Sparse Polynomial Arithmetic Part I: High Performance

Roman Pearce

CECM, Simon Fraser University

May 2008

Joint work with Michael Monagan, Simon Fraser University

Data Structures

"Which Polynomial Representation is Best?"

David Stoutemyer, 1984 Macsyma Users Conference

- Distributed or recursive?

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

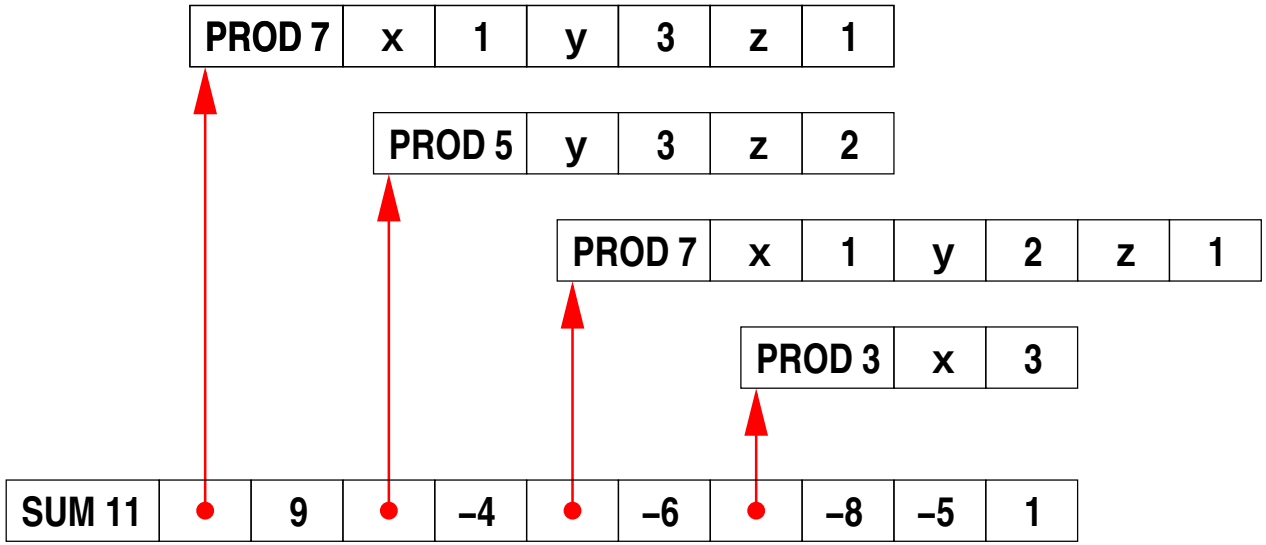
or

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

- Sparse or dense?
- Variables in or out?
- Arrays or linked lists?
- Sorted how?

Data Structures

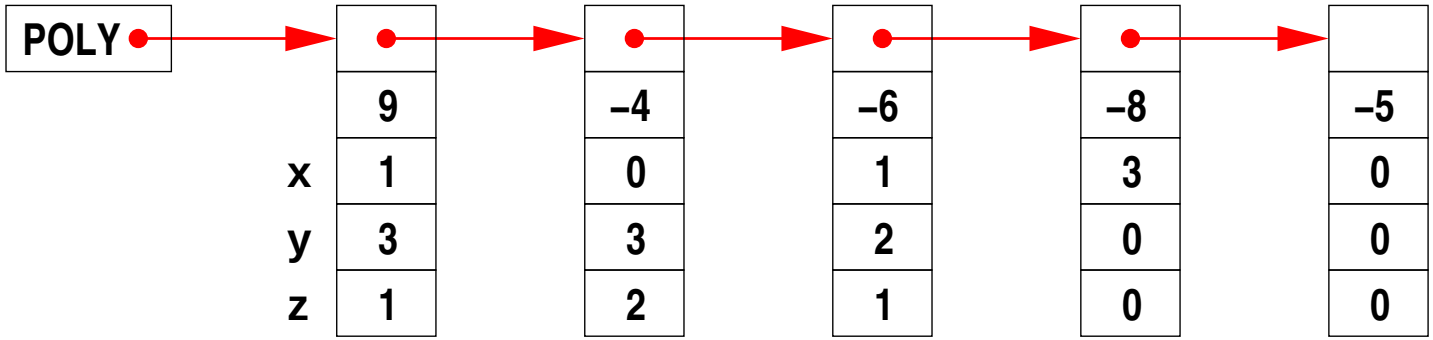
Maple's sum of products: $9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5$



- monomials are sparse and unsorted, but they are hashed
- storage for polynomial: $T(2n+3)+1$ words
- terms sorted by address of monomial
- monomial access incurs a cache miss: 150-200 cycles

Data Structures

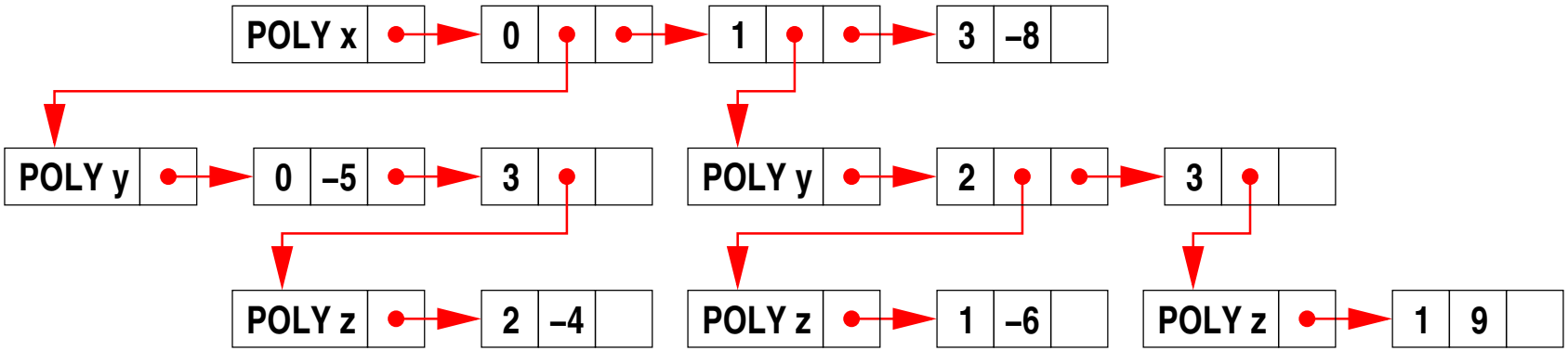
Singular's linked list: $9xy^3z - 4y^3z^2 - 6xy^2z - 8x^3 - 5$



