FAST MULTIPLICATION OF INTEGER POLYNOMIALS

JOACHIM VON ZUR GATHEN

October 3, 2012

Abstract. Fürer's (2007) method is the currently fastest procedure for multiplying integers. In this paper, fairly standard algorithmic technology provides an adaption of this result to univariate polynomials with integer coefficients.

1. Introduction

The seminal paper of Schönhage & Strassen (1971) gave an FFT-based multiplication for n-bit integers using $O(n \log n \log \log n)$ bit operations. This record stood for a quarter of a century, until Fürer (2007, 2009) took a fresh look at the FFT and obtained an algorithm with $\mathsf{M}(n)$ operations, where

$$\mathsf{M}(n) \in n \log n \, 2^{O(\log^* n)}.$$

Here $\log^* n$ is the number of times one has to apply the function $\log_2$ to get below 1. For example, if

$$m = 2^{2^{2^{2^{2}}}} = 2^{65\,536},$$

then $\log^* m = 6$ and $\log^* n \leq 5$ for all $n < m$. A number with m bits requires storage of over $2.5 \cdot 10^{19\,709}$ exabytes. Thus for practical purposes, $\log^*$ is constant.

Fürer used complex roots of unity, and De *et al.* (2008) transferred the approach to a p-adic domain, thus obviating the need for complex error estimates.

It is easy to adapt this to multiplication in $\mathbb{Z}[x]$. One substitutes a sufficiently large integer b for x, so that in a product $h = f \cdot g \in \mathbb{Z}[x]$, the b-adic digits of $h(b)$ are separated and can be read off, to yield $h(x)$. The only problem is that the b-adic representation of $f(b)$ has, in general, both positive and negative b-adic digits. This can be circumvented by splitting f and g into their positive and negative parts and performing four integer multiplications (see von zur Gathen & Gerhard (2003), § 8.4).

The purpose of the present paper is to reduce this factor of 4 to 1. This is achieved by choosing b as a power of 2 and converting the positive/negative b-adic representation of $f(b)$ into one with only positive digits. We then have the usual binary representation of integers, apply a single fast integer multiplication, and convert back.

2. Conversion algorithms

We consider two b-adic representations of integers: the *standard representation* with a sign bit and digits v satisfying $0 \leq v < b$, and a *signed representation* with digits w for which $-b/2 \leq w < b/2$ holds. Since the latter digits form a complete residue system modulo the integer $b \geq 2$, the representation is unique. A digit w is represented as (sign(w), binary representation of $|w|$), and 0 as $(+, 0)$.

For d-digit representations with positive leading digits in an even base b, the maximal values in the two representations are

(2.1)

$$(b-1, \ldots, b-1) \longleftrightarrow b^d - 1 \geq \frac{(b-2)(b^d-1)}{2(b-1)} \longleftrightarrow (\frac{b}{2} - 1, \ldots, \frac{b}{2} - 1),$$

and the minimal values, with a positive standard sign, are

$$(1, 0, \ldots, 0) \longleftrightarrow b^{d-1} > \frac{b^d - 2b^{d-1} + b}{2(b-1)} \longleftrightarrow (1, -\frac{b}{2}, \ldots, -\frac{b}{2}).$$

Furthermore, for $b \geq 4$ we have

$$b^d - 1 < \frac{(b-2)(b^{d+1}-1)}{2(b-1)}.$$

It follows that for $b \geq 4$ any number with d standard digits has at most $d+1$ signed digits, and with d signed digits it has at most d standard digits. This slight asymmetry is due to the fact that having a positive leading digit is a restriction on the signed representation, but not on the standard one.

The two conversion algorithms that follow work for positive inputs.

ALGORITHM 2.2. Conversion from signed b-adic to standard b-adic.

Input: integers $b \geq 4$, $d \geq 1$, $u_0, \ldots, u_{d-1}$ with $u_{d-1} \geq 1$ and $-b/2 \leq u_i < b/2$ for all $i < d$.

Output: integers $v_0, \ldots, v_{d-1}$ with $0 \leq v_i < b$ for all $i < d$.

