How fast can we multiply and divide sparse polynomials?

Michael Monagan

CECM, Simon Fraser University

Joint work with Roman Pearce, Simon Fraser University.
Supported by the MITACS NCE of Canada.
How do we multiply and divide sparse distributed polynomials?

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

\[ h = f \cdot g = (((((f_1 g + f_2 g) + f_3 g) + f_4 g) \cdots + f_n g) \]
\[ h : g = (((((h - f_1 g) - f_2 g) - f_3 g) - f_4 g) \cdots - f_n g) \]
How do we multiply and divide sparse distributed polynomials?

\[ f = a_1X_1 + a_2X_2 + \cdots + a_nX_n \]
\[ g = b_1Y_1 + b_2Y_2 + \cdots + b_mY_m \]  \hspace{1cm} \text{(sorted)}

\[ h = f \cdot g = (((((f_1g + f_2g) + f_3g) + f_4g) \cdots + f_ng) \]
\[ h \div g = (((((h - f_1g) - f_2g) - f_3g) - f_4g) \cdots - f_ng) \]

Example:

\[ f = x^n + x^{n-1} + \cdots + x \]
\[ g = y^m + y^{m-1} + \cdots + y \]

\textbf{Example:}

\[ f = x^n + x^{n-1} + \cdots + x \]
\[ g = y^m + y^{m-1} + \cdots + y \]

\[ i^{th} \text{ merge can do } O(im) \text{ comparisons (sparse)} \]
\[ \implies \sum_{i=1}^{n-1} im \in O(n^2m) \text{ comparisons in total} \]
How do we multiply sparse polynomials?

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

Maple uses \textit{divide and conquer} – \( O(mn \log m) \) monomial comparisons.

\[ f \times g = f_1 \times g_1 + f_2 \times g_1 + f_1 \times g_2 + f_2 \times g_2 \]

where \( f_1 \) and \( g_1 \) (\( f_2 \) and \( g_2 \)) are the first (second) half of the terms of \( f \) and \( g \).
How do we multiply sparse polynomials?

\[ f = a_1X_1 + a_2X_2 + \cdots + a_nX_n \]  
\[ g = b_1Y_1 + b_2Y_2 + \cdots + b_mY_m \]  
(sorted)

Maple uses divide and conquer – \( O(mn \log m) \) monomial comparisons.

\[ f \times g = f_1 \times g_1 + f_2 \times g_1 + f_1 \times g_2 + f_2 \times g_2 \]

where \( f_1 \) and \( g_1 \) (\( f_2 \) and \( g_2 \)) are the first (second) half of the terms of \( f \) and \( g \).

Magma uses hashing – \( mn \) hashes on monomials \( X_i \cdot Y_j \).

\[
\text{for } i = 1, 2, \ldots, n \text{ do for } j = 1, 2, \ldots, m \text{ do}
\set\ Z = X_i \cdot Y_j \text{ and } h[Z] = h[Z] + a_i \times b_j.
\]
How do we multiply sparse polynomials?

Singular uses geobuckets (Yan, 1998).

Split $f$ into buckets where bucket $i$ has at most $2^i$ terms

<table>
<thead>
<tr>
<th>Bucket</th>
<th>$f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$2xyz$</td>
</tr>
<tr>
<td>2</td>
<td>$-6x^3yz + 5xz^2 + 3xz$</td>
</tr>
<tr>
<td>3</td>
<td>$+4x^3yz - 3xyz^3 + 2xyz^2 + 7xyz + 4$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

$log(\#f) = -7x^4y^3 + 3xyz^3 + 7xyz - 7xz + 4x - 3y + 2$
How do we multiply sparse polynomials?

Singular uses geobuckets (Yan, 1998).

Split \( f \) into buckets where bucket \( i \) has at most \( 2^i \) terms

<table>
<thead>
<tr>
<th>Bucket</th>
<th>( f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( 2xyz )</td>
</tr>
<tr>
<td>2</td>
<td>( -6x^3yz + 5xz^2 + 3xz )</td>
</tr>
<tr>
<td>3</td>
<td>( +4x^3yz - 3xyz^3 + 2xyz^2 + 7xyz + 4 )</td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
</tr>
<tr>
<td>( \log(#f) )</td>
<td>( -7x^4y^3 + 3xyz^3 + 7xyz - 7xz + 4x - 3y + 2 )</td>
</tr>
</tbody>
</table>

Multiplication and also division are

**Sparse case:** \( O(nm \log(mn)) \) comparisons.

**Dense case:** \( O(nm) \) comparisons.
How do we multiply sparse polynomials?

ALTRAN uses a binary heap (S. Johnson, 1974).

■ Heap property: \( H_i \geq H_{2i} \) and \( H_i \geq H_{2i+1} \).
How do we multiply sparse polynomials?

