overline{A}, \overline{B}$) the following steps are needed.
(i) Zero-pad $\overline{A}$ and $\overline{B}$ with N zeroes in higher significant positions
(ii) evaluate the FFT of the zero-paded vectors (each FFT takes $\Theta(N \lg N)$ work).
(iii) point-wise multiply the two FFTs (this takes only $\Theta(N)$ work)
(iv) Inverse transform the result (this requires another $N \lg N$ work).
(v) Carry-release (this is also $\Theta(N)$).
A linear convolution takes $3N \lg N$ work in general. If one of the two operands is known ahead of

time (so that its transform can be assumed to be pre-computed), then only one forward transform is required. In that case a linear convolution requires $2N \lg N$ effort.

(b) A cyclic convolution is obtained by
(i) taking forward transforms of A and B (without the zero padding). Each FFT is now of length N and the data is real. So each FFT requires $\Theta(\frac{N}{2} \lg N)$ work.
(ii) point-wise multiply the two FFTs
(iii) Inverse FFT the result ($\Theta(\frac{N}{2} \lg N)$)
(iv) release the carries.
Cyclic convolution takes half the effort but yields the sum of upper and lower halves ($U + L$). Note that with precomputation, a cyclic convolution requires $\Theta(N \lg N)$ work.

(c) The negacyclic convolution is obtained by
(i) modulate $\overline{A}$ and $\overline{B}$ with $\overline{\Omega}_{2N}$ which is a vector consisting of first N powers of the $2N$th root of unity:

$$\overline{\Omega}_{2N} = [1,\ \omega_{2N},\ \omega_{2N}^2,\ \cdots\ \omega_{2N}^{N-1}] \tag{6}$$

Modulating a vector $\overline{A}$ means multiplying the kth element of $\overline{A}[k] = A_k$ by z^k.
(ii) FFT the two vectors. It can be shown that the particular modulation by $\overline{\Omega}_{2N}$ does not change the effort level, i.e., the FFT of the modulated vectors takes $\Theta(\frac{N}{2} \lg N)$ effort.
(iii) point-wise multiply the FFTs
(iv) Inverse transform ($\Theta(\frac{N}{2} \lg N)$)
(iv) Demodulate
(v) release carries.
Note that with precomputation, a negacyclic convolution also takes $\Theta(N \lg N)$ work, which is the same order of complexity as a cyclic convolution and half the work required by a linear convolution (it yields the difference of the two halves ($L - U$).

[2] Even at word lengths where Karatsubba's method is optimal, cyclic convolutions are substantially cheaper: if $A = (A_H|A_L)$ and $B = (B_H|B_L)$ then full linear convolution needs $\{A_H B_H, A_L B_L$ and $(A_H B_L + A_L B_H)\}$. Karatsubba's method computes the last value as $(A_H + A_L)(B_H + B_L) - A_H B_H - A_L B_L$ replacing 4 products by 3 products.

If only a cyclic convolution is required, then only two values are required $(A_H B_H + A_L B_L)$ and $(A_H B_L + A_L B_H)$. These can be generated with only two products C_1 and C_2 (instead of the 3

required by a linear convolution):

$$\begin{aligned} C1 &= (A_H + A_L)(B_H + B_L) \\ C2 &= (A_H - A_L)(B_H - B_L) \\ (A_H B_H + A_L B_L) &= \frac{C1 + C2}{2} \\ (A_H B_L + A_L B_H) &= \frac{C1 - C2}{2} \end{aligned} \tag{7}$$

A negacyclic convolution would need the same effort as a linear convolution if the operands are split into only 2 halves. (In other words, at wordlengths where the Karatsubba algorithm is the optimal, negacyclic convolution requires the same effort as a full linear convolution).

However, all the new algorithms presented hrerein use both cyclic and negacyclic convolutions and hence will require less effort than if the convolutions were linear ones.