- multiple exponents can be packed into a word
- storage: $T(n+2)$ words $\Rightarrow 3T$ with packing
- monomial operations are word operations
- next term incurs a cache miss: 150-200 cycles
- terms are allocated and freed $\Rightarrow$ memory fragmentation

Data Structures

Trip's recursive sparse: $(-5y - 4z^2y^3) + (-6zy^2 + 9zy^3)x - 8x^3$

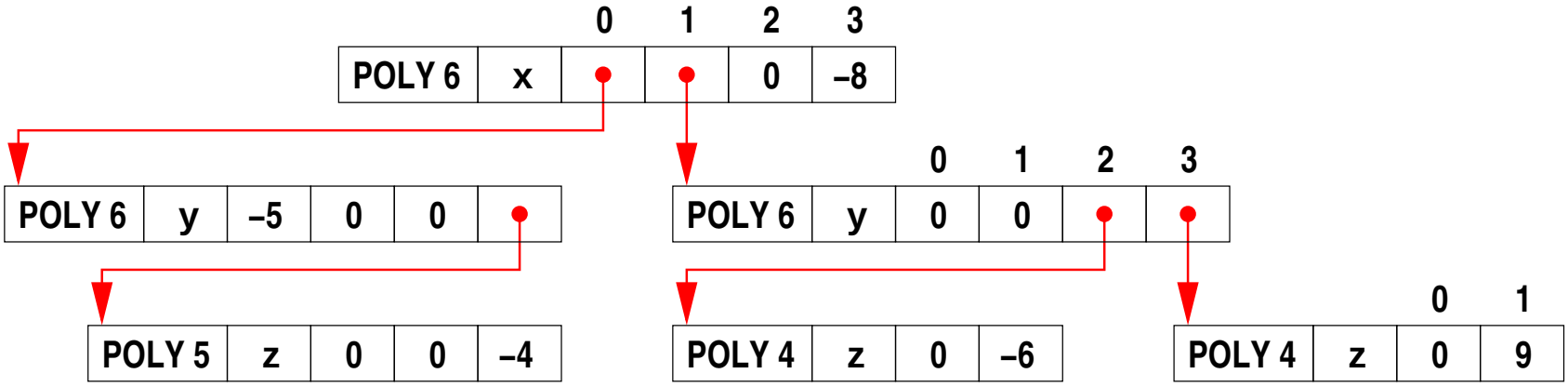


- algorithms are univariate and recursive
- not very space efficient
- pointers allow us to insert terms without copying data
- recursive computation amortizes searching and cache misses

Example: insert $y(z+1) \cdot y^2(z-1)$

Data Structures

Pari's recursive dense: $(-5y - 4z^2y^3) + (-6zy^2 + 9zy^3)x - 8x^3$



- no monomial operations, sorting is implicit
- storage: depends on degree, sparsity pattern
- recursive computation amortizes cache misses

Fast algorithms (Karatsuba, FFT) are rarely applicable:

- they fill in all lower levels (sparse $\Rightarrow$ dense - disaster!)
- degrees in the last variable are low

Data Structures

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

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

Pentium III 933 MHz:

```
Pari/GP 2.0.17              2.3 sec (dense recursive array)
MockMMA ACL 6.1/GMP4.1      3.3 sec (dense recursive array)
Hashing/GMP4.1,ACL6.1       4.7 sec (hash monomial=coefficient)
Reduce 3.7 (in CSL)         5.0 sec (sparse recursive list)
Singular 2.0.3              6.1 sec (sparse distributed list)
Macsyma (in ACL 6.1)        6.9 sec (sparse recursive list)
Maple VR4                  17.9 sec (sparse distributed DAG)
Maple 7                    50.0 sec (sparse distributed DAG)
```

Conclusions:

- recursive is faster than distributed
- dense recursive is faster than sparse recursive
- arrays are faster than lists

Sparse Multiplication

- But Fateman's benchmark is dense!
- In the recursive representation it is 100% dense!

Let $\#f$ denote the number of terms in f.

The dispersion of $f \cdot g$ is $D = \dfrac{\#fg}{(\#f\,\#g)} = \dfrac{\text{distinct monomials}}{\text{total number of terms}}$.

Then $1/D = (\#f\,\#g)/\#fg$ is the average number of terms added to get each term of the result.

Example (sparse):
$f = (1 + x + \cdots + x^n),\ g = (1 + y + \cdots + y^m),\ D = 1.$

Example (dense):
$f = (1 + x + y + z)^{20},\ g = f + 1,\ \#f = \#g = 1771,$
$\#fg = 12341.\ D = 12341/1771^2 = 0.003935,\ 1/D = 254.15.$

Sparse Multiplication

How can we multiply $f \cdot g$?