ALTRAN uses a **binary heap** (S. Johnson, 1974).

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$x^{13}$</td>
<td>$x^{10}$</td>
<td>$x^9$</td>
<td>$x^1$</td>
<td>$x^6$</td>
<td>$x^7$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Heap property**: $H_i \geq H_{2i}$ and $H_i \geq H_{2i+1}$.
- **Creating** is $O(n)$ comparisons where $n = \#H$.
- **Heap extraction** is $O(\log_2 n)$.
How do we multiply sparse polynomials?

ALTRAN uses a binary heap (S. Johnson, 1974).

<table>
<thead>
<tr>
<th></th>
<th>13</th>
<th>10</th>
<th>9</th>
<th>1</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

- Heap property: $H_i \geq H_{2i}$ and $H_i \geq H_{2i+1}$.
- Creating is $O(n)$ comparisons where $n = \#H$.
- Heap extraction is $O(\log_2 n)$.
- Hence, sorting using a heap is $O(n \log_2 n)$. 
Multiplication using a binary heap.

\[
f = a_1 X_1 + a_2 X_2 + \cdots + a_n X_n \quad \text{(sorted)}
\]

\[
g = b_1 Y_1 + b_2 Y_2 + \cdots + b_m Y_m
\]

<table>
<thead>
<tr>
<th>$X_1Y_1$</th>
<th>$X_1Y_2$</th>
<th>$X_2Y_1$</th>
<th>\cdots</th>
<th>$X_iY_j$</th>
<th>\cdots</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(a_1, b_1)$</td>
<td>$(a_1, b_2)$</td>
<td>$(a_2, b_1)$</td>
<td>\cdots</td>
<td>$(a_i, b_j)$</td>
<td>\cdots</td>
</tr>
</tbody>
</table>

O($nm \log(nm)$) comparisons, O($nm$) space.

coefficient arithmetic using O(1) temporary registers.
Multiplication using a binary heap.

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

<table>
<thead>
<tr>
<th>(X_1 Y_1)</th>
<th>(X_1 Y_2)</th>
<th>(X_2 Y_1)</th>
<th>(\cdots)</th>
<th>(X_i Y_j)</th>
<th>(\cdots)</th>
</tr>
</thead>
<tbody>
<tr>
<td>((a_1, b_1))</td>
<td>((a_1, b_2))</td>
<td>((a_2, b_1))</td>
<td>(\cdots)</td>
<td>((a_i, b_j))</td>
<td>(\cdots)</td>
</tr>
</tbody>
</table>

- \(O(nm \log(nm))\) comparisons, \(O(nm)\) space.
- coefficient arithmetic using \(O(1)\) temporary registers.
Multiplication using a binary heap.

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

\[
\begin{align*}
f &= a_1X_1 + a_2X_2 + \cdots + a_nX_n \\
g &= b_1Y_1 + b_2Y_2 + \cdots + b_mY_m
\end{align*}
\]

(sorted)

Products

- $a_1X_1(b_1Y_1 + b_2Y_2 + \cdots + b_mY_m)$
- $a_2X_2(b_1Y_1 + b_2Y_2 + \cdots + b_mY_m)$
- $\vdots$
- $a_nX_n(b_1Y_1 + b_2Y_2 + \cdots + b_mY_m)$

- $O(nm \log n)$ comparisons.
- Space for $\leq n$ monomials in the heap.
- Can pick $n \leq m$. 
High Performance

- L1 (32Kbytes): 3 cycles
- L2 (2MBytes): 20 cycles
- DRAM (2Gbytes): 150-200 cycles
- larger polynomial is *streamed* into the cache
- products *generated* inside cache
- heap fits *on chip*
- pointers updated in L1/L2
- result written out to memory
Division using a heap.

Johnson’s quotient heap algorithm.

Dividing $f \div g$ compute

$$f - \sum_{i=1}^{\#q} q_i \times g$$

- $O(\#f + \#q \#g \log \#q)$ comparisons
- $O(\#q)$ working memory
## Division using a heap.

### Johnson’s quotient heap algorithm.

Dividing $f \div g$ compute

\[
f - \sum_{i=1}^{\#q} q_i \times g
\]

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

A divisor heap algorithm.

Dividing $f \div g$ compute

\[
f - \sum_{i=2}^{\#g} g_i \times q
\]

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

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

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

\( \text{quotient heap} \)

\( \text{divisor heap} \)

▶ Does \( O(\#f + \#q\#g \log \min(\#q, \#g)) \) comparisons
▶ using \( O(\min(\#q, \#g)) \) working memory.
Pseudo Division

Pseudo division scales terms to avoid fractions:

$$f \div g = (((f - \frac{q_1}{d_1}g) - \frac{q_2}{d_2}g) - \frac{q_3}{d_3}g) - \cdots - \frac{q_n}{d_n}g)$$

$$\Rightarrow (d_n \cdots (d_3(d_2(d_1f - q_1g) - q_2g) - q_3g) - \cdots - q_ng)$$

How many multiplications can this do?

