What’s the best data structure for multivariate polynomials in a world of 64 bit multicore computers?

Michael Monagan

Center for Experimental and Constructive Mathematics
Simon Fraser University
British Columbia

ECCAD 2013, Annapolis, Maryland
April 27, 2012

This is joint work with Roman Pearce.
Representations for $9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5$.

Maple 16

PROD 7  x  1  y  3  z  1
PROD 5  y  3  z  2
PROD 7  x  1  y  2  z  1
PROD 3  x  3
SUM 11  9  -4  -6  -8  -5  1

Memory access is not sequential.
Monomial multiplication costs $O(100)$ cycles.

Michael Monagan
ECCAD, Annapolis, 2013
Representations for $9 \, x y^3 z - 4 \, y^3 z^2 - 6 \, x y^2 z - 8 \, x^3 - 5$.

Maple 16

Singular 3.1.0

- Memory access is not sequential.
- Monomial multiplication costs $O(100)$ cycles.

Michael Monagan  
ECCAD, Annapolis, 2013
Our representation \(9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5\).

Monomial encoding for graded lex order with \(x > y > z\)

Monomial > and \(\times\) cost one instruction !!!!

Advantages
Our representation $9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5$.

<table>
<thead>
<tr>
<th>SEQ 4</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>POLY 12</td>
<td>5131</td>
<td>9</td>
<td>5032</td>
</tr>
</tbody>
</table>

Monomial encoding for graded lex order with $x > y > z$

Monomial $>$ and $\times$ cost one instruction !!!!

Advantages

- It’s about four times more compact.
- Memory access is sequential.
- The simpl table is not filled with PRODs.
- Division cannot cause exponent overflow in a graded lex order.
Core i7 920 @ 2.67 GHz
45nm lithography, Q4 2008

Core i7-3930K @ 3.20 GHz
32 nm lithography, Q4 2011
Overclocked @ 4.2 GHz
Multicore Computers: AMD FX 8350 Intel i7 4770

**AMD FX 8350 @ 4.2 GHz**
- 8 core
- 32nm
- Q4, 2012
- Full integer support.

**Intel Core i7-4770 @ 3.5 GHz**
- 4 core
- 22 nm
- Q2 2013
- Only 5–10% faster.

How should we parallelize Maple?
How would that speed up polynomial factorization?
Let’s parallelize polynomial multiplication and division.

- Johnson’s sequential polynomial multiplication
- Our parallel polynomial multiplication
- A multiplication and factorization benchmark

Why is parallel speedup poor?

- Maple 17 integration of POLY
- New timings for same benchmark.
- Notes on integration into Maple 17 kernel.
- Future work.
Sequential multiplication using a binary heap.

Let \( f = f_1 + \cdots + f_n = c_1 X_1 + \cdots c_n X_n. \)
Let \( g = g_1 + \cdots + g_m = d_1 Y_1 + \cdots d_m Y_m. \)
Compute \( f \times g = f_1 \cdot g + f_2 \cdot g + \cdots + f_n \cdot g. \)

Johnson (1974) simultaneous \( n \)-ary merge (heap): \( O(mn \log n). \)
Sequential multiplication using a binary heap.

Let $f = f_1 + \cdots + f_n = c_1 X_1 + \cdots c_n X_n$.
Let $g = g_1 + \cdots + g_m = d_1 Y_1 + \cdots d_m Y_m$.
Compute $f \times g = f_1 \cdot g + f_2 \cdot g + \cdots + f_n \cdot g$.

Johnson (1974) simultaneous $n$-ary merge (heap): $O(mn \log n)$.

$|Heap| \leq n \implies O(nm \log n)$ comparisons.

Delay coefficient arithmetic to eliminate garbage!
Parallel multiplication using a binary heap.

Target architecture

One thread per core.

Local Heaps

Global Heap

Threads try to acquire global heap as buffer fills up to balance load.

Threads write to a finite circular buffer.

Michael Monagan  ECCAD, Annapolis, 2013
Parallel multiplication using a binary heap.

Target architecture

Local Heaps

Global Heap

\[ g \]
\[ f \]

One thread per core.

Threads write to a finite circular buffer.

\[ r \quad w \]

\[ r \mod N \quad w \mod N \]

Threads try to acquire global heap as buffer fills up to balance load.
Maple 16 multiplication and factorization benchmark.

