Literature Review: Adaptive Polynomial Multiplication

Daniel S. Roche

November 27, 2007

While output-sensitive algorithms have gained a fair amount of popularity in the computer algebra community, adaptive algorithms whose difficulty measure is something more subtle than the size of the input or output seem to be less common. Rather, different representations of structures (e.g. polynomials, matrices) are studied which may take advantage of some difficulty measure.

In this spirit, I plan to investigate adaptive algorithms for perhaps the most classical and well-studied area in computer algebra, polynomial multiplication. This is mostly an attempt to unify and provide a finer gradient between the two commonly-used polynomial representations and multiplication algorithms. As such, I will first present the commonly-used model and assumptions, then discuss various techniques and progress on multiplying dense and sparse polynomials, and finally present an overview of some ideas for an adaptive approach.

1 Polynomial Representation

Let R be an arbitrary ring (commutative with identity), n a natural number, and $f(x) \in \mathsf{R}[x]$ a polynomial of degree $n-1$. To be as general as possible, we will count ring operations $(+, -, \times)$ for time complexity and ring elements for space complexity. This model is more accurate for rings with finite characteristic, such as the integers mod a prime, rather than rings of characteristic 0 such as the integers or the complex numbers.

There are two obvious (and of course related) ways to write down $f(x)$:

$$\begin{aligned} f(x) &= f_0 + f_1 x + f_2 x^2 + \cdots + f_{n-1} x^{n-1} \\ &= a_1 x^{e_1} + a_2 x^{e_2} + \cdots + a_t x^{e_t}, \end{aligned}$$

where $a_1, \ldots, a_t \neq 0$ and $e_1 < e_2 < \cdots < e_t = n-1$.

The first equation corresponds to the *dense representation* by a vector of coefficients of length n: $\langle f_0, f_1, \ldots, f_{n-1} \rangle$, size $\Theta(n)$.

The second equation corresponds to *sparse representation* by a vector of nonzero coefficient-exponent pairs: $\langle (a_1, e_1), (a_2, e_2), \ldots, (a_t, e_t) \rangle$, size $\Omega(\log n + t \log t)$, $O(t \log n)$. We call t the "sparsity" of f.

For the remainder, let $g(x) \in \mathsf{R}[x]$ of degree $m-1$, such that $g(x) = \sum_{i=0}^{m-1} g_i x^i = \sum_{i=1}^{s} b_i x^{d_i}$ as before. And let $h(x) = f(x)g(x)$ with sparsity r.

2 Dense Multiplication

von zur Gathen and Gerhard point out in [14] the similarities between dense polynomial multiplication and comparison-model sorting. For both, the naïve or "school" method has $O(n^2)$ complexity, whereas asymptotically fast algorithms can improve this to $O(n \log n)$ (or close to it). Of course, this is also part of the motivation to investigate adaptive algorithms for this problem.

Despite the similarities, polynomial multiplication does seem to be a little more difficult than sorting. The asymptotically fast algorithms are significantly more complicated than, say, Mergesort, and a lower bound of $\Omega(n \log n)$ for an arbitrary ring is still not known despite numerous attempts. However, in a

rather restricted model of an algebraic circuit over $\mathbb{C}$ with bounded coefficients, it has been proven that $\Omega(n \log n)$ ring operations are needed [2].

To easier explain the algorithms, we will initially assume that $f(x)$ and $g(x)$ have the same degree — that is, $\deg f = \deg g = n - 1$.

2.1 Karatsuba's Method [8]

This is a divide-and-conquer technique, based on the fact that addition is fast. It is very similar to to Strassen's matrix multiplication algorithm (which would come a few years later), and was the first algorithm to break the $O(n^2)$ barrier.

The basic idea is to split each of the input polynomials into halves, and then use some clever manipulation and the fact that dense polynomial addition is cheap (i.e. linear time) to gain a slight improvement.

