POLY : A new polynomial data structure for Maple 17 *

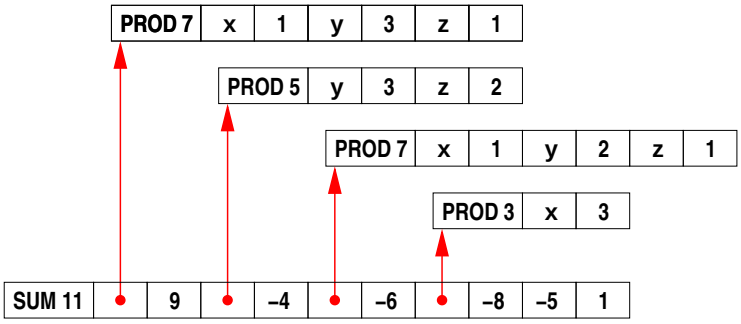
Michael Monagan and Roman Pearce
Department of Mathematics, Simon Fraser University
Burnaby B.C. V5A 1S6, Canada

Abstract

We demonstrate how a new data structure for sparse distributed polynomials in the Maple kernel significantly accelerates several key Maple library routines. The POLY data structure and its associated kernel operations (degree, coeff, subs, has, diff, eval, ...) are programmed for high scalability, allowing polynomials to have hundreds of millions of terms, and very low overhead, increasing parallel speedup in existing routines and improving the performance of high level Maple library routines.

1 Introduction

The figure below shows the default polynomial data structure in Maple 16 and all previous versions for the polynomial $9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5$. It is a "sum-of-products" where each term has a separate Maple object, a `PROD`, to represent the monomial. To compute the degree of f, a coefficient in x, test for a subexpression, or do almost anything else, the Maple kernel must recursively descend through multiple levels of dags. This involves extensive branching and random memory access, both of which are slow, and will prevent Maple from achieving high-performance on modern CPUs.

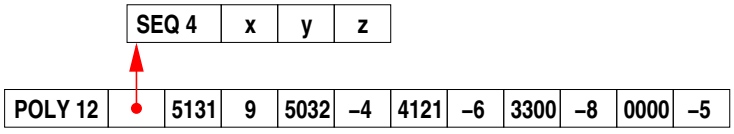


Maple's sum-of-products representation has
irregular Maple dags for each term.

Another operation that is very slow is monomial multiplication. Consider multiplying f by xyz^2. Maple must allocate memory for each new monomial in the product and, in a loop, add exponents of like variables. Then, because exponents can be negative, Maple must simplify the monomials. Finally, because Maple stores unique copies of objects, the resulting monomial is hashed, and inserted in an internal table. In all, there are many function calls and many loops. We estimate that Maple takes over 200 clock cycles for each monomial multiplication.

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

The figure below shows our new data structure for sparse distributed polynomials. The first word is a pointer to the variables which are sorted in Maple's canonical ordering for sets. This is followed by monomials and coefficients where the monomials encode the exponents together with the total degree in a single machine word. E.g. for xy^2z^3 we store the values $(6,1,2,3)$ as $6 \cdot 2^{48} + 2^{32} + 2 \cdot 2^{16} + 3$ on a 64-bit machine. The terms are sorted into graded lex order by comparing the monomials as unsigned integers. This gives a canonical representation for the polynomial.



The new packed distributed representation.

Five advantages of the new representation are readily apparent.

1. It is much more compact. Polynomials use two words per term instead of $2n+3$ words, where n is the number of variables. For polynomials in 3 variables we save over a factor of 4.

2. By explicitly storing the variables and sorting the terms, we can execute many common Maple idioms without looking at all the terms, e.g. `degree`(f), `indets`(f) (extract the set of variables in f), `has`(f,x), and `type`$(f,polynom)$.

3. Other operations such as `degree`(f,x), `diff`(f,x), and `coeff`(f,x,i) (extract the coefficient of x^i in f) now access memory sequentially and will execute faster.

4. For large polynomials we avoid creating a lot of small Maple objects (the PRODs) each of which must be simplified by Maple's internal simplifier and then stored in Maple's `simpl` table, an internal hash table of all Maple objects. They fill the `simpl` table and slow down Maple's garbage collector.