2 Results

2.1 True remaindering with $\Theta(2.5N\lg N)$ effort

Let $X = RX_u + X_l$ where $X < P^2$ and X_u and X_l are the upper and lower halves w.r.t. R.

$$X \mod P = (RX_u \mod P + X_l \mod P) \mod P \tag{8}$$

$$RX_u = \frac{R^2 X_u}{R} \tag{9}$$

Let $R^2 = PP_{\text{inv}} + \delta$ where P_{inv}, δ are precomputed quotient and remainder when R^2 is divided by P

$$RX_u = \frac{(PP_{\text{inv}} + \delta)X_u}{R} = \frac{(P(P_{\text{inv}}X_u) + \delta X_u)}{R}$$

Let

$$P_{\text{inv}}X_u = Q'R + L \quad \text{then}$$

$$RX_u = \frac{P(Q'R + L) + \delta X_u}{R} = PQ' + \frac{PL + \delta X_u}{R} \tag{10}$$

$$RX_u \mod P = \left(\frac{PL + \delta X_u}{R}\right) \mod P \tag{11}$$

where the last modulo w.r.t P is at most one subtraction because

$$LP + \delta X_u < R \cdot P + P \cdot P = RP + P^2 < 2RP \quad \text{so that}$$

$$\frac{LP + \delta X_u}{R} < 2P \quad \text{and}$$

$$\frac{LP + \delta X_u}{R} \quad \text{must be an integer, as seen from (10).} \tag{12}$$

$$\text{Let } t = \frac{LP + \delta X_u}{R} \quad \text{then, desired remainder} \tag{13}$$

$$RX_u \mod P = t \text{ or } t - P \tag{14}$$

Note that selecting $R \in \{R_+, R_-\}$ yields minimal complexity because

Step 1: $L = P_{\text{inv}}X_u \mod R$ becomes a wrap-around convolution requiring $\Theta(N\lg N)$ work.

Lemma 1 : step 2:

$$t = \frac{LP + \delta X_u}{R}$$

can also be accomplished via a sum of wrap-around convolutions:

Proof : (i) When $R = 2^n + 1 = R_+$ then from equation (10) t is an integer implies

$$\begin{aligned} mP + \delta X_u &= 2^n t + t && (15) \\ &\Rightarrow \text{ sum of upper and lower halves} = 2t \\ 2t &= \mathcal{U}(LP + \delta X_u) + \mathcal{L}(LP + \delta X_u) \\ &= [\mathcal{U}(LP) + \mathcal{L}(LP)] + [\mathcal{U}(\delta X_u) + \mathcal{L}(\delta X_u))] \\ &= (LP \mod R_-) + (\delta X_u \mod R_-) && (16) \\ &\quad \text{and a sum of cyclic convolution is sufficient} v \end{aligned}$$

Transforms representing the cyclic convolutions of L, P and δ, X_u can be added befor taking the inverse transform so that this step can be done in $\Theta(1.5N \lg N)$: (forward transforms of X_u and L, and the inverse transform of the sum).

(ii) When $R = 2^n - 1 = R_-$ then t is an integer implies

$$\begin{aligned} LP + \delta X_u &= 2^n t - t && (17) \\ &\Rightarrow \text{ difference of upper and lower halves} = 2t \\ 2t &= \mathcal{U}(LP + \delta X_u) - \mathcal{L}(LP + \delta X_u) \\ &= [\mathcal{U}(LP) - \mathcal{L}(LP)] + [\mathcal{U}(\delta X_u) - \mathcal{L}(\delta X_u)] \\ &= (LP \mod R_+) + (\delta X_u \mod R_+) && (18) \\ &\quad \text{and a sum of negacyclic convolutions is sufficient} \end{aligned}$$

Thus the total effort required is $\Theta(2.5N \lg N)$ □.

This is almost identical to Montgomery's method, but there are some key differences:
(1) There was no use of any coprimeness or modular inverses, in other words the above derivation does not restrict P in any manner. (2) The fact that δ is now multiplying X necessicitates the evaluation of one more forward transform in the reduction phase. This transform is the additional work over what montgomery's method needs to do. However it should be noted that montgomery's method calculates $XR^{-1} \mod P$ instead of the true remainder $X \mod P$. Now we briefly state the algorithm.