Let \( \#q = n, \#g = m, \#f = nm \):

Then \( \sum_{i=1}^{n} (i + 1)m \in O(n^2m) \) multiplications.
Pseudo Division

Theorem.

We can divide $f$ by $g$, producing a quotient $q$ using $O(#f + #q#g \log \min(#q, #g))$ comparisons.

Additionally:
Pseudo Division

**Theorem.**
We can divide $f$ by $g$, producing a quotient $q$ using $O(#f + #q#g \log \min(#q, #g))$ comparisons.

Additionally:
- Exact polynomial division over $\mathbb{Z}$ requires $#q(#g - 1)$ integer multiplications and $#q$ divisions.
- Pseudo division with remainder over $\mathbb{Q}$ does at most $#f + #q(2#g - 1)$ integer multiplications, $#q(#g + 1)$ divisions, and $#q$ gcds.
- We need $O(1)$ temporary storage registers for coefficient arithmetic and $O(\min(#f, #g))$ storage for the heap. No garbage is created.
Optimizations

Chaining terms in the heap:

- terms are chained on insertion
- dense case: $O(nm \log n) \Rightarrow O(nm)$ comparisons
Optimizations

Chaining terms in the heap:

- terms are chained on insertion
- dense case: $O(nm \log n) \Rightarrow O(nm)$ comparisons

Also:
- one word monomials stored directly in the heap
- wordsize integer arithmetic coded in assembly
Benchmark 1: sparse unbalanced divisions.

\[ q = (1 + x + y + 2z^2 + 3t^3 + 5u^5)^\alpha \]
\[ g = (1 + u + t + 2z^2 + 3y^3 + 5x^5)^\beta \]

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>( \beta )</th>
<th>#q</th>
<th>#g</th>
<th>( f = q \cdot g )</th>
<th>( f \div g )</th>
<th>max heap</th>
<th>real max</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>30</td>
<td>126</td>
<td>324632</td>
<td>2.99</td>
<td>2.77</td>
<td>126</td>
<td>126</td>
</tr>
<tr>
<td>8</td>
<td>18</td>
<td>1287</td>
<td>33649</td>
<td>2.27</td>
<td>2.21</td>
<td>1287</td>
<td>1161</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>6188</td>
<td>6188</td>
<td>2.44</td>
<td>2.24</td>
<td>12079</td>
<td>3895</td>
</tr>
<tr>
<td>18</td>
<td>8</td>
<td>33649</td>
<td>1287</td>
<td>2.38</td>
<td>2.46</td>
<td>2572</td>
<td>1231</td>
</tr>
<tr>
<td>30</td>
<td>4</td>
<td>324632</td>
<td>126</td>
<td>2.84</td>
<td>2.53</td>
<td>250</td>
<td>70</td>
</tr>
</tbody>
</table>

• chaining reduces the size of the heap in practice
• division is as fast as multiplication
Representation of polynomials.

“Which Polynomial Representation is Best?”

David Stoutemyer, 1984 Macsyma Users Conference

Distributed or recursive?

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

or \(( -5y - 4z^2y^3 ) + ( -6zy^2 + 9zy^3 )x - 8x^3 \) ?
Representation of polynomials.

“Which Polynomial Representation is Best?”

David Stoutemyer, 1984 Macsyma Users Conference

Distributed or recursive?

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

or $$(-5y - 4z^2y^3) + (-6zy^2 + 9zy^3)x - 8x^3$$?

Sparse or dense?
Variables in or out?
Arrays or linked lists?
Maple’s sum of products representation.

\[
9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5
\]
Maple’s sum of products representation.

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

Singular’s distributed representation.
Trip’s recursive sparse representation.

\[ (-5y - 4z^2 y^3) + (-6zy^2 + 9zy^3)\cdot x - 8x^3 \]
Trip’s recursive sparse representation.

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

Pari’s recursive dense representation.
So which representation is best?
So which representation is best?

Stoutemyer concluded

1. recursive is better than distributed
So which representation is best?

Stoutemyer concluded

1. recursive is better than distributed
2. and recursive dense is better than recursive sparse!
Fateman’s 2003 benchmark.

“Comparing the speed of programs for sparse polynomial multiplication”, Richard Fateman, March 2003:

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

Pentium III, 933 MHz, 32 bit machine.

