Sparse Polynomial Multiplication and Division in Maple 14

Michael Monagan and Roman Pearce
Department of Mathematics, Simon Fraser University
Burnaby B.C. V5A 1S6, Canada

Abstract

We demonstrate new routines for sparse multivariate polynomial multiplication and division over the integers that we have integrated into Maple 14 through the `expand` and `divide` commands. These routines are currently the fastest available, and the multiplication routine is parallelized with superlinear speedup. The performance of Maple is significantly improved. We describe our polynomial data structure and compare it with Maple's. Then we present one benchmark comparing Maple 14 with Maple 13, Magma, Mathematica, Singular, Pari, and Trip.

This work was supported by the MITACS NCE of Canada and Maplesoft.

1 Introduction

There are two polynomial representations that computer algebra systems mainly use: the *distributed* representation and the *recursive* representation. In the distributed representation a polynomial is stored as a sum of terms, e.g.:

$$f = 9\,xy^3z - 4\,y^3z^2 - 6\,xy^2z + 8\,x^3 + 5\,xy^2.$$

The terms are often sorted. They are sorted above in the graded lexicographical ordering with $x > y > z$, so the terms of highest total degree come first and ties are broken by lexicographical (alphabetical) order. Computer algebra systems that use the distributed representation by default include Maple, Mathematica, Magma, and Singular.

In the recursive representation polynomials are stored as univariate polynomials in one variable with coefficients that are (recursive) polynomials in the remaining variables. For example,

$$f = 8\,x^3 + \left((9\,z)y^3 + (-6\,z+5)\,y^2\right)x + (-4\,z^2)y^3.$$

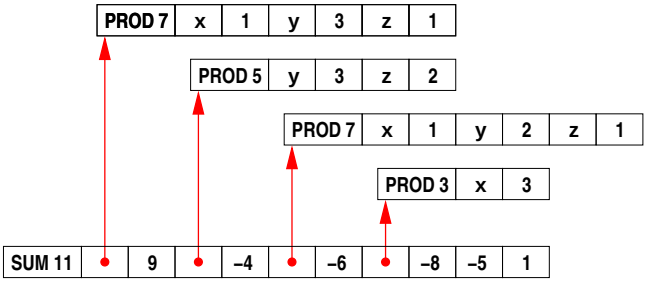
Polynomial arithmetic uses univariate algorithms (in x) with coefficient arithmetic (in $\mathbb{Z}[y,z]$) performed recursively. Computer algebra systems that use the recursive representation by default include Maxima, Reduce, Pari, and Trip.

It is widely believed that the recursive representation is generally more efficient than the distributed representation for multiplying and dividing sparse polynomials. See, for example, the work of Stoutemyer (15) and Fateman (3). However, in (10) we found that algorithms based on heaps together with efficient monomial representations can make the distributed representation faster than the recursive representation. Heap based algorithms can run in the CPU cache without accessing memory heavily or generating any intermediate "garbage". This lends them to efficient parallelization on multicore processors.

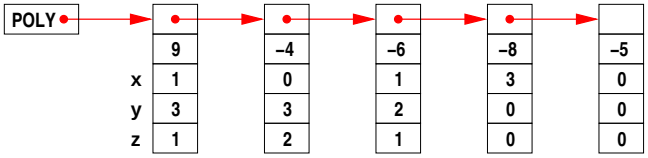
In 1975 Johnson (8) presented an algorithm for multiplying two polynomials in the distributed representation. Let $f = f_1 + f_2 + \cdots + f_m$ and $g = g_1 + g_2 + \cdots + g_n$. Johnson's algorithm uses a binary heap to merge the partial products $\{f_1 \cdot g,\ f_2 \cdot g,\ \ldots, f_m \cdot g\}$ in $O(mn \log \min(m,n))$ monomial comparisons. We show in (9) how to modify the heap so that for dense polynomials only $O(mn)$ comparisons. This is needed to have good performance on both sparse and dense problems. In (10) we present an algorithm to divide sparse polynomials that uses a heap and has the same complexity as multiplication. And in (11) and (12) we present parallel algorithms for sparse polynomial multiplication and division that achieve superlinear speedup often, a factor of 5 on a 4 core machine.