1. For $0 \leq i < d$ do $v_i \leftarrow u_i$.

2. For $i = 0, \ldots, d-1$ do
3. If $v_i < 0$ then $v_i \leftarrow v_i + b$, $v_{i+1} \leftarrow v_{i+1} - 1$ { carry step }.
4. Return $v_0, \ldots, v_{d-1}$.

ALGORITHM 2.3. Conversion from standard b-adic to signed b-adic.
Input: integers $b \geq 4$, $d \geq 1$, $v_0, \ldots, v_{d-1}$ with $0 \leq v_i < b$ for all $i < d$.
Output: $u_0, \ldots, u_d$ with $-b/2 \leq u_i < b/2$ for all $i \leq d$.

1. For $0 \leq i < d$ do $u_i \leftarrow v_i$.
2. $u_d \longleftarrow 0$.
3. For $i = 0, \ldots, d-1$ do
4. If $u_i \geq b/2$ then $u_i \leftarrow u_i - b$, $u_{i+1} \leftarrow u_{i+1} + 1$ { carry step }.
5. Output $u_0, \ldots, u_d$.

For symmetry, we set $u_d = 0$ in the input to Algorithm 2.2.

THEOREM 2.4. *Both conversion algorithms work correctly, that is,*

$$\sum_{0 \leq i \leq d} u_i b^i = \sum_{0 \leq i < d} v_i b^i.$$

If $b = 2^m$, for an integer $m \geq 2$, then each of them takes at most $3dm$ bit operations for the arithmetic.

PROOF. For the correctness of Algorithm 2.2, we have

$$\sum_{0 \leq i \leq d} u_i b^i = \sum_{0 \leq i < d} v_i b^i \tag{2.5}$$

after step 1. If v_i', v_{i+1}' denote the values after a carry step for i, we have

$$v_i' b^i + v_{i+1}' b^{i+1} = (v_i + b) b^i + (v_{i+1} - 1) b^{i+1} = v_i b^i + v_{i+1} b^{i+1}.$$

Thus the right-hand sum in (2.5) remains unchanged throughout the algorithm.

We next claim that $0 \leq v_i < b$ for all i. Each v_i gets changed at most twice in step 3, to $v_i + b$ for i or to $v_i - 1$ for $i - 1$. After step 4 for $i - 1$, we have

$$-b/2 - 1 \leq v_i < b/2,$$

and after step 4 for i,

$$0 \leq v_i < b,$$

as claimed. Since $u_{d-1} \geq 1$, we have $v_{d-1} \geq 0$ after step 4 for $d-2$, and the condition in step 4 for $d-1$ is not satisfied.

The proof for Algorithm 2.3 is similar. Denoting by u'_i, u'_{i+1} the values after a carry step for i, we have

$$u'_i b^i + u'_{i+1} b^{i+1} = (u_i - b)b^i + (u_{i+1} + 1)b^{i+1} = u_i b^i + u_{i+1} b^{i+1}.$$

In the condition of step 4 for i, we have $0 \leq v_i \leq u_i \leq v_i + 1 \leq b$. If at i no carry occurs, then $0 \leq u'_i = u_i < b/2$. If it is a carry step, then $u_i \geq b/2$ and

$$-b/2 = b/2 - b \leq u'_i = u_i - b \leq 0.$$

This shows all claims about the output.

For the claimed running time when $b = 2^m$, it is sufficient to realize addition or substraction of either 1 or b to a digit with m bit operations. In step 3 of Algorithm 2.2, we compute $v_i + b$ for some $v_i < 0$. Let $\overline{v_i}$ be the integer whose m-digit binary representation complements that of $|v_i|$. Then

$$\begin{aligned} |v_i| + \overline{v_i} &= 2^m - 1 = b - 1, \\ v_i + b &= -|v_i| + b = \overline{v_i} + 1, \end{aligned}$$

so that a complementation and addition of 1 suffice. Similarly, in step 4 of Algorithm 2.3 we need $u_i - b$ for some $u_i \geq b/2$. Using its complement $\overline{u}_i$, we have

$$u_i - b = -(\overline{u_i} + 1).$$

There are standard techniques to handle the potential carries in adding or subtracting 1 to or from an m-bit number in m bit operations when we count bit addition with carry as one operation. We do not go into the details. One carry step can be done with at most $3m$ bit operations, and the whole conversion with at most $3dm$ operations. $\square$

ALGORITHM 2.6. Reducing multiplication in $\mathbb{Z}[x]$ to that in $\mathbb{Z}$.

