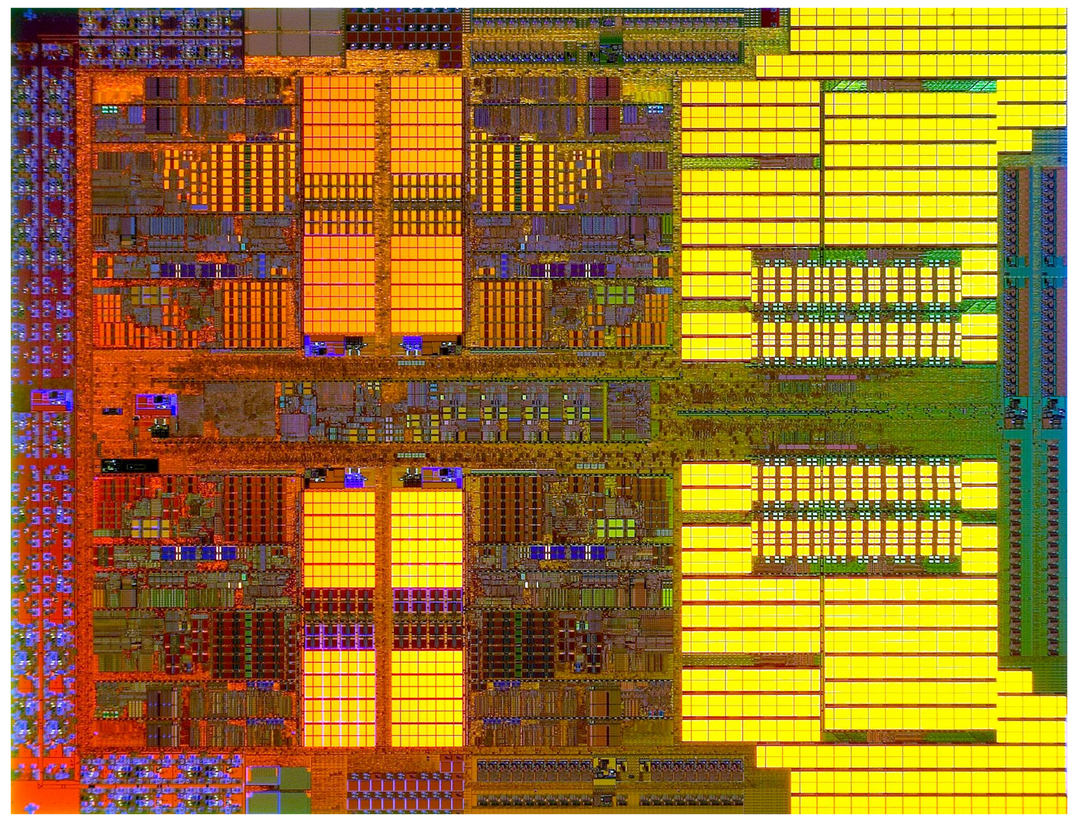
Parallel Sparse Polynomial Multiplication Using Heaps

Roman Pearce   Michael Monagan

Multicore Processors



The AMD Phenom II is an entry level CPU in 2009. It has four cores (left), each with 512KB L2 cache, and a 6 MB L3 cache (right) which can be used for communication between the cores.

Most programs do not benefit from additional cores.

Sparse Polynomials

Computer algebra systems like Maple spend a lot of time multiplying and dividing polynomials. E.g.:

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

$$g = 5x^3y^2 + 2xyz^3 - 3y^2z^2 + 7xy + 1$$

The polynomials are often stored in a *sparse format* which represents only non-zero terms.

To compute $f \times g$ we multiply each term of f by all the terms of g, sort the products, and add like terms.

Doubling the size of f and g quadruples the time or worse. A lot of time is spent doing large problems.

Even seemingly unrelated tasks like integration use sparse polynomial routines. Their speed is critical.

