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?

$$\begin{aligned} f &= a_1 X_1 + a_2 X_2 + \cdots + a_n X_n \\ g &= b_1 Y_1 + b_2 Y_2 + \cdots + b_m Y_m \end{aligned} \quad \text{(sorted)}$$

$$h = f \cdot g = ((((f_1 g + f_2 g) + f_3 g) + f_4 g) \cdots + f_n g)$$

$$h \div g = ((((h - f_1 g) - f_2 g) - f_3 g) - f_4 g) \cdots - f_n g)$$

How do we multiply and divide sparse distributed polynomials?

$$\begin{aligned} f &= a_1X_1 + a_2X_2 + \cdots + a_nX_n \\ g &= b_1Y_1 + b_2Y_2 + \cdots + b_mY_m \end{aligned} \quad \text{(sorted)}$$

$$h = f \cdot g = ((((f_1g + f_2g) + f_3g) + f_4g) \cdots + f_ng)$$

$$h \div g = ((((h - f_1g) - f_2g) - f_3g) - f_4g) \cdots - f_ng)$$

Example:

$$\begin{aligned} f &= x^n + x^{n-1} + \cdots + x \\ g &= y^m + y^{m-1} + \cdots + y \end{aligned}$$

- i^{th} merge can do $O(im)$ comparisons (sparse) $\Longrightarrow \sum_{i=1}^{n-1} im \in O(n^2m)$ comparisons in total

How do we multiply sparse polynomials?

$$\begin{aligned} f &= a_1 X_1 + a_2 X_2 + \cdots + a_n X_n \\ g &= b_1 Y_1 + b_2 Y_2 + \cdots + b_m Y_m \end{aligned} \quad \text{(sorted)}$$

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

$$f \times g \quad = \quad f_1 \times g_1 + f_2 \times g_1 + f_1 \times g_2 + f_2 \times g_2$$

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?

$$\begin{aligned} f &= a_1X_1 + a_2X_2 + \cdots + a_nX_n \\ g &= b_1Y_1 + b_2Y_2 + \cdots + b_mY_m \end{aligned} \quad \text{(sorted)}$$

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

$$f \times g \quad = \quad f_1 \times g_1 + f_2 \times g_1 + f_1 \times g_2 + f_2 \times g_2$$

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 $X_i \cdot Y_j$.

for $i = 1, 2, ..., n$ *do* *for* $j = 1, 2, ..., m$ *do*
set $Z = X_i \cdot Y_j$ *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

Bucket	f
1	$2xyz$
2	$-6x^3yz + 5xz^2 \quad + 3xz$
3	$+4x^3yz - 3xyz^3 + 2xyz^2 + 7xyz + 4$
$\vdots$	$\vdots$
$\log(\#f)$	$-7x^4y^3 + 3xyz^3 + 7xyz \quad - 7xz \quad + 4x - 3y + 2$

How do we multiply sparse polynomials?

Singular uses geobuckets (Yan, 1998).

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

Bucket	f
1	$2xyz$
2	$-6x^3yz + 5xz^2 + 3xz$
3	$+4x^3yz - 3xyz^3 + 2xyz^2 + 7xyz + 4$
$\vdots$	$\vdots$
$\log(\#f)$	$-7x^4y^3 + 3xyz^3 + 7xyz - 7xz + 4x - 3y + 2$

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).

1	2	3	4	5	6	7	8	
x^{13}	x^{10}	x^{9}	x^{1}	x^{6}	x^{7}			

- 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).

1	2	3	4	5	6	7	8	
x^{13}	x^{10}	x^{9}	x^{1}	x^{6}	x^{7}			

- 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).

1	2	3	4	5	6	7	8	
x^{13}	x^{10}	x^{9}	x^{1}	x^{6}	x^{7}			

- 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.

$$\begin{aligned} f &= a_1 X_1 + a_2 X_2 + \cdots + a_n X_n \\ g &= b_1 Y_1 + b_2 Y_2 + \cdots + b_m Y_m \end{aligned} \quad \text{(sorted)}$$

X_1Y_1 (a_1, b_1)	X_1Y_2 (a_1, b_2)	X_2Y_1 (a_2, b_1)	$\cdots$	X_iY_j (a_i, b_j)	$\cdots$

Multiplication using a binary heap.

$$\begin{aligned} f &= a_1 X_1 + a_2 X_2 + \cdots + a_n X_n \\ g &= b_1 Y_1 + b_2 Y_2 + \cdots + b_m Y_m \end{aligned} \quad \text{(sorted)}$$

