Project Description : Exploring Fundamental Computation vs. Memory trade-offs: Does a 512-bit wide Floating point Datapath obviate the need for integer-only methods in most applications ?

1 Introduction

The Fast Fourier Transform (FFT) [8, 9] has revolutionized multiple fields including signal processing (thereby impacting the entie field of Electrical Engineering, control theory and practise ...), optics and **theoretical mathematics**. FFT makes it possible to realize a convolution in one domain as a point-wise product in the other domain thereby making it possible to realize a convolution of vectors of size N with $O(N \lg N)$ computations. Since polynomial multiplication is a convolution, multiplication of long integers is done via FFTs. All long word-length cryptographic algorithms therefore heavily rely on FFTs. Beyond long integer operations, Increasingly larger number of problems from diverse areas like information retrieval (ex: approximate string/text matching, scoring/ranking the closeness of vectors of anything in general, multi dimensional matching...), bioinformatics (ex: genome sequencing) are being shown to benefit from FFT based processing methods.

Fundamentally FFT based processing involves a memory versus computation tradeoff (for instance the sine and cosine values are assumed to be precomputed or generated by a hardware functional unit). Access patterns required by FFT methods result in less efficient processor-cache utilization. As is very often the case, memory becomes the limitation much before computation does. Indeed if memory is the bottleneck, then computationally the algorithm is adequately efficient. In other words, further enhancements in computation speed will not be useful because memory is the bottleneck.

Today, available precision is a bottlenck not the memory. That suggests that more effort should be put into creating hardware with higher precision.

We illustrate this with a concrete problem : FFT based multiplication of long integers of length n bits. Here, the integers are treated as polynomials of some radix β so that each operand is represented as a vector of length $N = \left\lceil \frac{n}{\lg \beta} \right\rceil$ where each element is a radix β digit. For efficiency reasons, N should be a power of 2 and this is achieved by zero padding the vectors appropriately. Likewise, radix β is also selected to be a power of 2 for obvious reasons. The FFT based multiply then involves the following steps
(i) Zero-pad the vectors in higher significant places.
(ii) evaluate the FFT of the zero-padded 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 $\Theta(N)$).
In comparison, the integer-only methods (for instance the Schoenhage-Strassen multiplication method) require $O(N \lg N \lg \lg N)$ *integer* operations which is higher than the $N \lg N$ oprations required by complex floating point FFT methods. So whenever possible it is advantageous to use complex-floating point methods.

However, Today's double precisioin floating point datapath (64 bit-wide) can handle multiplication of operands of length only upto (about) 2^{20} bits (this is simply a round-figure conservative estimate for the purpuose of illustration. For the exact word length that can be handled please refer to [35]).
For word-lengths larger than 2^{20} bits, one has to use the ineger-only methods that have a higher operation count.

The total operand bit length n that can be handled by FFT based methods depends on the available precision: it varies as $\Theta(\lg n)$.

So increasing the width of the datapath from 64 to $8 \times 64 = 512$ bits should make it possible to handle wordlengths of $(2^{20})^8 = 2^{160}$ bits.

Now 2^{160} is larger than the number of molecules on earth. In other words, 512 bit floating point datapath will be able to support complex floating point FFT based mulitplicaction of such long numbers that they cannot be physically stored anywhere.
In essence the ultrawide datapath **removes precision as a constraining factor and makes memory the limiting factor**.

Even though we illustrated the problem from the perspective of multiplying long integers, convolutions of other forms (Boolean convolution and alike) are also implemented via FFT. Consequently a very wide floating point datapath should benefit all types of approximate "matchings" in general which has implications for genome sequencing and other problems in bioinformatics. Clustering and information retrieval in a more general sense involves multi-dimensional matching/correlations/convolutions. All these operations should benefit from a wide floating point datapath. In our view, this alone is a sufficient justification for creating such a datapath.

Furthermore It is known that "interior methods" that explore the interior of constraint space can be more efficient than methods that explore only the corners/hypersurfaces. For instance, the original simplex method for optimizing linear objective function under linear constraints essentially progresses on the surfaces/edges while more recent enhancements such as Karmarkar's algorithm [23] approach the optimum through the interior of the volume defined by the constraints).
Increasingly larger number of methods are leveraging complex roots of unity for approximate solutions to NP and hard problems (for instance [40, 17]) An ultrawide datapath is a valuable tool in all such computation paradigms.

1.1 Current state-of-the-art of Modular Reduction

Since we encountered the problem during our work on fast modular reduction techniques, we first summarize the current state of the art in modular reduction techniques and allied areas such as modular exponentiation. We then briefly describe the state of the art in substring matching and performing "other products" via FFT.

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

$$X = Q \cdot P + S \quad \text{where} \quad 0 < X < P^2,\ 0 \leq R < P \tag{1}$$

Modulus P is assumed to be n bits long: $2^{n-1} < P < 2^n$ 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.