5. Provided no overflow occurs, monomial multiplication is now integer addition, thus one machine instruction. This improves the efficiency of polynomial multiplication and division.

The idea of packing monomials in a computer word is not new; the ALTRAN computer algebra system [8] allowed the user to pack monomials in lexicographical order to conserve memory. In [1], Bachmann and Schönemann compared the graded packing with packings for other monomial orderings for Gröbner basis computation. However, as far as we know, none of the current general purpose computer algebra systems pack monomials.

We have integrated the new POLY data structure into the Maple kernel which we hope to have available for Maple 17, the next release of Maple. We describe here when the new Maple uses the new POLY dag representation. A polynomial in n variables with integer coefficients of total degree d with t terms in our new Maple is automatically stored in the POLY dag representation on a 64 bit computer if (i) $t > 1$, (ii) $d > 1$, (iii) $d < 2^b$ where $b = \lfloor 64/(n+1) \rfloor$. Otherwise it is stored in the "sum-of-products" representation. All conversions between representations are automatic and invisible to the Maple user.

Note, for polynomials with total degree $d = 1$, we chose not to store them in as a POLY dag because Maple's sum-of-products representation is better in this case. For example $f = 2x+3y+4z+5$ is represented as `SUM` $|2|x|3|y|4|z|5|1$. This is compact and monomials are not explicitly represented.

Note, the monomial encoding is determined solely by the number of variables in the polynomial. This means that operations between polynomials in the same variables require no repacking. For example, on a 64 bit computer, if a polynomial has 8 variables then we have to store 9 integers for each monomial

which means we have $\lfloor 64/9 \rfloor = 7$ bits each so the maximum total degree for a polynomial in 8 variables in the POLY dag representation is 127. The 64 bit word which is the norm on todays desktop and laptop computers, really makes this packing design workable as we expect the majority of practical problems will fit in a 64 bit word.

We chose a graded ordering as the default rather than pure lexicographical ordering for several reasons. Firstly, the graded ordering is the more natural ordering for output and secondy, unlike lexicographical order, in a graded ordering, the division algorithm cannot cause an overflow of the exponents from one bit field to another. Also, when multiplying $a \times b$ if the total degree $d = \deg a + \deg b$ does not overflow, then, unlike lexicographical order, the entire product can be computed without overflow. Thus we do not need to use up any bits to detect overflow.

2 Algorithms

The new representation has allowed us to write many high performance algorithms for the Maple kernel. In the old data structure, most operations are $O(nt)$, where n is the number of variables and t is the number of terms. Maple must examine the entire "sum-of-products" structure because its contents are unknown. In the new data structure, we can often avoid doing expensive operations on all of the terms. We measured the speedup on a polynomial with one million terms in three variables, constructed as $f := expand(mul(randpoly(i, degree = 100, dense), i = [x, y, z]))$: The cost for evaluation is added to the other commands if you are using Maple interactively.

command	description	Maple 16	new dag	speedup	notes
f;	evaluation	0.162 s	0.000 s	$\to O(n)$	evaluate the variables
$coeff(f, x, 20)$	coefficient of x^{20}	2.140 s	0.005 s	420x	binary search for univariate f
$coeffs(f, x)$	extract all coefficients in x	0.979 s	0.119 s	8x	reorder exponents and radix sort
$frontend(g, [f])$	subs functions for variables	3.730 s	0.000 s	$\to O(n)$	looks at variables only
$degree(f, x)$	degree in x	0.073 s	0.003 s	24x	stop early using monomial degree
$degree(f)$	total degree in all variables	0.175 s	0.000 s	$\to O(1)$	first term in polynomial
$diff(f, x)$	differentiate wrt x	0.956 s	0.031 s	30x	terms remain sorted
$eval(f, x = 6)$	compute $f(6, y, z)$	3.760 s	0.175 s	21x	use Horner form recursively
$expand(2xf)$	multiply by a term	1.190 s	0.066 s	18x	terms remain sorted
$has(f, x^{101})$	search for subexpression	0.040 s	0.002 s	20x	$O(n)$ for names, $O(\log t)$ for terms
$indets(f)$	set of indeterminates	0.060 s	0.000 s	$\to O(1)$	first word in dag
$lcoeff(f, x)$	leading coefficient in x	0.058 s	0.005 s	11x	stop early using monomial degree
$op(f)$	extract terms of f	0.634 s	1.740 s	0.36x	has to construct old structure
$subs(x = y, f)$	replace variable	1.160 s	0.076 s	15x	combine exponents, sort, merge
$taylor(f, x, 50)$	Taylor series to $O(x^{50})$	0.668 s	0.055 s	12x	get coefficients in one pass
$type(f, polynom)$	type check	0.029 s	0.000 s	$\to O(n)$	type check the variables

