Modular Multiplication Without Trial Division

Peter L. Montgomery


Stable URL: http://links.jstor.org/sici?sici=0025-5718%28198504%2944%3A170%3C519%3AMMWTD%3E2.0.CO%3B2-4

*Mathematics of Computation* is currently published by American Mathematical Society.

Your use of the JSTOR archive indicates your acceptance of JSTOR’s Terms and Conditions of Use, available at http://www.jstor.org/about/terms.html. JSTOR’s Terms and Conditions of Use provides, in part, that unless you have obtained prior permission, you may not download an entire issue of a journal or multiple copies of articles, and you may use content in the JSTOR archive only for your personal, non-commercial use.

Please contact the publisher regarding any further use of this work. Publisher contact information may be obtained at http://www.jstor.org/journals/ams.html.

Each copy of any part of a JSTOR transmission must contain the same copyright notice that appears on the screen or printed page of such transmission.

JSTOR is an independent not-for-profit organization dedicated to creating and preserving a digital archive of scholarly journals. For more information regarding JSTOR, please contact support@jstor.org.
Modular Multiplication Without Trial Division

By Peter L. Montgomery

Abstract. Let \( N > 1 \). We present a method for multiplying two integers (called \( N \)-residues) modulo \( N \) while avoiding division by \( N \). \( N \)-residues are represented in a nonstandard way, so this method is useful only if several computations are done modulo one \( N \). The addition and subtraction algorithms are unchanged.

1. Description. Some algorithms [1], [2], [4], [5] require extensive modular arithmetic. We propose a representation of residue classes so as to speed modular multiplication without affecting the modular addition and subtraction algorithms.

Other recent algorithms for modular arithmetic appear in [3], [6].

Fix \( N > 1 \). Define an \( N \)-residue to be a residue class modulo \( N \). Select a radix \( R \) coprime to \( N \) (possibly the machine word size or a power thereof) such that \( R > N \) and such that computations modulo \( R \) are inexpensive to process. Let \( R^{-1} \) and \( N' \) be integers satisfying \( 0 < R^{-1} < N \) and \( 0 < N' < R \) and \( RR^{-1} - NN' = 1 \).

For \( 0 \leq i < N \), let \( i \) represent the residue class containing \( iR^{-1} \mod N \). This is a complete residue system. The rationale behind this selection is our ability to quickly compute \( TR^{-1} \mod N \) from \( T \) if \( 0 \leq T < RN \), as shown in Algorithm REDC:

```plaintext
function REDC(T)
    m ← (T mod R)N' mod R [so 0 ≤ m < R]
    t ← (T + mN)/R
    if t ≥ N then return t − N else return t
```

To validate REDC, observe \( mN \equiv TN'N \equiv -T \mod R \), so \( t \) is an integer. Also, \( tR \equiv T \mod N \) so \( t \equiv TR^{-1} \mod N \). Thirdly, \( 0 \leq T + mN < RN + RN \), so \( 0 \leq t < 2N \).

If \( R \) and \( N \) are large, then \( T + mN \) may exceed the largest double-precision value. One can circumvent this by adjusting \( m \) so \( -R < m \leq 0 \).

Given two numbers \( x \) and \( y \) between 0 and \( N - 1 \) inclusive, let \( z = REDC(xy) \). Then \( z \equiv (xy)R^{-1} \mod N \), so \( (xR^{-1})(yR^{-1}) \equiv zR^{-1} \mod N \). Also, \( 0 \leq z < N \), so \( z \) is the product of \( x \) and \( y \) in this representation.

Other algorithms for operating on \( N \)-residues in this representation can be derived from the algorithms normally used. The addition algorithm is unchanged, since \( xR^{-1} + yR^{-1} \equiv zR^{-1} \mod N \) if and only if \( x + y \equiv z \mod N \). Also unchanged are...
the algorithms for subtraction, negation, equality/inequality test, multiplication by
an integer, and greatest common divisor with \( N \).