Modular reduction is a fundamental operation in cryptographic systems. Most well known modular reduction methods including Barrett's [5] and Montgomery's [29] algorithms leverage some-pre computations to avoid divisions so that the main complexity of these methods lies in a sequence of two long multiplications. For large wordlengths a multiplication which is tantamount to a linear convolution is performed via the Fast Fourier Transform (FFT) or other transform-based techniques as in the Schonhage-Strassen multiplication algorithm [42].

$\langle 1 \rangle$ We have recently developed a fast remaindering algorithm for reduction w.r.t. a fixed modulus P that has the following characteristics:
(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) it works for any modulus P, irrespective of whether it is relatively co-prime with anything else.

This method covers all cases where Montgomery's algorithm is not applicable (either because the modulus P is not relatively co-prime w.r.t the radicii of interest, or 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)

$\langle 2 \rangle$ 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 factors than $2^n - 1$ so that it is more likely to be coprime w.r.t. moduli 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.

$\langle 3 \rangle$ We extend Montgomery's method to include moduli P that are not relatively co-prime w.r.t. R but are relatively co-prime w.r.t. R_{gi} (subscript "gi" denotes the generalized inverse of R. The modular inverse of R does not exist when$\gcd(R, P) > 1$). This further enhances the applicability of the proposed implementations of Montgomery's methods.

$\langle 4 \rangle$ We are currently verifying ultrafast modular exponention algorithms that evolved during the writing of this document (outlined in Section **??**).

Ⓑ String matching and scoring the closeness of a match (approximate matching) algorithms also employ FFT. An excellent treatment of the problem can be found in [41]. Given a Text/Data block of length n and a pattern P of length m, string matching involves calculating the hamming distance between P and T at every relative position of P with respect to T as P is **"slid-past"** the text block. This process is a convolution of a slightly different type. The deterministic method in [16] computes a mask and non-masked version for each alphabet that results in replacing T and P with vectors binary vectors. A convolution then measures hamming distance at all positions (of P w.r.t. T). The complexity of this method is $O(\sigma n \lg m)$ where σ is the size of the alphabet.

More recent methods [4, 41, 3] randomly map the letters of the alphabet onto integers in the range 0 to σ and then onto σth complex roots of unity. The same letters in P are mapped onto the multiplicative inverses (complex conjugates) of the corresponding values in T. Padding and some intermediate steps are preformed, followed by a fast convolution and a small amount of post processing.

It appears that a wider floating point datapath would allow a direct matching of very long strings. This has implications for genome sequencing and other bioinformatics problems. A systematic plan to explore this issue is presented in the proposed research section (Section **??**).

1.2 Overview of Proposed Research

$\boxed{1}$ Buliding upon our recent past and on-going work on fast modular reduction algorithms we intend to
(i) Implement all the algorithms developed in GMP and perform a thorough timing analysis
(ii) Continue the investigation of open-ended problems including the exploration of whether a "carry-release" in the original domain can be approximated in the Transform Domain. We believe that the carry release is the main factors that thwarts FFT based integer division fails even when the remainder is known to be 0 ahead of time.
(iii) Investigate allied issues such as efficient/fast multiplicative and modular inverse computations.

$\boxed{2}$ Explore the capabilities and limitations afforded by an ultrawide floating point datapath.
(i) Refine the precision analysis in [35]. We believe it is possible to derive tighter bounds on errors (as outlined in Section **??**). This should bolster the precsion analysis.
(ii) Explore in depth the total complexity of achieving the computation via an ultrawide floating point data path versus using integer-only operations.
(iii) Identify/characterize the more general class of problems (beyond convolutions) that can be imple-

mented more efficiently with floating point operations than by resorting to integer-only methods.

$\boxed{3}$ (i) Investigate the application of an ultrawide floating point datapath to substring matching, approximate/partial matching/scoring. It is expected that the investigation will reveal connections to other problems in bioinformatics, information retrieval/querying as well as clustering.

(ii) Bloom filtering is used to signal potential membership in a set. It can be effectively used as a fast pre-processing technique. For instance a list of "good customers" or "bad customers" could be summarized in the form of bloom filter banks and an incoming request can be quickly and *most of the time correctly* classified as belonging to one of groups. The classification is not guaranteed to be always corrrct, but the false positives and false negatives can be adjusted by trading off more memory and computation. In essence Bloom filters leverage the fundemantal memory-computation vs classification accuracy tradeffs in hashing.

We believe that there is a connection between Bloom filtering and substring matching. Bloom filtering can be considered to capture gradually coarser/less-specific information about "matches/overlaps". We propose to formally investigate this issue.

$\boxed{5}$ Investigate the transformations of integer programming and other discrete problems into ultrawide floating point operations. We conjeture that ultrawide floating point processing capability could usher in fundamentally new ways of handling problems previosuly thought to be non-floating point in nature (string matching is a good example).
We intend to explore how an ultrawide datapath helps methods that employ roots of unity for approxiate solutions to NP problems [40, 17] or take Interior point approaches to NP complete problems [23, 24, 22]).