Let $k = \lfloor \frac{n}{2} \rfloor$ and write $f(x) = F_1(x) + F_2(x)x^k$, $g(x) = G_1(x) + G_2(x)x^k$ (where F_1, F_2, G_1, G_2 all have degree less than $\lceil \frac{n}{2} \rceil$). Then we can write

$$h = F_1G_1 + F_2G_2x^{2k} + \\ + \left[(F_1 + F_2)(G_1 + G_2) - F_1G_1 - F_2G_2\right] x^k$$

So multiplication of two polynomials with degree less than n is reduced to three multiplications of polynomials with degree less than $\lceil \frac{n}{2} \rceil$ (and six additions/subtractions). Solving the recurrence gives asymptotic cost $O(n^{\log_2 3}) = O(n^{1.59})$.

2.2 Fast Fourier Transform Method

This is the asymptotically fastest method currently known for dense multiplication. It was first proposed as an integer multiplication method by Schönhage and Strassen in [13]. The algorithm was first extended to work on polynomials over some rings [12], and then to arbitrary rings [3]. The approach is still divide-and-conquer, and is based on a change of representation.

The Discrete Fourier Transform (DFT) is a mathematical mapping from $\mathsf{R}[x] \to \mathsf{R}^n$, given a primitive n^{th} root of unity $\omega \in \mathsf{R}$, which is defined by

$$\text{DFT}_\omega(f) = \left(f(1), f(\omega), f(\omega^2), \ldots, f(\omega^{n-1})\right).$$

That is, the DFT maps a polynomial of degree less than n to the values of that polynomial at the n points $1, \omega, \omega^2, \ldots, \omega^{n-1}$. We know that this map is also invertible since there is exactly one polynomial of degree less than n which will pass through the given evaluation points (all the ω^i's are distinct because ω is an n^{th} root of unity). And in fact, the inverse transform can be performed with another DFT by dividing all values by n and then applying the DFT with primitive n^{th} root ω^{-1}.

The FFT is a famous divide-and-conquer algorithm which can perform the DFT with just $O(n \log n)$ ring operations [4]. The only trouble is if our base ring R does not contain a primitive n^{th} root of unity, but in this case we can create and adjoin a "virtual" root for an extra $O(\log \log n)$ cost per ring operation.

Then, since $h(\omega^i) = f(\omega^i)g(\omega^i)$, we can compute the DFT of the result $h(x)$ via pointwise multiplication of the vectors $\text{DFT}_\omega(f)$ and $\text{DFT}_\omega(g)$. Finally, we convert back to the dense representation of $h(x)$ via an inverse DFT.

So each conversion step costs $O(n \log n \log \log n)$ and the multiplication step (not the dominating step) is $O(n)$. Therefore the total worst-case cost using this method is $O(n \log n \log \log n)$.

2.3 "Multiplication Time" and other operations

A convenent unifying notation is a "multiplication time" of dense polynomials. This is a function $\mathsf{M}(n)$, defined as the time to multiply two dense polynomials of degree less than n. We will assume that $\mathsf{M}(n)$ is $\Omega(n \log n)$ and $O(n^2)$.

Suppose $\deg g = m < n$. Then we can partition the coefficients of f into n/m polynomials each of degree less than m, multiply each of these by g, and sum them, to obtain complexity $O(\frac{n}{m}\mathsf{M}(m))$ for computing $f(x) \cdot g(x)$.

After asymptotically fast methods for multiplying polynomials had been discovered, much effort was put into reducing other polynomial problems to multiplication. Some examples of such reductions are summarized in the following table:

Operation	Complexity
Euclidian Division: $f = qg + r$	$O(\mathsf{M}(n))$
Evaluation/Interpolation	$O(\mathsf{M}(n)\log n)$
$\gcd(f(x), g(x))$	$O(\mathsf{M}(n)\log n)$
Change of basis: $f(x) = \sum c_i g^i$	$O(\mathsf{M}(n)\log \frac{n}{m})$

3 Sparse Multiplication

We now turn to sparse multiplication algorithms. These have a shorter history, and results are less spectacular, no doubt due to the fact that the worst-case output size is quadratic in the size of the input (as opposed to linear for the dense case).

3.1 Standard Algorithms

Recall that the sparsities of f, g, h are t, s, r, respectively, and assume $s \leq t$. Notice that each pair of terms from f and g could form a distinct term in the product h (if all the exponent sums $e_i + d_j$ are distinct). So r could be as large as ts — this is why the worst-case output size could be quadratic for sparse polynomials.