<table>
<thead>
<tr>
<th>multiply</th>
<th>Maple 16</th>
<th>Maple 16</th>
<th>Magma 2.16-8</th>
<th>Singular 3.1.0</th>
<th>Mathem atica 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1 := f_1(f_1 + 1)$</td>
<td>1.60</td>
<td>0.063</td>
<td>0.30</td>
<td>0.58</td>
<td>4.79</td>
</tr>
<tr>
<td>$p_4 := f_4(f_4 + 1)$</td>
<td>95.97</td>
<td>2.14</td>
<td>13.25</td>
<td>30.64</td>
<td>273.01</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>factor</th>
<th>Hensel lifting is mostly polynomial multiplication!</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$ 12341 terms</td>
<td>31.10</td>
</tr>
<tr>
<td>$p_4$ 135751 terms</td>
<td>2953.54</td>
</tr>
</tbody>
</table>

$f_1 = (1 + x + y + z)^{20} + 1$                                         1771 terms
$f_4 = (1 + x + y + z + t)^{20} + 1$                                     10626 terms

Parallel speedup for $f_4 \times (f_4 + 1)$ is $2.14 / .643 = 3.33 \times$. Why?
To expand sums $f \times g$ Maple calls ‘expand/bigprod(f,g)’
if $\#f > 2$ and $\#g > 2$ and $\#f \times \#g > 1500$.

‘expand/bigprod’ := proc(a,b) # multiply two large sums
    if type(a,polynom(integer)) and type(b,polynom(integer)) then
        x := indets(a) union indets(b); k := nops(x);
        A := sdmp:-Import(a, plex(op(x)), pack=k);
        B := sdmp:-Import(b, plex(op(x)), pack=k);
        C := sdmp:-Multiply(A,B);
        return sdmp:-Export(C);
    else
        ...
    end;

‘expand/bigdiv’ := proc(a,b,q) # divide two large sums
    ...
    x := indets(a) union indets(b); k := nops(x)+1;
    A := sdmp:-Import(a, grlex(op(x)), pack=k);
    B := sdmp:-Import(b, grlex(op(x)), pack=k);
    ...

Make POLY the default representation in Maple.

If we can pack all monomials into one word use

\[
\begin{array}{cccc}
\text{SEQ} & 4 & x & y & z \\
\end{array}
\]

\[
\begin{array}{cccccccccccc}
\text{POLY} & 12 & 5131 & 9 & 5032 & -4 & 4121 & -6 & 3300 & -8 & 0000 & -5 \\
\end{array}
\]

otherwise use the sum-of-products structure.
Make POLY the default representation in Maple.

If we can pack all monomials into one word use

\[
\text{SEQ 4 } x \ y \ z
\]

\[
\text{POLY 12 } 5131 \ 9 \ 5032 \ -4 \ 4121 \ -6 \ 3300 \ -8 \ 0000 \ -5
\]

otherwise use the sum-of-products structure.

**But must reprogram entire Maple kernel for new POLY !!**

\[
O(1) \quad \text{degree}(f); \ \text{lcoeff}(f); \ \text{indets}(f);
\]
\[
O(n + t) \quad \text{degree}(f,x); \ \text{expand}(x*t); \ \text{diff}(f,x);
\]

For \( f \) with \( t \) terms in \( n \) variables.
To compute \( \text{coeff}(f,y,3) \) we need to

We can do step 1 in \( O(1) \) bit operations.

Can we do step 2 faster than \( O(n) \) bit operations?
High performance solutions.

/* pre-compute masks for compress_fast */
static void compress_init(M_INT mask, M_INT *v)

/* compress monomial m using precomputed masks v */
/* in O( log_2 WORDSIZE ) bit operations */
static M_INT compress_fast(M_INT m, M_INT *v)
{
    M_INT t;
    if (v[0]) t = m & v[0], m = m ^ t | (t >> 1);
    if (v[1]) t = m & v[1], m = m ^ t | (t >> 2);
    if (v[2]) t = m & v[2], m = m ^ t | (t >> 4);
    if (v[3]) t = m & v[3], m = m ^ t | (t >> 8);
    if (v[4]) t = m & v[4], m = m ^ t | (t >> 16);
    #if WORDSIZE > 32
    if (v[5]) t = m & v[5], m = m ^ t | (t >> 32);
    #endif
    return m;
}

- Costs 24 bit operations per monomial.
- Intel Haswell (2013): 1 cycle (PEXT/PDEP)
Result: everything except op and map is fast!

