Proposal: Algorithms for Multivariate Polynomials

1 Research Area

The primary facility of Computer Algebra Systems like Maple, Magma, Mathematica, and Singular, is their ability to do polynomial arithmetic and polynomial algebra over various rings and fields. Many of the other facilities build on these core capabilities. I propose to work on problems in this area, including

(i) sparse polynomial multiplication and division,

(ii) polynomial GCD computation over algebraic function fields and quotient rings, and

(iii) polynomial factorization over algebraic number and function fields.

For (i), our sequential algorithms in [15] that use heaps to multiply and divide polynomials with *integer* coefficients are very good. They out perform Magma, Singular, and Maple by factors of 10 to 200. It is perhaps surprising that such improvements are still possible for core arithmetic operations. The challenge now, however, is to design *parallel* algorithms that are effective in practice.

For (ii) we have developed a first sparse modular GCD algorithm in [9]. It is a Las Vegas algorithm. Although it works well in practice, we do not yet have a complete analysis for the failure probability. There is also room for improvement in the sparse interpolation. And we'd like to generalize the algorithm to compute GCDs over rings with zero divisors.

Project (iii) is a new project for us. Our algorithm for (ii) improves considerably the effectiveness of the Trager-Kronecker algorithm (see Ch. 8 of [4]). But the Trager-Kronecker algorithm introduces a severe expression swell for multivariate polynomials. We propose to design and implement new Hensel lifting based methods.

2 Background, Literature Review and Recent Progress

2.1 Polynomial multiplication and division.

Let $f = f_1 + f_2 + ... + f_m$ and $g = g_1 + g_2 + ... + g_n$ be two sparse polynomials with $m = \#f$ terms and $n = \#g$ terms. If we order the terms in a monomial ordering so that $f_1 > f_2 > ... > f_m$ and $g_1 > g_2 > ... > g_n$, and multiply $f \times g$ by computing

$$(...((g_1 \times f + g_2 \times f) + g_3 \times f) + ...) + g_n \times f$$

then we can use merging to add polynomials. This algorithm does $O(mn)$ coefficient multiplications but may do as many as $O(mn^2)$ monomial comparisons. Similarly, if we divide h by f to get a quotient g and remainder r using

$$r = (...((h - g_1 f) - g_2 f) - ...) - g_n f$$

then we again can use merging to subtract polynomials. This too can do as many as $O(mn^2)$ monomial comparisons. An example where this bad behaviour occurs is $f = x + x^2 + ... + x^m$ and $g = y + y^2 + ... + y^n$.

Three approaches studied in the literature include Horowitz's binary trees [8], Yan's "geobuckets" [22], and Johnson's binary heaps [10]. For polynomial multiplication all do $O(\#f\#g \min\log(\#f,\#g))$ monomial comparisons. For division, Yan's and Horowitz's algorithms have the same complexity but Johnson's algorithm is $O(\#f\#g \log \#g)$. All assume rational arithmetic.

Maple uses a divide and conquer algorithm to multiply in $O(mn\log(mn))$ monomial comparisons. Magma uses a hash table, hashing on the monomials, to multiply using $O(mn)$ hashes. It then does $O(mn\log(mn))$ monomial comparisons to sort the result. In [5], Fateman reports that hashing does not work well for large polynomials because of the large number of random memory accesses.

In [15] we show how to divide over the integers using a heap in $O(\#f\#g \log\min(\#f,\#g))$ monomial comparisons and $O(\#f\#g)$ integer operations using a fraction-free algorithm. In practice, the heap algorithms perform very well. They have a small working set of memory – they need $O(\min(m,n))$ storage for the heap and they access the terms of f and g sequentially and write out the terms of $f \times g$ sequentially. Thus they have good locality. Another nice property is that one can re-use working storage for the coefficient arithmetic so that no garbage is created, thus eliminating the need for garbage collection.