2 Polynomial Data Structures

The performance of any algorithm also depends on the data structure that is used to represent polynomials. Here we describe `sdmp`, our data structure, and compare it with the distributed representation used by Maple and Singular. The following figures show how the polynomial $9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5$ is represented in Maple and in Singular. Magma uses a representation similar to Singular's. Mathematica's representation is similar to Maple's.



Maple's sum-of-products representation
uses ~ 9 words per term.



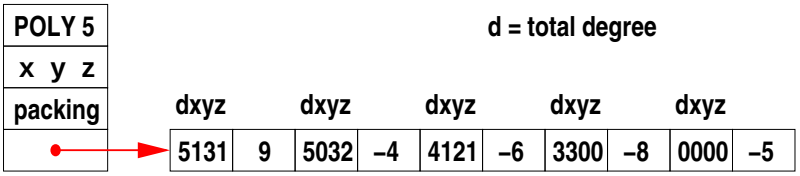
Singular's distributed representation
uses ~ 5 words per term.

The main reason why Maple is slow is that multiplying and comparing monomials is slow. For example, to multiply xy^3z by yz^2, Maple allocates an array long enough to store the product $x^1y^4z^3$ and computes it in a loop with a merge. Since exponents can be integers, rational numbers, or floating point constants, exponent addition involves a function call and a dispatch. Next Maple hashes the resulting monomial to determine if this monomial already exists in memory. All of this requires many function calls and many loops. Singular does much less work but still uses many machine cycles. How fast can one multiply and compare monomials?

In the graded lexicographical ordering with $x > y > z$, we can encode monomials $x^iy^jz^k$ in 64-bit machine words as the integer $(i+j+k)2^{48} + 2^{32}i + 2^{16}j + k$, that is, storing the 4 integers $(i+j+k, i, j, k)$ using 16 bits each. A monomial comparison becomes one machine integer comparison and a monomial multiplication becomes one machine integer addition provided the degrees are not too large to cause overflow. The figure below shows how our system `sdmp` represents the polynomial

$$9\,xy^3z - 4\,y^3z^2 - 6\,xy^2z + 8\,x^3 - 5$$

in the above graded monomial ordering. Our software also supports pure lexicographical order and the ability to pack monomials into multiple machine words.



sdmp's packed array representation using 2 words per term.

An integer coefficient x is stored in place as $2x+1$ if $|x| < 2^{B-2}$, where $B = 64$ is the word size of the machine. For larger coefficients we store pointers to GMP integers (see (5)) which we distinguish by their least significant bit.

This idea of packing multiple exponents into a word dates back to the Altran computer algebra system (7) in 1971. This was a period when RAM memory was very limited. In 1998 (1), Bachmann and Schönemann experimented with packings for different monomial orderings to speed up polynomial divisions in the distributed representation for Gröbner basis computations. However, none of the general purpose computer algebra systems since Altran have used this idea, presumably because they did not want to limit the degrees of polynomials or the number of variables. Today, however, CPUs and operating systems have moved to 64-bits decisively. This allows us to pack monomials in many variables with high degrees into one word.

3 Maple 14 Integration

In Maple, the `expand` and `divide` commands perform multiplication and exact division of multivariate polynomials with integer and rational coefficients. When the polynomials involved have few terms, multiplication or division is done directly in the Maple kernel which is programmed in C. Otherwise, a library routine `'expand/bigprod'` or `'expand/bigdiv'` is invoked which allows for more sophisticated algorithms to be used. For example, in Maple 13, for dense multivariate polynomials, Maple switches to using a recursive representation in one of the variables and then expands (divides) resursively the coefficients in the remaining variables. These two routines are programmed in Maple's interpreted language.

Our approach has been to modify these routines. For polynomials with integer coefficients, we convert them to our `sdmp` data structure, automatically packing monomials into as few words as possible, multiply (divide) using our C library, and then convert the product (quotient) back to Maple's sum-of-products representation. For polynomial multiplication we pack monomials using pure lexicographical order to minimize the number of machine words needed to encode the monomials in the product. For polynomial division, however, we use the graded monomial ordering because no exponent overflow in a multivariate division can occur in this ordering.