<table>
<thead>
<tr>
<th>command</th>
<th>Maple 16</th>
<th>Maple 17</th>
<th>speedup</th>
<th>notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>coeff($f, x, 20$)</td>
<td>2.140 s</td>
<td>0.005 s</td>
<td>420x</td>
<td>terms easy to locate</td>
</tr>
<tr>
<td>coeff($f, x$)</td>
<td>0.979 s</td>
<td>0.119 s</td>
<td>8x</td>
<td>reorder exponents and radix sort</td>
</tr>
<tr>
<td>frontend($g, [f]$)</td>
<td>3.730 s</td>
<td>0.000 s</td>
<td>$\rightarrow O(n)$</td>
<td>looks at variables only</td>
</tr>
<tr>
<td>degree($f, x$)</td>
<td>0.073 s</td>
<td>0.003 s</td>
<td>24x</td>
<td>stop early using monomial degree</td>
</tr>
<tr>
<td>diff($f, x$)</td>
<td>0.956 s</td>
<td>0.031 s</td>
<td>30x</td>
<td>terms remain sorted</td>
</tr>
<tr>
<td>eval($f, x = 6$)</td>
<td>3.760 s</td>
<td>0.175 s</td>
<td>21x</td>
<td>use Horner form recursively</td>
</tr>
<tr>
<td>expand($2 * x * f$)</td>
<td>1.190 s</td>
<td>0.066 s</td>
<td>18x</td>
<td>terms remain sorted</td>
</tr>
<tr>
<td>indets($f$)</td>
<td>0.060 s</td>
<td>0.000 s</td>
<td>$\rightarrow O(1)$</td>
<td>first word in dag</td>
</tr>
<tr>
<td>op($f$)</td>
<td>0.634 s</td>
<td>2.420 s</td>
<td>0.26x</td>
<td>has to construct old structure</td>
</tr>
<tr>
<td>for $t$ in $f$ do</td>
<td>0.646 s</td>
<td>2.460 s</td>
<td>0.26x</td>
<td>has to construct old structure</td>
</tr>
<tr>
<td>subs($x = y, f$)</td>
<td>1.160 s</td>
<td>0.076 s</td>
<td>15x</td>
<td>combine exponents, sort, merge</td>
</tr>
<tr>
<td>taylor($f, x, 50$)</td>
<td>0.668 s</td>
<td>0.055 s</td>
<td>12x</td>
<td>get coefficients in one pass</td>
</tr>
<tr>
<td>type($f, polynom$)</td>
<td>0.029 s</td>
<td>0.000 s</td>
<td>$\rightarrow O(n)$</td>
<td>type check variables only</td>
</tr>
</tbody>
</table>

For $f$ with $n = 3$ variables and $t = 10^6$ terms created by

```maple
f := expand(mul(randpoly(v,degree=100,dense),v=[x,y,z]));
```
Maple 17 multiplication and factorization benchmark

Intel Core i5 750 2.66 GHz (4 cores)

<table>
<thead>
<tr>
<th></th>
<th>Maple 16</th>
<th>Maple 17</th>
<th>Magma 2.19-1</th>
<th>Singular 3.1.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiply</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p_4 := f_4(f_4 + 1)</td>
<td>2.140 0.643</td>
<td>1.770 0.416</td>
<td>13.43 31.59</td>
<td></td>
</tr>
<tr>
<td>p_6 := f_6g_6</td>
<td>0.733 0.602</td>
<td>0.203 0.082</td>
<td>0.90 2.75</td>
<td></td>
</tr>
<tr>
<td>factor</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p_4 135751 terms</td>
<td>59.27 46.41</td>
<td>24.35 12.65</td>
<td>325.26 61.05</td>
<td>61.05</td>
</tr>
<tr>
<td>p_6 417311 terms</td>
<td>51.98 49.07</td>
<td>8.32 6.32</td>
<td>364.67 42.08</td>
<td></td>
</tr>
</tbody>
</table>


\[
\begin{align*}
    f_4 &= (1 + x + y + z + t)^{20} + 1 \\
    f_6 &= (1 + u^2 + v + w^2 + x - y)^{10} + 1 \\
    g_6 &= (1 + u + v^2 + w + x^2 + y)^{10} + 1
\end{align*}
\]

10626 terms
3003 terms
3003 terms

Parallel speedup for \( f_4 \times (f_4 + 1) \) is \( 1.77/0.416 = 4.2 \times \).
Given a polynomial $f(x_1, x_2, ..., x_n)$, we store $f$ using POLY if

1. $f$ is expanded and has integer coefficients,
2. $d > 1$ and $t > 1$ where $d = \deg f$ and $t = \#\text{terms}$,
3. we can pack all monomials of $f$ into \textbf{one 64 bit word}, i.e. if $d < 2^b$ where $b = \left\lfloor \frac{64}{n+1} \right\rfloor$

Otherwise we use the sum-of-products representation.
Given a polynomial \( f(x_1, x_2, ..., x_n) \), we store \( f \) using POLY if

1. \( f \) is expanded and has integer coefficients,
2. \( d > 1 \) and \( t > 1 \) where \( d = \deg f \) and \( t = \# \text{terms} \),
3. we can pack all monomials of \( f \) into one 64 bit word, i.e. if \( d < 2^b \) where \( b = \left\lfloor \frac{64}{n+1} \right\rfloor \)

Otherwise we use the sum-of-products representation.