Less work has been done on parallel algorithms. For polynomials $f = a_0+a_1x+...+a_mx^m$ and $g = b_0 + b_1x + ... + b_nx^n$, to multiply $f \times g$, Wang in [20] used

$$\sum_{k=0}^{m+n} C_k x^k \quad \text{where} \quad C_k = \sum_{i+j=k} a_i b_j.$$

Here, the C_k can be computed *independently* in parallel. This approach can also be applied to multivariate $f(x,y,z,...)$ and $g(x,y,z,...)$. Just represent them recursively as polynomials in x with coefficients a_i and b_j polynomials in the other variables. Even if f and g are sparse, their univariate representation in x is likely to be dense.

If the polynomials are dense then FFT based methods are likely to be much faster. Xavier and Iyengar's text on parallel algorithms describes how to use an FFT to multiply in parallel and Bini and Pan in [1] give an FFT based parallel division algorithm. In [13], Maza et. al. implemented multiplication modulo a (dense) triangular set modulo a prime using the FFT. Although many parallel experiments have been done, with the possible exception of MuPad, none of the computer algebra systems, as far we know, exploit parallelism in their polynomial multiplication and division routines.

2.2 Polynomial GCD computation.

If we are simplifying ratios of univariate polynomials over a field, GCDs exist and can be computed by the monic Euclidean algorithm and cancelled out. It is well known that for multivariate polynomials, an intermediate expression swell occurs that makes the Euclidean algorithm ineffective for polynomials of even modest degree. In the last 40 years, efficient algorithms for computing multivariate polynomial GCDs over $\mathbf{Z}$ have been developed.

The first was the modular method of Brown [2]. We mention also Moses and Yun's EZGCD algorithm [16], Zippel's sparse modular GCD algorithm [23, 24], the heuristic algo-

rithm of Char, Geddes and Gonnet [4], and Kaltofen's black box algorithm [3], each of which introduced a new technique for GCD computation over **Z**.

Some work on parallel implementations of Zippel's algorithm has been done. We mention the work of Rayes, Wang and Weber in [21] and Murao and Fujise in [17].

In [11], as one step in the process of solving a system of polynomial equations, Lazard describes an algorithm which needs to compute polynomial GCDs modulo triangular sets. Let $P = \mathbf{Q}(t_1, ..., t_k)[y_1, ..., y_n]$ and $I = \langle f_1, ..., f_n \rangle \subset P$ be a zero-dimensional ideal in P. If $f_i \in P[y_1, ..., y_i]$ with $\deg_{x_i}(f_i) > 0$, then the set $\{f_1, ..., f_n\}$ is called a triangular set. Lazard's algorithm needs to compute GCDs of univariate polynomials A and B over the quotient ring $R = P/I$ where it will often be the case that R has zero divisors, i.e., I is not prime.

In [14], Moreno-Maza and Rioboo show how to do this. However, their algorithm is non-modular and thus results in an expression swell. In [6] we developed a modular algorithm for the case $k = 0$, I maximal, i.e., R is an algebraic number field with multiple extensions. Subsequently, we modified our algorithm to treat zero divisors: we reconstruct and output either the gcd of the input polynomials or a zero divisor – a factor of some $f_i(y_i)$.

In [7] we developed and implemented a *dense* multivariate modular GCD algorithm for the case $k \geq 0$, $n = 1$, I maximal, i.e., R is an algebraic function field in one field extension. In [9] we developed a *sparse* multivariate modular GCD algorithm for the case $k \geq 0$, $n \geq 0$, I maximal, i.e., R is an algebraic function field with multiple field extensions. What remains to be done is a treatment of zero-divisors in R for the case $k > 0$.

2.3 Polynomial factorization.

The literature includes several algorithms for factoring polynomials over algebraic number fields, including those of Trager (see [4]), Wang [19], and Lenstra [12]. Of these, Trager's algorithm generalizes immediately to function fields so we outline the major steps of his algorithm here for later reference.