$\boxed{5}$ Guided by the results of the analysis of fundamental tradeoffs between computation and memory we propose to design and implement an ultrawide datapath. The PI has substantial experience designing and implementing Arithmetic Algorithms in VLSI. The Co-PI has substantial VLSI design and testing experience and was braught on board specifically for handling the VLSI implementation aspects of the project.

2 Background and Related Work

2.0.1 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 N to about 2^{17} [11, 35, 12]
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.

2.1 Problem Definition

Modular Reduction was defined in the Introduction above.

2.1.1 Modular Exponentiation

Here the problem is to evaluate $X^Y \mod P$. Repeated squaring and reducing yields the result. The pseudo-code to find the result is:

```
Let $Y = (y_{k-1}y_{n-2}\cdots y_0)$ in binary where $y_{k-1} = 1$ initialize:  $M_e = X^2 \mod P$
for (i = k-2; i >= 1; i--){
        if $(y_i == 1)$ $M_e = M_e \cdot X \mod P$
        $M_e = (M_e)^2 \mod P$
}
if $(y_0 == 1)$ $M_e = M_e \cdot X \mod P$              /* $M_e$ is the result */
```

Note that this loop corresponds to evaluating the exponent by the Horner's polynomial evaluation method. We call one iteration the above loop as the **assimilate-square-reduce** operation. If y_i is zero then the multiplication by X is skipped and the loop operation simplies to a **square-and-reduce** operation.

2.2 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/nega-cyclic polynomial convolution is defined as the product modulo $(x^n \mp 1)$ respectively:

$$\text{cyclic: sum of upper/lower halves: } C(x)\%(x^n-1) = (U(x)x^n + L(x))\%(x^n-1) = U(x)+L(x) \quad (2)$$

$$\text{nega-cyclic: difference of the halves: } C(x)\%(x^n+1) = (U(x)x^n + L(x))\%(x^n+1) = L(x)-U(x) \quad (3)$$

Wrap-around convolutions take less effort :

It is well known that For transform based methods, cyclic/negacyclic convolutions require half the work of a linear convolution. 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 $\{A_HB_H, A_LB_L$ and $(A_HB_L + A_LB_H)\}$. Karatsubba's method computes the last value as $(A_H+A_L)(B_H+B_L) - A_HB_H - A_LB_L$ replacing 4 products by 3 products.

If only a cyclic convolution is required, then only two values are required $(A_HB_H + A_LB_L)$ and $(A_HB_L + A_LB_H)$. These can be generated with only two products C_1 and C_2 (instead of the 3 required by a linear convolution):

$$\begin{aligned} C1 &= (A_H+A_L)(B_H+B_L) & C2 &= (A_H-A_L)(B_H-B_L) \\ (A_HB_H + A_LB_L) &= \frac{C1+C2}{2} & (A_HB_L + A_LB_H) &= \frac{C1-C2}{2} \end{aligned} \quad (4)$$

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). As the number of parts into which a vector is split increase, negacylic convolution takes lesser effort. Asymptotically both cyclic and nega cyclic convolutions take same effort.

2.2.1 Basic Montgomery Reduction

For efficient modular reductio w.r.t. a given modulus P, Montgomery selects an $R > P$ which is relatively prime w.r.t. P, i.e., $\gcd(R,P) = 1$.

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

$$RR^{-1} - P'P = 1 \quad \text{and} \quad 0 < R^{-1} < P \quad \text{and} \quad 0 < P' < R$$

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 $\text{MonProd}(A,B)$ which is defined as
$\text{MonProd}(A,B) = ABR^{-1} \mod P$
. *Step 1 :* Compute $T = A \times B$
Step 2 : $m = ((T \mod R)P^{'}) \mod R$ so that $0 \leq m \leq R-1$
Step 3 : $t = (T + mP)/R$
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, each multiplication modulo P can be replaced by $\text{MonProd}(.,.)$ which is a bit more efficient than regular modular reduction. At the end there is a need to convert the result back from the montgomery residue class.

2.3 Efficient Implementation of Montgomery's method

The framework in [28] selects an integer Q satisfying

$$\frac{Q}{\gcd(Q,R)} > \frac{(P-1)^2 + (R-1)P}{R} \geq t \tag{5}$$

and performs the two modified steps:

$$m = ABP^{'} \mod R \tag{6}$$
$$t = (\frac{AB+mP}{R}) \mod (\frac{Q}{\gcd(Q,R)}) \tag{7}$$

Given a double length operand X It shows a method to perform Montgomry reduction (to evaluate $XR^{-1} \mod P$) with one cyclic and one negacyclic convolutin, requring $\Theta(2N \lg N)$ effort which is the same as a linear convolution or a multiply operation. It then shows a way to implement $[ABR^{-1} \mod P]$ with $\Theta(3.5N \lg N)$ effort using cyclic convolutions followed by negacyclic ones.

While this set of equations broadly defines a framework, we believe the most interesting variation has been left out as explained in section Section 3.3.

3 Proposed Research

First We summarize our current work on fast modular reduction and demonstrate the new results. We propose to thourougly evaluate the real complexity (including issues such as caching efficiency which often throws off theoretical predictions) of the new algorithms via exhaustive testing. We'll then implement the new algorithms in GMP package.