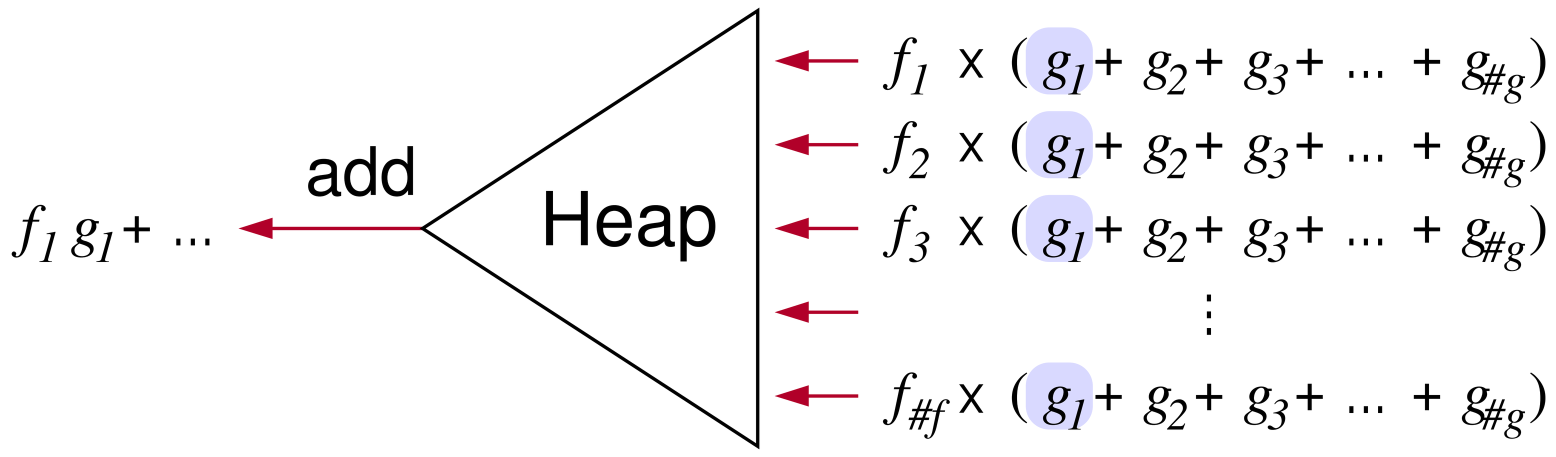
So how fast can we multiply sparse polynomials ?

Multiplication Using a Heap

The key to high performance is the CPU cache. Main memory is slow.

Johnson's algorithm uses a heap to simultaneously merge each $f_i \times g$.

The first term of each $f_i \times g$ is put into a heap, from which we extract terms in descending order. When $f_i \times g_j$ is extracted from the heap it is added to the end of the result and we insert $f_i \times g_{j+1}$ if it exists.