Input: $f = \sum_{0 \leq i < d} f_i x^i$ and $g = \sum_{0 \leq i < d} g_j x^j$ in $\mathbb{Z}[x]$, with $d \geq 2$ and all integer coefficients in standard binary representation.

Output: The coefficients of $h = f \cdot g = \sum_{0 \leq k \leq 2d-2} h_k x^k$ in standard binary representation.

1. If $f = 0$ or $g = 0$, then return 0.
2. $B \longleftarrow \max\{|f_i|, |g_i| \colon 0 \leq i < d\}$.
3. $m \longleftarrow \lceil \log_2(dB^2) \rceil + 2$, $b \longleftarrow 2^m$.

4. Let $f_p = \varepsilon_f |f_p|$ and $g_q = \varepsilon_g |g_q|$ be the highest nonzero coefficient of f and g, respectively, with $0 \le p, q < d$ and $\varepsilon_f, \varepsilon_g \in \pm 1$.
5. For $0 \le i < d$ do $f_i^* \longleftarrow \varepsilon_f f_i$, $g_i^* \longleftarrow \varepsilon_g g_i$.
6. $u_f \longleftarrow (f_{d-1}^*, \ldots, f_0^*)$, $u_g \longleftarrow (g_{d-1}^*, \ldots, g_0^*)$.
7. $v_f \longleftarrow$ conversion(u_f), $v_g \longleftarrow$ conversion(u_g), using Algorithm 2.2.
8. $v_h \longleftarrow v_f \cdot v_g$ in standard b-adic, using fast integer multiplication.
9. $u_h \longleftarrow$ conversion(v_h) in signed b-adic, using Algorithm 2.3. Parse u_h as $(h_{2d-2}^*, \ldots, h_0^*)$, with each h_k^* a signed b-adic digit.
10. Return $h_k = \varepsilon_f \varepsilon_g h_k^*$ for $0 \le k \le 2d-2$.

The *height* $H(f)$ of an integer polynomial f is the maximal absolute value of its coefficients.

THEOREM 2.7. *Algorithm 2.6 correctly multiplies two polynomials in $\mathbb{Z}[x]$ of degree less than $d \ge 3$ and height at most B. It takes at most $\mathsf{M}(n) + 13n$ bit operations, where $m = \lceil \log_2(dB^2) \rceil + 2$ and $n = dm$.*

PROOF. We set $f^* = \varepsilon_f f$ and $g^* = \varepsilon_g g$, so that the f_i^* and g_i^* are the coefficients of f^* and g^*, respectively. We first note that $m \ge 4$, $4d \le n$, and $2d \le 2dB^2 \le b - 1 \le (d-1)b$, so that

$$(b^d - 1)^2 = b^{2d} - 2b^d + 1 \le db^{2d} - 2db^{2d-1} - bd + 2d = d(b-2)(b^{2d-1} - 1),$$

$$|(f^* \cdot g^*)(b)| \le B^2 \left(\frac{b^d - 1}{b - 1} \right)^2 \le \frac{(b-2)(b^{2d-1} - 1)}{2(b-1)}.$$

It follows from (2.1) that the signed b-adic representation of $(f^* \cdot g^*)(b)$ has at most $2d-1$ digits. Furthermore, we let $w_k = \sum_{i+j=k} f_i^* g_j^*$ for $0 \le k \le 2d-2$. The integers represented by u_f and v_f are equal to $f^*(b)$, and u_g and v_g represent $g^*(b)$. Thus u_h is the signed b-adic representation of $f^*(b) \cdot g^*(b) = (f^* \cdot g^*)(b)$. It follows that

$$\sum_{0 \le k \le 2d-2} h_k^* b^k = (f \cdot g)(b) = \sum_{0 \le k \le 2d-2} w_k b^k, \tag{2.8}$$

$$|w_k| = |\sum_{i+j=k} f_i^* g_j^*| = |\varepsilon_f \varepsilon_g| \cdot |\sum_{i+j=k} f_i g_j| \le d \cdot B^2 < b/2.$$

Thus each w_k is a signed b-adic digit, so that both sums in (2.8) are signed b-adic representations of $(f \cdot g)(b)$. Since this representation is unique, we have $h_k^* = w_k$ for all k, and the output is correct.