3.1 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.

$$X \mod P = (RX_u \mod P + X_l \mod P) \mod P \tag{8}$$
$$RX_u = \frac{R^2 X_u}{R} \tag{9}$$
$$\text{Let } R^2 = PP_{\text{inv}} + \delta \quad \text{where} \quad P_{\text{inv}}, \delta \quad \text{are precomputed quotient and remainder when } R^2 \text{ is divided by } P \tag{10}$$

$$\begin{aligned}
RX_u &= \frac{(PP_{\text{inv}}+\delta)X_u}{R} = \frac{(P(P_{\text{inv}}X_u)+\delta X_u)}{R} \\
\text{Let} & \\
P_{\text{inv}}X_u &= Q^{'}R+L \quad \text{then} \\
RX_u &= \frac{P(Q^{'}R+L)+\delta X_u}{R} = PQ^{'} + \frac{PL+\delta X_u}{R} && (11) \\
RX_u \mod P &= \left(\frac{PL+\delta X_u}{R}\right) \mod P && (12)
\end{aligned}$$

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

$$\begin{aligned}
LP+\delta X_u &< R\cdot P + P\cdot P = RP + P^2 < 2RP \quad \text{so that} \\
\frac{LP+\delta X_u}{R} &< 2P \quad \textit{and} \\
\frac{LP+\delta X_u}{R} & \quad \text{must be an integer, as seen from (11).} && (13) \\
\text{Let} \quad t &= \frac{LP+\delta X_u}{R} \quad \text{then, desired remainder} && (14) \\
RX_u \mod P &= t \quad \text{or} \quad t-P && (15)
\end{aligned}$$

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 (11), t is an integer implies

$$\begin{aligned}
LP+\delta X_u &= 2^n t + t && (16) \\
&\Rightarrow \text{sum of upper and lower halves} = 2t \\
2t &= \mathcal{U}(LP+\delta X_u) + \mathcal{L}(LP+\delta X_u) \\
&= [\mathcal{U}(LP)+\mathcal{L}(LP)] + [\mathcal{U}(\delta X_u)+\mathcal{L}(\delta X_u))] \\
&= (LP \mod R_-) + (\delta X_u \mod R_-) && (17) \\
& \quad \text{and a sum of cyclic convolution is sufficient}
\end{aligned}$$

Transforms representing the cyclic convolution $\langle L \otimes_C P\rangle$ and $\langle \delta \otimes_C X_u\rangle$ can be added before taking the inverse transform so that this step can be done in $\Theta(1.5N\lg N)$ work: (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

$$\begin{aligned}
LP+\delta X_u &= 2^n t - t && (18) \\
&\Rightarrow \text{difference of upper and lower halves} = 2t \\
2t &= \mathcal{U}(LP+\delta X_u) - \mathcal{L}(LP+\delta X_u) \\
&= [\mathcal{U}(LP)-\mathcal{L}(LP)] + [\mathcal{U}(\delta X_u)-\mathcal{L}(\delta X_u)] \\
&= (LP \mod R_+) + (\delta X_u \mod R_+) && (19) \\
& \quad \text{and a sum of negacyclic convolutions is sufficient}
\end{aligned}$$

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

Algorithm StraightRemaidering `/* 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

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

Step 1 : $L = (X_u P_{\text{inv}}) \mod R_+$
`/*` **negacyclic convolution**, `requires` $\Theta(N \lg N)$ `work */`
Step 2 : Compute

$$t_1 = \langle L \otimes_C P \rangle + \langle \delta \otimes_C X_u \rangle \tag{20}$$
$$t_2 = (t_1 - \langle R_+ \otimes_C P \rangle) \mod R_- \tag{21}$$