Problem definition /* given $X = RX_u + X_l$ where $X < P^2$ find $X \mod P$ */
Without loss of generality, we illustrate the method for
$R = R_+$ (the other case $R = R_-$ is similar).
Pre-computation: Given P, compute quotient and remainder obtained by dividing R_+^2 by P

$$\delta = R_+^2 \mod P \quad \text{and} \quad P_{\text{inv}} = (R_+^2 - \delta)/P \quad \text{and}$$
$$\langle R_+ \otimes_C P \rangle = \text{cyclic convolution of } R_+ \text{and } P$$

Algorithm StraightRemaidering

Step 1 : $L = (X_u P_{\text{inv}}) \mod R_+$
/* **negacyclic convolution**, requires $\Theta(N \lg N)$ work */

Step 2 : Compute

$$t_1 = \langle L \otimes_C P \rangle + \langle \delta \otimes_C X_u \rangle \tag{19}$$
$$t_2 = (t_1 - \langle R_+ \otimes_C P \rangle) \mod R_- \tag{20}$$

/* $\Theta(1.5N \lg N)$ work */

Step 3 : small post-processing: $\Theta(n)$

ExpectedLeastSignificantDigit$(t) = d_0 = (L_0 P_0 + \delta_0 X_{u0}) \mod \beta$
srem = NULL; /* initialize to some invalid remainder value */

foreach t in $\{t_1, t_2\}$ do /* one of t_1, t_2 must yield correct remainder */
  if ($t \mod 2 != 0$) then
    if ($t \geq R_+$) then
      $t = t - R_-$
    else if ($0 \leq t < R_+$) then
      $t = t + R_-$
    end if
  end if
  $t = t/2$
  if ($t \mod \beta = d_0$) then
    srem = t
    break /* if t_1 matches, break out of the for loop */
  end if
end for
srem = (srem + X_l) mod P
return (srem) □ Total work is $\Theta(2.5N \lg N)$

2.2 Explanation of the post processing steps

$(LP + \delta X_u) < RP + P^2 < 2R^2$. If it exceeds R^2 then the IFFT yields incorrect result. FFT based computations are inherently wrap-around-R convolutions where $R \in \{R_+, R_-\}$. So any overflow gets added back to the least-significant digit so that **overflow necessarily changes the least significant digit**. We exploit this fact to check and correct.

Lemma 2 : If there is overflow the correction is simply to subtract the (precomputed) cyclic convolution $\langle R_+ \otimes_C P \rangle$ of R_+ and P.

Proof : The sequence of operations required to find the correct remainder is
(i) Divide $LP + \delta X_u$ by R
(ii) Then take modulo-P, i.e.

$$t = (\frac{LP + \delta X_u}{R}) \mod P \tag{21}$$

$$\text{Since} \quad LP + \delta X_u < RP + P^2, \quad \text{it must be expressable as} \tag{22}$$

$$LP + \delta X_u = RP + \theta \quad \text{where} \tag{23}$$

$$\theta < P^2 \quad \text{and } \theta \text{ is divisible by } R, \text{ so that} \tag{24}$$

$$t = (P + \frac{\theta}{R}) \mod P = \frac{\theta}{R} \quad \Rightarrow \tag{25}$$

$$t = (LP + \delta X_u - RP)/R \quad \text{which in turn implies that} \tag{26}$$

$$2t = \langle m \otimes_C P \rangle + \langle t_i \otimes_C t_i \rangle - \langle R \otimes_C P \rangle \qquad \square \tag{27}$$

So we calculate t_1 assuming no-overflow and t_2 assuming overflow and use the least significant digits to select one of the two.

The cyclic convolutions are values modulo-R_-. So if during the carry release an extra $\pm R_-$ gets left-in that will make the result odd. The modulo-2 test simply checks and corrects for this.