Table 1: Improvements for Maple kernel operations.

To achieve these gains, we employ a bit-level programming style [14] to avoid branches and loops. For example, to compute the degree of a monomial $x^3y^5z^7$ in $\{x, z\}$, we would mask the exponents for x and z and sum all of the fields using a parallel-prefix algorithm, which is $O(\log n)$. This is illustrated below, for a 32-bit monomial.

monomial $x^3y^5z^7$	00001111	00000011	00000101	00000111
mask for $\{x, z\}$	00000000	11111111	00000000	11111111
sum fields of	00000000	00000011	00000000	00000111

In the graded ordering, many of the above operations can still be done without need to sort the result. For example, consider our polynomial $f = 9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5$. If we differentiate f with respect to x we obtain $f' = 9y^3z + 0 - 6y^2z - 24x^2 + 0$. Notice that the non-zero terms in the derivative are sorted in the graded ordering. Thus we can compute the derivative in $O(n+t)$ instead of $O(nt)$. Another operation which can be done in $O(n+t)$ instead of $O(nt)$ is multiplication by a single term or monomial.

The one case where the new data structure loses is when we need to convert to the old data structure. The Maple command `op(f)` construct a sequence of all terms of f. Each term, e.g. $8xy^2$ is stored $\boxed{\texttt{SUM}\ 3 \mid \uparrow P \mid 3}$ where P is the monomial stored as $\boxed{\texttt{PROD}\ 5 \mid x \mid 1 \mid y \mid 2}$. Thus the new Maple must build a `SUM` and a `PROD` whereas the old Maple need only build the `SUM` as the `PROD` is already there.

The `coeffs(f,x)`, `eval(f,x=6)` and `taylor(f,x,n)` commands all need the coefficients of f in x. Suppose f is a polynomial in w, x, y. For each monomial $w^i x^j y^k$ in f, encoded as $\boxed{\texttt{dijk}}$ where $d = i+j+k$ is its degree, using a constant number of masks and bit operations (7 are sufficient), we move j, the degree of x to the front to obtain the $\boxed{\texttt{jdik}}$. Next we sort the terms of f on the new monomial encodings $\boxed{\texttt{jdik}}$ using a radix sort.[1] This groups the terms of f in x^n together, and for each group, sorts them in graded lex ordering in w and z so that the coefficient in x^n when extracted is already sorted.

The biggest improvement we have seen for a Maple library command was the `collect` command. It is used to write a polynomial in recursive form. For example, if $f = xy^3 + x^2y - x^2z + xyz - 2$, the command `collect(f,x)` writes f as $(y-z)x^2 + (y^3+yz)x - 2$. The Maple code for the `collect` command uses the `series(f,x,3);` command to implement this. Since the `series` command is 8x faster in our new Maple, we were not expecting collect to be 31 times faster. Here is a profile showing that most of the time in Maple 16 was not in the `series` command, but in the `frontend` and `indets` commands.

			Maple 16		New Maple	
function	depth	calls	time	time%	time	time%
frontend	1	1	3.932	59.43	0.000	0.00
indets	1	2	1.522	23.00	0.000	0.00
series	1	1	0.919	13.89	0.109	88.62
collect/recursive	1	1	0.160	2.42	0.010	8.13
collect/series	1	1	0.083	1.25	0.004	3.25
collect	1	1	0.000	0.00	0.000	0.00
total:	6	7	6.616	100.00	0.123	100.00

Profile for executing `collect(f,x)` in Maple 16 and Maple 17.

