How fast can we multiply and divide polynomials?


Michael Monagan
Center for Experimental and Constructive Mathematics,
Simon Fraser University,
Vancouver, British Columbia,
CANADA.

Joint work with Roman Pearce.
The Mathematics Of Computer Algebra and Analysis project.

\[ \text{factor}(2 x^3 + x^2 y^2 - x^3 y + x^2 - 5 y x - 3 y^3 + 3 y^2 x - 3 y - 1); \]

\[ \text{solve}(\{ x^2 + y^2 + z^2 - 4 , \ x y z + 2 , \ x y + z^3 - 1 \}); \]

\[ \text{Determinant}( \begin{bmatrix} t & 1 - 2 t & 1 \\ t^2 & t & 1 \\ 1 + t & 1 & 1 + t + t^2 \end{bmatrix} ); \]

\[ \int x^2 \ln(x)e^{-x} + (1 - x) \ln(x)e^{-x} - 2xe^{-x} \ dx; \]
The Mathematics Of Computer Algebra and Analysis project.

\[ \text{factor}(2x^3 + x^2y^2 - x^3y + x^2 - 5yx - 3y^3 + 3y'^2x - 3y - 1); \]

\[ \text{solve}\left\{ x^2 + y^2 + z^2 - 4, \; xy + 2, \; xy + z^3 - 1 \right\}; \]

\[ \text{Determinant}\left( \begin{bmatrix} t & 1-2t & 1 \\ t^2 & t & 1 \\ 1+t & 1 & 1+t+t^2 \end{bmatrix} \right); \]

\[ \int x^2 \ln(x)e^{-x} + (1-x) \ln(x)e^{-x} - 2xe^{-x} \; dx; \]

**Risch**

\[
\begin{align*}
e^{-x} & \rightarrow \theta_1 \\
\ln x & \rightarrow \theta_2 \\
\int & \left\{ \frac{a \text{ polynomial}}{x^2 \theta_2 \theta_1 + (1-x)\theta_2 \theta_1 - 2x\theta_1} \right\} \; dx \quad \text{where} \quad \theta_1' = -\theta_1 \\
\theta_2' & = 1/x.
\end{align*}
\]
Polynomials are the key!

Talk Outline:

- How do CAS represent polynomials?
- How do CAS multiply and divide polynomials?
- Our representation and algorithms.
- How fast we compared with other CAS?
- Immediate Monomial Project (for Maple 15)
- Parallel Multiplication (for Maple 15)
How do CAS *represent* polynomials?
Recursive and distributed polynomial representations.

The **distributed** representation: monomials $x^i y^j z^k$ are sorted in *lexicographical order* (Magma, Mathematica):

$$f = -6x^3 + 9xy^3 z - 8xy^2 z + 7y^2 z^2 + 5$$

or *graded lex order* (Singular, Maple 15):

$$f = 9xy^3 z - 8xy^2 z + 7y^2 z^2 - 6x^3 + 5.$$ 

Key property: if $X, Y, Z$ are monomials then $Y > Z \implies XY > XZ.$
Recursive and distributed polynomial representations.

The **distributed** representation: monomials $x^i y^j z^k$ are sorted in 
*lexicographical order* (Magma, Mathematica):

$$f = -6x^3 + 9xy^3z - 8xy^2z + 7y^2z^2 + 5$$

or *graded lex order* (Singular, Maple 15):

$$f = 9xy^3z - 8xy^2z + 7y^2z^2 - 6x^3 + 5.$$  

Key property: if $X, Y, Z$ are monomials then $Y > Z \implies XY > XZ$.

The **recursive** representation (Macsyma, REDUCE, Derive, Pari):

$$f = (-6)x^3 + ( (9z)y^3 + (-8z)y^2)x^1 + ((7z^2)y^2 + 5y^0)x^0.$$
Maple’s sum of products representation.

\[
\begin{align*}
\text{PROD 7} & \quad x & 1 & y & 3 & z & 1 \\
\text{PROD 5} & \quad y & 3 & z & 2 \\
\text{PROD 7} & \quad x & 1 & y & 2 & z & 1 \\
\text{PROD 3} & \quad x & 3 \\
\text{SUM 11} & \quad 9 & -4 & -6 & -8 & -5 & 1
\end{align*}
\]

\[9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5\]

Singular’s distributed representation.

\[
\begin{align*}
\text{POLY} & \quad 9 & -4 & -6 & -8 & -5 \\
x & 1 & 0 & 1 & 3 & 0 \\
y & 3 & 3 & 2 & 0 & 0 \\
z & 1 & 2 & 1 & 0 & 0
\end{align*}
\]
Trip’s recursive sparse representation.

\[ (-5y - 4z^2y^3) + (-6zy^2 + 9zy^3)x - 8x^3 \]

Pari’s recursive dense representation.
Our representation uses packed monomials.