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

1. Merge each $f_i g$ into a sum:

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

$$f \div g = ((((f - q_1 g) - q_2 g) - q_3 g) \cdots - q_n g)$$

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

Sparse Multiplication

2. Divide and conquer merge of $f_i g$:

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

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

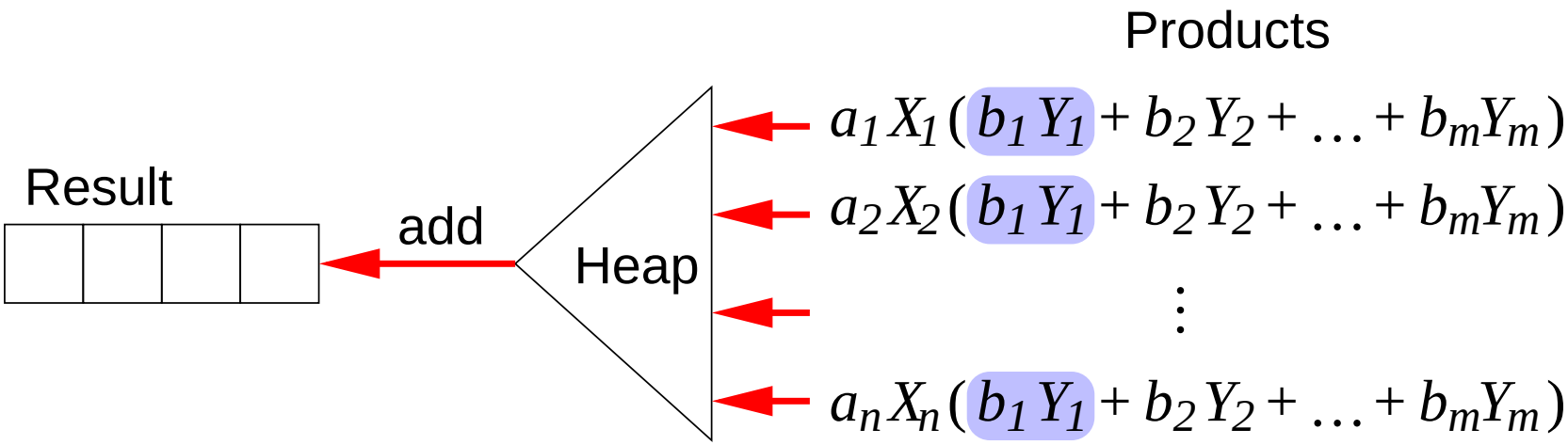
- $O(nm \log n)$ comparisons, $O(nm)$ memory
- Geobuckets (Yan, 1998) bucket i has at most 2^i terms

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

Sparse Multiplication

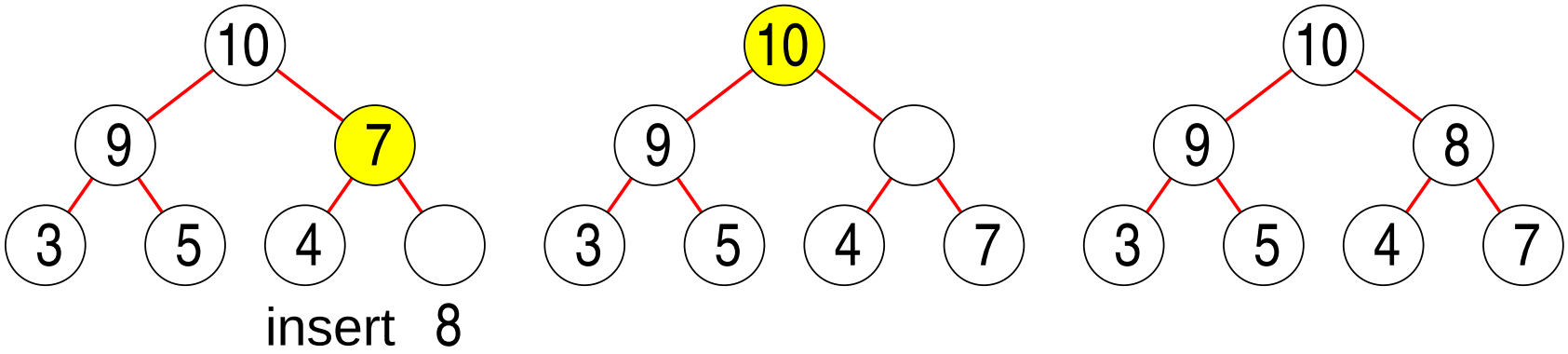
3. Simultaneous n-ary merge (Johnson, 1974):

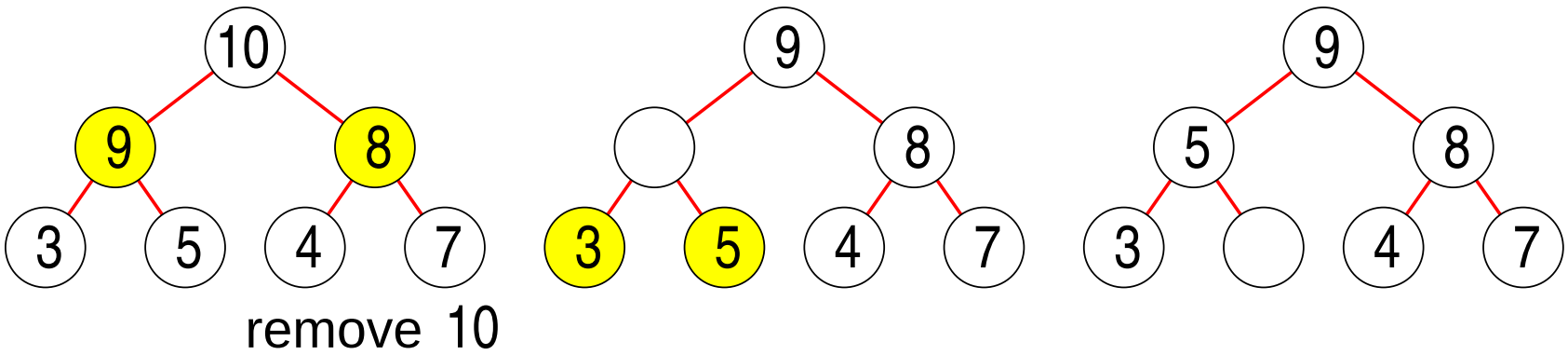
$$f = a_1 X_1 + a_2 X_2 + \cdots + a_n X_n$$
$$g = b_1 Y_1 + b_2 Y_2 + \cdots + b_m Y_m \quad \text{(sorted)}$$

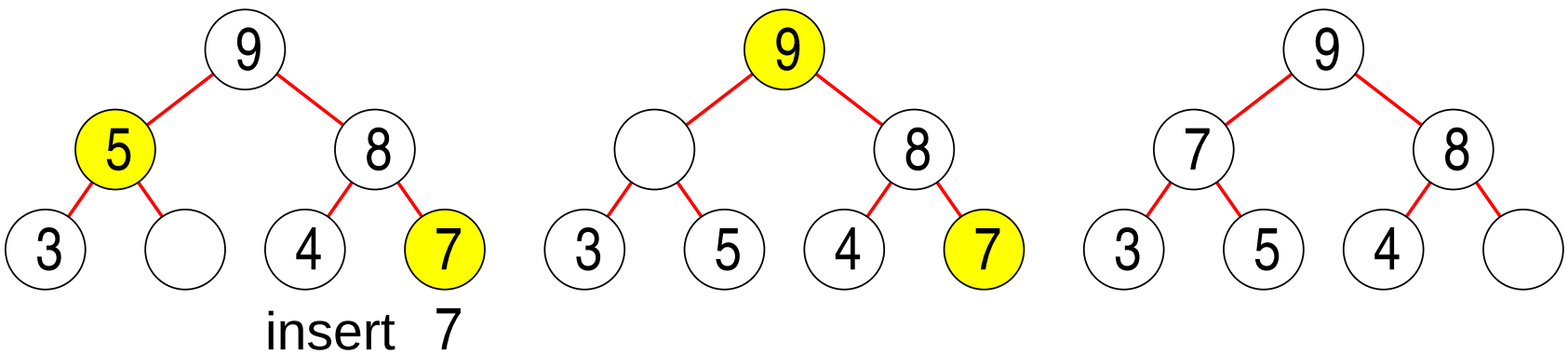


- $|heap| = n$, $O(\log n)$ comparisons to insert and extract
- $O(nm \log n)$ comparisons, $O(n)$ *working memory*
- no intermediate "garbage"