In our new Maple, the cost of `frontend` and `indets` are now negligible since they no longer need to descend into the sum-of-products dag. In the new Maple, they only need to look at the variables which costs $O(n)$. Why were `frontend` and `indets` so expensive? They need to search the sum-of-products dag to see if there are any indeterminates which are not variables. They are looking for objects like $x^{1/2}$, $\sin(x)$, 2^n, etc. But our polynomial has none; it only has variables x, y and z in it. In order to do this they pick appart each products and each power, e.g., given x^3yz^4 it recursively constructs x^3 and z^4 as new objects before descending them to see $x, 3, z, 4$.

3 Benchmarks

What impact on Maple's performance does the new POLY dag have for high level computations? And since the new POLY dag reduces the sequential overhead of computing with polynomials in Maple, how does this improve parallel speedup? Do we see any parallel speedup for high level operations? We consider two problems; computing determinants of matrices of polynomials and factoring polynomials.

[1] We use American flag sort, an in-place radix sort.

3.1 A determinant benchmark.

Our first high level benchmark computes the determinant of the $n \times n$ symmetric Toeplitz matrix A for $6 \le n \le 11$. This is a matrix in n variables $x_1, \ldots, x_n$ with x_i appearing along the i^{th} diagonal and i^{th} subdiagonal. We implemented the Bareiss algorithm [2] in Maple and Magma (see Appendix for code) to compute $\det(A)$. At the k^{th} elimination step, ignoring pivoting, the Bareiss algorithm computes

$$A_{i,j} := \frac{A_{k,k}A_{i,j} - A_{i,k}A_{k,j}}{A_{k-1,k-1}} \quad \text{for} \quad i = k+1, \ldots, n \quad \text{and} \quad j = k+1, \ldots, n \tag{1}$$

where the division is exact. At the end of the algorithm $A_{n,n} = \pm\det(A)$. Thus the Bariess algorithm does a sequence of $O(n^3)$ polynomial multiplications and divisions which grow in size, the largest of which occurs at the last step when $k = n-1$.

In Maple 16, the larger multiplications and divisions are done by our external library. This includes our software for parallel polynomial multiplication and parallel polynomial division from [12, 13]. Polynomials are converted from the old sum-of-products representation into our new POLY dag, and back. We observed that for very sparse products, for example $(x + x^2 + \ldots + x^n) \times (y + y^2 + \ldots y^n)$ whose product has n^2 terms, up to 90% of the time is spent converting the POLY dag for the product to the sum-of-products dag, and simplifying it. In our new Maple where the POLY dag is the default; the same library is used but there are now no conversions.

In the table below column #det is the number of terms in the determinant, which has total degree n. Column #num is the number of terms in $A_{n-1,n-1}A_{n,n} - A_{n,n-1}A_{n-1,n}$ which has degree $2n-2$ and is much larger than $\det(A)$. We used a quad core Intel Core i5 CPU @ 2.66 GHz running 64-bit Mac OS X. Timings are real times in seconds, not cpu times. On 4 cores, we achieve a factor of 3 to 4 speedup over Maple 16, which is huge. These gains are entirely from reducing the overhead of Maple data structures; there is no change to the polynomial arithmetic over Maple 16. The reduction of overhead increases parallel speedup to 2.59x, from 1.6x in Maple 16.

			Maple 13	Maple 14		Maple 16		new POLY dag		Magma 2.17
n	#det	#num	1 core	1 core	4 cores	1 core	4 cores	1 core	4 cores	1 core
6	120	575	0.015	0.010	0.010	0.008	0.009	0.002	0.002	0.000 s
7	427	3277	0.105	0.030	0.030	0.030	0.030	0.006	0.006	0.020 s
8	1628	21016	1.123	0.180	0.180	0.181	0.169	0.050	0.040	0.200 s
9	6090	128530	19.176	1.330	1.330	1.450	1.290	0.505	0.329	2.870 s
10	23797	813638	445.611	18.100	13.800	14.830	12.240	6.000	3.420	77.020 s
11	90296	5060172	–	217.020	145.800	151.200	94.340	88.430	34.140	2098.790 s

Table 2: timings (real times in seconds) for determinants using the Bareiss algorithm.