`/*` $\Theta(1.5N \lg N)$ `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 $\Theta(2.5N \lg N)$
```

3.2 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 signifi cant 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.

$$
\begin{aligned}
t &= (\frac{LP+\delta X_u}{R}) \quad \bmod P && (22)\\
\text{Since} \quad LP+\delta X_u &< RP+P^2, \quad \text{it must be expressable as} && (23)\\
LP+\delta X_u &= RP+\theta \quad \text{where} && (24)\\
\theta &< P^2 \quad \text{and } \theta \text{ is divisible by } R\text{, so that} && (25)\\
t &= (P+\frac{\theta}{R}) \quad \bmod P = \frac{\theta}{R} \quad \Rightarrow && (26)\\
t &= (LP+\delta X_u - RP)/R \quad \text{which in turn implies that} && (27)\\
2t &= \langle m \otimes_C P\rangle + \langle t_i \otimes_C t_i\rangle - \langle R \otimes_C P\rangle \qquad \square && (28)
\end{aligned}
$$

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.

3.3 Substantially Increasing Applicability of Montgomery's Method

Clearly, the results of previous section can be carried over to Montgomery's method as well, and R can be selected to be R_+or R_-. The selection $R = R_+$ is a new variation in the framework proposed in [28] as explained below. This seemingly trivial change of order of cyclic and negacyclic convolutions substantially increases the number of moduli for which Montgomery's method is applicable.

Realizing Montgomery Reduction with one negacyclic and one cyclic convolution

While [28] broadly defines a framework, we believe the most interesting variation has been left out.

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

$$
\begin{aligned}
& \gcd(R,Q) = \gcd(R,Q') = 1 \quad sothat \\
(\frac{Q}{\gcd(Q,R)}) = 2R_- = 2R_+ - 4 > \frac{(P-1)^2+(R-1)P}{R} \quad & since \qquad (29)\\
& R \geq P-1 \qquad (30)
\end{aligned}
$$

Now step 1 performs a negacyclic convolution and step2 performs a of cyclic convolution, i.e., if the square double length product $T = AB$ is fully evaluated, then Montgomery reduction takes $\Theta(2N\lg N)$ work. The square-and-reduce iteration ($M_e = \text{MonProd}(M_e, M_e)$) can be realized with $\Theta(3.5N\lg N)$ work with any order of cyclic/negacyclic convolutions. in [].

3.3.1 Handle more moduli 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, variation 3 proposed herein makes it feasible to dynamically select R so as to make it co-prime w.r.t. P. Consequently, 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 (see the discussion on the "half cyclic convolution" on page 220 of [1] basically says that there's no known way (that scales with n) of just getting one half without the other.). Isolating the lower half therefore 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 3.1.

3.4 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

$$RR_{gi} - P_{gi}P = g \quad \text{where } g = \gcd(R,P) \tag{31}$$

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, $(\frac{1}{R_{gi}} \mod P)$ exists and the Montgomery reduction procedure can be modified as follows:

$$m = (TP_{gi}) \mod R \quad \text{so that} \tag{32}$$

$$(mP) \mod R = -gT \mod R \quad \text{and}$$

$$\frac{(gT+mP)}{R} = t \quad \text{where} t = T \cdot R_{gi} \mod P \quad \text{and} \quad tR < gP^2 + PP = (g+1)P^2 \tag{33}$$

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

$$T = g \cdot T^{'} + T \mod g \quad \text{where}$$

$$T^{'} = \left\lfloor \frac{T}{g} \right\rfloor \quad \text{and} \quad T \mod P = g \cdot \left[T^{'} \mod P \right] + T \mod g$$

$$\text{use } T^{'} \text{ in place of } T \text{ in the reduction} \tag{34}$$

$$T^{'}g + mP \leq 2R^2 \tag{35}$$

Consequently, 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^{-1} in the calculation of $t = (mP + gT^{-1})/R$ 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 $\Theta(2N\lg N)$

3.4.1 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 dominant effort is still $\Theta(3.5N\lg N)$ as before. Instead of explicitly evaluating T and then T^{-1} the above method fuses the operations to obtain a saving of $(N/2)\lg N$.

3.5 Fast Modular Exponentiation

The modular exponention loop has the following two computations
`Assimilate :  if` $(y_i == 1)$ `then` $M_e = M_e \cdot X \mod P$
`Square-and-reduce :` $M_e = M_e^2 \mod P$

Note that initial operand X required in the assimilation step is independent of loop iteration index. In other words X can be considered a constant within the loop itself. This in turn implies that a few quantities related to X can be computed once before the loop and re-used in all the iterations of the loop. This is tantamount to a precompution. We show ways to leverage such loop-invariant precomputation.