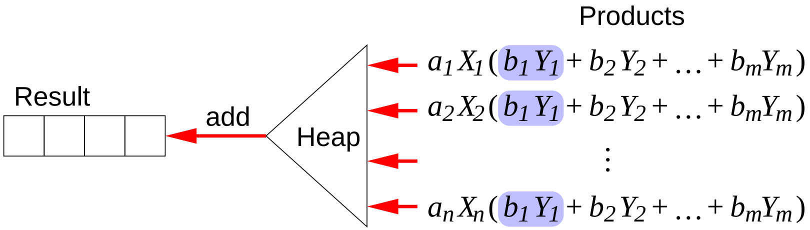
$X_1 Y_1$ (a_1, b_1)	$X_1 Y_2$ (a_1, b_2)	$X_2 Y_1$ (a_2, b_1)	$\cdots$	$X_i Y_j$ (a_i, b_j)	$\cdots$

- $O(nm \log(nm))$ comparisons, $O(nm)$ space.
- coefficient arithmetic using O(1) temporary registers.

Multiplication using a binary heap.

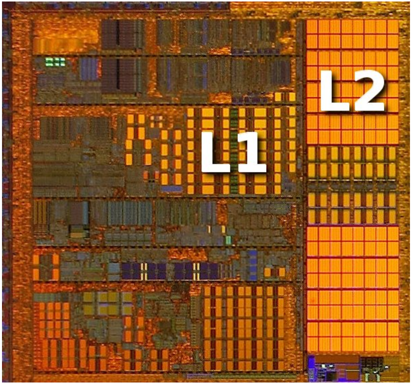
Johnson, 1974, a simultaneous n-ary merge:

$$\begin{aligned} f &= a_1 X_1 + a_2 X_2 + \cdots + a_n X_n \\ g &= b_1 Y_1 + b_2 Y_2 + \cdots + b_m Y_m \end{aligned} \quad \text{(sorted)}$$



- $O(nm \log n)$ 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.

$$\text{Dividing } f \div g \text{ compute } \boxed{f - \sum_{i=1}^{\#q} q_i \times g}$$

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

A divisor heap algorithm.

$$\text{Dividing } f \div g \text{ compute } \boxed{f - \sum_{i=2}^{\#g} g_i \times q}$$

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

Minimal heap division (Monagan & Pearce, 2008)

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

$$f \;-\; \underbrace{\sum_{i=1}^{\min(\#q,\#g)} q_i \times g}_{\text{quotient heap}} \;-\; \underbrace{\sum_{i=2}^{\#g} g_i \times (q_{\#g+1} + \cdots)}_{\text{divisor heap}}$$

- 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:

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

$$\Rightarrow\; (d_n \ldots (d_3(d_2(d_1 f - q_1 g) - q_2 g) - q_3 g) - \cdots - q_n g)$$

How many multiplications can this do ?

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

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

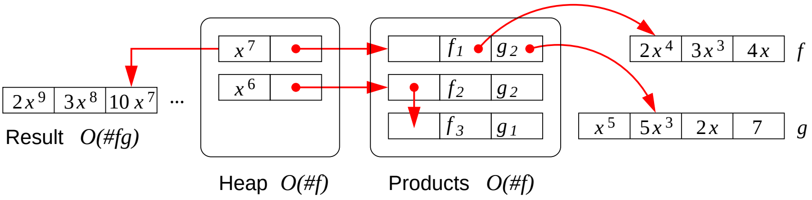
We can divide f by g, producing a quotient q using
$O(\#f + \#q\#g \ \log\min(\#q, \#g))$ comparisons.

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

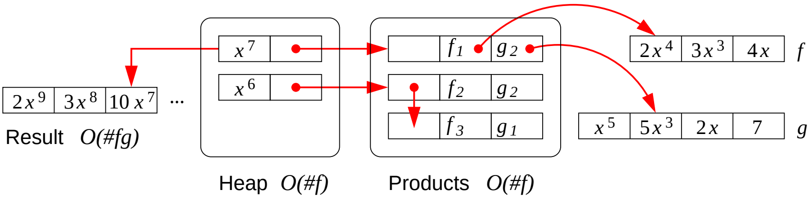
Chaining terms in the heap:



- 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.

$$\begin{aligned} q &= (1 + x + y + 2z^2 + 3t^3 + 5u^5)^\alpha \\ g &= (1 + u + t + 2z^2 + 3y^3 + 5x^5)^\beta \end{aligned}$$

Intel Core2 3.0 GHz 64-bit

α	β	$\#q$	$\#g$	$f = q \cdot g$	$f \div g$	max heap	real max
4	30	126	324632	2.99	2.77	126	126
8	18	1287	33649	2.27	2.21	1287	1161
12	12	6188	6188	2.44	2.24	12079	3895
18	8	33649	1287	2.38	2.46	2572	1231
30	4	324632	126	2.84	2.53	250	70