<table>
<thead>
<tr>
<th>Software</th>
<th>Time (s)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pari/GP 2.0.17</td>
<td>2.3</td>
<td>(recursive dense array)</td>
</tr>
<tr>
<td>MockMMA ACL6.1/GMP4.1</td>
<td>3.3</td>
<td>(recursive dense array)</td>
</tr>
<tr>
<td>Hashing ACL6.1/GMP4.1</td>
<td>4.7</td>
<td>(hash on monomial)</td>
</tr>
<tr>
<td>Reduce 3.7 (in CSL)</td>
<td>5.0</td>
<td>(sparse recursive list)</td>
</tr>
<tr>
<td>Singular 2.0.3</td>
<td>6.1</td>
<td>(sparse distributed list)</td>
</tr>
<tr>
<td>Macsyma (in ACL 6.1)</td>
<td>6.9</td>
<td>(sparse recursive list)</td>
</tr>
<tr>
<td>Maple VR4</td>
<td>17.9</td>
<td>(sparse distributed array)</td>
</tr>
</tbody>
</table>

Remark: \( f \) is 100% dense in the recursive representation.
Fateman’s 2003 benchmark.

“Comparing the speed of programs for sparse polynomial multiplication”, Richard Fateman, March 2003:

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

Pentium III, 933 MHz, 32 bit machine.

Pari/GP 2.0.17 2.3s (recursive dense array)
MockMMA ACL6.1/GMP4.1 3.3s (recursive dense array)
Hashing ACL6.1/GMP4.1 4.7s (hash on monomial)
Reduce 3.7 (in CSL) 5.0s (sparse recursive list)
Singular 2.0.3 6.1s (sparse distributed list)
Macsyma (in ACL 6.1) 6.9s (sparse recursive list)
Maple VR4 17.9s (sparse distributed array)

**Remark:** \( f \) is 100% dense in the recursive representation.
What has changed since 2003?

- Computers are now 64 bits.
- Level 2 cache is on the chip.
- New desktops are quad-core.
What has changed since 2003?

- Computers are now 64 bits.
- Level 2 cache is on the chip.
- New desktops are quad-core.
Our SDMP data structure

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

One word: $i + j + k i j k$

- monomial $>$ and $\times$ are one machine instruction.
Our SDMP data structure

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

One word: $i + j + k$ $i$ $j$ $k$

- monomial $>$ and $\times$ are one machine instruction.

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 dxyz dxyz dxyz dxyz</td>
</tr>
<tr>
<td>packing</td>
<td>5131 9 5032 -4 4121 -6 3300 -8 0000 -5</td>
</tr>
</tbody>
</table>
Our SDMP data structure

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

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

- monomial $>$ and $\times$ are one machine instruction.

Packed array for: $9xyz - 4y^3z^2 - 6xy^2z - 8x^3 - 5$

<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>dxyz</td>
</tr>
<tr>
<td></td>
<td>5131 9 5032 -4 4121 -6 3300 -8 0000 -5</td>
</tr>
</tbody>
</table>

Why **graded lex order**? Because it’s good for polynomial **division**.
Our data structure: general case

\[ Axy^3z - By^3z^2 - Cxy^2z - 8x^3 - 5 \]

- memory access is sequential
- 8K blocks of terms allocated at a time, chained together
Our SDMP data structure: one word packing

<table>
<thead>
<tr>
<th>#variables</th>
<th>#bits</th>
<th>max deg</th>
<th>#bits</th>
<th>max deg</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>21</td>
<td></td>
<td>10</td>
<td>1023</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>65535</td>
<td>8</td>
<td>255</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>2047</td>
<td>6</td>
<td>63</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>1023</td>
<td>5</td>
<td>31</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>511</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>255</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>127</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>63</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>31</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>15</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>21</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>63</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
## Space Data

<table>
<thead>
<tr>
<th>Polynomials</th>
<th>#terms</th>
<th>density</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A = (1 + x + y + z)^{20} )</td>
<td>1771</td>
<td>1.0000</td>
</tr>
<tr>
<td>( B = (1 + x^2 + y^2 + z^2)^{20} )</td>
<td>1771</td>
<td>0.1445</td>
</tr>
<tr>
<td>( C = (w + x + y + z)^{20} )</td>
<td>1771</td>
<td>0.1667</td>
</tr>
<tr>
<td>( D = (w^2 + x^2 + y^2 + z^2)^{20} )</td>
<td>1771</td>
<td>0.0131</td>
</tr>
<tr>
<td>( E = (1 + x_1 + x_2 + \ldots + x_{50})^2 )</td>
<td>1326</td>
<td>1.0000</td>
</tr>
<tr>
<td>( E = (1 + x_1^2 + x_2^2 + \ldots + x_{50}^2)^2 )</td>
<td>1326</td>
<td>0.0042</td>
</tr>
</tbody>
</table>

Table: \( \text{density} = \frac{\# \text{terms}}{\binom{n+m}{m}} \) where \( n = \deg f \) and \( m = \#\text{vars} \).  