2.3 Substantially Increasing Applicability of Montgomery's Method

Clearly, the results of previous section apply and R can be selected to be R_+or R_-. The selection $R = R_+$ is a new variatiion in the framework proposed in [**?**] as explained below. This seemingly trivial change of order of cyclic and negacyclic convolutions substantially increases the number of modulii for which Montgomery's method is applicable.

2.3.1 Basic Montgomery Reduction

Pre-computation : Compute R^{-1} and $P^{'}$ such that

$$\begin{aligned} RR^{-1} - P^{'}P &= 1 \quad \text{and} \quad 0 < R^{-1} < P \\ &\quad \text{and} \quad 0 < P^{'} < R \end{aligned} \tag{28}$$

This pre-computation is essentially the extended GCD algorithm.

Now given two numbers A and B in the proper residue class satisfying $AB < RP$, Montgomery's method evaluates $ABR^{-1} \mod P$ as follows.

Step 0 : Compute $T = A \times B$ (full linear convolution).

Step 1 :

$$\begin{aligned} m &= ((T \mod R)P^{'}) \mod R \\ \text{so that} \quad & 0 \leq m \leq R \end{aligned} \tag{29}$$

Step 2 :

$$t = (T + mP)/R \tag{30}$$

Here, t is guaranteed to be an integer, so that the above division by R is exact.

Step 3 : if $t \geq P$ return $(t - P)$ else return t

Montgomery's algorithm is the best way of performing modular exponentiation.
Given an X, it calculates $t_0 = (XR) \mod P$ in the beginning (this is like "forward transformation of X into the proper residue class (not to be confused with a forward FFT)). From here on, it performs

the following two computations in a loop (see Section 1.1.2).

$$
\begin{aligned}
&\texttt{Loop :}\\
&\qquad t_{i+1} = T_i R^{-1} \mod P \quad \text{where} \quad T_i = t_i^2\\
&\qquad S_{i+1} = \begin{cases} S_i & \text{if } y_i = 0 \\ S_i \cdot t_{i+1} \mod P & \text{if } y_i = 1 \end{cases}\\
&\texttt{End Loop :}
\end{aligned}
$$

At the end it performs one extra iteration to reverse-transform the result back.

We call the implementation of the first equation within the loop as the **square-and-reduce** operation. Both the computations together are referred to as the **square-reduce-assimilate** operation.

2.3.2 Realizing Montgomery Redudction with one negacyclic and one cyclic convolution

The framework in [?] selects an integer Q satisfying

$$\frac{Q}{\gcd(Q,R)} > \frac{(P-1)^2+(R-1)P}{R} \geq t \tag{31}$$

and performs the two modified steps:

$$m = ABP^{'} \mod R \tag{32}$$

$$t = (\frac{AB+mP}{R}) \mod (\frac{Q}{\gcd(Q,R)}) \tag{33}$$

While this set of equations broadly defines a framework, we believe the most interesting variation has been left out.

Variation 3 : Select $R = R_+, \quad Q^{'} = R_-$ and $Q = 2Q^{'}$.

$$\gcd(R,Q) = \gcd(R,Q^{'}) = 1(\frac{Q}{\gcd(Q,R)}) = 2R_- = 2R_+ - 4(N-1)$$

$$2R_+ - 4 > \frac{(P-1)^2+(R-1)P}{R} \quad \text{since} \tag{34}$$

$$R \geq P-1 \tag{35}$$

Now step 1 performs a negacyclic convolution and step2 performs a of cyclic convolution, i.e., if the square $T = t_i^2$ is fully evaluated, then Montgomery reduction takes $\Theta(2N \lg N)$ work.

2.3.3 Realizing the square-and-reduce iteration with $\Theta(3.5N \lg N)$ work with any order of cyclic/negacyclic convolutions

A straight evaluation of the square followed by the reduction would take $\Theta(4N \lg N)$ work. As in [?], however, the square-and reduce step can be realized with $\Theta(3.5N \lg N)$ work. We briefly state the algorithm for variation 3 proposed above.(the case when $R = R_-$ is variation 2 in [?]).

