Montgomery Algorithm for Modular Multiplication

Professor Dr. D. J. Guan *

August 25, 2003

The Montgomery algorithm for modular multiplication is considered to be the fastest algorithm to compute $xy \bmod n$ in computers when the values of x, y, and n are large. In this lecture note, we shall describe the Montgomery algorithm for modular multiplication. This is one of the notes for the algorithms which my students have difficulty to fully understand.

Suppose we want to compute $xy \bmod n$ in a computer. We first choose a positive integer, r, greater than n and relative prime to n. The value of r is usually 2^m for some positive integer m. This is because multiplication, division and modulo by r can be done by shifting or logical operations in computers.

Since $\gcd(r, n) = 1$, there are two numbers r^{-1} and n' with $0 < r^{-1} < n$ and $0 < n' < r$, satisfying

$$rr^{-1} - nn' = 1.$$

The algorithm is based on the fact that the computation of $xr^{-1} \bmod n$ can be done very efficiently by the algorithm *reduce*.

function *reduce* (x)
begin
    $q := (x \bmod r)n' \bmod r$;
    $a := (x + qn)/r$;
    if $a > n$ **then** $a := a - n$;
    return a;
end

Figure 1: Algorithm for the computation of $a = xr^{-1} \bmod n$.

*Department of Computer Science, National Sun Yar-Sen Univeristy, Kaohsiung, Taiwan 80424 (guan@cse.nsysu.edu.tw).

The reason why the above algorithm works is explained as follows. First,

$$xr^{-1} = xrr^{-1}/r = x(nn' + 1)/r$$

Note that, for any integer l,

$$((xn' + lr)n + x)/r \bmod n = (xn'n + lrn + x)/r \bmod n = (xn'n + x)/r \bmod n$$

Therefore, instead of computing $q = xn'$, we can compute $q = xn' \bmod r$.

Now, suppose $0 \leq x < rn$. The value of $a = (x + qn)/r$ will be less than $2n$. Therefore, computing $a \bmod n$ can be done very easily by subtracting n from a if $a > n$.

Suppose that a number x, $0 \leq x \leq n$, is mapped to xr. It is easy to verify that

1. $x + y$ is mapped to $(x + y)r = xr + yr$,
2. xy is mapped to $(xy)r = (xr)(yr)r^{-1}$,

Therefore, if the numbers x and y are represented by xr and yr in a computer, then the multiplication algorithm is $reduce(xy)$. The addition algorithm is unchanged since $xr^{-1} + yr^{-1} \equiv zr^{-1} \bmod n$ if and only if $x + y \equiv z \bmod n$. The algorithms for subtraction, negation, equality test, inequality test, multiplication by an integer, and the greatest common divisor with n are also unchanged.

When the numbers X, Y, and N are large, we can apply the above algorithm to compute $XY \bmod N$ in an efficient way. Suppose that

$$X = \sum_{k=0}^{m-1} x_k \beta^k, \quad Y = \sum_{k=0}^{m-1} y_k \beta^k, \quad N = \sum_{k=0}^{m-1} n_k \beta^k,$$

and we want to compute $A = XY\beta^{-m}$. Let $A = \sum_{k=0}^{m-1} a_k \beta^k$. In the algorithm *montgomery*, we use the notation $(\alpha)_\beta^{-1}$ to denote the inverse of α in Z_β. The reason why the above algorithm works is explained as follows. The algorithm computes the answer A incrementally. At each step, x_k is used to do the computation. The computation is similar to the first algorithm. That is, compute $A = A + x_k Y + qN$ and then $A = A/\beta$.

The rationale behind this computation is to find a proper value of q so that, at the k-th iteration, the value of $A + x_k Y + qN$ is a multiple of β. and that this value is bounded by $(R + \beta^k)N$. As explained above, the value of q is

$$q = (A + x_k Y)N' \bmod \beta,$$

function *montgomery* (X, Y)
begin
    $A := 0$;
    for $k := 0$ **to** $m - 1$ **do begin**
        $q := (a_0 + x_k y_0)(\beta - n_0)^{-1}_{\beta} \bmod \beta$;
        $A := A + x_k Y + qN$;
        $A := A/\beta$;
    end;
    return A;
end

Figure 2: Algorithm for the computation of $A = XY\beta^{-m} \bmod N$.

where N' is the inverse of N in Z_{β^m}. Therefore,

$$q = (A + x_k Y)N' \bmod \beta = (a_0 + x_k y_0)(\beta - n_0)^{-1}_{\beta} \bmod \beta,$$

In the above equation, we use the fact that

$$RR^{-1} - NN' \equiv 1 \pmod{\beta^m}.$$

Thus, $-NN' \bmod \beta = 1$, which implies that

$$N' \bmod \beta = (-n)^{-1}_{\beta} = (\beta - n_0)^{-1}_{\beta}.$$

Suppose that $\beta = 2^k$ for some positive integer k, and that the value of $n_0 = \beta - 1$. Then the value of $(\beta - n_0)^{-1}_{\beta} = 1$. In this case, $q = (a_0 + x_k y_0) \bmod \beta$, which is the value of the last digit of $A + x_k Y$. Therefore, we can save the computation of the value of q at each iteration.

References

[1] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, April 1985.

[2] Jean-Claude Bajard, Laurent-Stephane Didier, and Peter Komerup. An RNS Montgomery modular multiplication algorithm. *IEEE Transaction on Computers*, 47(7):766–776, July 1998.