All the following methods use $\Theta(ts)$ ring operations and thus are worst-case optimal in that respect.

School method Compute $b_i \cdot g$ for $1 \leq i \leq t$, and merge each product into the result, one by one. Successive merges cost $t, 2t, 3t, \ldots, st$, and each comparison costs $O(\log n)$ word operations (we must account for this because the exponents of sparse polynomials can be multiple-precision integers). This gives a total cost of $O(s^2 t \log n)$ word operations and $O(st)$ space.

Geobuckets [15] Instead of merging the $b_i g$'s one-by-one, recursively pairwise-merge them in a manner similar to Mergesort so that each merge is between two polynomials with roughly the same number of terms (in the worst case). This reduces the total cost to $O(st \log s \log n)$ word operations and $O(st)$ space.

Heaps [6, 10] Some brand-new results build on an older, forgotten paper from the 1970's. Here, the result is stored dynamically in an array of heaps so that each term in the product is not computed until needeed. This therefore gives an output-sensitive measure on the space complexity. Using a number of little tricks, the complexity becomes $O(st \log s \log n)$ word operations (again), but just $O(t + r)$ space.

3.2 Output Sensitivity

If $r \ll st$, the heaps algorithm gives a significant reduction in space usage. In order to also reduce the time complexity in terms of ring and word operations, we can use a result from [1, 7] which gives an algorithm for *sparse interpolation*. This is similar to the DFT algorithm above for dense polynomials, with the differences being that we only need $\Theta(r)$ evaluation points to recover h, and we never have to create any "virtual" roots. Unfortunately, this method is much slower, and uses $O(r^{1.87} + rt \log n)$ ring *and* word operations. And even this is a bit of a cheat, since it requires a fast logarithm operation in the base field, which we know for many fields (e.g. integers mod a large prime) is in fact intractable.

3.3 Uses and Results

The sparse representation is especially useful in dealing with multivariate polynomials, as the total number of terms (i.e. dense size) grows exponentially with the number of indeterminates. Sparse univariate polynomials are often called "supersparse" or "lacunary" in the literature (these two names are actually a bit of a battle between Erich Kaltofen and H. W. Lenstra, Jr.).

Note that the size of the sparse representation can be exponentially smaller than the dense size. This raises important and interesting questions regarding the complexity of operations with sparse polynomials. For example, factoring and computing GCD are provably NP-hard [11], but interpolation and root-finding can be done in polynomial time [1, 5, 9]. The attempt to completely partition sparse polynomial operations in this way is a significant area of current active research, and many important questions remain open, such as a divisibility test for sparse polynomials.

4 Adaptive Multiplication

The goal is an algorithm (or algorithms) which adapts to the difficulty of the input polynomials, but which is still never worse than the asymptotically best methods for dense *or* sparse multiplication.

The basic approach will be first to define carefully a standard representation form that captures more finely the difficulty of the input and always guarantees performance at least as good as the standard sparse and dense multiplication methods. Multiplying in this new representation will likely be relatively simple and use existing tools; the challenge will be to construct linear-time algorithms to convert between the new representation and the existing dense and sparse ones to give a complete and useful adaptive algorithm.

The basic premise for most of this is that the complexity of dense methods grows much more slowly (superlinear vs. quadratic) than sparse ones, so combining dense methods with sparse representations can be advantageous. Some ideas for the change of representation:

4.1 Dense "chunks"

Suppose the input polynomials have few nonzero terms, and that most of those terms seem to be bunched together into clusters, or "chunks". Then we might represent the polynomial similarly to the sparse representation, but where each nonzero coefficient is instead a nonzero dense polynomial.

Then multiplication would follow the scheme of standard sparse multiplication on the outside to combine terms, and the pairwise products of terms could be computed via fast dense methods.

If f and g are each written as the sum of t dense polynomials of degree less than d, then the complexity of multiplication is reduced to $O(t^2\mathsf{M}(d)\log n)$ vs. $O(\mathsf{M}(n))$ for dense or $O(t^2d^2\log n)$ for sparse.

4.2 Equal Spacing

Suppose all the e_i's (the powers of the nonzero terms of f) are divisible by an integer $k > 1$. So we can write $f(x) = F(x^k)$, where F is a dense polynomial of degree $\frac{n}{k}$. Similarly, write $g(x) = G(x^l)$.

