QuickMul: Practical Fast Integer Multiplication

Chee Yap

Jan, 1993. Updated: July, 2000

1 The One-Prime FFT Multiplication

We are interested in "practical" methods for exploiting modular FFT techniques in the multiplication of very large integers. At issue is to reduce the overhead in implementing such algorithms. Our method exploits the fact that real machines has non-negligible machine word sizes (say 32- or 64-bit).

In the following, we are given two N-bit positive integers U, V and we want to compute their produce $W = UV$. The method shown below works for N up to 250 Million bits on a 32-bit machine.

1.1 Computation Modulo $M = 2013265921$

We want to compute in $\mathbb{Z}_M = \{0, 1, \ldots, M-1\}$ where $M = 2,013,265,921$ is is a prime number that is less than 32 bits. We can express M as

$$M = 2^{27}m + 1$$

where $m = 15$. In the following, we write $a \equiv b$ to mean $a - b \equiv 0(\bmod M)$ and write $(a)_M$ for the value $(a \bmod M) \in \mathbb{Z}_M$. We claim that the number 31 is a primitive element of $\mathbb{Z}_M$ in the sense that $\{(31^i)_M : i = 0, 1, \ldots, M-2\} = \{1, 2, \ldots, M-1\}$. It is easy to verify this using Lucas' criterion: this amounts to checking that $31^{(M-1)/p} \neq 1 \bmod M)$ for $p = 2, 3, 5$. Setting

$$\omega := (31^m)_M = 440564289,$$

we easily check that ω is a 2^{27}-th primitive root of unity.

We use these facts to implement the DFT algorithm on vectors of length 2^{27} in $\mathbb{Z}_M$. To implement the FFT, we compute everything mod M. At the level of machine instructions, we can multiply $U, V \in \mathbb{Z}_M$ as to normal integers at double precision, and then reduce the result mod M. Alternatively, to avoid double precision machine arithmetic, we can split a number $U \bmod M$ into two parts (U_1, U_0), each part with at most 16 bits. Then we can multiply two such numbers with 3 single-precision multiplications (as in Karatsuba's method).

The implementation of FFT is then straightforward.

1.2 Fast Multiplication

Recall our simplified version of the Schöhage-Strassen integer multiplication algorithm. We break up an N-bit integer U into K integers each with L-bits, where

$$N = KL.$$

We view this as a $(2K)$-vector, after padding with zeros. The steps are:

1. Compute the $(2K)$-point DFT of vectors obtained from U and V.
2. Recursively multiply the above two DFT's, componentwise.
3. Compute the inverse DFT of the resulting vector. This gives us the convolution of the original vector.
4. Evaluate the polynomial corresponding to the convolution.

1.3 Non-Recursive Multiplication

Our goal is to carry out this algorithm with values of K and L which will allow all arithmetic operations to be done with 32-bit machine arithmetic. Also, we insist that step 2 is to be non-recursive: it must be carried out directly, with machine multiplications. Let us estimate the largest value of N which can be achieved this way.

Suppose we break up U and V into K pieces and perform steps 1-3 as outlined above. This gives us a $(2K)$-vector $\overline{W} = (W_{2K-1}, \ldots, W_0)$ which is the convolution of the input vectors $\overline{U}$ and $\overline{V}$. We must make sure that $\overline{W}$ has not lost any information because we computed in $\mathbb{Z}_M$. But each W_i is the sum of $\le 2K$ numbers, each with $\le 2L$ bits. Thus W_i has $\le 2L + \lg(2K)$ bits. So we need

$$2L + \lg(2K) \le \lg M.$$

Note that $\lg M = 30.9\ldots$. Clearly $N = KL$ is maximized by making K as large as possible. Let N_{max} be this maximum value. Letting $L = 1$, we have $\lg(2K) \le (\lg M) - 2$ or $K \le M/8$. Thus we have $N_{max} = \lfloor M/8 \rfloor = 251,658,240$. Thus the method works for integers up to 250 megabits or over 7.8 megabytes.

In general, if we are given two numbers with $N \le N_{max}$ bits, we choose positive K, L so as to maximize L (and hence minimize K) while subject to

$$KL \ge N, \qquad 2L + \lg(2K) \le \lg M.$$

So it is enough to choose

$$L = \left\lfloor (\lg\left(\frac{M}{2N}\right)) + \lg(L))/2 \right\rfloor, \qquad K = \lceil N/L \rceil.$$

For instance, with $N = 1,000,000$ then we choose $L = 6$ and $K = 166,667$. The following table gives the upper bound on N for any $L = 1, 2, \ldots, 8$.

L	Largest Possible N
8	122,880
7	430,080
6	1,474,560
5	4,915,200
3	47,185,920
2	125,829,120
1	251,658,240

How fast can we multiply two such numbers? Well, FFT on $2K$-vectors takes $1.5(2K)\lg(2K)$ arithmetic operations. If $N = N_{max}$, the number of double-precision machine multiplications is $3 \cdot (2.52 \times 10^8) \cdot 28.9$ which is less than 22 megaflops. Counting a factor of 10 for the overhead per flop, this computation takes 220 seconds on a megaflop machine.