Maple and Magma do not use the Bareiss algorithm to compute these determinants. They use the method of minor expansion as presented by Gentleman and Johnson in [6]. Recall that given an n by n matrix A

$$\det(A) = \sum_{i=1}^{n} (-1)^{n+1} A_{i,1} \det(M(1,i)) \tag{2}$$

where the $M(1,i)$ is $n-1$ by $n-1$ matrix obtained from A by deleting column 1 and row i. If one applies this identity naively, one will recompute determinants of sub-matrices. To avoid recomputation, Gentleman and Johnson compute determinants of the sub-matrices bottom up, that is of size 1×1, 2×2, 3×3, etc. This is still exponential in n; it computes $\binom{n}{k}$ determinants of sub-matrices of size k by k for a total of $\sum_{k=1}^{n} \binom{n}{k} = 2^n - 1$ determinants.

For our Toeplitz matrices, the polynomial multiplications in (2) are all of the form variable $\times$ polynomial. For example, to multiply $f = 9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5$ by y, we add the monomial representation for y, namely $\boxed{1010} = 2^{48} + 2^{16}$ to each monomial in f, namely the integer encodings of $\boxed{5131}$, $\boxed{5032}$, $\boxed{4121}$, $\boxed{3300}$, $\boxed{0000}$. Notice that result will already be sorted in the graded ordering. The polynomial additions and subtractions in (2) are also done in linear time by a simple merge. The improvement is huge. It surprised us.

n	#det	Maple 16	new POLY dag	Magma 2.17
6	120	0.002	0.002	0.001
7	427	0.010	0.004	0.003
8	1628	0.049	0.013	0.019
9	6090	0.305	0.047	0.116
10	23797	1.991	0.252	0.77
11	90296	19.37	1.322	6.21
12	350726	274.99	6.737	44.50
13	1338076	2024.37	39.819	337.77

3.2 A factorization benchmark.

In our second benchmark we see a large gain in performance on polynomial factorization. To provide some perspective, we include timings for Magma [3], Singular [7], Mathematica, and Trip [4], a computer algebra system for celestial mechanics.

We report two times for Trip. The (RS) time is for Trip's optimized recursive sparse polynomial data structure POLYV. The (RD) time is the optimized recursive dense data structure POLPV. Both use multiprecision rational coefficients and Trip's parallel routines [5].

We used an Intel Core i5 750 @ 2.66GHz and an Intel Core i7 920 @ 2.66GHz which had identical times in Maple 16. These are 64-bit quad core cpus. All of the times in the following table are real times, not cpu times, in seconds. Both timings reported for Trip are for 4 cores.

	Maple 13	Maple 16		new POLY dag		Magma	Singular	Mathem	Trip 1.2	
		1 core	4 cores	1 core	4 cores	2.17-1	3-1-4	atica 7.0	(RS)	(RD)
multiply										
$p_1 := f_1(f_1+1)$	1.60	0.053	0.029	0.042	0.017	0.30	0.57	4.79	0.010	0.008
$p_2 := f_2(f_2+1)$	1.55	0.054	0.028	0.042	0.016	0.30	0.58	5.06	0.018	0.016
$p_3 := f_3(f_3+1)$	26.76	0.422	0.167	0.398	0.137	4.09	6.77	50.36	0.088	0.073
$p_4 := f_4(f_4+1)$	95.97	1.810	0.632	1.730	0.508	13.25	30.99	273.01	0.433	0.336
divide										
$q_1 := p_1/f_1$	1.53	0.053	0.026	0.042	0.016	0.36	0.40	6.09	0.200	0.122
$q_2 := p_2/f_2$	1.53	0.053	0.026	0.042	0.018	0.36	0.39	6.53	0.170	0.144
$q_3 := p_3/f_3$	24.74	0.440	0.162	0.402	0.135	4.31	3.64	46.39	1.676	0.950
$q_4 := p_4/f_4$	93.42	1.880	0.662	1.760	0.560	20.23	14.96	242.87	7.292	4.277
factor										
p_1 12341 terms	31.10	2.58	2.46	1.06	0.93	6.15	2.01	11.82		
p_2 12341 terms	296.32	2.86	2.74	1.18	1.06	6.81	2.10	64.31		
p_3 38711 terms	391.44	15.19	13.00	8.22	6.13	117.53	12.48	164.50		
p_4 135751 terms	2953.54	53.52	44.84	26.43	16.17	332.86	61.85	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

Table 3: timings (real times in seconds) for polynomial multiplication, division and factorization. Maple timings are for executing the commands `expand(f1*(f1+1))`, `divide(p1,f1,'q1')` and `factor(p1)`.