Although the conversions are programmed in C, for sparse multiplications where the product has $O(\#f \cdot \#g)$ terms, e.g., for $f(x) = a_n x^n + \cdots a_1 x + a_0$ and $g(y) = b_m y^m + \ldots b_1 y + b_0$, the conversion of the product to Maple's sum-of-products representation can account for more than 70% of the time. Partly for this reason, we are working with Maplesoft to make our `sdmp` data representation the new default representation in Maple. Maple will support two main representations. If all monomials of a polynomial in one or more variables can be packed into one machine word in the graded monomial ordering, Maple will use our `sdmp` representation. Otherwise the polynomial will be stored in Maple's existing sum-of-products representation. Our thinking is that on a 64 bit computer, many (most?) multivariate polynomials encountered in practice will be stored using our representation. This will not only eliminate conversion overhead but also reduce significantly the space needed to store them.

4 Benchmarks

The following table illustrates the gains in performance that are obtained from using our routines in Maple 14 when compared with Maple 13. We used an Intel Core i7 920 CPU running at 2.66GHz with hyperthreading disabled. This is a 64 bit machine with 4 cores. All times reported are real times, not cpu times, in seconds. The Maple 13 and 14 timings are for excecuting the `expand(f1*g1)`, `divide(p1,f1,'q1')` and `factor(p1)` commands, that is, they include conversion overhead. For Maple 14, we report two timings, firstly, the time for Maple 14 running with 1 core, and secondly, with 4 cores. Also shown is the %age of the real time spent in conversion and other overhead.

Our closest competitor is Trip (4), a computer algebra system for celestial mechanics. We report two times for Trip. The first (RS) is for Trip's recursive sparse polynomial data structure, the second (RD) is for Trip's recursive dense polynomial data structure. Like Maple 14, polynomial multiplication in Trip is parallelized. Both timings reported for Trip are for 4 cores.

	Maple 13	Maple 14		Magma	Singular	Mathematica	Pari	Trip 1.0	
		(1 core)	(4 cores)	2.16-8	3.1	7.0	2.3	(RS)	(RD)
multiply									
$p_1 := f_1(f_1+1)$	1.60	0.062	0.028 (50%)	0.30	0.58	4.79	0.45	0.043	0.035
$p_2 := f_2(f_2+1)$	1.55	0.062	0.028 (50%)	0.30	0.57	5.06	1.86	0.047	0.048
$p_3 := f_3(f_3+1)$	26.76	0.532	0.166 (28%)	4.09	6.96	50.36	3.96	0.376	0.314
$p_4 := f_4(f_4+1)$	95.97	2.293	0.675 (24%)	13.25	30.64	273.01	39.15	1.648	1.309
divide									
$q_1 := p_1/f_1$	1.53	0.09	0.09 (11%)	0.36	0.42	6.09	0.27	0.216	0.202
$q_2 := p_2/f_2$	1.53	0.09	0.09 (11%)	0.36	0.43	6.53	0.77	0.221	0.254
$q_3 := p_3/f_3$	24.74	0.73	0.74 (5%)	4.31	3.98	46.39	2.88	1.880	1.722
$q_4 := p_4/f_4$	93.42	3.17	3.17 (7%)	20.23	15.91	242.87	17.67	8.071	7.286
factor									
$p_1 =$ 12341 terms	31.10	2.80	2.66	6.15	12.28	11.82			
$p_2 =$ 12341 terms	296.32	20.80	20.00	6.81	23.67	64.31			
$p_3 =$ 38711 terms	391.44	17.15	15.59	117.53	97.10	164.50			
$p_4 =$ 135751 terms	2953.54	66.35	54.73	332.86	404.86	655.49			

$f_1 = (1+x+y+z)^{20}+1$ 1771 terms $f_2 = (1+x^2+y^2+z^2)^{20}+1$ 1771 terms $f_3 = (1+x+y+z)^{30}+1$ 5456 terms $f_4 = (1+x+y+z+t)^{20}+1$ 10626 terms

We also see a substantial gain in performance on polynomial factorization from the improvements to polynomial multiplication and division. This is because the algorithm for factoring $f(x,y,z)$ starts by factoring the univariate polynomial $f(x, y=a, z=b)$, where $\{a,b\}$ are small integers, and it recovers $\{y,z\}$ in the factors of $f(x,y=a,z=b)$ using a process called "Hensel lifting". Hensel lifting consists of a sequence of polynomial multiplications and some divisions. Most of the time spent factoring $f(x,y,z)$ is in the polynomial multiplications in the Hensel lifting.