### Maple vs. Pari, Trip, Singular, SDMP (packed)

<table>
<thead>
<tr>
<th></th>
<th>Maple</th>
<th>Pari</th>
<th>Trip</th>
<th>Singular</th>
<th>SDMP (packed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>14,544</td>
<td>2,463</td>
<td>6,465</td>
<td>8,855</td>
<td>3,542</td>
</tr>
<tr>
<td>B</td>
<td>14,553</td>
<td>4,233</td>
<td>6,465</td>
<td>8,855</td>
<td>3,542</td>
</tr>
<tr>
<td>C</td>
<td>17,634</td>
<td>15,938</td>
<td>14,165</td>
<td>10,626</td>
<td>3,543</td>
</tr>
<tr>
<td>D</td>
<td>17,634</td>
<td>26,563</td>
<td>14,165</td>
<td>10,626</td>
<td>3,543</td>
</tr>
<tr>
<td>E</td>
<td>8,928</td>
<td>5,150</td>
<td>10,350</td>
<td>68,952</td>
<td>5,304</td>
</tr>
<tr>
<td>F</td>
<td>9,078</td>
<td>6,575</td>
<td>10,350</td>
<td>68,952</td>
<td>6,630</td>
</tr>
</tbody>
</table>

Table: Space in words assuming coefficients are immediate integers.
Benchmarks: How should we measure sparsity?

Let \( \#f = \#\text{terms}(f) \), \( m = \#\text{vars}(f) \), \( d = \text{deg}(f) \).

The density of \( f \) is \( D_f = \frac{\#f}{(d+m)} \).
Benchmarks: How should we measure sparsity?

Let \( \#f = \#\text{terms}(f) \), \( m = \#\text{vars}(f) \), \( d = \deg(f) \).

The \textbf{density} of \( f \) is \( D_f = \frac{\#f}{d+m} \).

Let \( f = a_1 X_1 + a_2 X_2 + \ldots + a_n X^n \), \( g = b_1 Y_1 + b_2 Y_2 + \ldots + b_m Y^m \).

The \textbf{work} of \( f \times g \) is \( W_{f \times g} = \frac{\#f \#g}{|\{X_i Y_j\}|} \).
Benchmarks: How should we measure sparsity?

Let \( \#f = \# \text{terms}(f) \), \( m = \# \text{vars}(f) \), \( d = \text{deg}(f) \).

The density of \( f \) is \( D_f = \frac{\#f}{(d+m)} \).

Let \( f = a_1 X_1 + a_2 X_2 + \ldots + a_n X^n \), \( g = b_1 Y_1 + b_2 Y_2 + \ldots + b_m Y^m \).

The work of \( f \times g \) is \( W_{f \times g} = \frac{\#f \#g}{|\{X_i Y_j\}|} \). \( 1 \leq W \frac{n^m}{2^m m!} \).
Benchmarks: How should we measure sparsity?

Let $\# f = \#\text{terms}(f)$, $m = \#\text{vars}(f)$, $d = \text{deg}(f)$.

The density of $f$ is $D_f = \frac{\# f}{(d+m)}$.

Let $f = a_1 X_1 + a_2 X_2 + \ldots + a_n X^n$, $g = b_1 Y_1 + b_2 Y_2 + \ldots + b_m Y^m$.

The work of $f \times g$ is $W_{f \times g} = \frac{\# f \# g}{|\{X_i Y_j\}|}$. $1 \leq W \frac{n^m}{2^m m!}$.

Example: $f = (1 + x + y + z)^{20}$, $g = f + 1$.
$D_f = 1.00$, $W = 254.15$. ($\# f = \# g = 1,771$, $\# fg = 12,341$).
Benchmarks: How should we measure sparsity?

Let \( \#f = \#\text{terms}(f) \), \( m = \#\text{vars}(f) \), \( d = \text{deg}(f) \).

The density of \( f \) is \( D_f = \frac{\#f}{(d+m)} \).

Let \( f = a_1X_1 + a_2X_2 + \ldots + a_nX^n \), \( g = b_1Y_1 + b_2Y_2 + \ldots + b_mY^m \).

The work of \( f \times g \) is \( W_{f \times g} = \frac{\#f \#g}{|\{X_iY_j\}|} \cdot \frac{n^m}{2^m m!} \).

Example: \( f = (1 + x + y + z)^{20} \), \( g = f + 1 \).
\( D_f = 1.00 \), \( W = 254.15 \). (\( \#f = \#g = 1,771 \), \( \#fg = 12,341 \)).

Example: \( f = (1 + x^2 + y^2 + z^2)^{20} \), \( g = f + 1 \).
Now \( D_f = 0.1435 \) but \( W = 254.15 \) is the same!
Benchmarks: How should we measure sparsity?

Let \( \#f = \#\text{terms}(f), \ m = \#\text{vars}(f), \ d = \deg(f) \).