There are some anomalies in Table 3. Maple's timings for division on 1 core and 4 cores are very close to those for multiplication. However Singular's division timings for p_3/f_3 and p_4/f_4 are more than twice as fast as the time for multiplication. This is because Singular uses a recursive representation for the polynomials for division but not multiplication. On the other hand, Trip's timings for division are much slower than for multiplication. This is because division in Trip 1.2 has not been parallelized. In comparing the timings for factoring p_1 and p_2 we see that factoring p_2 is much slower in Maple 13 and Mathematica but this is not the case for Maple 16, Magma and Singular. This is because Maple 16, Magma and Singular are using a substitution $p_2(x^2 = u, y^2 = v, z^2 = w)$ to reduce the degree of the input polynomial before factoring it. This halves the number of Hensel lifting steps. We note that Singular timings for factorization have improved from version 3-1-0 to 3-1-4 by a factor of 6. Timings for version 3-1-0 for the four factorizations were 12.28, 23.67, 97.10, 404.86 seconds. The factorization code was changed to use a recursive representation for polynomials by Michael Lee.

The first improvement (compare Maple 13 and Maple 16) is due to our improvements to polynomial multiplication and division in [11, 12, 13] which we reported at ISSAC 2010 in [10]. The speedup for factorization is due to the speedup in polynomial multiplication and division. This is because most of the time in multivariate factorization is spent in "Hensel lifting" which consists of many polynomial multiplications and some exact divisions. We note that Maple's factorization code has not changed since 1984. However, there is little parallel speedup. We achieve significant additional speedup (compare Maple 16 with the new POLY dag) with the POLY dag used by default. For factoring p_4 we obtained a sequential improvement of a factor of $53.52/26.43 = 2.02\times$ and an improvement of a factor of $44.84/16.17 = 2.77\times$. Parallel speedup for factoring p_4 improved from $53.52/44.84 = 1.19\times$ to $26.43/16.17 = 1.63\times$ in our new Maple.

A closer examination of the timings shows that parallel speedup for the multiplication $p_4\times(p_4+1)$, which is a factor of $1.810/0.632 = 2.76\times$ is still quite poor even though our parallel C library for multiplication is 4 times faster on the actual multiplication. Why is this? There are two reasons. One is that on the Core i5, if one uses one core only, that core will run in *turbo boost* mode which on our Core i5 is $\leq$ 3.33Ghz/2.66Ghz $= 1.25\times$ faster. The other reason is the sequential overhead in the integration of our parallel multiplication and division software. For a polynomial multiplication $c := a \times b$, Maple 16 first converts the input polynomials a and b from the sum-of-products data structure to our POLY data structure, then multiplies them using our external parallel C library, which does achieve a factor of 4 speedup on 4 cores, then converts the product c back to Maple's sum-of-products data structure. There is additional sequential overhead required to determine how many words are required to pack the monomials. Maple must compute the union of the sets of variables in a and b and the total degree of a and b in those variables. This involves many passes through the sum-of-products data structures for a, b and c. This overhead is largely eliminated in the new Maple. Adjusting for the turbo boost the parallel speedup in the new Maple for 4 cores is $\frac{1.73}{0.508} \times \frac{3.33}{2.66} = 3.86\times$.

To see where the improvements in the factorization have come from we have profiled the main parts of the factorization code. The profile (see Figure 1) shows the %age of the time in the main parts of the factorization algorithm for Maple 16 and our new Maple. The data under *improved coeftayl* includes a further algorithmic improvement.