Given an n bit modulus $P > 2^{(n-1)}$ and n bit scaled remainder t_i
find $(t_i^2 R^{-1})\%P$.

Without loss of generality, let $R = 2^n + 1$. (this is simply for the purpose of illustraiton, the maple code dynamically selects either R_+ or R_-, whichever is relatively coprime with P).

Pre-computation : Several entities are precomputed:
(1) Numbers R^{-1} and $P^{'}$ such that

$$RR^{-1} - P^{'}P = 1 \tag{36}$$

The FFT's of those numbers are also pre-computed

Algorithm FusedSquareAndReduce

Step 1 : $T_l = T \mod R_+ = \langle t_i \otimes_N t_i \rangle$
/* **negacyclic convolution** of t_i with itself requires $\Theta(N \lg N)$ work. */

Step 2 : $m = (T_l P^{'}) \mod R_+$
/* **negacyclic convolution**, requires $\Theta(N \lg N)$ work */

Step 3 : Compute

$$t_1 = \langle m \otimes_C P \rangle + \langle t_i \otimes_C t_i \rangle \quad (37)$$
$$t_2 = (t_1 - \langle R_+ \otimes_C P \rangle) \mod R_- \quad (38)$$

/* $\Theta(1.5N \lg N)$ work */

Step 4 : small post-processing: $\Theta(n)$
ExpectedLeastSignificantDigit$(t) = d_0 = (m_0 P_0 + T_0) \mod \beta$ /* $\Theta(1)$ */
srem = NULL; /* initialize to some invalid remainder value */

```
foreach t in {t1,t2} do /* one of t1,t2 must yield correct remainder */
   if (t mod 2 != 0) then
       if (t ≥ R+) then
           t = t − R−
       else if (0 ≤ t < R+) then
           t = t + R−
       end if
   end if
   t = t/2
   if (t mod β = d0) then
       srem = t
       break                  /* if t1 matches, break out of the for loop */
   end if
end for
if (srem > P) srem = srem - P
return (srem)        □
```

2.3.4 Handle more modulii P by dynamically selecting between $R = R_+$ or $R = R_-$

$\{R_+, R_-\}$ are relatively coprime for all values of n. Furthermore, R_+ tends to factor into few large primes. Consequently, there is a high probability that either R_+ or R_- is relatively co-prime w.r.t. modulus P. In other words, the above choice of R substantially increases the cases where Montgomery's method can be applied.

Note that $R = 2^n$ is a bad choice: for this value of R, the first step of the montgomery reduction (evaluation of m) becomes equivalent to evaluating only the lower half of a product. To the best of our knowledge, there is no way to generate only the upper or lower half without also generating the other. Hence isolating the lower half needs a full linear convolution. The second step still needs a cyclic convolution, thereby making total effort $\Theta(3N \lg N)$. Instead of that, true remainder can be evaluated with $\Theta(2.5N \lg N)$ by the method shown in Section **??**.

2.4 Extending Montgomery's Method to cases where $\gcd(R, P) > 1$ but $\gcd(R_{gi}, P) = 1$

In general, R and P are not always coprime. In that case, there exist integers, R_{gi} and P_{gi} such that

$$RR_{gi} - P_{gi}P = g \quad \text{where } g = \gcd(R, P) \tag{39}$$

In general $\gcd(R_{gi}, P)$ can be > 1. However, such values of P are even less frequent (because now gcd w.r.t both R and R_{gi} must be > 1).