The density of \( f \) is \( D_f = \frac{\#f}{(d+m)} \).

Let \( f = a_1X_1 + a_2X_2 + \ldots + a_nX^n, \ g = b_1Y_1 + b_2Y_2 + \ldots + b_mY^m \).

The work of \( f \times g \) is \( W_{f \times g} = \frac{\#f \#g}{|\{X_i Y_j\}|}. \ 1 \leq W \frac{n^m}{2^m m!} \).

Example: \( f = (1 + x + y + z)^{20}, \ g = f + 1. \)
\( D_f = 1.00, \ W = 254.15. \ (#f = #g = 1,771, \ #fg = 12,341) \).

Example: \( f = (1 + x^2 + y^2 + z^2)^{20}, \ g = f + 1. \)
Now \( D_f = 0.1435 \) but \( W = 254.15 \) is the same!

Example: \( f = (1 + x + \cdots + x^n), \ g = (1 + y + \cdots + y^n), \)
Here \( D_f = D_g = D_{f \times g} = 1.00 \), but the work \( W = 1.00! \).
Benchmark 2: A dense Fateman problem.

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

- \( f \) and \( g \) have 61 bit coefficients
- \( h = f \cdot g \) has 128 bit coefficients

Intel Core2 3.0 GHz 64-bit

<table>
<thead>
<tr>
<th>( W = 3,385 )</th>
<th>( 46,376 \times 46,376 = 635,376 ) terms</th>
<th>multiply ( p = f \times g )</th>
<th>divide ( q = p/f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maple 11</td>
<td>15986.16</td>
<td>13039.24</td>
<td></td>
</tr>
<tr>
<td>Singular 3-0-4 (distributed)</td>
<td>1482.36</td>
<td>364.49</td>
<td></td>
</tr>
<tr>
<td>Magma V2.14-7</td>
<td>679.07</td>
<td>610.62</td>
<td></td>
</tr>
<tr>
<td>Pari 2.3.3 (w/ GMP)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trip v0.99 (rationals) (recursive)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sdmp (unpacked)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sdmp (packed)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arithmetic cost</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Benchmark 2: A dense Fateman problem.

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

- \( f \) and \( g \) have 61 bit coefficients
- \( h = f \cdot g \) has 128 bit coefficients

Intel Core2 3.0 GHz 64-bit

<table>
<thead>
<tr>
<th>( 46,376 \times 46,376 = 635,376 ) terms ( W = 3,385 )</th>
<th>multiply ( p = f \times g )</th>
<th>divide ( q = p/f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maple 11</td>
<td>15986.16</td>
<td>13039.24</td>
</tr>
<tr>
<td>Singular 3-0-4 (distributed)</td>
<td>1482.36</td>
<td>364.49</td>
</tr>
<tr>
<td>Magma V2.14-7</td>
<td>679.07</td>
<td>610.62</td>
</tr>
<tr>
<td>Pari 2.3.3 (w/ GMP)</td>
<td>512.18</td>
<td>283.44</td>
</tr>
<tr>
<td>Trip v0.99 (rationals) (recursive)</td>
<td>108.22</td>
<td>-</td>
</tr>
</tbody>
</table>
Benchmark 2: A dense Fateman problem.

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

- \( f \) and \( g \) have 61 bit coefficients
- \( h = f \cdot g \) has 128 bit coefficients

Intel Core2 3.0 GHz 64-bit

<table>
<thead>
<tr>
<th></th>
<th>multiply ( p = f \times g )</th>
<th>divide ( q = p/f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>46,376 \times 46,376 = 635,376 terms</td>
<td>15986.16</td>
<td>13039.24</td>
</tr>
<tr>
<td>( W = 3,385 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Maple 11: 15986.16 13039.24
- Singular 3-0-4 (distributed): 1482.36 364.49
- Magma V2.14-7: 679.07 610.62
- Pari 2.3.3 (w/ GMP): 512.18 283.44
- Trip v0.99 (rationals) (recursive): 108.22 -
- Trip v0.99 (rationals) (recursive): 119.94 135.05
- sdmp (unpacked): 47.33 58.44
Benchmark 2: A dense Fateman problem.