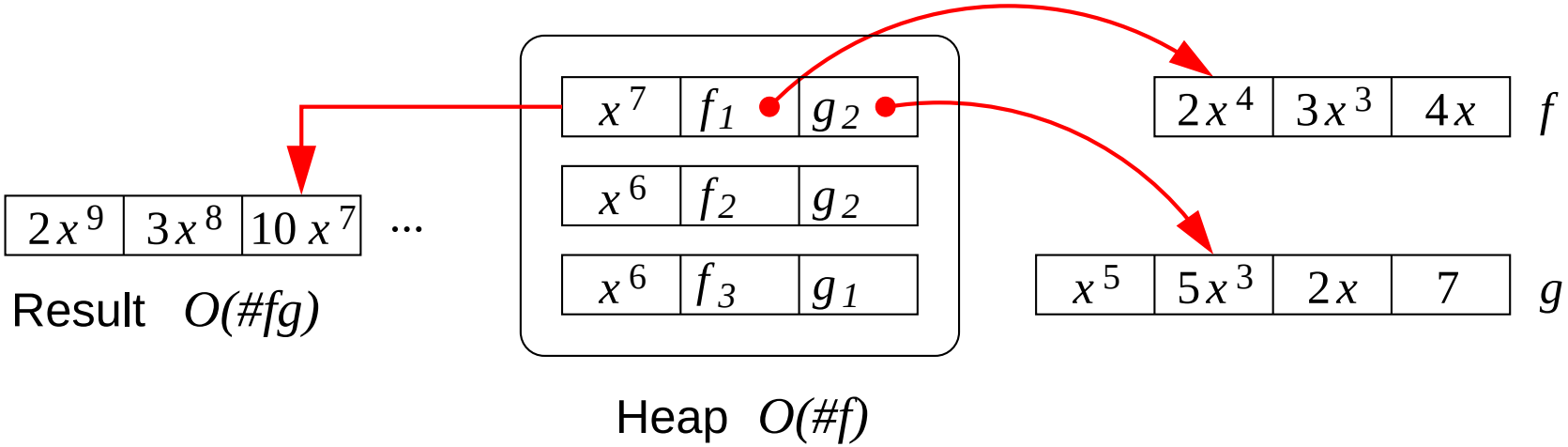
Binary Heaps





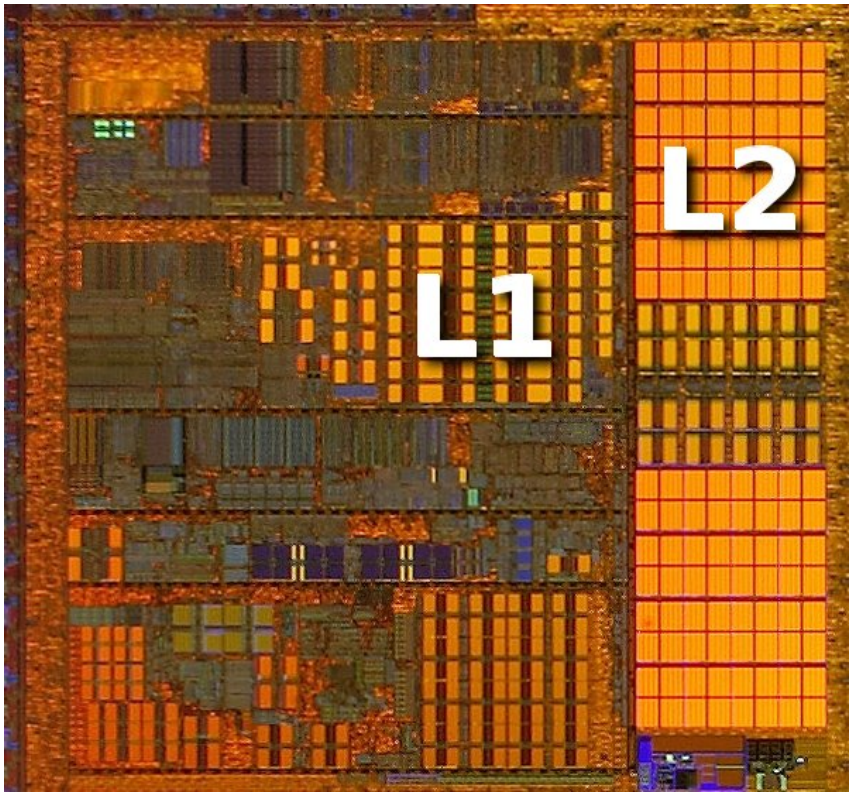


Sparse Multiplication

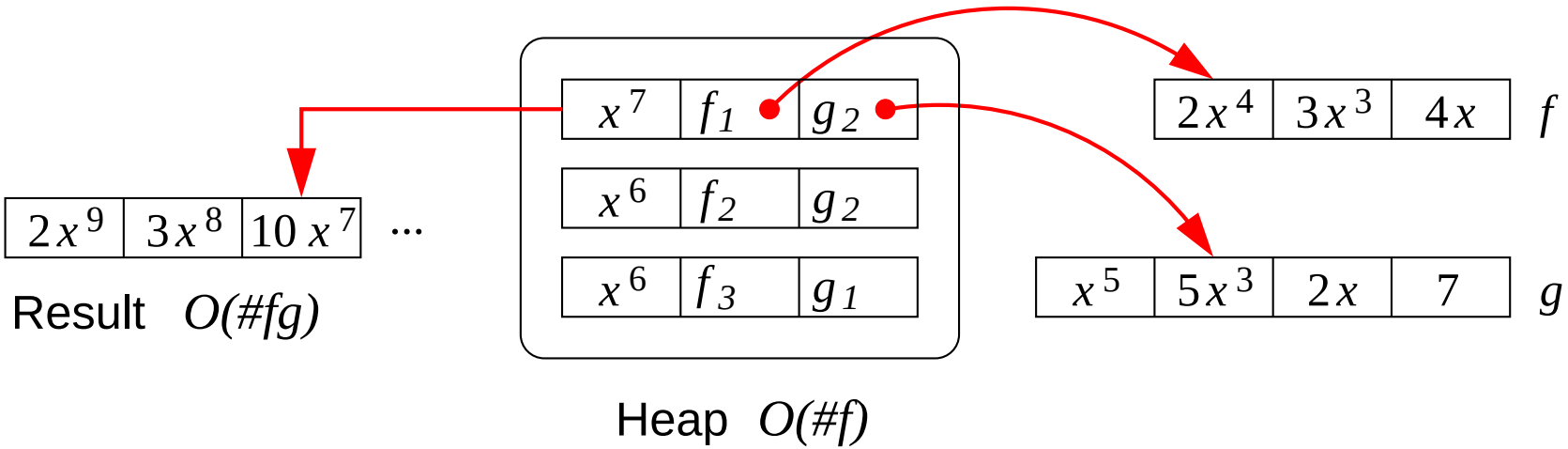


- the largest product is extracted from the heap
- coefficients are multiplied and added to a sum
- the term is copied to the end of the result

High Performance

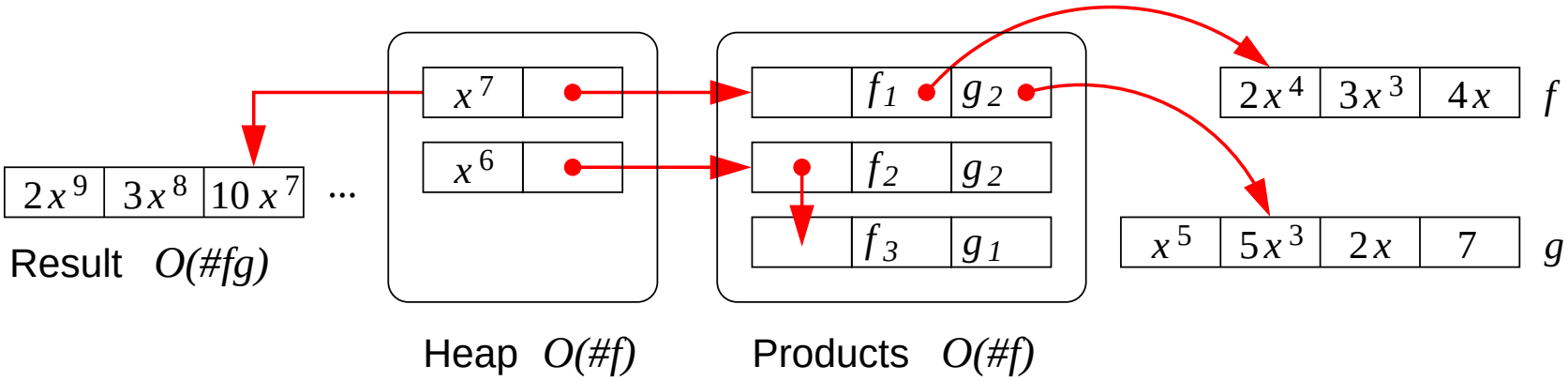


- L1: 3 cycles, L2: 20 cycles, DRAM: 150-200 cycles