3.5.1 Speeding up Modular Exponention based on True Remaindering

result 1 : Assimilation can be done with $\Theta(2.5N \lg N)$ work since X is a loop-invariant.
Sketch-of-Proof/Procedure : Identical to true-remaindering

$$\text{Let } C \quad \text{be an invariant, precompute Quotient, Rem before loop} \tag{36}$$

$$RC = PQ + \theta \quad \text{then} \tag{37}$$

$$Z \cdot C = \frac{RZC}{R} = \frac{(PQ+\theta)Z}{R} \qquad \text{Let} \qquad QZ = RU + L \quad \text{so that} \tag{38}$$

$$ZC \mod P = \frac{PL + \theta Z}{R} \tag{39}$$

Complexity (dominent terms)

Compute $L = QZ \mod R$	$\Theta(N \lg N)$ (Negacyclic, assuming $R = R_+$)
Forward Transform L, Z, Inverse Transform sum	$\Theta(\frac{3N}{2} \lg N)$
total	$\Theta(2.5N \lg N)$

result 2 : $(X \cdot C_1 + Y \cdot C_2) \mod P$ can be done with $\Theta(3.5N \lg N)$ work when C_1, C_2 can be considered constants (for example, loop invariants)

Procedure : Precompute quotient and remainders Q_i, θ_i :

$$RC_i = PQ_i + \theta_i \quad ; \quad i = 1,2$$

$$(X \cdot C_1 + Y \cdot C_2) \mod P = \frac{L_1 P + \theta_1 X}{R} + \frac{L_1 P + \theta_1 Y}{R} = \frac{(L_1 + L_2)P + \theta_1 X + \theta_2 Y}{R} \tag{40}$$

$$L_1 + L_2 = (Q_1 X + Q_2 Y) \mod R \quad \text{negacyclic,} \tag{41}$$

$$\text{2 forward FFTs (of negacyclic-modulated } X \text{ and } Y \text{ and} \tag{42}$$

$$\text{one reverse transform} \Theta(1.5N \lg N) \tag{43}$$

$$\text{then 3 forward FFT's and 1 reverse FFT } \Theta(2N \lg N) \tag{44}$$

$$\text{total effort } \Theta(3.5N \lg N) \tag{45}$$

result 3 : Assimilate-square-and-reduce can be realized with $\Theta(5.5N \lg N)$ work.

Proof : (a) if (y_i == 1) then at the end of the loop

$$M_e = (M_e X)^2 \mod P = (M_e^2 (X^2 \mod P)) \mod P \tag{46}$$

$$= (UR + L)\delta \mod P \quad \text{where} \tag{47}$$

$$M_e^2 = UR + L \text{ Full Linear convolution } \Theta(2N \lg N) \text{ and} \tag{48}$$

$$\delta = X^2 \mod P \text{ precomputed once before loop} \tag{49}$$

$$M_e = U(R\delta) \mod P + L\delta \mod P \tag{50}$$

$$= (U\gamma + L\delta) \mod P \quad \gamma \text{ precomputed at loop setup} \tag{51}$$

$$= \text{by result 2 the last step } \Theta(3.5N \lg N) \tag{52}$$

$$= \text{total complexity } \Theta(5.5N \lg N) \tag{53}$$

(b) if (y_i == 1) then $M_e = M_e^2 \mod P$ which needs $\Theta(4.5N \lg N)$ work.

The average work per iteration of the Loop is $\Theta(\frac{5.5+4.5}{2} N \lg N = 5N \lg N)$ a factor of 2 improvement over prior results in [].

3.5.2 Speeding up Modular Exponention based on modified Montogmery's method

Result 4 : When C is an invariant, MonProd(Z,C) = $ZCR^{-1} \mod P$ also requires $\Theta(2.5N \lg N)$ work. (Here inputs Z and C are assumed to be in proper residue class and so is the output of their Montgomery-Product).
In other words scaling by a constant followed by remaindering is such a simple operation that Montgomery Product has no speed/simplicity advantage over true remaindering.

Note that Cyclic and Negacyclic FFTs of Z are required and they cannot be combined $\Rightarrow$
even if $C_1 = CR^{-1} \mod P$ is assumed to be precomputed $[(ZC_1) \mod R]$ needs a wrap-around convolution. Then the second step needs 2 forward and one reverse FFT. The overall total effort is therefore $\Theta(2.5N \lg N)$.

Result 5 : MonProd(M_e^2, X^2) takes $\Theta(5.5N \lg N)$ work

3.6 Investigation of further enhancements including the new "constant-scale-factor" modular reduction method

We belive further enhancements are feasible. we illustrate one new algorithm that eveolved during the writing of this document. Assume (R,P) are relatively co-prime and find R^{-1}, P^{-1} such that

$$RR^{-1} - PP^{-1} = 1 \qquad R^2R_2^{-1} - PP_2^{-1} = 1 \quad \cdots \quad R^kR_k^{-1} - PP_k^{-1} = 1 \tag{54}$$

i.e., all powers of R are also co-prime wrt P. Note that P_k^{-1} increases with powers of R but that has no effect on the algorithm because the quantity that gets used is $P_k^{-1} \mod R$.

$$\text{Let } m_k = TP_k^{-1} \mod R = (T \mod R)(P_k^{-1} \mod R) \mod R \tag{55}$$

$$m_kP \mod R = TP_k^{-1} \mod R = T(R^kR_k^{-1} - 1) \mod R = -T mod R \tag{56}$$

$$t = \frac{m_kP + T}{R} \tag{57}$$

i.e., any power of R^{-1} can be introduced by using the appropriate P^{-1}

Note that if instead of regular product XY of regular integers (not transformed into the montgomery residue class) if a montgomery product is done, it yields $(XYR^{-1} \mod P)$. We leverage this fact and adjust the number of monproduct operations and the degree of R^{-1} introduced so that at the end the total power of R^{-1} factored into the result is a constant.