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

- \( f \) and \( g \) have 61 bit coefficients
- \( h = f \cdot g \) has 128 bit coefficients

Intel Core2 3.0 GHz 64-bit

\[
\begin{array}{|c|c|c|}
\hline
\text{46,376} \times \text{46,376} = \text{635,376 terms} & \text{multiply} & \text{divide} \\
W = 3,385 & p = f \times g & q = p/f \\
\hline
\text{Maple 11} & 15986.16 & 13039.24 \\
\text{Singular 3-0-4 (distributed)} & 1482.36 & 364.49 \\
\text{Magma V2.14-7} & 679.07 & 610.62 \\
\text{Pari 2.3.3 (w/ GMP)} & 512.18 & 283.44 \\
\text{Trip v0.99 (rationals) (recursive)} & 108.22 & - \\
\text{sdmp (unpacked)} & 119.94 & 135.05 \\
\text{sdmp (packed)} & 47.33 & 58.44 \\
\text{Arithmetic cost} & 15.50 & 15.50 \\
\hline
\end{array}
\]
Benchmark 3: A sparse 10 variable problem.

\[ f = (x_1x_2 + x_2x_3 + x_3x_4 + x_4x_5 + x_5x_6 + x_6x_7 + x_7x_8 + x_8x_9 + x_9x_{10} + x_1x_{10} + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + 1)^5 \]

\[ g = (x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2 + x_7^2 + x_8^2 + x_9^2 + x_{10}^2 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + 1)^5 \]

Intel Core2 3.0 GHz 64-bit

<table>
<thead>
<tr>
<th>26,599 × 36,365 = 19,631,157 terms</th>
<th>multiply ( p = f \times g ) (megabytes) seconds</th>
<th>divide ( q = p/f ) (megs) secs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maple 11</td>
<td>14053.37</td>
<td>10760.36</td>
</tr>
<tr>
<td>Singular 3-0-4</td>
<td>(1538) 655.25</td>
<td>(1390) 206.60</td>
</tr>
<tr>
<td>Magma V2.14-7</td>
<td>(2365) 313.02</td>
<td>(1753) 5744.60</td>
</tr>
<tr>
<td>sdmp (unpacked)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sdmp (packed)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Benchmark 3: A sparse 10 variable problem.

\[ f = (x_1x_2 + x_2x_3 + x_3x_4 + x_4x_5 + x_5x_6 + x_6x_7 + x_7x_8 + x_8x_9 + x_9x_{10} + x_1x_{10} + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + 1)^5 \]

\[ g = (x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2 + x_7^2 + x_8^2 + x_9^2 + x_{10}^2 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + 1)^5 \]

Intel Core 2 3.0 GHz 64-bit

<table>
<thead>
<tr>
<th>Software</th>
<th>Multiply Time (megabytes)</th>
<th>Divide Time (megs)</th>
<th>Processor Time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maple 11</td>
<td>14053.37</td>
<td></td>
<td>10760.36</td>
</tr>
<tr>
<td>Singular 3-0-4</td>
<td>(1538) 655.25</td>
<td>(1390) 206.60</td>
<td></td>
</tr>
<tr>
<td>Magma V2.14-7</td>
<td>(2365) 313.02</td>
<td>(1753) 5744.60</td>
<td></td>
</tr>
<tr>
<td>Trip v0.99 (rationals)</td>
<td>(1218) 221.91</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pari 2.3.3 (w/ GMP)</td>
<td>109.27</td>
<td></td>
<td>109.69</td>
</tr>
<tr>
<td>sdmp (unpacked)</td>
<td>(1617) 175.97</td>
<td>(14.4) 162.37</td>
<td></td>
</tr>
<tr>
<td>sdmp (packed)</td>
<td>(304) 40.33</td>
<td>(3.4) 41.33</td>
<td></td>
</tr>
</tbody>
</table>
Benchmark 4: A very sparse 5 variable problem.

\[ f = (1 + x + y + 2z^2 + 3t^3 + 5u^5)^{12} \]
\[ g = (1 + u + t + 2z^2 + 3y^3 + 5x^5)^{12} \]

- \( f \) and \( g \) have 37 bit coefficients
- \( h = f \cdot g \) has 75 bit coefficients

Intel Core2 3.0 GHz 64-bit

<table>
<thead>
<tr>
<th>( 6188 \times 6188 = 5821335 ) terms</th>
<th>multiply ( p = f \times g ) (megabytes) seconds</th>
<th>divide ( q = f / g ) (megs) secs</th>
</tr>
</thead>
<tbody>
<tr>
<td>( W = 6.58 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maple 11</td>
<td>(2157) 332.71</td>
<td>(2157) 367.46</td>
</tr>
<tr>
<td>Singular 3-0-4</td>
<td>(595) 58.91</td>
<td>(572) 39.25</td>
</tr>
<tr>
<td>Magma V2.14-7</td>
<td>(1690) 23.77</td>
<td>(180) 151.99</td>
</tr>
</tbody>
</table>

Pari 2.3.3 (w/ GMP)

Pari 2.3.3 (w/ GMP)

Trip v0.99 (rationals)

Trip v0.99 (rationals)

sdmp (unpacked)

sdmp (unpacked)

sdmp (packed)

sdmp (packed)
Benchmark 4: A very sparse 5 variable problem.