- the larger polynomial is *streamed* into the cache
- products are *generated* inside the cache
- the heap sorts *on the chip*
- pointers updated in L1/L2
- result is stored to memory



Optimizations

Chaining terms in the heap:



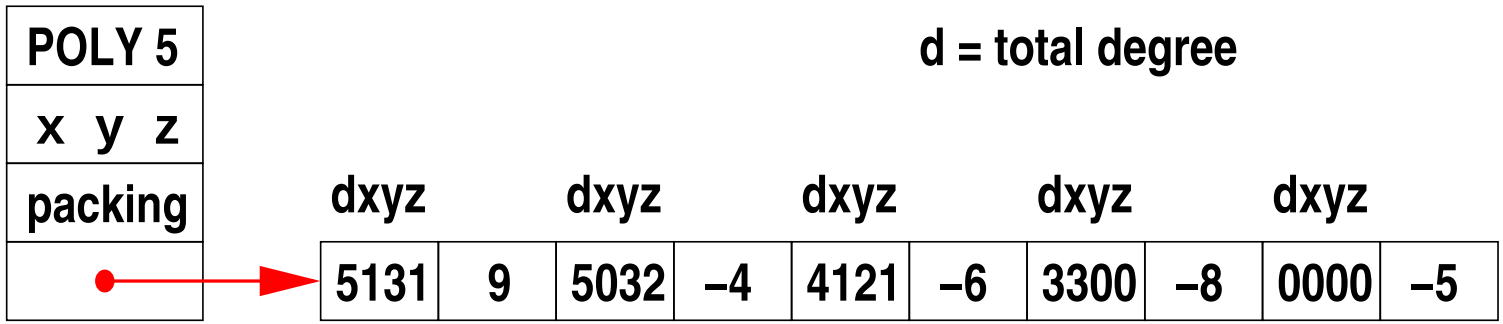
- terms are chained on insertion
- dense case: $O(nm \log n) \Rightarrow O(nm)$ comparisons

Also:

- arithmetic for wordsize integers coded in assembly
- one word monomials stored directly in the heap