In this case,

$$\frac{1}{R_{gi}} \mod P \quad \text{exists}$$

and the montgomery reduction procedure can be modified as follows:

$$\begin{aligned}
m &= (TP_{gi}) \mod R \quad \text{so that} && (40)\\
(mP) \mod R &= -gT \mod R \quad \text{and} &&\\
\frac{(gT + mP)}{R} &= t \quad \text{where} &&\\
t &= T \cdot R_{gi} \mod P && (41)\\
tR &< gP^2 + PP = (g+1)P^2 && (42)
\end{aligned}$$

The overflow (above R^2, which can happen as eenn from the last equation) can be handled by pre-scaling T:

$$\begin{aligned} T &= g \cdot T^{'} + T \mod g \quad \text{where} \\ T^{'} &= \left\lfloor \frac{T}{g} \right\rfloor \qquad (43) \\ T \mod P &= g \cdot \left[T^{'} \mod P \right] + T \mod g \\ &\quad \text{use } T^{'} \text{ in place of } T \text{ in the reduction} \qquad (44) \\ T^{'} g + mP &\leq 2R^2 \qquad (45) \\ & \qquad (46) \end{aligned}$$

Consequelty, the modified Montgomery reduction requires only one additional task:
Pre-Scaling T to find $T^{'}$ and ($T \mod g$). Note that this calculation immediately makes $gT^{'}$ available so that the uses of $gT^{'}$ in the caculation of $t = (mP+T)/R$ incurs no additional cost.

The prescaling can be considered $O(N)$ if gcd(R,P) can be considered $O(1)$. In that case the complexity remains $\Theta(2N \lg N)$

2.4.1 Realizing the Square-and-reduce iteration with the same $\Theta(3.5N \lg N)$ work when $\gcd(R,P) > 1$ but $\gcd(R_{gi},P) = 1$

The computations required are

(1) $\varepsilon = (AB \mod g) = (A \mod g)(B \mod g) mod g$ $\Theta(N)$
(2) $l = (AB) \mod R_+$ $\Theta(N \lg N)$
(3) $m = lP_{\text{inv}} \mod R_+ - \varepsilon P_{\text{inv}} \mod R_+$ $\Theta(N \lg N)$
(4) $2t = (mP \mod R_-) + (AB \mod R_-) - \varepsilon$ $\Theta(1.5N \lg N)$ (5) small post processing (identical to that in all previous algorithms) $\Theta(N)$

Total effort is still $\Theta(3.5N \lg N)$ as before. Instead of explicitly evaluating T and then $T^{'}$ the above method fuses the operations to obtain a saving of $(N/2) \lg N$.

2.4.2 Fermat Numbers

For most efficient computation transform length N and hence bit-world-length n need to be powers of 2. In this case

$$R_+ = 2^n + 1 = 2^{2^m} + 1 = F_m \tag{47}$$

Fermat numbers are well studied and factor into very few relatively large factors (for an up-to-date listing of known factorizations of Fermat numbers, visit the web-site [**?**]). Since any factor which is less than 2^{32} (i.e., about one machine word long) can be considered to be $O(1)$, small factors do not pose a problem; as long as $\gcd(R_{gi}, P) = 1$ cases where $\gcd(R_+, P) > 1$ but can be considered $O(1)$ can be handled as explained in section Section **??** above).

When $\gcd(R_+, P) > O(1)$), then the complexity of modified montgomery method (modified to deal with order $O(1)$ gcds) increases, and another value of R must be selected. For all the above algorithms to be not-applicable the modulus P must simultaneously satisfy

$$\gcd(P, R_+) > O(1) \quad \text{and} \quad \gcd(P, R_{gi+}) > 1 \quad \text{AND} \tag{48}$$

$$\gcd(P, R_-) > O(1) \quad \text{and} \quad \gcd(P, R_{gi-}) > 1 \quad \text{AND} \tag{49}$$

$$2^n > P > 2^{n-1} \tag{50}$$

Given that fermat numbers have very few factors, we expect the number of modulii that satisfy the above conditions to be very small. For instance for $n = 1024$

$$\begin{aligned} R_+ = F_{10} &= 45592577 \cdot 6487031809 \cdot \\ &\quad 4659775785220018543264560743076778192897 \cdot P252 \\ R_- &= (F_9)(F_8)(F_7)\cdots 9.5.3 \end{aligned}$$

In this case, the first 2 factors of R_+ can be considered to be $O(1)$. Likewise for , all factors of R_- below F_7 can be considered to be $O(1)$. That leaves very few choices to create a P that satifies all the (pathological) conditions above.