Then multiplication can be performed in time

$$O\left(\frac{kl}{\gcd(k,l)^2}\mathsf{M}\left(\frac{n}{\operatorname{lcm}(k,l)}\right)\right),$$

compared to $O(\mathsf{M}(n))$ for dense or $O(n^2/(kl)\log n)$ for sparse.

4.3 Coefficients in Sequence

Perhaps $f(x)$ does not have many coefficients which are 0, but most of its coefficients form some arithmetic or geometric sequence $(\alpha_i)_{i\geq 0}$. Then we can write $f(x) = \sum_{i=0}^{n}\alpha_i x^i + \hat{f}(x)$, where $\hat{f}(x)$ is sparse with (say) t' nonzero terms.

To multiply by another polynomial $g(x)$, first compute $g(x)\sum_{i=0}^{n}\alpha_i x^i$ in *linear time*, then add the result to $\hat{f}(x)g(x)$ using whatever method is most appropriate.

4.4 Combination of ideas

Using a combination of the above ideas can result in a multiplication time that is an order of magnitude faster than the asymptotically dense *and* sparse methods for many classes of input.

For example, suppose $\deg g = \deg f = n = k^3$ for some positive integer k, and suppose f and g both have roughly the same form: they can be written as a sum of approximately $\log k$ polynomials of degree k^2, each of which has coefficients all equal to each other and whose powers are all divisible by k. So we can write

$$f(x) = \sum_{i=1}^{\log k} F_i x^{e_i}, \quad F_i = \alpha(\sum_{j=0}^{k} x^j) \circ (x^k).$$

Dense multiplication will cost $\mathsf{M}(k^3)$, or $\Omega(k^3\log k)$. Sparse multiplication will cost $\Omega(k^2(\log k)^3)$. But the adaptive multiplication will cost just $O(k(\log k)^3)$.

References

[1] Michael Ben-Or and Prasoon Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 301–309, New York, NY, USA, 1988. ACM Press.

[2] Peter Bürgisser and Martin Lotz. Lower bounds on the bounded coefficient complexity of bilinear maps. *J. ACM*, 51(3):464–482 (electronic), 2004.

[3] David G. Cantor and Erich Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Inform.*, 28(7):693–701, 1991.

[4] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comp.*, 19:297–301, 1965.

[5] Felipe Cucker, Pascal Koiran, and Steve Smale. A polynomial time algorithm for Diophantine equations in one variable. *J. Symbolic Comput.*, 27(1):21–29, 1999.

[6] Stephen C. Johnson. Sparse polynomial arithmetic. *SIGSAM Bull.*, 8(3):63–71, 1974.

[7] Erich Kaltofen and Wen-shin Lee. Early termination in sparse interpolation algorithms. *J. Symbolic Comput.*, 36(3-4):365–400, 2003. International Symposium on Symbolic and Algebraic Computation (ISSAC'2002) (Lille).

[8] A. Karatsuba and Yu. Ofman. Multiplication of multidigit numbers on automata. *Dokl. Akad. Nauk SSSR*, 7:595–596, 1963.

[9] H. W. Lenstra, Jr. Finding small degree factors of lacunary polynomials. In *Number theory in progress, Vol. 1 (Zakopane-Kościelisko, 1997)*, pages 267–276. de Gruyter, Berlin, 1999.

[10] Michael Monagan and Roman Pearce. Polynomial division using dynamic arrays, heaps, and packed exponent vectors. Preprint; accepted to CASC 2007.

[11] David A. Plaisted. New NP-hard and NP-complete polynomial and integer divisibility problems. *Theoret. Comput. Sci.*, 31(1-2):125–138, 1984.

[12] A. Schönhage. Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2. *Acta Informat.*, 7(4):395–398, 1976/77.

[13] A. Schönhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. *Computing (Arch. Elektron. Rechnen)*, 7:281–292, 1971.

[14] Joachim von zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge University Press, Cambridge, second edition, 2003.

[15] Thomas Yan. The geobucket data structure for polynomials. *J. Symbolic Comput.*, 25(3):285–293, 1998.