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

October 15, 2009

Abstract

We report on new codes for sparse multivariate polynomial multiplication and division over the integers that we have integrated into Maple 14's `expand` and `divide` commands. We describe our polynomial data structure and compare it with the one used by Maple. We then present some benchmarks comparing our software with the Magma, Maple, Singular, Trip and Pari computer algebra systems and also illustrating how multivariate polynomial factorization benefits from our improvements to polynomial multiplication and division.

This work was supported by the MITACS NCE of Canada.

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 represented as a sum of terms, for example,

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

In our example, the terms of f are sorted in the graded lex monomial ordering with $x > y > z$. In this ordering monomials of higher degree (the degree of $x^iy^jz^k$ is $i + j + k$) come first and ties are broken by lexicographical (alphabetical) order. Algorithms for distributed polynomial arithmetic have a distinctly classical feel – their performance depends on how you sort terms and perform coefficient arithmetic. Computer algebra systems that use the distributed representation by default include Maple, Mathematica, Magma, and Singular.

In the recursive representation polynomials are represented as univariate polynomials in a main variable with coefficients that are univariate polynomials in the next variable, and so on. 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, Derive, Reduce, Pari, and Trip.

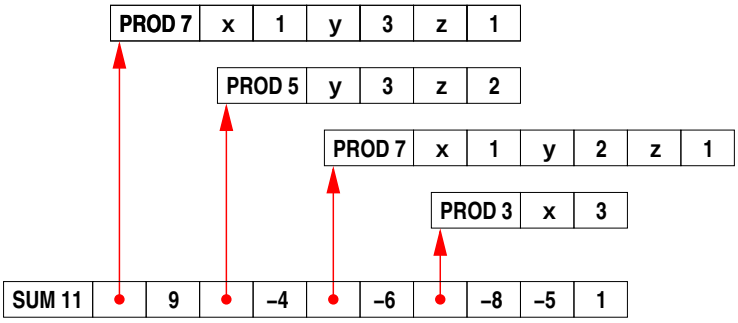
It is widely believed that the recursive representation is more efficient than the distributed representation for multiplying polynomials. See for example the work of Stoutemyer in (13) and Fateman in (3). Indeed our own benchmarks in Section 4 show Pari and Trip are generally faster than Maple, Magma, and Singular. However, we have found that if one uses good algorithms and packs monomials, multiplication and division in the distributed representation can be faster than the recursive representation.

In 1975, Johnson in (8) presented an algorithm for multiplying two polynomials $h := f \times g$ in the distributed representation. Letting $f = f_1 + f_2 + \ldots f_m$ and $g = g_1 + g_2 + \ldots + g_n$, Johnson's algorithm uses a binary heap to merge the sums $f_1 \times g, f_2 \times g, \ldots, f_m \times g$ in $O(mn \log \min(m,n))$ monomial comparisons. For dense polynomials we showed in (10) how to modify the heap so that multiplication does $O(mn)$ monomial comparisons. This is important so that performance is good for both sparse and dense problems. In (11) we presented an algorithm for dividing polynomials, also using a heap, that achieves the same complexity – $O(mn \log \min(m,n))$ monomial comparisons. But the overall performance of an algorithm will also depend on the data structure that is used for representing polynomials.

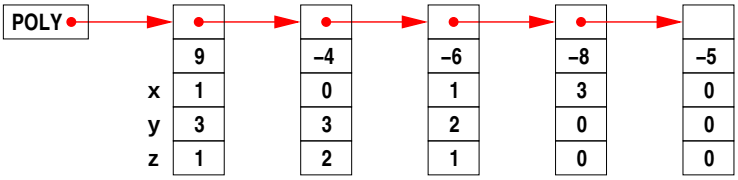
In this report we describe our data structure `sdmp` and compare it with Maple's "sum-of-products" data structure. We then give two benchmarks comparing our algorithms for multiplication and division with those in the Magma, Pari, Singular and Trip computer algebra systems. Finally, we describe how we have integrated our new software into Maple 14. We give some benchmarks that show that by speeding up polynomial multiplication and division, we often get a very good speedup in other parts of Maple, for example, in polynomial factorization.

2 Maple's Sum-Of-Products Data Structure

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 similar representation to Singular.



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



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

One of the reasons why Maple, Singular, and Magma are slow compared with Pari and Trip is that multiplying and comparing monomials is slow. For example, to multiply xy^3z by yz^2, Maple first must allocate an array long enough to store the product $x^1y^4z^3$, then compute it in a loop using a merge. Maple then hashes the resulting monomial to determine if it already exists in memory. All this means several function calls and several loops. Singular does less work, but still uses many machine cycles. How fast can one multiply monomials?