To convert an integer \( x \) to an \( N \)-residue, compute \( xR \mod N \). Equivalently,
compute \( \text{REDC}((x \mod N)(R^2 \mod N)) \). Constants and inputs should be converted
once, at the start of an algorithm. To convert an \( N \)-residue to an integer, pad it with
leading zeros and apply Algorithm \( \text{REDC} \) (thereby multiplying it by \( R^{-1} \mod N \)).

To invert an \( N \)-residue, observe \((xR^{-1})^{-1} \equiv zR^{-1} \mod N \) if and only if \( z \equiv R^2x^{-1} \mod N \). For modular division, observe \((xR^{-1})(yR^{-1})^{-1} \equiv zR^{-1} \mod N \) if and only if \( z \equiv x(\text{REDC}(y))^{-1} \mod N \).

The Jacobi symbol algorithm needs an extra negation if \( (R/N) = -1 \), since
\((xR^{-1}/N) = (x/N)(R/N) \).

Let \( M \mid N \). A change of modulus from \( N \) (using \( R = R(N) \)) to \( M \) (using \( R = R(M) \))
proceeds normally if \( R(M) = R(N) \). If \( R(M) \neq R(N) \), multiply each \( N \)-residue by
\((R(N)/R(M))^{-1} \mod M \) during the conversion.

2. Multiprecision Case. If \( N \) and \( R \) are multiprecision, then the computations of
\( m \) and \( mN \) within \( \text{REDC} \) involve multiprecision arithmetic. Let \( b \) be the base
used for multiprecision arithmetic, and assume \( R = b^n \), where \( n > 0 \). Let \( T =
(T_{2n-1}T_{2n-2} \cdots T_0)_b \) satisfy \( 0 \leq T < RN \). We can compute \( TR^{-1} \mod N \) with \( n 
\) single-precision multiplications modulo \( R \), \( n \) multiplications of single-precision
integers by \( N \), and some additions:

\[
c \leftarrow 0
\]

\[
\text{for } i := 0 \text{ step } 1 \text{ to } n - 1 \text{ do}
\]

\[
(dT_{i+n-1} \cdots T_i)_b \leftarrow (0T_{i+n-1} \cdots T_i)_b + N^*(T_iN' \mod R)
\]

\[
(cT_{i+n})_b \leftarrow c + d + T_{i+n}
\]

\[
[T \text{ is a multiple of } b^{i+1}]
\]

\[
[T + cb^{i+n+1} \text{ is congruent mod } N \text{ to the original } T]
\]

\[
[0 \leq T < (R + b')N]
\]

\[
\text{end for}
\]

\[
\text{if } (cT_{2n-1} \cdots T_n)_b \geq N \text{ then}
\]

\[
\text{return } (cT_{2n-1} \cdots T_n)_b - N
\]

\[
\text{else}
\]

\[
\text{return } (T_{2n-1} \cdots T_n)_b
\]

\[
\text{end if}
\]

Here variable \( c \) represents a delayed carry—it will always be 0 or 1.

3. Hardware Implementation. This algorithm is suitable for hardware or software.
A hardware implementation can use a variation of these ideas to overlap the
multiplication and reduction phases. Suppose \( R = 2^n \) and \( N \) is odd. Let \( x =
(x_{n-1}x_{n-2} \cdots x_0)_2 \), where each \( x_i \) is 0 or 1. Let \( 0 \leq y < N \). To compute
\( xyR^{-1} \mod N \), set \( S_0 = 0 \) and \( S_{i+1} \) to \((S_i + x_iy)/2 \) or \((S_i + x_iy + N)/2 \), whichever
is an integer, for \( i = 0, 1, 2, \ldots, n - 1 \). By induction, \( 2S_i \equiv (x_{i-1} \cdots x_0)y \mod N \)
and \( 0 \leq S_i < N + y < 2N \). Therefore \( xyR^{-1} \mod N \) is either \( S_n \) or \( S_n - N \).