If such a P (whose gcd wrt $R_+, R_- > O(1)$) is at hand, then it is better to use the straight remaindering algorithm presented in Section **??**. In these cases P contains factors of fermat numbers which are of the form $(k.2^q + 1)$ where $q \geq m + 2$. We conjecture that the presence of such factors with a specific form in the modulus P should make it possible to tailor the straight remaindering methods of section Section **??** and drop their complexity further below $2.5N \lg N$,

3 Discussion and Conclusions

3.1 Alternate forms of straight remaindering in $\Theta(2.5N \lg N)$

Originally we had derived an alternate $\Theta(2.5N \lg N)$ remaindering algorithm based on the associativity of cyclic convolution. It was an improvement over our prior results [**?**] wherein we first evaluated Q_e, a very close approximation to the quotiont $Q = \lfloor (X)/P \rfloor$ via a full linear convolution.

$$
\begin{aligned}
Q_e &= \left\lfloor \frac{X}{P} \right\rfloor && (51)\\
\frac{X}{P} &= \frac{X}{R} \cdot \frac{R^2}{P} \cdot \frac{1}{R} \quad \text{where,} \quad R = 2^n && (52)\\
Q_e &= \left\lfloor \left(\left\lfloor \frac{X}{R} \right\rfloor \cdot \left\lfloor \frac{R^2}{P} \right\rfloor \right) \cdot \frac{1}{R} \right\rfloor && (53)\\
&= \left\lfloor \frac{X_u P_{\text{inv}}}{R} \right\rfloor \quad \text{where} && (54)\\
\left\lfloor \frac{R^2}{P} \right\rfloor &= P_{\text{inv}} \text{ (precomputed)} && (55)
\end{aligned}
$$

Accordingly, the prior $\Theta(3N \lg N)$ method we proposed in [?] had the following steps.

$$
\begin{aligned}
&\text{Step 1 :} \quad \text{Do the full linear convolution } X_u \times P_{\text{inv}} \text{ and retrieve the upper half } Q_e && (56)\\
&\text{Step 2 :} \quad \text{perform a cyclic convolution of } Q_e \text{ and } P && (57)\\
&\qquad \text{and subtract } X_u \text{ and perform some simple corrections} && (58)
\end{aligned}
$$

To derive a $\Theta(2.5N \lg N)$ method, note that

$$
\begin{aligned}
2Q_e &= (X_u P_{\text{inv}}) \mod R_+ + (X_u P_{\text{inv}}) \mod R_- && (59)\\
\langle 2Q_e \otimes_C P \rangle &= 2Q_e \cdot P \mod R_-\\
&= [(X_u P_{\text{inv}}) \mod R_+ + (X_u P_{\text{inv}}) \mod R_-]P \mod R_-\\
&= [(X_u)(P \cdot P_{\text{inv}})] \mod R_- + [(X_u P_{\text{inv}}) \mod R_+)] \cdot P \mod R_-\\
&= (X_u \Delta) \mod R_- + (mP) \mod R_- \quad \text{where} && (60)\\
&\qquad m = X_u P_{\text{inv}} \mod R_+ \quad \text{and } \Delta = P_{\text{inv}} P \mod R_-\\
\langle 2Q_e \otimes_C P \rangle &= (X_u \Delta + mP) \mod R_- && (61)
\end{aligned}
$$

It thrns out that this (older) method is numerically not as well behaved and needs somewhat involved intermediate checks. We've mentioned it for the sake of completeness.