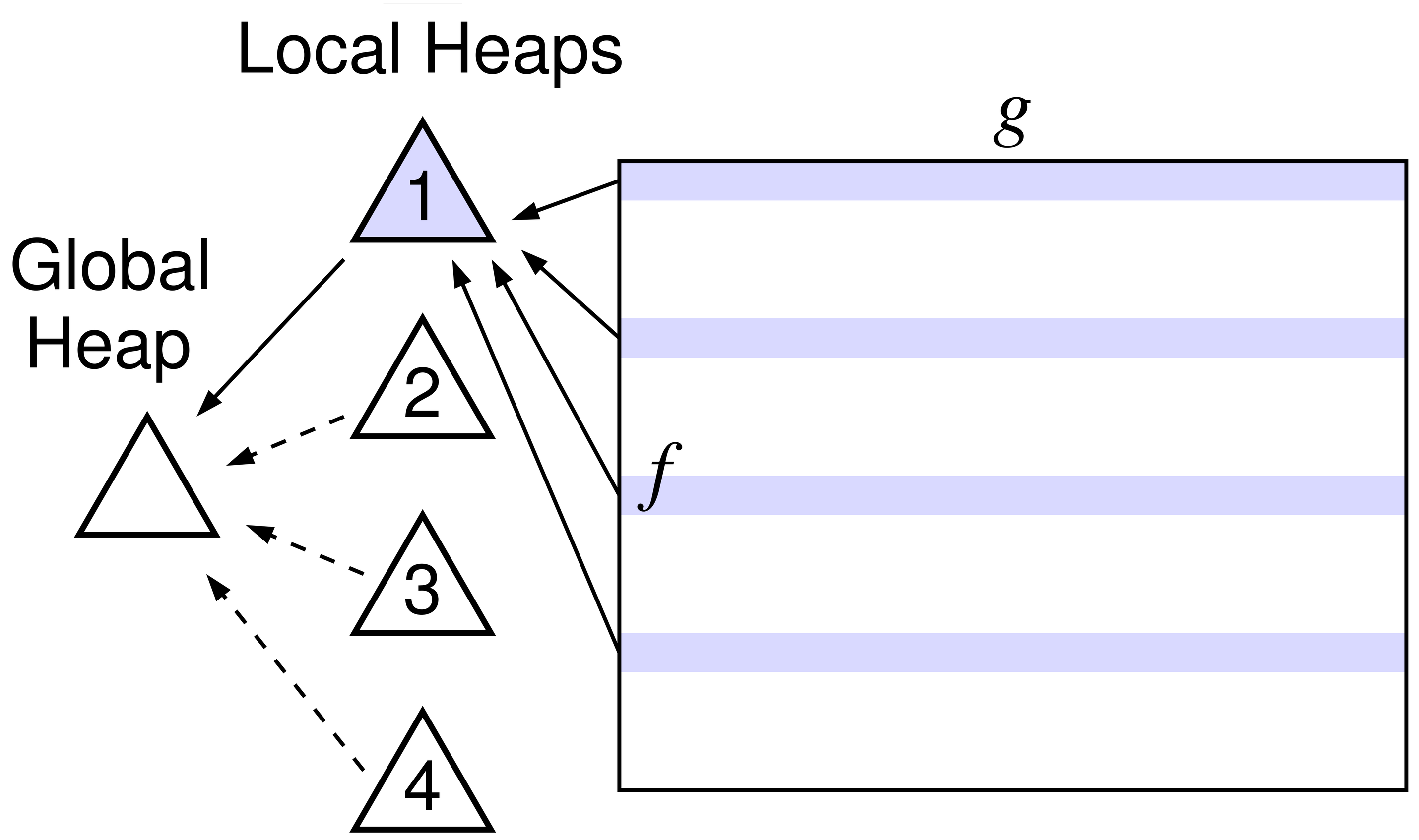
The heap is $O(\#f)$ so it fits in the CPU cache.

Inserting and extracting terms is $O(\log \#f)$ monomial comparisons.

We do at most $O(\#f \#g \log \#f)$ comparisons in total.