- 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

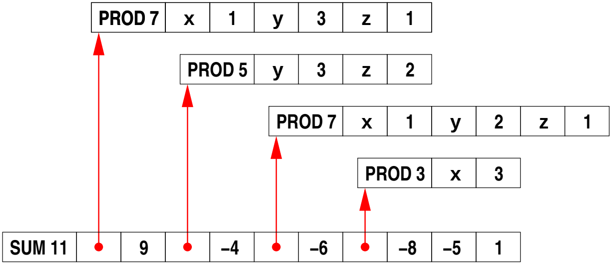
Distributed or recursive?

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

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

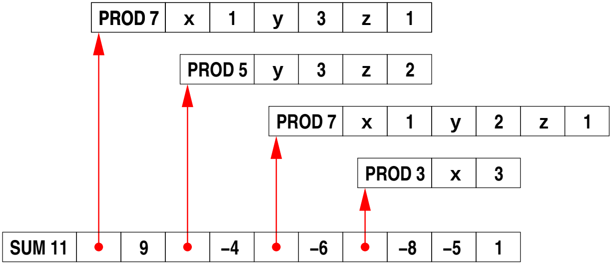
Sparse or dense?
Variables in or out?
Arrays or linked lists?

Maple's sum of products representation.



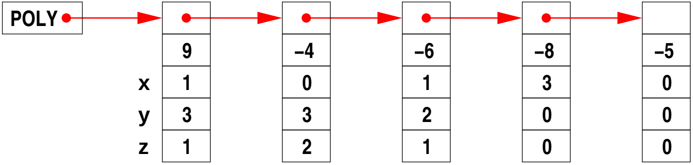
$$9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5$$

Maple's sum of products representation.

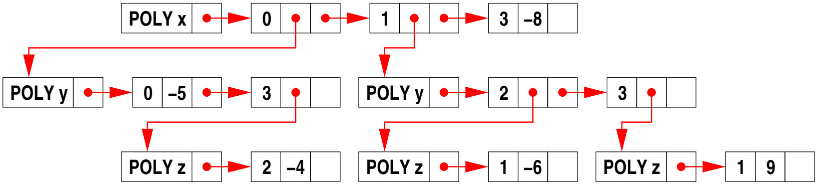


$$9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5$$

Singular's distributed representation.

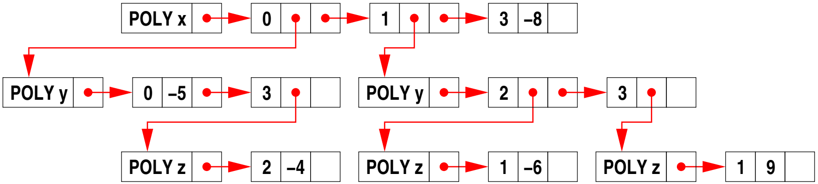


Trip's recursive sparse representation.



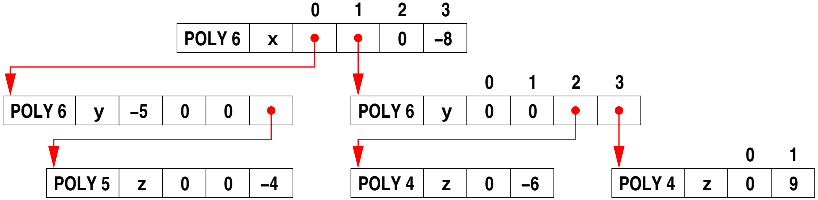
$$(-5y - 4z^2y^3) + (-6zy^2 + 9zy^3)x - 8x^3$$

Trip's recursive sparse representation.



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

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:

$$f := (1 + x + y + z)^{20} \qquad g := f + 1 \qquad p := f \cdot g$$

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)

Fateman's 2003 benchmark.

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

$$f := (1 + x + y + z)^{20} \qquad g := f + 1 \qquad p := f \cdot g$$

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?