Let $F = \mathbf{Q}(t_1, ..., t_k)$. Let $m(z)$ be an irreducible polynomial in $F[z]$ and $L = F[z]/(m)$. So L is an algebraic function field in k parameters $t_1, ..., t_k$. Given a polynomial $f(x) \in L[x]$, the Trager-Kronecker algorithm factors $f(x)$ into irreducible factors over L using a reduction to a factorization problem in $F[x]$ for which we already have effective algorithms. Let $||f||$ denote the norm of $f(x)$. The norm can be computed using $||f|| = \mathrm{res}_z(f(z), m(z)) \in F[x]$, that is, by computing a polynomial resultant. If f factors as $f = f_1 f_2 ... f_k$ over L, Trager's algorithm uses the fact that

(i) $||f|| = ||f_1|| \times ||f_2|| \times ... \times ||f_k||$, and

(ii) f_i is irreducible in $L[x] \iff ||f_i||$ is irreducible in $F[x]$.

It computes and factors $||f||$ and obtains f_i using $f_i = \gcd(f, ||f_i||)$, a GCD computation in $L[x]$. Note, it can happen that $||f_i|| = ||f_j||$. In which case a workaround is required. This algorithm is used by many (all?) computer algebra systems for factoring polynomials over function fields. Applying our GCD algorithm from [9] to compute $\gcd(f, ||f_i||)$ in $L[x]$ greatly improves the effectiveness of this algorithm.

3 Research Proposal

3.1 Polynomial multiplication and division.

We have seen that the complexity of several algorithms for multiplying polynomials is $O(\#f\#g \log \min(\#f, \#g))$ monomial comparisons. I believe it is still open as to whether this is optimal.

We want to extend our multiplication and fraction-free division algorithms in [15] to be able to multiply and divide modulo a triangular set. This would give us polynomial multiplication and division over algebraic number fields and function fields as special cases – cases of practical importance.

We wish to develop parallel algorithms for sparse polynomial multiplication and division. Let $f = f_1X_1 + f_2X_2 + ... + f_nX_n$ and $g = g_1Y_1 + ... + g_mY_m$ be sparse polynomials with terms sorted in a monomial ordering so that i.e, $X_1 > X_2 > ...$ and $Y_1 > Y_2 > ....$ Presently, one of my students is looking at the following approaches to multiply $f \times g$ in parallel.

Let T be a hash table indexed by monomials with entries initialized to 0. For each (i, j) in parallel, hash X_iY_j and add f_ig_j to $T[X_iY_j]$. Next, sort the terms T into the monomial ordering using a parallel sorting algorithm. This assumes a shared memory computer and a fast memory lock/unlock primitive for locking hash buckets.

The second algorithm is a divide and conquer algorithm. Let $f = A + B$ and $g = C + D$ where the terms in A are greater than B and C are greater than D. Thus $f \times g = AC + AD + BC + BD$. We multiply AC, AD, BC, BD in parallel, either recursively, or using our heap algorithms. We have all terms in AC are greater than those in BD so they do not overlap. Identify where AC, AD, and BC overlap and merge using two processors (merging from both ends). Similarly for AD, BC with BD.

The third algorithm uses the recursive representation for sparse multivariate polynomials. It computes the C_k (see section 2.1) in parallel, recursively.

3.2 Polynomial GCD computation.

We propose to improve our algorithm in [9] in the following ways.

1 Generalize it to treat the case when the coefficient ring has zero divisors but the GCD exists.

2 Our algorithm in [9] reduces GCD computation in $L[x_1, ..., x_n]$ to many univariate GCD computations in x_1 over $R_p = L(t_1 = \alpha_1, ..., t_k = \alpha_k)$ modulo a prime p, a finite ring, which are computed using the Euclidean algorithm. Often, over 90% of the total time is spent doing this. By designing "in-place" algorithms for arithmetic in $R_p[x_1]$ we aim to obtain an improvement of a factor of 2 to 5 overall.