```
    for ($i = k-1; i > 0; i = i-1$) {
      if ($y_i == 1$) {
      /* MonProd has no advantage do straight Remaindering
      but introduce $R^{-1}$ via values precompuated from $X$ */
          $M_e = M_e^2 \cdot [(R^{-1})X^2] \mod P$
      } else { /* MonProd is better for simply a square-and-reduce */
          $M_e = \text{MonProd}(M_e, M_e)$
      }
} /* each iteration of the loop introduces one $R^{-1}$ factor.  */
/* total power of $R^{-1}$ factored in = value of binary string 1 1 1 1 1 ....  of length k
= $2^k - 1 = I$ So correct the final answer by multiplying by precomputed $R^k \mod P$ */
$M_e = R^I M_e \mod P$   The average number of operation per is $\Theta(4.5N \lg N)$.
```

Other possible improvements include processing the exponent 2 bits at a time (like Radix-4 booth recoding of a multiplier).

3.7 Comparison of Montgomery versus straight remaindering

Operation	True Remaindering	Montgomery (finds $(.)R^{-1} \mod P$)
$T = AB$ is fully evaluated	$\Theta(2.5N \lg N)$	$\Theta(2N \lg N)$
$AB \mod P$	$\Theta(5.5N \lg N)$	$\Theta(4N \lg N)$.
$X^2 \mod P$	$\Theta(4.5N \lg N)$	$\Theta(3.5N \lg N)$.
Avg effort per iteration of the main loop in modular exponentiation	$\Theta(5N \lg N)$	$\Theta(4.5N \lg N)$

We intend to experimentally validate these results (the theory does not account for cache/memory interactions etc).

3.8 Related problems

Since remaindering is fundamental, the above improvements to remaindering will have a ripple effect. In particular we propose to investigate fast modular inversion.

We also believe it is possible to derive a tighter bound (tighter than the one presented in [35] on the precision required to support FFT based multiplication of vectors of length n bits. The interesting aspect of our derivation is that it shows a computation effort vs. accuracy tradeoff.

The method of proof essentially starts off with a butterfly operation that transforms

$$(z_0, z_1) \Longrightarrow (z_0 + \omega z_1, z_0 + \omega z_1) \tag{58}$$

There are $\lg N$ such stages where N is the length of the transform.

References

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

[2] BigSky: a system to translate expressions into VHDL code and a library of on-line arithmetic modules, http://arith.cs.ucla.edu.

[3] AMIR, A., LEWENSTEIN, M., AND LEWENSTEIN, N. Pattern matching in hypertext. *J. Algorithms 35*, 1 (2000), 82–99.

[4] ATALLAH, CHYZAK, AND DUMAS. A randomized algorithm for approximate string matching. *ALGRTHMICA: Algorithmica 29* (2001).

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

[6] BRUCEK KHAILANY, WILLIAM J. DALLY, SCOTT RIXNER ET. AL. Imagine: Media Processing with Streams. *IEEE Micro* (March/April 2001).

[7] CARTER, T. M., AND ROBERTSON, J. E. The Set Theory of Arithmetic Decomposition. *IEEE Transactions on Computers C-39*, 9 (August 1990), 993–1005.

[8] COOLEY, J. W., AND TUKEY, J. W. An algorithm for the machine calculation of complex Fourier series. *j-MATH-COMPUT 19*, 90 (Apr. 1965), 297–301.

[9] COOLEY, J. W., AND TUKEY, J. W. On the origin and publication of the FFT paper. *Current Contents*, 51–52 (1993), 8–9.

[10] CORTADELLA, J., AND LLABERIA, J. M. Evaluation of $A+B=K$ conditions without carry propagation. *IEEE Transactions on Computers C-41*, 11 (Nov. 1992), 1484–1488.

[11] CRANDALL, R., AND FAGIN, B. Discrete weighted transforms and large-integer arithmetic. *Mathematics of Computation 62*, 205 (Jan. 1994), 305–324.

[12] D. PHATAK, AND T. GOFF. Fast modular reduction for large wordlengths via One Linear and One Cyclic Convolution. *Proc. of the 17th IEEE International Symposium on Computer Arithmetic, Cape Cod, MA* (June 2005).

[13] ERCEGOVAC, M., AND GRANROV, A. L. On the performance of on-line arithmetic. In *Prof. of Int. conf. Parallel Processing (ICPP)* (1980).

[14] ERCEGOVAC, M., AND LANG, T. On recoding in arithmetic algorithms. *J. VLSI Signal Processing 14* (1996), 283–294.

[15] ERCEGOVAC, M. D. On-Line Arithmetic: An Overview. *Real Time Signal Processing VII SPIE-495* (1984), 86–93.

[16] FISCHER, AND PATERSON. String-matching and other products. In *SIAMAMS: Complexity of Computation: Proceedings of a Symposium in Applied Mathematics of the American Mathematical Society and the Society for Industrial and Applied Mathematics* (1974).

[17] GOEMANS, M. X., AND WILLIAMSON, D. Approximation algorithms for MAX-3-CUT and other problems via complex semidefinite programming. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing: Hersonissos, Crete, Greece, July 6–8, 2001* (pub-ACM:adr, 2001), ACM, Ed., pub-ACM, pp. 443–452.