Packing for $x^i y^j z^k$ in graded lex order with $x > y > z$:

One 64 bit word: $\begin{bmatrix} i + j + k & i & j & k \end{bmatrix}$.

$(i + j + k)2^{48} + 2^{32}i + 2^{16}j + k$.

Why?
Our representation uses packed monomials.

Packing for \( x^i y^j z^k \) in **graded lex order** with \( x > y > z \):

One 64 bit word: \[
\begin{array}{ccc}
  i + j + k & i & j & k \\
\end{array}
\]

\[(i + j + k)2^{48} + 2^{32}i + 2^{16}j + k.\]

Why? Because monomial > and \( \times \) are one machine instruction.
Our representation uses packed monomials.

Packing for $x^i y^j z^k$ in **graded lex order** with $x > y > z$:

One 64 bit word: $\begin{array}{c}
i+j+k \\
i \\
j \\
k \end{array}$.

$$(i+j+k)2^{48} + 2^{32}i + 2^{16}j + k.$$ 

**Why?** Because monomial $>$ and $\times$ are one machine instruction.

Our packed array for $9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5$.

<table>
<thead>
<tr>
<th>POLY 5</th>
<th>d = total degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>x y z</td>
<td>dxyz</td>
</tr>
<tr>
<td>packing</td>
<td>5131 9 5032 -4 4121 -6 3300 -8 0000 -5</td>
</tr>
</tbody>
</table>

Why **graded lex order**?
Our representation uses packed monomials.

Packing for $x^i y^j z^k$ in **graded lex order** with $x > y > z$:

One 64 bit word: $i + j + k \ b i \ b j \ b k$.

$(i + j + k)2^{48} + 2^{32}i + 2^{16}j + k$.

Why? Because monomial $>$ and $\times$ are one machine instruction.

Our packed array for $9xyz - 4y^3z^2 - 6xy^2z - 8x^3 - 5$.

| POLY 5 |  
|-------|---|---|---|---|---|---|---|
| x     | y | z | packing | dxyz | dxyz | dxyz | dxyz | dxyz |
| 9     | 5032 | -4 | 4121 | -6 | 3300 | -8 | 0000 | -5 |

Why **graded lex order**? No exponent overflow in division.
How do CAS multiply and divide polynomials?
Let $f = f_1 + f_2 + \cdots + f_n$ and $g = g_1 + g_2 + \cdots + g_m$ where $f_1 > f_2 > \cdots > f_n$ and $g_1 > g_2 > \cdots > g_m$.

Using

$$h = f \times g = ((f_1g + f_2g) + f_3g) + \cdots + f_ng$$
and

$$h ÷ g = f : (((h - f_1g) - f_2g) - f_3g) - \cdots - f_ng$$
Let \( f = f_1 + f_2 + \cdots + f_n \) and \( g = g_1 + g_2 + \cdots + g_m \) where \( f_1 > f_2 > \cdots > f_n \) and \( g_1 > g_2 > \cdots > g_m \).

Using

\[
    h = f \times g = ((f_1g + f_2g) + f_3g) + \cdots + f_ng \quad \text{and}
\]

\[
    h ÷ g = f : (((h - f_1g) - f_2g) - f_3g) - \cdots - f_ng
\]

takes \( O(n^2m) \) comparisons of monomials and \( O(nm) \) multiplications of coefficient and monomials.