The data shows that we have eliminated $0.599 - 0.377 = 0.222s$ of overhead from the polynomial multiplications (see row `expand`) or 37%. The biggest speedup is division (see row `divide`). This is because the divisions are mostly trial divisions which fail quickly. In such cases almost all the time is in conversion which is wasted.

function	Maple 16 time	Maple 16 time%	New Maple time	New Maple time%	improved coeftayl time	improved coeftayl time%
coeftayl	1.086s	41.06	0.310s	28.21	0.095s	12.03
expand	0.506s	19.13	0.263s	23.93	0.255s	32.28
diophant	0.424s	16.03	0.403s	34.94	0.299s	37.85
divide	0.256s	9.68	0.034s	3.09	0.035s	4.43
factor	0.201s	7.60	0.011s	1.00	0.010s	1.27
factor/hensel	0.127s	4.80	0.064s	5.82	0.063s	7.97
factor/unifactor	0.045s	1.70	0.033s	3.00	0.033s	4.18
total:	2.645s	100.00%	1.099s	100.00%	0.790s	100.00%

Figure 1: profile for `factor(p1);` (1 core).

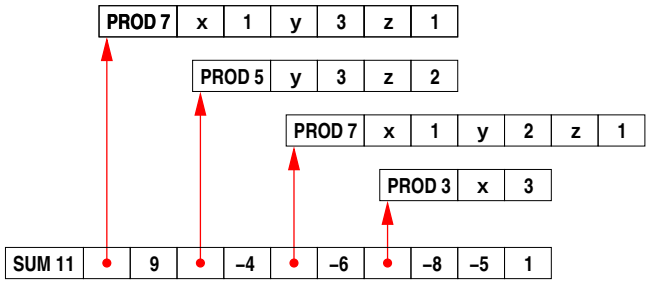
The biggest absolute gain is for the routine `coeftayl(f,x-a,k)` which computes the coefficient of f in $(x-a)^k$. This computation is not done by expanding f as a Taylor series about $x=a$ but rather by using the formula $g(x=a)/k!$ where $g = \frac{\mathrm{d}f}{\mathrm{d}^k x}$, the k'th derivative of f. Referring back to Table 1, we can see that the speedup is due to the improvement of differentiation and polynomial evaluation. We also tried the following formula to compute the coefficient

$$\sum_{i=k}^{\deg_x f} \mathrm{coeff}(f, x^i) a^i \binom{i}{k}.$$

We can see that this is 3× faster again (see improved coeftayl). The total real time is reduced from 2.59s to 1.07s to 0.790s.

4 Conclusion

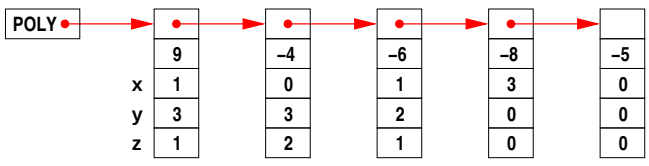
Maple, Mathematica, Magma and Singular all use a distributed representation for multivariate polynomials. Maple's sum-of-products data structure and Singular's linked list data structure are illustrated in the figure below. Mathematica's data structure is similar to Maple's and Magma's data structure is similar to Singular's. These data structures, which were designed in the 1980s when memory access was constant time, will not yield high-performance on todays computers because memory access is not sequential.



Maple's sum-of-products representation.

Representations for the polynomial

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



Singular's linked list representation.

One way to speed up polynomial multiplication, division, or factorization would be to convert the input to a more suitable data structure, compute the result, then convert back. This is what we did in [10] for Maple 14 for polynomial multiplication and division. Singular does this for polynomial division and factorization. It switches to using a recursive representation. However, the conversion overhead will

limit parallel speedup. Ahmdah's law states that if the sequential proportion of a task is S then parallel speedup on N cores is limited to

$$speedup \leq \frac{1}{S + (1-S)/N}.$$

When S is large (30% or more say), then in order to get good parallel speedup we have to also speed up the sequential part of the problem.

What we have done in this work for Maple is to make our POLY data structure the default data structure in Maple. The POLY data structure is used when all monomials in a polynomial can be packed into a single word. This enabled us to eliminate conversion overhead in multiplication and division. The data in Table 3 shows improved parallel speedup for polynomial multiplication and division. However, we were also able to implement highly efficient algorithms for many Maple kernel operations. This substantially improved the speedup of multivariate polynomial factorization. The data in Table 3 shows speedups of factors of between 2 and 3 for large polynomial factorizations which is a huge gain. Although not reported here, we also find speedups of a factor of 2 for large multivariate polynomial gcd computations.