[18] GOFF, T., PHATAK, D. S., AND KOREN, I. Redundancy Management in Arithmetic Processing via Redundant Binary Representations. *Proceedings of ASILOMAR'99 (Annual Conference on Signals Systems and Computers), Pacific Grove, California* (Oct. 1999), 1475–1479.

[19] HANAWA M., ET. AL. A 4.3ns 0.3 μm CMOS 54×54b Multiplier Using Precharged Pass-Transistor Logic. In *in ISSCC Digest of Technical Papers* (1996), pp. 364–365.

[20] IRWIN, M. J., AND OWENS, R. M. Fully Digit On-Line Networks. *IEEE Transactions on Computers C-32* (1983), 402–406.

[21] IRWIN, M. J., AND OWENS, R. M. Digit–Pipelined Arithmetic as Illustrated by the Paste–Up System: A Tutorial . *IEEE Computer* (Apr. 1987), 61–73.

[22] KAMATH, A. P., KARMARKAR, N. K., RAMAKRISHNAN, K. G., AND RESENDE, M. G. C. An interior point approach to Boolean vector function synthesis. *Proceedings of the 36th MSCAS* (1993), 185–189.

[23] KARMARKAR, N. K. A new polynomial–time algorithm for linear programming. *Proceedings of the 16th Annual ACM Symposium on Theory of Computing* (1984), 302–311.

[24] KARMARKAR, N. K. An interior-point approach to NP-complete problems. An extended abstract, 1990.

[25] KORNERUP, P. Digit-Set Conversions: Generalizations and Applications. *IEEE Transactions on Computers C-43*, 6 (May 1994), 622–629.

[26] KORNERUP, P. Necessary and Sufficient Conditions for Parallel and Constant Time Conversion and Addition. In *Proc. 14th IEEE Symposium on Computer Arithmetic* (April 1999), IEEE Computer Society, pp. 152–156.

[27] KUNINOBU, S., NISHIYAMA, T., EDAMATSU, H., TANIGUCHI, T., AND TAKAGI, N. Design of high speed MOS multiplier and divider using redundant binary representation. *Proc. of the 8th Symposium on Computer Arithmetic* (1987), 80–86.

[28] MCLAUGHLIN, P. B. New Frameworks for Montgomery's Modular Multiplication Method. *Mathematics of Computation 73*, 246 (2003), 899–906.

[29] MONTGOMERY, P. L. Modular multiplication without trial division. *Mathematics of Computation 44*, 170 (1985), 519–521.

[30] N. OHKUBO AND SUZUKI, M., ET. AL. A 4.4-ns CMOS 54 × 54-b Multiplier Using Pass-Transistor Multiplexor. *IEEE Journal of Solid-State Circuits 30*, 3 (Mar. 1995), 251–256.

[31] NAGENDRA, C., OWENS, R. M., AND IRWIN, M. J. Unifying Carry-Sum and Signed-Digit Number Representations. Tech. Rep. CSE–96–036, Computer Science and Engineering Department, Pennsylvania State University, 1996.

[32] NIELSEN, A. M., AND KORNERUP, P. MSB-First Digit Serial Arithmetic. *Journal of Universal Computer Science 1*, 7 (July 1995).

[33] NIELSEN, A. M., AND KORNERUP, P. Redundant Radix Representation of Rings. November 1999.

[34] PARHAMI, B. Generalized signed-digit number systems: a unifying framework for redundant number representations. *IEEE Transactions on Computers C-39* (Jan. 1990), 89–98.

[35] PERCIVAL, C. Rapid multiplication modulo the sum and difference of highly composite numbers. *Mathematics of Computation 72*, 241 (2002), Pages 387–395.

[36] PHATAK, D. S. Comments on Duprat and Muller's Branching CORDIC Paper. *IEEE Transactions on Computers* (Sep. 1998), 1037–1040.

[37] PHATAK, D. S. Double Step Branching CORDIC: A New Algorithm for Fast Sine and Cosine Generation. *IEEE Transactions on Computers TC–47*, 5 (May 1998), 587–602.

[38] PHATAK, D. S., GOFF, T., AND KOREN, I. Efficient Arithmetic Implementations Based on Carry-Save Representations. In *Proceedings of SPIE's 45th Annual Meeting, the International Symposium on Optical Science and Technology, San Diego, CA* (2000), pp. 258–266.

[39] PHATAK, D. S., GOFF, T., AND KOREN, I. Constant-time Addition and Simultaneous Format Conversion Based on Redundant Binary Representations. *IEEE Trans. on Computers TC–50*, 11 (Nov. 2001), 1267–1278.

[40] ROJAS, J. M. A subexponential algorithm for containment and intersection between translated subtori and algebraic sets. *Proceedings of MEGA 2005* (2005). available via http://www.math.tamu.edu/ rojas.

[41] SCHOENMEYR, T., AND ZHANG, D. Y. FFT-based algorithms for the string matching with mismatches problem. *J. Algorithms 57*, 2 (2005), 130–139.

[42] SCHÖNHAGE, A., AND STRASSEN, V. Schnelle Multiplikation großer Zahlen. (German) [Fast multiplication of large numbers]. *Computing 7*, 3–4 (1971), 281–292.

[43] TAKAGI, N., YASUURA, H., AND YAJIMA, S. High-speed VLSI multiplication algorithm with a redundant binary addition tree. *IEEE Transactions on Computers C-34* (Sep. 1985), 789–796.