Example:
\[
    f = x^n + x^{n-1} + \cdots + x \quad \text{and} \quad g = y^n + y^{n-1} + \cdots + y.
\]
Our algorithms for multiplication and division use heaps.
Heaps

A binary heap $H$ with $n$ entries is a partially ordered array satisfying

$$H_i \geq H_{2i} \quad \text{and} \quad H_i \geq H_{2i+1}.$$  

<table>
<thead>
<tr>
<th>$H_1$</th>
<th>$H_2$</th>
<th>$H_3$</th>
<th>$H_4$</th>
<th>$H_5$</th>
<th>$H_6$</th>
<th>$H_7$</th>
<th>$H_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>9</td>
<td>8</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>7</td>
<td>—</td>
</tr>
</tbody>
</table>

- $H_1$ is the biggest entry in a heap.
- We can extract the maximum entry in $O(\log_2 n)$ comparisons.
- We can insert a new entry in $O(\log_2 n)$ comparisons.
Multiplication using a binary heap.

Johnson, 1974, a simultaneous $n$-ary merge:

\[ f = a_1 X_1 + a_2 X_2 + \cdots + a_n X_n \]
\[ g = b_1 Y_1 + b_2 Y_2 + \cdots + b_m Y_m \] (sorted)

\[ f_1 g_1 + \cdots + f_n g_n \]

\[ O(nm \log n) \] comparisons.

\[ \text{Space for } \leq n \text{ monomials in the heap.} \]

\[ \text{Can pick } n \leq m. \]
Division using a heap.

Johnson’s quotient heap algorithm.

\[
\text{Dividing } f \div g = q \text{ compute } \quad f - \sum_{i=1}^{\#q} q_i \times g
\]

- \(O(\#f + \#q \#g \log \#q)\) comparisons
- \(O(\#q)\) working memory

Our divisor heap algorithm.

\[
\text{Dividing } f \div g = q \text{ compute } \quad f - \sum_{i=2}^{\#g} g_i \times q
\]

- \(O(\#f + \#q \#g \log \#g)\) comparisons
- \(O(\#g)\) working memory
Minimal heap division (Monagan & Pearce, 2008)

**Problem:** we don’t know if $\#q > \#g$ when starting a division.

E.g. \((x^7 - y^7) ÷ (x - y) = x^6 + yx^5 + y^2x^4 + \cdots + y^6\).

Start with quotient heap, switch to divisor heap when $\#q = \#g$.

\[
f = \min(\#q, \#g) \sum_{i=1}^{\#q} q_i \times g - \sum_{i=2}^{\#g} g_i \times (q_{\#g+1} + \cdots)
\]

- $O(\#f + \#q \#g \log \min(\#q, \#g))$ comparisons
- $O(\min(\#q, \#g))$ working memory
Which CAS is fastest?
Benchmark 1: A dense Fateman problem.

\[ f = (1 + x + y + z + t)^{20} \quad g = f + 1 \]

- \( f \) and \( g \) have 39 bit coefficients and 10,626 terms
- \( h = f \cdot g \) has 83 bit coefficients and 135,751 terms

<table>
<thead>
<tr>
<th>Intel Core2 3.0 GHz</th>
<th>multiply ( p = f \times g )</th>
<th>divide ( q = p/f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maple 12</td>
<td>289.23 s</td>
<td>187.72 s</td>
</tr>
<tr>
<td>Maple 13</td>
<td>187.35 s</td>
<td>159.12 s</td>
</tr>
<tr>
<td>Singular 3-0-4</td>
<td>62.00 s</td>
<td>20.00 s</td>
</tr>
<tr>
<td>Magma V2.14-7</td>
<td>23.02 s</td>
<td>22.76 s</td>
</tr>
<tr>
<td>Pari 2.3.3 (w/ GMP)</td>
<td>32.43 s</td>
<td>14.76 s</td>
</tr>
<tr>
<td>Trip v0.99</td>
<td>5.93 s</td>
<td>-</td>
</tr>
<tr>
<td>sdmp (unpacked)</td>
<td>5.15 s</td>
<td>5.44 s</td>
</tr>
<tr>
<td>sdmp (packed)</td>
<td><strong>2.26 s</strong></td>
<td><strong>2.77 s</strong></td>
</tr>
<tr>
<td>Maple 14</td>
<td><strong>3.33 s</strong></td>
<td><strong>4.46s</strong></td>
</tr>
</tbody>
</table>
Benchmark 2: A sparse 10 variable problem.