Data Structure

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



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

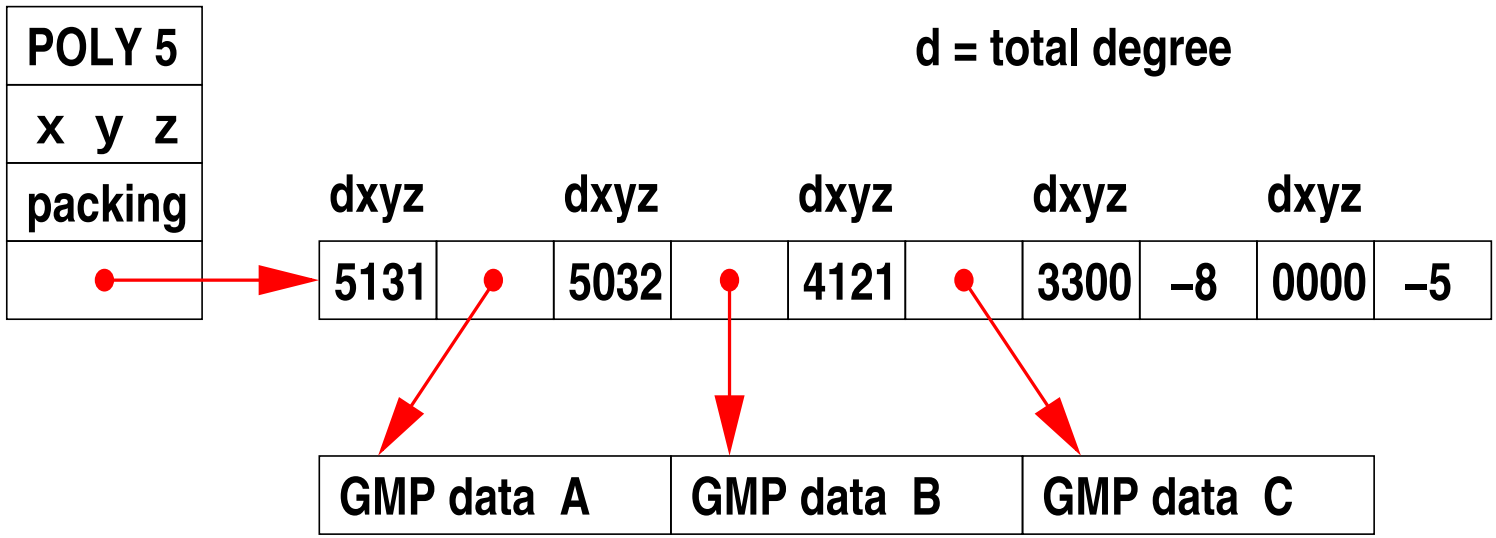
$i+j+k$	i	j	k

- exponents use an overflow bit for division
- comparisons and multiplications are one instruction:

$$\begin{aligned} 3 \text{ variables} &\Rightarrow 16 \text{ bits} \Rightarrow \text{degree} < 65536 \\ 7 \text{ variables} &\Rightarrow 8 \text{ bits} \Rightarrow \text{degree} < 256 \\ 8 \text{ variables} &\Rightarrow 7 \text{ bits} \Rightarrow \text{degree} < 128 \\ 15 \text{ variables} &\Rightarrow 4 \text{ bits} \Rightarrow \text{degree} < 16 \end{aligned}$$

Data Structure

Packed array: $Axy^3z - By^3z^2 - Cxy^2z - 8x^3 - 5$



- packing reduces the size and speeds everything up
- monomial operations are word operations
- no cache miss for sequential access
- blocks of terms allocated at a time
- data in rtables $\Rightarrow O(1)$ garbage collection
- size of polynomials limited only by RAM

Benchmarks

Dense Fateman problem:

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

Intel Core2 3.0 GHz 64-bit

$46376 \times 46376 = 635376$ terms $D = 0.000295 \quad 1/D = 3385$	multiply $p = f \cdot g$	divide $q = p/f$
sdmp (packed)	54.720	68.160
sdmp (unpacked)	131.730	126.320
Trip v0.99 (doubles)	43.664	-
Trip v0.99 (rationals)	108.224	-
Pari 2.3.3 (w/ GMP)	512.184	283.445
Magma V2.14-7	679.070	610.620
Singular 3-0-4	1482.360	364.490
Maple 11	15986.169	13039.248

- f and g have 61 bit coefficients
- $p = f \cdot g$ has 128 bit coefficients

Mike's Test

Compare with a dense univariate multiplication:

```
dense_mul := proc(poly f, poly g)
    N = #f + #g - 1;
    p = new poly with N terms;
    for k from 1 to N do
      for i from max(1, k - #g + 1) to min(k, #f) do
        P[k] += f[i] * g[k-i+1];
      end do;
    end do;
    return p;
end proc;
```

- measures the cost of coefficient arithmetic
- time is a lower bound

Mike's Test

Dense univariate multiplications:

Intel Core2 2.4 GHz 64-bit, 4MB L2

size	our time	arithmetic	proportion	optimal	heap structs
5000×5000	0.450	0.180	40.0%	2.50x	0.38 MB
10000×10000	1.820	0.710	39.0%	2.56x	0.76 MB
20000×20000	7.420	2.850	38.4%	2.60x	1.52 MB
30000×30000	16.910	6.420	37.9%	2.63x	2.29 MB
40000×40000	32.710	11.370	34.7%	2.88x	3.05 MB
50000×50000	62.520	17.950	28.7%	3.48x	3.81 MB
60000×60000	114.560	25.740	22.5%	4.45x	4.58 MB

For large problems:

- polynomial data is eventually squeezed out of the cache
- it would help to shrink the heap structures

Benchmarks

Sparse 10 variables:

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

Intel Core2 3.0 GHz 64-bit

$26599 \times 36365 = 19631157$ terms $D = 0.0203 \qquad 1/D = 49.27$	multiply $p = f \cdot g$	divide $q = p/f$
sdmp (packed)	40.330	41.330
sdmp (unpacked)	175.970	162.370
Trip v0.99 (doubles)	49.090	-
Trip v0.99 (rationals)	221.910	-
Pari 2.3.3 (w/ GMP)	109.270	109.692
Magma V2.14-7	313.020	5744.600
Singular 3-0-4	655.250	206.600
Maple 11	14053.371	10760.364

Benchmarks

Very sparse 5 variables:

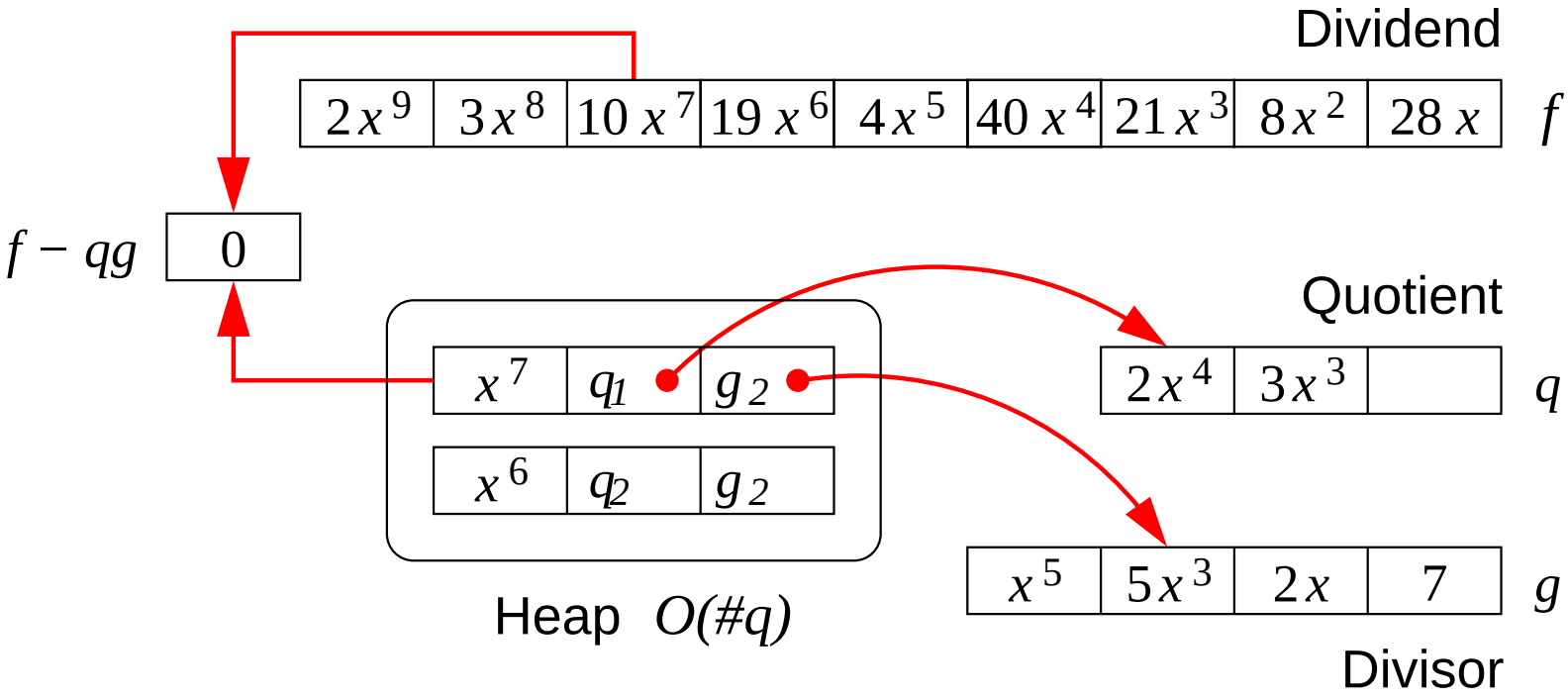
$$f = (1 + x + y + 2z^2 + 3t^3 + 5u^5)^{12}$$
$$g = (1 + u + t + 2z^2 + 3y^3 + 5x^5)^{12}$$

Intel Core2 3.0 GHz 64-bit

$6188 \times 6188 = 5821335$ terms $D = 0.152 \qquad 1/D = 6.58$	multiply $p = f \cdot g$	divide $q = p/f$
sdmp (packed)	2.020	2.100
sdmp (unpacked)	4.770	5.120
Trip v0.99 (doubles)	1.938	-
Trip v0.99 (rationals)	4.147	-
Pari 2.3.3 (w/ GMP)	53.980	30.689
Magma V2.14-7	23.770	151.990
Singular 3-0-4	58.910	39.250
Maple 11	332.716	367.462

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

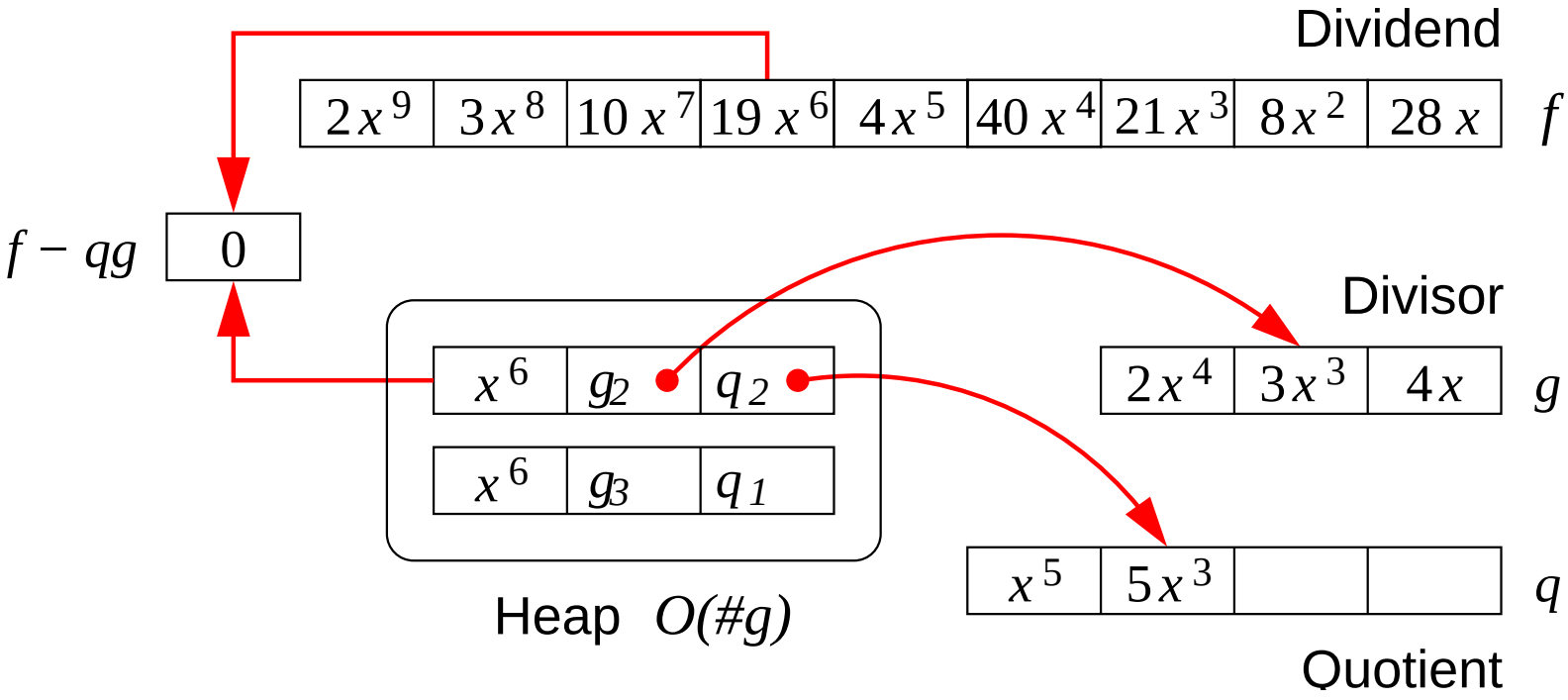
Sparse Division



Quotient Heap (Johnson, 1974): $f - \sum_{i=1}^{\#q} q_i \times (g - \mathrm{LT}(g))$