3 The sdmp Data Structure

Assuming we are using the graded lex monomial ordering with $x > y > z$, to speed up monomial multiplications and comparisons we encode a monomial $x^i y^j z^k$ into one 64 bit machine word by encoding the 4 integers $(i+j+k, i, j, k)$ using 16 bits each as the integer $(i+j+k)2^{48} + 2^{32}i + 2^{16}j + k$. Now a monomial comparison becomes one machine integer comparison and multiplication of two monomials $x^k y^j z^k$ by $x^l y^m z^n$ becomes one machine integer addition, provided the degrees are not too large to cause overflow. Figure 3 below shows how we represent the polynomial

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

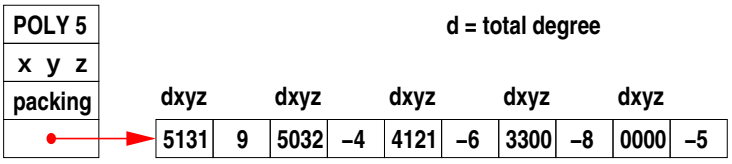


Figure 1: sdmp's representation with all monomials packed in the graded lex order with $x > y > z$.

An integer coefficient x is stored in place as $2x + 1$ when $|x| < 2^{B-2}$ where $B = 64$ is the base of the machine. Otherwise, for larger coefficients we store pointers GMP integers (see (5)), which we distinguish by the lowest bit. We illutrate this in Figure 3 below for the polynomial $Axy^3z + By^3z^2 + Cxy^2z - 8x^3 - 5$.

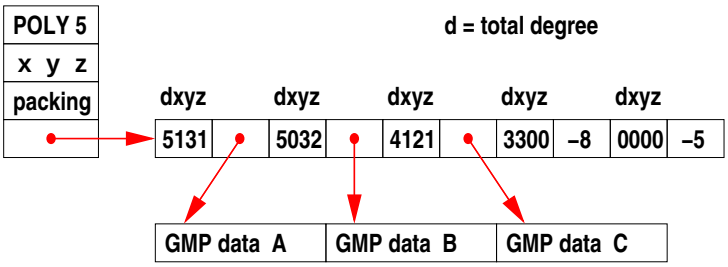


Figure 2: sdmp's packed representation with coefficients A, B, C large integers.

The benchmarks in the next section show the significant benefit of packing exponents of monomials. Obviously we save both space and time. This idea of packing multiple exponents into a word dates back to the Altran computer algebra system (see (7)) from 1971. In 1998 Bachmann and Schönemann in (1) experimented with packings for different monomial orderings to speed up the polynomial divisions in the distributed representation for Gröbner basis computations. Yet none of the general purpose computer algebra systems since Altran have used it. This, we believe, is because they did not want to limit the degree of the polynomials or the number of variables. Most of them were designed when 32 bit word computers were the norm. Today, however, all new computers have 64 bit words which allow us to pack monomials with more variables and higher total degree into one word.

4 Some Benchmarks

To benchmark the computer algebra systems we used one core of an Intel Xeon 5160 (Core2) 3.0 GHz with 4 MB of L2 cache, 16 GB of RAM, 64-bit Linux, and GMP 4.2.1. We give two times for our C library (sdmp). In the slow time (unpacked) each exponent is stored as a 64 bit integer. For the fast time we pack all of the exponents into one 64 bit integer and use word operations to compare and multiply monomials.

We tested Pari/GP 2.3.3, Magma 2.14-7 (see (2)), Maple 12, Singular 3-0-4 (see (6)), and Trip 0.99 (see (4)). Maple and Singular use distributed representations to multiply polynomials and switch to recursive representations to divide. Magma and sdmp use distributed representations for both. Pari and Trip use recursive representations for both.

Our first problem is due to Fateman (3). Let $f = (1 + x + y + z + t)^{20}$ and $g = f + 1$. We multiply $h = f \cdot g$ and divide $q = h/f$. Fateman's benchmark is a dense computation. The coefficients of f and g are 39 bit integers and the coefficients of h are 83 bit integers.

$10626 \times 10626 = 135751$ terms	$h = f \times g$	$q = h/f$
sdmp (1 word monomial)	2.26 s	2.77 s
sdmp (4 word monomial)	5.18 s	5.44 s
Trip v0.99 (rationals)	5.93 s	-
Pari/GP 2.3.3	32.43 s	14.76 s
Magma V2.14-7	23.02 s	22.76 s
Singular 3-0-4	62.00 s	20.00 s
Maple 12	289.23 s	187.72 s

Table 1: CPU time (in seconds) for dense multiplication and division over $\mathbb{Z}$

Maple and Singular can both divide faster than they can multiply because they switch to using recursive algorithms. In sdmp division is slightly slower than multiplication because the code is more complex.