The cost incurred is mainly in code complexity. We must manage two data structures for polynomials, one where the coefficients are integers and the monomials can be packed into a single machine word, and one, Maple's sum-of-products data structure, which does not have these restrictions. The gains suggest this is worthwhile.

References

[1] O. Bachmann and H. Schönemann. Monomial representations for Gobner bases computations. *Proceedings of ISSAC '98*, pp. 309–316, 1998.

[2] E. Bariess, 1968. Sylvester's Identity and Multistep Integer-Preserving Gaussian Elimination, *Mathematics of computation* **22** (102): 565−578.

[3] Bosma, W., Cannon, J., Playoust, C., 1997. The Magma Algebra System I: The User Language. *J. Symb. Cmpt.* **24**(3-4), 235–265. See also `http://magma.maths.usyd.edu.au/magma`

[4] Gastineau, M., Laskar, J., 2006. Development of TRIP: Fast Sparse Multivariate Polynomial Multiplication Using Burst Tries. *Proceedings of ICCS 2006*, Springer LNCS **3992**, pp. 446–453.

[5] Gastineau, M., 2010. Parallel operations of sparse polynomials on multicores - I. Multiplication and Poisson bracket. *Proceedings of PASCO '2010*, ACM Press, pp. 44–52, 2010.

[6] Gentleman, W.M., Johnson, S.C. Analysis of Algorithms, A Case Study: Determinants of Matrices with Polynomial Entries. *ACM Transactions on Mathematical Software*, **2**(3), pp. 232–241, September 1976.

[7] 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

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

[9] Peter M. McIlroy, Keith Bostic, and M. Douglas McIlroy. Engineering Radix Sort, Computing Systems, **6**(1): 5–27, 1993.

[10] Monagan, M., Pearce, R., 2010. Sparse Polynomial Multiplication and Division in Maple 14. *Communications in Computer Algebra*, **44**:4, 205–209, December 2010.

[11] Monagan, M., Pearce, R., 2011. Sparse Polynomial Division using Heaps. *J. Symb. Cmpt.* **46**(7):807–822, 2011.

[12] M. Monagan, R. Pearce., 2009. Parallel Sparse Polynomial Multiplication Using Heaps. *Proceedings of of ISSAC 2009*, ACM Press, pp. 295–315.

[13] M. Monagan, R. Pearce., 2010. Parallel Sparse Polynomial Division Using Heaps. *Proc. of PASCO 2010*, ACM Press, pp. 105–111.

[14] Warren, Henry S. Hacker's Delight. Addison-Wesley, 2003.

Appendix

Maple code (no pivoting) for the Bareiss algorithm.

```
ffge := proc(A,n) local d,i,j,k,t;
   d := 1;
   for k to n-1 do
       for i from k+1 to n do
           for j from k+1 to n do
               t := expand(A[k,k]*A[i,j]-A[i,k]*A[k,j]);
               divide(t, d, evaln(A[i,j]));
           od;
           A[i,k] := 0;
       od;
       d := A[k,k];
   od:
   A[n,n];
end;
n := 8;
T := linalg[toeplitz]([seq(x[i],i=1..n)]);
A := array(1..n,1..n):
for i to n do for j to n do A[i,j] := T[i,j] od od:
det := CodeTools[Usage]( ffge(A,n) ):
```

Magma code for the Bareiss algorithm.

```
Z := IntegerRing();
P<x,y,z,u,v,w,p,q,r,s,t,a> := PolynomialRing(Z,12);
X := [x,y,z,u,v,w,p,q,r,s,t,a];
n := 8;
A := Matrix(P,n,n,[0 : i in [1..n^2]]);
for i in [1..n] do
    for j in [1..n] do
        A[i,j] := X[AbsoluteValue(j-i)+1];
    end for;
end for;
d := 1;
time for k in [1..n-1] do
         for i in [k+1..n] do
            for j in [k+1..n] do
                t := A[k,k]*A[i,j]-A[i,k]*A[k,j];
                A[i,j] := ExactQuotient(t,d);
            end for;
         end for;
         d := A[k,k];
     end for;
det := A[n,n];
```