3 Complete the analysis for the failure probability of our algorithm.

4 Design and implement parallel sparse polynomial (or rational function) interpolation algorithms for the purpose of computing polynomial GCDs.

3.3 Polynomial factorization.

Illias Kotsirias tried to factor the following polynomial in Maple.

$$f = 3/2x_0^2 + 7/2x_1^2 + 11/2x_2^2 + 15/2x_3^2 + 19/2x_4^2 + 23/2x_5^2 - \sqrt{3}\sqrt{2}x_0x_1$$
$$- 2\sqrt{5}x_1x_2 - \sqrt{2}\sqrt{3}\sqrt{7}x_2x_3 - 6\sqrt{2}x_3x_4 - \sqrt{2}\sqrt{5}\sqrt{11}x_4x_5 - 10681741/1985.$$

Since Maple is using Trager's factorization algorithm, Maple begins by computing the norm $||f||$. But the polynomial $||f||$ is not easy to compute. It is degree 64 in $x_0, x_1, ..., x_5$ with 2,760,681 terms and integer coefficients of over 200 decimal digits in length. Factoring $||f||$ will be even more difficult! However, a Hensel lifting approach should be very feasible: pick small random integers $\alpha_1, ..., \alpha_5$, factor $\bar{f} = f(x_0, x_1 = \alpha_1, ..., x_5 = \alpha_5)$ using Trager's method, then Hensel lift the remaining variables $x_1, ..., x_5$.

For a polynomial $f \in L[x_0, x_1, ..., x_n]$ over an algebraic function field L in $k \geq 0$ parameters $t_1, ..., t_k$, there are several ways to approach this. We could evaluate the parameters first at small integers, factor the resulting polynomial (over a number field) and lift the parameters. Or we could evaluate the variables $x_1, ..., x_n$ first, factor the resulting univariate polynomial in $L[x_0]$ using Trager's method then Hensel lift $x_1, ..., x_n$. What we are presently investigating is how to do leading coefficient predetermination when working over a algebraic number or function field L.

Suppose we factor $c \in L[x_1, ..., x_n]$, the leading coefficient of f in the main variable x_0. In [18], for $L = \mathbf{Q}$, Wang gives a clever way to determine which factors of c belong to which factors of $\bar{f}$ so that the Hensel lifting can be started with correct leading coefficients. His algorithm computes only integer GCDs of the leading coefficient of $\bar{f}$ and the leading coefficients of the factors of c evaluated at $(\alpha_1, ..., \alpha_n)$.

Wang's technique assumes that GCDs exist and are unique (up to multiplication by units) in the underlying coefficient ring, $\mathbf{Z}$. For $L = \mathbf{Q}(i)$, since $\mathbf{Z}[i]$, the Gaussian integers, is a UFD, Wang's technique works. But for $L = \mathbf{Q}(\sqrt{-5})$ and $L = \mathbf{Q}(s)[c]/(s^2 + c^2 - 1)$, the underlying rings $\mathbf{Z}[\sqrt{-5}]$ and $\mathbf{Z}[s, c]/(s^2 + c^2 - 1)$ are not UFDs. The approach we are trying is to use GCDs of norms of algebraic numbers and functions to do leading coefficient predetermination.

4 Training Aspect of the Proposal

I presently have eight graduate students. Two will graduate by the end of 2008 or early in 2009 so I will be supporting 6 students, 1 Ph.D. and 5 Masters. Of the Masters students, two want to do a Ph.D. I have budgeted approximately (90%) of the funds for graduate student RA support and travel support for them to attend conferences in the field. The research problems in this proposal will provide graduate students the opportunity to develop their mathematical skills and algorithm design and implementation skills.

References

[1] D. Bini, V. Pan. Improved parallel polynomial division. *SIAM J. Comp.* **22** (3) 617–626, 1993.