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 $x > y > z$:

$$\text{One word :} \quad \boxed{i+j+k \mid i \mid j \mid k}$$

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

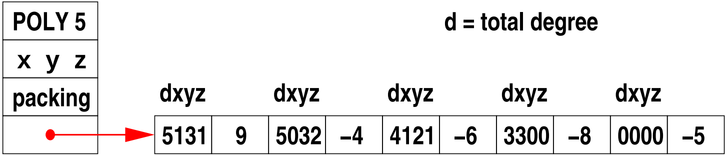
Our SDMP data structure

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

$$\text{One word : } \boxed{i+j+k \mid i \mid j \mid k}$$

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

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



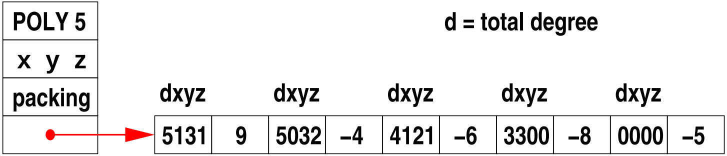
Our SDMP data structure

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

$$\text{One word}: \quad \boxed{i+j+k \mid i \mid j \mid k}$$

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

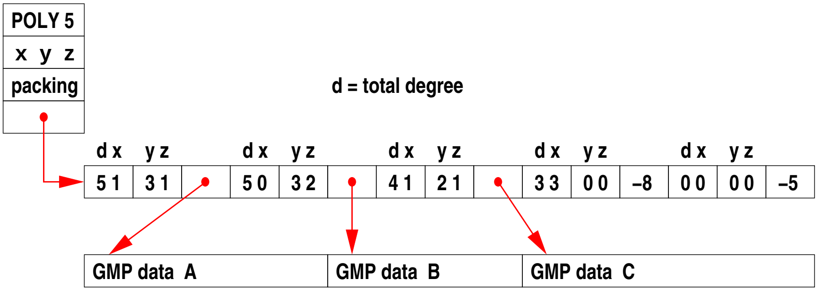
Packed array for: $9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5$



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

	64 bit		32 bit	
#variables	#bits	max deg	#bits	max deg
2	21		10	1023
3	16	65535	8	255
4	12	2047	6	63
5	10	1023	5	31
6	9	511	4	15
7	8	255	4	15
8	7	127	3	7
9	6	63	3	7
11	5	31	2	3
15	4	15	2	3
21	3	7	1	1
31	2	3	1	1
63	1	1	-	-

Space Data

Polynomials	#terms	density
$A = (1+x+y+z)^{20}$	1771	1.0000
$B = (1+x^2+y^2+z^2)^{20}$	1771	0.1445
$C = (w+x+y+z)^{20}$	1771	0.1667
$D = (w^2+x^2+y^2+z^2)^{20}$	1771	0.0131
$E = (1+x_1+x_2+\ldots+x_{50})^2$	1326	1.0000
$E = (1+x_1^2+x_2^2+\ldots+x_{50}^2)^2$	1326	0.0042

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

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

Table: Space in words assuming coefficients are immediate integers.

Benchmarks: How should we measure sparsity?

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

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

Benchmarks: How should we measure sparsity?

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

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

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 $W_{f\times g} = \dfrac{\#f\ \#g}{|\{X_iY_j\}|}$.

Benchmarks: How should we measure sparsity?

Let $\#f = \#terms(f),\; m = \#vars(f),\; d = \deg(f)$.

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

Let $f = a_1X_1 + a_2X_2 + ... + a_nX^n,\; g = b_1Y_1 + b_2Y_2 + ... + b_mY^m$.

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

Benchmarks: How should we measure sparsity?

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

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

Let $f = a_1X_1 + a_2X_2 + ... + a_nX^n$, $g = b_1Y_1 + b_2Y_2 + ... + b_mY^m$.

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

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 $\#f = \#terms(f),\ m = \#vars(f),\ d = \deg(f)$.

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

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

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 $\#f = \#terms(f)$, $m = \#vars(f)$, $d = \deg(f)$.

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

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

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

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

$46,376 \times 46,376 = 635,376$ terms $W = 3,385$	multiply $p = f \times g$	divide $q = p/f$
Maple 11	15986.16	13039.24
Singular 3-0-4 (distributed)	1482.36	364.49
Magma V2.14-7	679.07	610.62

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

$46,376 \times 46,376 = 635,376$ terms $W = 3,385$	multiply $p = f \times g$	divide $q = p/f$
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	-

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

$46,376 \times 46,376 = 635,376$ terms $W = 3,385$	multiply $p = f \times g$	divide $q = p/f$
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	-
sdmp (unpacked)	119.94	135.05
sdmp (packed)	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

$46,376 \times 46,376 = 635,376$ terms $W = 3,385$	multiply $p = f \times g$	divide $q = p/f$
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	-
sdmp (unpacked)	119.94	135.05
sdmp (packed)	47.33	58.44
Arithmetic cost	15.50	15.50