3.2 Crossover: word lengths where a speedup can be seen

Note that the methods use both cyclic and negacyclic convolutions. At any wordlength above machine-word-length, a cyclic convolution is always faster than a full linear convolution. A negacyclic convolution takes same effort as a linear convolution at small lengths and requires only half the effort at large wordlengths. In any case, nagacyclic convolution never requires more effort than a linear convolution. So, replacing 2 linear convolutions with one cyclic and one negacyclic will always be faster. Consequently, even at small wordlengths (512 bits and above) a gain can be expected (such a gain was seen in the experimental data presented in [?] at relatively small worldlengths: 768 bits and above)

3.3 Straight Remaindering vs. Montgomery Reduction

The above results demonstrate the fundamental reason why whenever applicable, Montgomery's method is always faster. Montgomery's method calculates something other than the true remainder ($X \mod P$). The value it calculates is carefully chosen to make its evaluation easy (which is the essential inginuity of Montgomery's method). The ability get-away with an easy-to-calculate alternate value must result in less work required, which is reflected in the $\Theta(\frac{N}{2}\lg N)$ less work for montgomery as opposed to straight remaindering.

3.4 Modular Exponentiation

The best known prior results [**?**, **?**] show ways to accomplish one entire iteration of this **square-reduce-assimilate** loop with $\Theta(10N\lg N)$ effort.

(1) If the straight-remaindering is used one iteration can be done in $\Theta(9N\lg N)$ work. (To the best of our knowledge square-and-true-remainder iteration requires the full linear convolution to compute t_i^2 followed by the straightremaindering which makes the total effort $\Theta(4.5N\lg N)$. None of the methods of fusing operations (that we've tried) seems to yield a lower total effort).

(2) Using the proposed fused-square-and-montgomery-reduce method one iteration can be done in $\Theta(7N\lg N)$ work.

Acknowledgment The authors would like to thank Dr. David Fu from the National Security Agency (NSA) for his feedback during the course of this project.

References

[1] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996. http://www.cacr.math.uwaterloo.ca/hac/.

[2] D. Naccache and D. M'Raïhi, "Cryptographic smart cards," *IEEE Micro*, pp. 14–24, June 1996.

[3] P. Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor," in *Advances In Cryptology – CRYPTO '86 (LNCS 263)* (A. M. Odlyzko, ed.), pp. 311–323, 1987.

[4] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, 1985.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 2nd ed., 2001.

[6] D. E. Knuth, *The Art of Computer Programming – Seminumerical Algorithms*, vol. 2. Addison-Wesley, 3rd ed., 1998.

[7] D. J. Bernstein, "Multidigit multiplication for mathematicians," *Advances in Applied Mathematics*, Aug. 12 2001.

[8] Kerry Bloodworth's web-site http://careybloodworth.tripod.com/index.htm.

[9] D. G. Cantor and E. Kaltofen, "On fast multiplication of polynomials over arbitrary algebras," *Acta Informatica*, vol. 28, no. 7, pp. 693–701, 1991.

[10] A. Schönhage and V. Strassen, "Schnelle Multiplikation großer Zahlen. (German) [Fast multiplication of large numbers]," *Computing*, vol. 7, no. 3–4, pp. 281–292, 1971.

[11] A. Bosselaers, R. Govaerts, and J. Vandewalle, "Comparison of three modular reduction functions," in *Advances In Cryptology – CRYPTO '93 (LNCS 773)* (D. R. Stinson, ed.), pp. 175–186, 1994.

[12] GNU Multi Precision Library Reference Manual http://www.gnu.org/software/gmp/manual/.

[13] R. Crandall and B. Fagin, "Discrete weighted transforms and large-integer arithmetic," *Mathematics of Computation*, vol. 62, pp. 305–324, Jan. 1994.

[14] FFT De-mystified http://http://www.eptools.com/tn/T0001/INDEX.HTM.

[15] Dan Bernstein's web-site, http://cr.yp.to.

[16] see the discussion on the half cyclic convolution on page 220 of a recent textbook draft (2004) http://www.jjj.de/fxt/fxtbook.pdf.

[17] C. Percival, "Rapid multiplication modulo the sum and difference of highly composite numbers," *Mathematics of Computation*, vol. 72, no. 241, pp. Pages 387–395, 2002.

[18] http://www.fftw.org.

[19] A. Schönhage, "Fast algorithms: a multitape turing machine implementation," 1994.