\[ f = (x_1 x_2 + x_2 x_3 + x_3 x_4 + x_4 x_5 + x_5 x_6 + x_6 x_7 + x_7 x_8 + x_8 x_9 + x_9 x_{10} + x_{11} x_{10} + x_{10} + 1)^4 \]

\[ g = (x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2 + x_7^2 + x_8^2 + x_9^2 + x_{10}^2 + x_9 + x_{10} + 1)^4 \]

<table>
<thead>
<tr>
<th>6,746 × 8,361 = 3,157,883 terms</th>
<th>multiply ( p = f \times g )</th>
<th>divide ( q = p / f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maple 12</td>
<td>305.76s</td>
<td>280.65s</td>
</tr>
<tr>
<td>Maple 13</td>
<td>293.74s</td>
<td>312.29s</td>
</tr>
<tr>
<td>Singular 3-0-4</td>
<td>31.00s</td>
<td>18.00s</td>
</tr>
<tr>
<td>Magma V2.14-7</td>
<td>17.43s</td>
<td>197.72s</td>
</tr>
<tr>
<td>Pari 2.3.3 (w/ GMP)</td>
<td>7.06s</td>
<td>7.05s</td>
</tr>
<tr>
<td>Trip v0.99 (rationals)</td>
<td>8.13s</td>
<td>–</td>
</tr>
<tr>
<td>sdmp (unpacked)</td>
<td>11.12s</td>
<td>10.37s</td>
</tr>
<tr>
<td><strong>sdmp (packed)</strong></td>
<td><strong>2.46s</strong></td>
<td><strong>2.61s</strong></td>
</tr>
<tr>
<td>Maple 14</td>
<td>11.74s</td>
<td>14.45s</td>
</tr>
</tbody>
</table>
Benchmark 3: Factorization speedup in Maple 14.

In Maple 13,
\[
> \ h := \text{expand}(f*g); \\
> \ \text{divide}(h,f,'q');
\]
call ‘expand/bigprod’(f,g) and ‘expand/bigdiv’(h,f,q) for large inputs.

In Maple 14, we reprogrammed ‘expand/bigprod’ and ‘expand/bigdiv’ to convert to SDMP, multiply (divide) in SDMP, then convert back to Maple.

<table>
<thead>
<tr>
<th>Benchmark (a =)</th>
<th>factor( (h) ) where (h = (a + 1)(a + 2))</th>
</tr>
</thead>
<tbody>
<tr>
<td>((x + y + z)^{30})</td>
<td>Maple13 368.88  Magma 4.47  Maple14 18.61  Speedup 19.8 x</td>
</tr>
<tr>
<td>((1 + x + y + z)^{20})</td>
<td>Maple13 38.38  Magma 10.95  Maple14 4.01  Speedup 9.6 x</td>
</tr>
<tr>
<td>((1 + x + y + z)^{30})</td>
<td>Maple13 679.01  Magma 400.4  Maple14 23.38  Speedup 29.0 x</td>
</tr>
<tr>
<td>((1 + x + y + z + t)^{20})</td>
<td>Maple13 5390.32  Magma 1286.8  Maple14 99.00  Speedup 54.4 x</td>
</tr>
</tbody>
</table>

**Table:** Factorization Benchmark Timings (in CPU seconds)
The Immediate Monomial Project.

A joint MITACS project with Maplesoft.
A new data structure being implemented by Paul de Marco.

<table>
<thead>
<tr>
<th>POLY 12</th>
<th>5131</th>
<th>9</th>
<th>5032</th>
<th>-4</th>
<th>4121</th>
<th>-6</th>
<th>3300</th>
<th>-8</th>
<th>0000</th>
<th>-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQ 4</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How will we pack monomials? E.g. $x^i y^j z^k$ on a 64 bit computer.
Always try to pack all monomials into one word.
If $i + j + k < 2^{16}$ pack $i+j+k$ \( i \) \( j \) \( k \) in one word.
If $i + j + k \geq 2^{16}$ use Maple’s existing representation.
So the number of variables determines the packing.
Let $f(x_1, x_2, \cdots, x_n) = f_1 + f_2 + \cdots + f_m$.