Arithmetic computation takes place in steps 5 through 10; we ignore the other steps. The most expensive step is the integer multiplication in step 8.

Its first argument $f^*(b)$, represented as v_f, satisfies

$$0 \le f^*(b) \le B\frac{b^d - 1}{b - 1} < 2Bb^{d-1}.$$

Setting

$$c = \lfloor \log_2 B \rfloor + 2, \tag{2.9}$$

we have

$$\log_2(2Bb^{d-1}) \le 1 + c - 1 + (d-1)m \le dm = n,$$

since $c \le m$. The same bound holds for $g^*(b)$, and step 8 can be perfomed with $\mathsf{M}(n)$ operations. All other steps take at most the following number of bit operations:

step 5: $2d$,

step 6: 0,

step 7: $6n$,

step 8: $\mathsf{M}(n)$,

step 9: $6n$,

step 10: $1 + 2d - 1 = 2d$. □

The range of practicality of fast integer multiplication has yet to be determined, and no attempt has been made to optimize the algorithms presented here (except for the constant 1 in $1 \cdot \mathsf{M}(n)$ of Theorem 2.7). Each coefficient of f or g can be represented with c bits (as in (2.9)), for a total of $2cd$ bits. We have

$$n = dm \approx d(2c + \log_2 d) = 2cd + d\log_2 d.$$

For each pair (c, d) of parameters, the algorithm can be implemented on a Boolean circuit of the stated size. Besides the usual gates, we use an addition gate whose 2-bit output is the integer sum of the two input bits.

Well-known algorithms of Borodin & Moenck (1974); Cook (1966); Kung (1974); Sieveking (1972); Strassen (1973), and others perform division with remainder using Newton iteration. For n-bit integers, they yield algorithms using $O(\mathsf{M}(\mathsf{n}))$ bit operations (see von zur Gathen & Gerhard (2003), Theorem 9.8). This leads to fast multiplication in finite fields and for polynomials over finite fields.

3. Acknowledgement

This work was supported by the B-IT Foundation and the Land Nordrhein-Westfalen.

References

A. Borodin & R. Moenck (1974). Fast Modular Transforms. *Journal of Computer and System Sciences* **8**(3), 366–386.

S. A. Cook (1966). *On the minimum computation time of functions*. Doctoral Thesis, Harvard University, Cambridge MA.

Anindya De, Piyush Kurur, Chandan Saha & Ramprasad Saptharishi (2008). Fast Integer Multiplications Using Modular Arithmetic. In *Proceedings of the Fourtieth Annual ACM Symposium on Theory of Computing,* Victoria, BC, Canada, 499–505. ACM Press.

Martin Fürer (2007). Fast Integer Multiplication. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing,* San Diego, California, USA, 57–66. ACM. URL `http://dx.doi.org/10.1145/1250790.1250800`. Preprint available at `http://www.cse.psu.edu/~furer/Papers/mult.pdf`.

Martin Fürer (2009). Faster Integer Multiplication. *SIAM Journal on Computing* **39**(3), 979–1005. URL `http://dx.doi.org/10.1137/070711761`.

Joachim von zur Gathen & Jürgen Gerhard (2003). *Modern Computer Algebra*. Cambridge University Press, Cambridge, UK, Second edition. ISBN 0-521-82646-2, 800 pages. URL `http://cosec.bit.uni-bonn.de/science/mca/`. Other available editions: first edition 1999, Chinese edition, Japanese translation.

H. T. Kung (1974). On Computing Reciprocals of Power Series. *Numerische Mathematik* **22**, 341–348.

A. Schönhage & V. Strassen (1971). Schnelle Multiplikation großer Zahlen. *Computing* **7**, 281–292.

M. Sieveking (1972). An Algorithm for Division of Powerseries. *Computing* **10**, 153–156.

Volker Strassen (1973). Vermeidung von Divisionen. *Journal für die reine und angewandte Mathematik* **264**, 182–202.

Joachim von zur Gathen
B-IT
Universität Bonn
D-53012 Bonn
gathen@bit.uni-bonn.de
http://cosec.bit.uni-bonn.de/