\[
\begin{align*}
  f &= (1 + x + y + 2z^2 + 3t^3 + 5u^5)^{12} \\
  g &= (1 + u + t + 2z^2 + 3y^3 + 5x^5)^{12}
\end{align*}
\]

- \( f \) and \( g \) have 37 bit coefficients
- \( h = f \cdot g \) has 75 bit coefficients

Intel Core2 3.0 GHz 64-bit

<table>
<thead>
<tr>
<th>( 6188 \times 6188 = 5821335 ) terms</th>
<th>multiply ( p = f \times g ) (megabytes) seconds</th>
<th>divide ( q = f / g ) (megs) secs</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.58</td>
<td>(2157) 332.71</td>
<td>(2157) 367.46</td>
</tr>
<tr>
<td>Maple 11</td>
<td>(595) 58.91</td>
<td>(572) 39.25</td>
</tr>
<tr>
<td>Singular 3-0-4</td>
<td>(1690) 23.77</td>
<td>(180) 151.99</td>
</tr>
<tr>
<td>Magma V2.14-7</td>
<td>53.98</td>
<td>30.68</td>
</tr>
<tr>
<td>Pari 2.3.3 (w/ GMP)</td>
<td>(552) 4.14</td>
<td>-</td>
</tr>
<tr>
<td>Trip v0.99 (rationals)</td>
<td>(336) 4.77</td>
<td>(0.3) 5.12</td>
</tr>
<tr>
<td>sdmp (unpacked)</td>
<td>(150) 2.02</td>
<td>(0.1) 2.10</td>
</tr>
<tr>
<td>sdmp (packed)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

Distributed can be faster than recursive.
But packing monomials is necessary.
Conclusion

Distributed can be faster than recursive.
But packing monomials is necessary.
Heaps are good!
Conclusion

Distributed can be faster than recursive.
But packing monomials is necessary.
Heaps are good!

- Heaps get us $\#C \in O(nm \log \min(m, n))$ worst case complexity. Optimal?
Conclusion

Distributed can be faster than recursive.
But packing monomials is necessary.
Heaps are good!

- Heaps get us $\#C \in O(nm \log \min(m, n))$ worst case complexity. Optimal?

- Coefficient arithmetic can be done in-place.
  No garbage!
Conclusion

Distributed can be faster than recursive.
But packing monomials is necessary.
Heaps are good!

- Heaps get us $\#C \in O(nm \log \min(m, n))$ worst case complexity. Optimal?
- Coefficient arithmetic can be done in-place. No garbage!
- Size(heap) $\in O(\min(m, n)) \Rightarrow$ heap fits in cache.
Conclusion

Distributed can be faster than recursive.
But packing monomials is necessary.
Heaps are good!

- Heaps get us $\#C \in O(nm \log \min(m, n))$ worst case complexity. Optimal?
- Coefficient arithmetic can be done in-place.
  No garbage!
- Size(heap) $\in O(\min(m, n)) \implies$ heap fits in cache.
- Multivariate pseudo-division is as efficient as exact division.
Conclusion

Distributed can be faster than recursive. But packing monomials is necessary. Heaps are good!

- Heaps get us $\# C \in O(nm \log \min(m, n))$ worst case complexity. Optimal?
- Coefficient arithmetic can be done in-place. No garbage!
- $\text{Size(heap)} \in O(\min(m, n)) \implies$ heap fits in cache.
- Multivariate pseudo-division is as efficient as exact division.
- But heaps reduce opportunity for parallelism.
The heap extract operation.

Algorithm 1: extract costs $2 \log n - O(1)$ comparisons on average.

![Heap diagram with nodes labeled and arrows indicating the extract operation.](image-url)
The heap extract operation.

Algorithm 1: extract costs $2 \log n - O(1)$ comparisons on average.

Heapsort is $2n \log n - O(n)$ average
Quicksort is $2n \log n + O(n)$ average
Mergesort is $n \log n - n + 1$ worst case
The heap extract operation.

Algorithm 2: extract costs $\log n - O(1)$ comparisons on average.

Heapsort is $n \log n + O(n)$ average
The heap and the cache.

So which heap extract algorithm is best?
The heap and the cache.

So which heap extract algorithm is best? It depends!
So which heap extract algorithm is best? It depends!

For one word monomials stored immediately in the heap, Algorithm 1 with $2 \log n - O(1)$ comparisons is faster.

For multi-word monomials pointed to in the heap, Algorithm 2 with $\log n + O(1)$ comparisons is faster.
The heap and the cache.

So which heap extract algorithm is best? It depends!

For one word monomials stored immediately in the heap, Algorithm 1 with $2 \log n - O(1)$ comparisons is faster.

For multi-word monomials pointed to in the heap, Algorithm 2 with $\log n + O(1)$ comparisons is faster.

The difference in speed ranged from 0% to 23%.