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

$26,599 \times 36,365 =$ 19,631157 terms $W = 49.27$	multiply $p = f \times g$ (megabytes) seconds	divide $q = p/f$ (megs) secs
Maple 11	14053.37	10760.36
Singular 3-0-4	(1538) 655.25	(1390) 206.60
Magma V2.14-7	(2365) 313.02	(1753) 5744.60

Benchmark 3: A sparse 10 variable problem.

$$
\begin{aligned}
f &= (x_1x_2 + x_2x_3 + x_3x_4 + x_4x_5 + x_5x_6 + x_6x_7 + x_7x_8 + x_8x_9 + x_9x_{10} + x_1x_{10} \\
&\quad + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + 1)^5 \\
g &= (x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2 + x_7^2 + x_8^2 + x_9^2 + x_{10}^2 \\
&\quad + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + 1)^5
\end{aligned}
$$

Intel Core2 3.0 GHz 64-bit

$26,599 \times 36,365 =$ $19,631157$ terms $W = 49.27$	multiply $p = f \times g$ (megabytes) seconds	divide $q = p/f$ (megs) secs
Maple 11	14053.37	10760.36
Singular 3-0-4	(1538) 655.25	(1390) 206.60
Magma V2.14-7	(2365) 313.02	(1753) 5744.60
Trip v0.99 (rationals)	(1218) 221.91	–
Pari 2.3.3 (w/ GMP)	109.27	109.69
sdmp (unpacked)	(1617) 175.97	(14.4) 162.37
sdmp (packed)	(304) 40.33	(3.4) 41.33

Benchmark 4: A very sparse 5 variable problem.

$$
\begin{aligned}
f &= (1 + x + y + 2z^2 + 3t^3 + 5u^5)^{12} \\
g &= (1 + u + t + 2z^2 + 3y^3 + 5x^5)^{12}
\end{aligned}
$$

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

Intel Core2 3.0 GHz 64-bit

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

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

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

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!
- $\text{Size(heap)} \in O(\min(m, n)) \Longrightarrow$ 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!
- $\mathrm{Size(heap)} \in O(\min(m, n)) \implies$ heap fits in cache.
- Multivariate pseudo-division is as efficient as exact division.

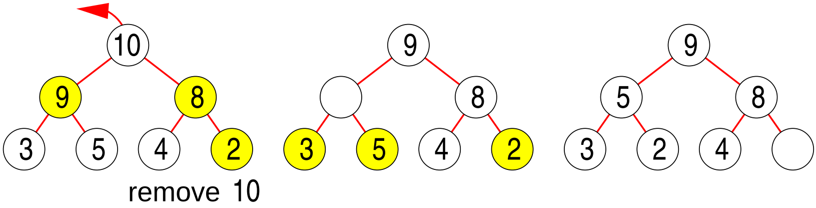
Conclusion

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.

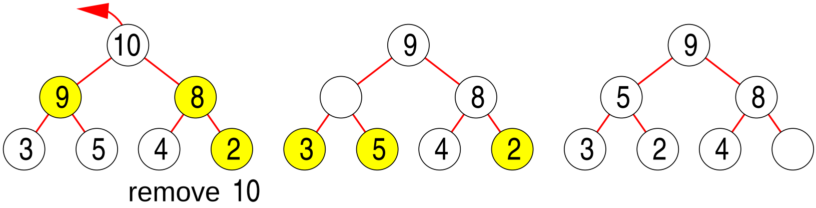
The heap extract operation.

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



The heap extract operation.

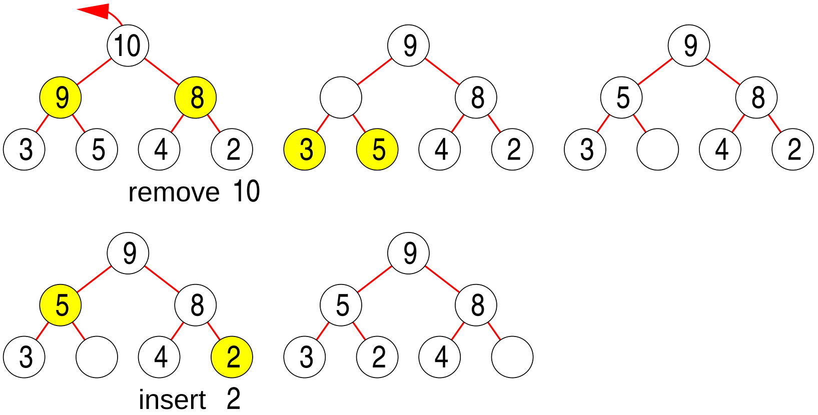
Algorithm 1: extract costs $2\log n - O(1)$ comparisons on average.



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!

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 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%.