- The representation is invisible to the Maple user.
- Conversions are automatic.
Given a polynomial $f(x_1, x_2, ..., x_n)$, we store $f$ using POLY if

1. $f$ is expanded and has integer coefficients,
2. $d > 1$ and $t > 1$ where $d = \deg f$ and $t = \#\text{terms}$,
3. we can pack all monomials of $f$ into one 64 bit word, i.e. if $d < 2^b$ where $b = \left\lfloor \frac{64}{n+1} \right\rfloor$

Otherwise we use the sum-of-products representation.

- The representation is invisible to the Maple user. Conversions are automatic.
- POLY polynomials will be displayed in sorted order.
Given a polynomial $f(x_1, x_2, ..., x_n)$, we store $f$ using POLY if

1. $f$ is expanded and has integer coefficients,
2. $d > 1$ and $t > 1$ where $d = \text{deg} \ f$ and $t = \# \text{terms}$,
3. we can pack all monomials of $f$ into one 64 bit word, i.e. if $d < 2^b$ where $b = \left\lfloor \frac{64}{n+1} \right\rfloor$

Otherwise we use the sum-of-products representation.

- The representation is invisible to the Maple user. Conversions are automatic.
- POLY polynomials will be displayed in sorted order.
- Packing is fixed by $n = \# \text{variables}$.
### Degree limits (64 bit word)

<table>
<thead>
<tr>
<th>$n$</th>
<th>per variable</th>
<th>total degree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#bits</td>
<td>max deg</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>511</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>255</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>127</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>63</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>31</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>31</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>20</td>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>

**Joris van der Hoven:** Do you use the extra bits for the total degree?

**My answer:** No, because it would complicate and slow down the code, e.g., polynomial division would require explicit overflow checking.

E.g. $b = 2x^2y^2 + y^3 \div x^2y + y^3 = y$ with remainder $-y^4$. 
## Degree limits (64 bit word)

<table>
<thead>
<tr>
<th>$n$</th>
<th>per variable</th>
<th>total degree</th>
<th>Vandermonde</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#bits</td>
<td>max deg</td>
<td>extra bits</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>511</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>255</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>127</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>63</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>31</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>31</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>15</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>15</td>
<td>8</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>15</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>20</td>
<td>3</td>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>

**Joris van der Hoven:** Do you use the extra bits for the total degree?  
**My answer:** No, we can multiply $f \times g$ in POLY if $\deg f + \deg g < 2^b$. Moreover, polynomial division would require explicit overflow checking.  
E.g. $x^2y^2 + y^3 \div x^2y + y^3 = y$ with remainder $y^4$.  

---

Michael Monagan  
ECCAD, Annapolis, 2013
POLY is in Maple 17!
Future Work

- POLY is in Maple 17!
- Use extra bits for total degree.
Future Work

- POLY is in Maple 17!
- Use extra bits for total degree.
- Rethink polynomial factorization for multi-core computers.

<table>
<thead>
<tr>
<th># cores</th>
<th>factor(p)</th>
<th>p := expand(f×g)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>97.51s</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>55.36s</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>36.85s</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>31.59s</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>5.60s</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2.50s</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1.18s</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>0.78s</td>
<td>6</td>
</tr>
</tbody>
</table>

Real time speedup:
- 1.8x
- 2.7x
- 3.1x

Intel Core i7 3930K, 6 cores, overclocked @ 4.2GHz
POLY is in Maple 17!
Use extra bits for total degree.
Rethink polynomial factorization for multi-core computers.

<table>
<thead>
<tr>
<th># cores</th>
<th>factor($p$)</th>
<th>$p := \text{expand}(f \times g)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>1.8x</td>
<td>–</td>
</tr>
<tr>
<td>2</td>
<td>2.7x</td>
<td>2.2x</td>
</tr>
<tr>
<td>4</td>
<td>3.1x</td>
<td>4.7x</td>
</tr>
<tr>
<td>6</td>
<td>–</td>
<td>7.1x</td>
</tr>
</tbody>
</table>

Intel Core i7 3930K, 6 cores, overclocked @ 4.2GHz

Let $f(u, v, w, x, y) = \left( \sum c_{i,j}(u, v, w)x^i y^j \right) \times \left( \sum d_{i,j}(u, v, w)x^i y^j \right)$.
Pick $\alpha = (\omega_1, \omega_2, \omega_3) \in \mathbb{Z}_p^3$ and for $k = 1, 2, \cdots$ factor

$$f(\alpha^k, x, y) = \left( \sum c_{i,j}(\alpha^k)x^i y^j \right) \times \left( \sum d_{i,j}(\alpha^k)x^i y^j \right) \mod p.$$
Conclusion

We will not get good parallel speedup using these

Even with conversions to a more suitable data structure, sequential overhead will limit parallel speedup.

Thank you for attending my talk.