- $O(\#f + \#q\#g \log \#q)$ comparisons, $O(\#q)$ memory
- no intermediate "garbage"

Sparse Division



Divisor Heap (Monagan & Pearce, 2007): $f - \sum_{i=2}^{\#g} g_i \times q$

- $O(\#f + \#q\#g \log \#g)$ comparisons, $O(\#g)$ *working memory*
- large quotients are streamed into the cache

Sparse Division

Minimal Heap (Monagan & Pearce, 2008):

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

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

The multiplication of $q \times g$ is optimal:

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

Benchmarks

Sparse unbalanced divisions:

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

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	2.990	2.770	126	126
8	18	1287	33649	2.270	2.210	1287	1161
12	12	6188	6188	2.440	2.240	12079	3895
18	8	33649	1287	2.380	2.460	2572	1231
30	4	324632	126	2.840	2.530	250	70

- chaining reduces the size of the heap in practice
- division is as fast as multiplication

Pseudo Division

Pseudo division scales terms to avoid fractions:

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

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

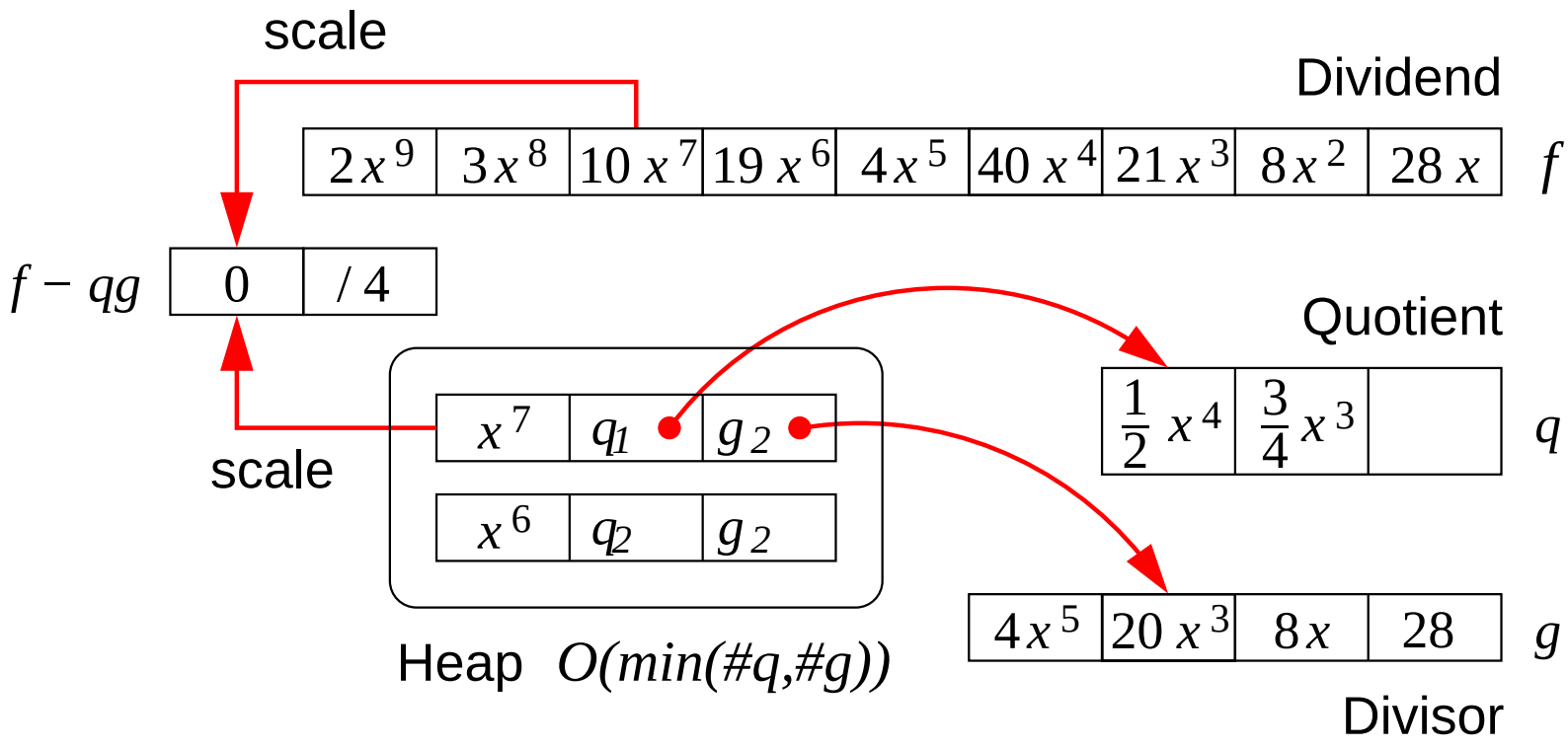
How many multiplications can this do ?

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

1^{st} merge: $nm + m$
2^{nd} merge: $(n-1)m + m$
3^{rd} merge: $(n-2)m + m$
...
n^{th} merge: $m + m$

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

Pseudo Division



- each term is scaled at most once
- pointer comparisons detect equal denominators
- as fast as a dedicated algorithm over $\mathbb{Z}$

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 divisions over $\mathbb{Z}$ require $\#q(\#g - 1)$ multiplications and $\#q$ divisions.*
- *Pseudo divisions over $\mathbb{Q}$ do at most $\#f + \#q(2\#g - 1)$ multiplications, $\#q(\#g + 1)$ divisions, and $\#q$ gcds.*
- *The algorithm uses $O(\min(\#q, \#g))$ working memory, plus $O(\#q)$ memory to store the quotient.*