$O(nm) \Rightarrow O(1): \text{lcoeff}(f); \text{degree}(f); \text{indets}(f);$  
$O(nm) \Rightarrow O(n): f; \text{has}(f,x); \text{type}(f,\text{polynom}(\text{integer}));$  
$O(nm) \Rightarrow O(m): \text{degree}(f,x); \text{diff}(f,x); \text{coeffs}(f,x);$  

A 10 – 20% gain in overall efficiency gain for Maple 14?
Paralllelizing Multiplication Using Heaps

Intel Core i7.
Parallel Algorithm

One heap per core, merge results in a global heap.

Don’t waste real or cpu time:

- partition terms
- transfer data
- balance load
Transferring Data

Threads write to a finite circular buffer.

```c
#define N 32768 /* size in words (256 K) */
#define CLINE 64 /* bytes per cache line */
struct buffer {
    long r;       /* words read */
    char pad1[CLINE - sizeof(long)];
    long w;       /* words wrote */
    char pad2[CLINE - sizeof(long)];
    long data[N]; }
```

![Diagram of buffer structure](image)

Intel Core i7
threads try to acquire a lock for the global heap
one thread per core avoids context switches and OS
threads independently adjust their share of global work

buffer full → do more global work
buffer empty → do less global work
Dense Benchmark

\[ f = (1 + x + y + z + t)^{30} \quad g = f + 1 \]

\[ 46376 \times 46376 = 635376 \text{ terms} \quad W(f, g) = 3332 \]

<table>
<thead>
<tr>
<th>threads</th>
<th>Core i7 2.66GHz</th>
<th>Core 2 2.4GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>sdmp (packed)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>11.48 s</td>
<td>14.15 s</td>
</tr>
<tr>
<td></td>
<td>6.15x</td>
<td>4.25x</td>
</tr>
<tr>
<td>3</td>
<td>16.63 s</td>
<td>19.43 s</td>
</tr>
<tr>
<td></td>
<td>4.24x</td>
<td>3.10x</td>
</tr>
<tr>
<td>2</td>
<td>28.26 s</td>
<td>28.29 s</td>
</tr>
<tr>
<td></td>
<td>2.50x</td>
<td>2.13x</td>
</tr>
<tr>
<td>1</td>
<td>70.59 s</td>
<td>60.25 s</td>
</tr>
</tbody>
</table>

| Trip 1.0 beta2 recursive dense |
| 4       | 23.76 s       | 26.86 s       |
|         | 3.89x         | 3.94x         |
| 3       | 31.05 s       | 35.65 s       |
|         | 2.97x         | 2.97x         |
| 2       | 46.56 s       | 52.98 s       |
|         | 1.98x         | 1.99x         |
| 1       | 92.38 s       | 105.78 s      |

| Trip 1.0 beta2 recursive sparse |
| 4       | 29.36 s       | 31.95 s       |
|         | 3.26x         | 3.38x         |
| 3       | 36.00 s       | 39.96 s       |
|         | 2.66x         | 2.71x         |
| 2       | 50.96 s       | 56.68 s       |
|         | 1.88x         | 1.91x         |
| 1       | 95.74 s       | 108.15 s      |

| Magma 2.15-8 | 1 | 526.12 s |
| Pari/GP 2.3.3 | 1 | 642.74 s |
| Singular 3-1-0 | 1 | 744.00 s |
| Maple 13     | 1 | 5849.48 s |
Parallel Speedup: Core i7

- \( W(f,g) = 2737 \) : 24500 terms
- \( W(f,g) = 2040 \) : 33000 terms
- \( W(f,g) = 133.7 \) : 502000 terms
- \( W(f,g) = 41.11 \) : 1.63 M terms
- \( W(f,g) = 21.72 \) : 3.09 M terms
- \( W(f,g) = 11.16 \) : 6.01 M terms
- \( W(f,g) = 5.912 \) : 11.4 M terms
- \( W(f,g) = 3.637 \) : 18.4 M terms
- \( W(f,g) = 2.054 \) : 32.7 M terms
- \( W(f,g) = 1.361 \) : 49.3 M terms
- \( W(f,g) = 1.021 \) : 65.7 M terms

- dense
- sparse

- random univariate polynomials
- \( 8192 \times 8192 = 67.1 \times 10^6 \) products
- linear speedup @ \( 18.4 \times 10^6 \) terms
- 5x faster @ \( 1.63 \times 10^6 \) terms