The parallel multiplication algorithm in Maple 14 generally uses fewer CPU cycles in total than the sequential algorithm, due to its efficient use of extra cache from each core – see (11). We obtain some parallel speedup on

the factorization benchmarks "for free", however the speedup is not as large as one would hope. This is because for smaller polynomial multiplications, the conversions to and from Maple's data structure are sequential and incur substantial overhead.

One can see that for the sparse benchmark, $p_2 = f_2(f_2+1)$, the systems which use a recursive dense representation (Pari and Trip) take longer but multiplication and division in the other systems is essentially the same as for the first benchmark. Note Magma (see (14)) factors p_2 by substituting $x^2 = u, y^2 = v, z^2 = w$ and factoring $p_2(u, v, w) = f_2(u, v, w)(f_2(u, v, w) + 1)$.

5 What we will present at ISSAC

1. Picture of the data structures used in Maple 13, Singular, Pari, Trip, with our new data structure indicating why Maple is slow and showing the monomial encoding we use.
2. A slide summarizing our heap based algorithms for multiplication and division with complexity results and the main advantages.
3. A slide giving a visual picture of our parallel multiplication algorithm to indicate that this is now in Maple 14.
4. A slide detailing how the integration of our code and data structure into Maple 14 has been done and indicating that the monomial encoding is automatic.
5. A benchmark slide detailing the improvements obtained.
6. A demonstration of our software package `sdmp` used from inside Maple showing different monomial packings and our parallel multiplication speedup.
7. A demonstration of a polynomial factorization in Maple showing our multiplication and division software being called.

References

[1] Bachmann, O., Schönemann, H., 1998. Monomial representations for Gröbner bases computations. *Proc. ISSAC '98*, pp. 309–316.

[2] Bosma, W., Cannon, J., Playoust, C., 1997. The Magma algebra system I: The user language. *J. Symb. Comput.* 24 (3-4), 235–265. See also `http://magma.maths.usyd.edu.au/magma`

[3] Fateman, R., 2003. Comparing the speed of programs for sparse polynomial multiplication. ACM SIGSAM Bulletin 37 (1), pp. 4–15.

[4] Gastineau, M., Laskar, J., 2006. Development of TRIP: Fast Sparse Multivariate Polynomial Multiplication Using Burst Tries. In: Proc. ICCS 2006, Springer LNCS 3992, pp. 446–453. http://www.imcce.fr/Equipes/ASD/trip

[5] Granlund, T., 2008. The GNU Multiple Precision Arithmetic Library, version 4.2.2. http://www.gmplib.org/

[6] Greuel, G.-M., Pfister, G., Schönemann, H., 2005. Singular 3.0: A Computer Algebra System for Polynomial Computations. Centre for Computer Algebra, University of Kaiserslautern. http://www.singular.uni-kl.de

[7] Hall, A.D. Jr., The ALTRAN System for Rational Function Manipulation – A Survey. *Communications of the ACM*, **14**, 517–521, ACM Press, 1971.

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

[9] Monagan, M., Pearce, R., 2007. Polynomial Division Using Dynamic Arrays, Heaps, and Packed Exponent Vectors. *Proc. of CASC '07*, Springer Verlag LNCS **4770**, 295–315.

[10] Monagan, M., Pearce, R., 2008. Sparse Polynomial Division using Heaps. Accepted September 2009 for *J. Symb. Comp.*, Preprint: http://www.cecm.sfu.ca/CAG/papers/MonHeaps.pdf

[11] M. Monagan, R. Pearce., 2009. Parallel Sparse Polynomial Multiplication Using Heaps. *Proc. of ISSAC 2009*, 295–315.

[12] M. Monagan, R. Pearce., 2010. Parallel Sparse Polynomial Division Using Heaps. Submitted to *PASCO 2010*, April 2010. Preprint: http://www.cecm.sfu.ca/CAG/papers/ParHeapDiv.pdf

[13] PARI/GP, version 2.3.4, Bordeaux, 2008. http://pari.math.u-bordeaux.fr/

[14] Allan Steel, private communication, May 2010.

[15] Stoutemyer, D., 1984. Which Polynomial Representation is Best? Proc. of the 1984 Macsyma Users Conference, Schenectedy, N.Y., pp. 221–244.