Our second benchmark is a sparse benchmark in 10 variables. Let $f = \left(\sum_{i=1}^{9} x_i(x_{i+1} + 1) + x_{10}(x_1 + 1) + 1\right)^4$ and $g = \left(\sum_{i=1}^{10}(x_i^2 + x_i) + 1\right)^4$. We multiply $h = f \cdot g$ and divide $q = h/f$. All coefficients are less than 20 bits long.

$6746 \times 8361 = 3157883$ terms	$h = f \cdot g$	$q = h/f$
sdmp (1 word monomial)	2.46 s	2.61 s
sdmp (10 word monomial)	11.12 s	10.37 s
Trip v0.99 (rationals)	8.13 s	-
Pari/GP 2.3.3	7.06 s	7.05 s
Magma V2.14-7	17.43 s	197.72 s
Singular 3-0-4	31.00 s	18.00 s
Maple 12	305.76 s	280.65 s

Table 2: CPU time (in seconds) for sparse multiplication and division over $\mathbb{Z}$

This benchmark clearly demonstrates the value of packing exponents. Pari's recursive dense algorithms perform well on many variables of low degree, while Magma's division (ExactQuotient) seems to sort terms inefficiently. It may be using repeated subtraction.

5 Maple 14 Integration

The Maple commands `expand` and `divide` do polynomial multiplication and exact division of multivariate polynomials over the integers as illustrated in the example below. Lines beginning with `>` are Maple input commands.

```
> f := expand( (1+x+y+z)^2 );
```

$$1 + 2\,x + 2\,y + 2\,z + x^2 + 2\,xy + 2\,xz + y^2 + 2\,yz + z^2$$

```
> g := f+1:
> h := expand(f*g):
> divide(h,f,'q'); # test if f|h
```

$$true$$

```
> q;
```

$$2 + 2\,x + 2\,y + 2\,z + x^2 + 2\,xy + 2\,xz + y^2 + 2\,yz + z^2$$

If either multiplicand f or g has few terms, the multiplication is done in the Maple kernel in C, otherwise, the Maple library routine `‘expand/bigprod‘` is invoked to do the multiplication. Similarly, if the dividend h or divisor f have few terms, the division is done in the Maple kernel, otherwise the Maple library routine `‘expand/bigdiv‘` is invoked to do the division. These library routines are programmed in Maple.

The way we have integrated our new software into Maple 14 is to reprogram `‘expand/bigprod‘` and `‘expand/bigdiv‘` to convert their input polynomials to our sdmp data structure, automatically packing monomials in the graded lex monomial ordering in as few words as possible, then multiply or divide, then convert the output polynomials back to Maple's sum-of-products representation. We have found that the conversions back to Maple's sum-of-products representation, even though coded carefully in C, can take longer than the multiplication of $f \times g$ if f and g are sparse and their product has many terms.

In the following benchmark we compare the time for polynomial factorization in Maple 13 with our new code in Maple 14. The columns in Table 3 for expand and factor are timings (in CPU seconds) for computing `f := expand((a+1)*(a+2))` and `factor(f)` respectively, for the polynomials a in the table.

To factor $f(x, y, z)$, the factorization algorithm first factors the univariate polynomial $f(x, y = a, z = b)$ where a, b are small integers, then recovers y, z in the factors of $f(x, y = a, z = b)$ in a process called "Hensel lifting". Hensel lifting consists of a sequence of polynomial multiplications and some polynomial divisions where most of the time spent factoring $f(x, y, z)$ is in the polynomial multiplications in the Hensel lifting. The benchmark demonstrates that by significantly speeding up polynomial multiplication and division, we get a very good speedup in polynomial factorization.

Benchmark	expand			factor		
$a =$	Maple13	Maple14	Speedup	Maple13	Maple14	Speedup
$(x+y+z)^{30}$	.213	0.009	23.7 x	368.881	18.615	19.8 x
$(1+x+y+z)^{20}$	1.869	0.069	27.1 x	38.379	4.009	9.6 x
$(1+x+y+z)^{30}$	38.746	0.720	53.8 x	679.010	23.384	29.0 x
$(1+x+y+z+t)^{20}$	159.971	3.330	48.0 x	5390.317	98.997	54.4 x

Table 3: Factorization Benchmark Timings (in CPU seconds)

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] B.W. Char, G.J. Fee, K.O. Geddes, G.H. Gonnet and M.B. Monagan. A Tutorial Introduction to Maple. *J. Symbolic Comp.*, **2** (2), 179–200, 1986.

[10] 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.

[11] Monagan, M., Pearce, R., 2007. Sparse Polynomial Division using Heaps. Accepted for *J. Symb. Comp.*, September 2009.
Preprint: http://www.cecm.sfu.ca/CAG/papers/MonHeaps.pdf

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

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