[2] W. S. Brown, On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors, *J. ACM* **18** (1971), pp. 476–504.

[3] Diaz A., Kaltofen E. On computing greatest common divisors with polynomials given by black boxes for their evaluation. *Proc. of ISSAC '95*, ACM Press, pp. 232–239, 1995.

[4] Geddes K.O., Labahn G., Czapor S. *Algorithms for Computer Algebra.* Kluwer, 1992.

[5] Richard Fateman. Comparing the speed of programs for sparse polynomial multiplication. *SIGSAM Bulletin*, **37** (1) (2003) 4–15.

[6] M. van Hoeij, M. B. Monagan, A Modular GCD Algorithm over Number Fields with Multiple Field Extensions. *Proc. of ISSAC '02*, ACM Press, pp. 109–116, 2002.

[7] M. van Hoeij, M. B. Monagan. Algorithms for Polynomial GCD Computation over Algebraic Function Fields. *Proc. of ISSAC '04*, ACM Press, pp. 297–304, 2004.

[8] E. Horowitz. A Sorting Algorithm for Polynomial Multiplication. *J. ACM*, **22** (1975) 450–462.

[9] Mahdi Javadi and Michael Monagan. A Sparse Modular GCD Algorithm for Polynomial GCD Computation over Algebraic Function Fields. *Proc. of ISSAC '07*, ACM Press, pp. 187–194, 2007. Preprint included.

[10] S.C. Johnson. Sparse polynomial arithmetic. *SIGSAM Bulletin*, **8** (3) (1974) 63–71.

[11] D. Lazard, Solving Zero-dimensional Systems. *J. Symbolic Comp.* **13**, 117–131, 1992.

[12] A.K. Lenstra. Lattices and factorization of polynomials over algebraic number fields. *Proc. of EUROCAM '82*, Springer-Verlag LNCS **144**, pp. 32–39, 1982.

[13] X. Li and M. Moreno Maza. Multithreaded parallel implementation of arithmetic operations modulo a triangular set. *Proc. of PASCO '07*, ACM Press, 53–59, 2007.

[14] M. Moreno Maza, R. Rioboo, Polynomial Gcd Computations over Towers of Algebraic Extensions, *Proc. AAECC-11* (1995), pp. 365–382.

[15] Michael Monagan and Roman Pearce. Sparse Polynomial Pseudo Division Using a Heap. Submitted to *J. Symbolic Comp.*, September 2008. Preprint included.

[16] Joel Moses and David Yun. The EZGCD Algorithm. *Proc. ACM '73*, pp. 159–166, 1973.

[17] H. Murao and T. Fujise. A modular algorithm for sparse multivariate polynomial interpolation and its parallel implementation. *J. Symbolic Comp.* **21** (1996) 377–396.

[18] Wang P.S. The EEZ-GCD Algorithm. *SIGSAM Bulletin* **14** pp. 50-60, 1980.

[19] Wang P.S. Factoring multivariate polynomials over algebraic number fields. *Math. Comp.* **30** (1976) 324–336.

[20] Wang P. S. Parallel Polynomial Operations on SMPs. *J. Symbolic. Comp.,* **21** 397–410, 1996.

[21] Rayes M., Wang P., Weber K. Parallelization of the Sparse Modular GCD Algorithm for Multivariate Polynomials on SMPs. *Proc. of ISSAC '94*, ACM Press, pp. 66–73, 1994.

[22] T. Yan. The geobucket data structure for polynomials. *J. Symb. Comput.* **25** (1998) 285–293.

[23] Zippel R., Probabilistic Algorithms for Sparse Polynomials. *Proc. of Eurosam 79'*, Springer-Verlag LNCS, **72**, 216-226, 1979.

[24] Zippel R., Interpolating Polynomials from Their Values. *J. Symbolic Comp.* **9** (1990) 375–403.