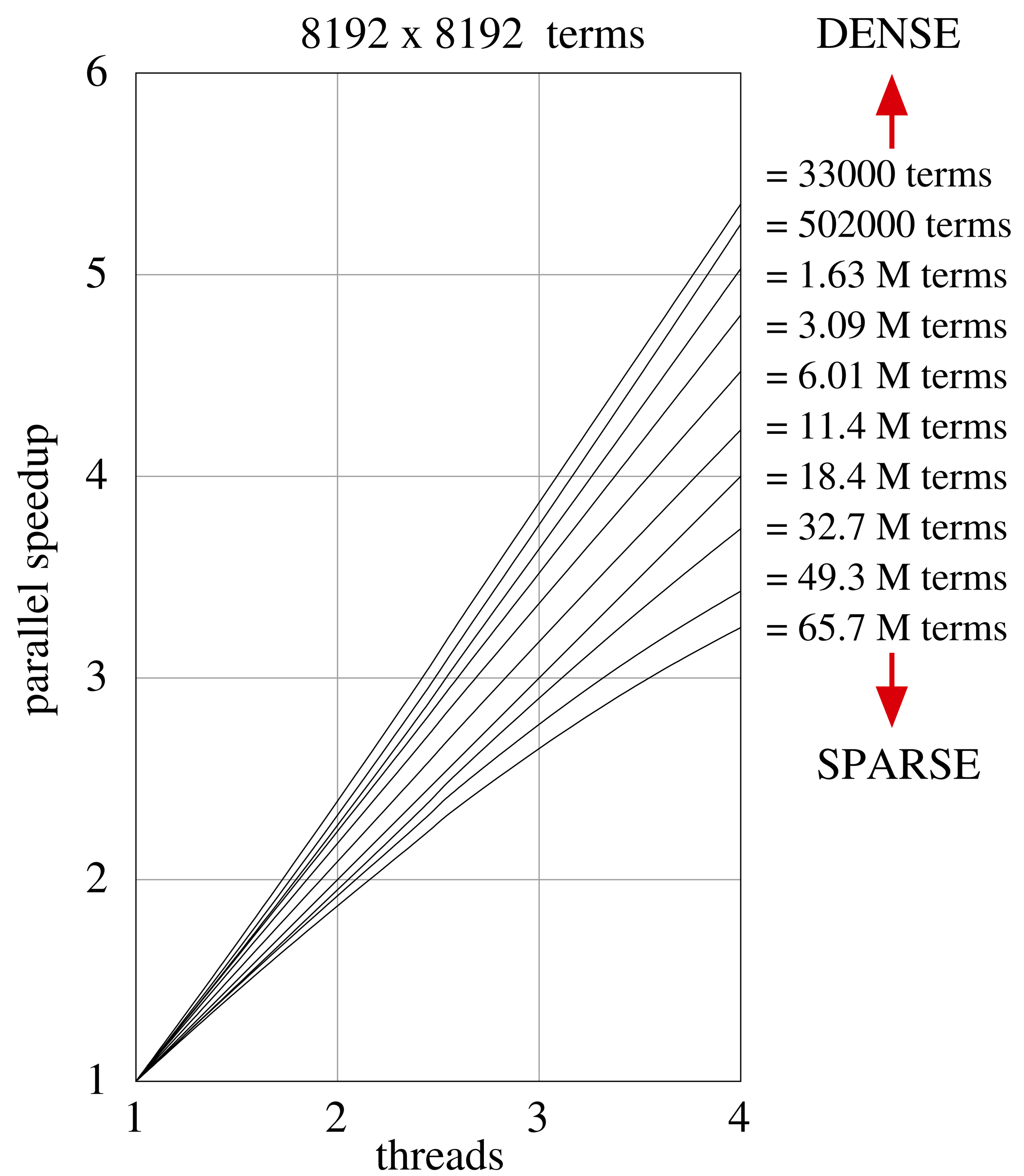
Parallel Algorithm

- Each thread uses a local heap to multiply some of the $f_i \times g$.
- Intermediate results are written to buffers in shared L3 cache.
- The threads take turns combining the buffers to form the result.



Parallel Performance

We multiplied random univariate polynomials with 8192 terms. Typical problems run 5x faster with 4 cores on a Core i7 CPU.



The speedup is relative to the fastest sequential code available. Our code is 50x faster than other computer algebra systems.

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

$46376 \times 46376 = 635376$ terms $W(f,g) = 3332$

	threads	Core i7		Core 2 Quad	
our software (sdmp)	4	11.48 s	6.15x	14.15 s	4.25x
	3	16.63 s	4.24x	19.43 s	3.10x
	2	28.26 s	2.50x	28.29 s	2.13x
	1	70.59 s		60.25 s	
Magma 2.15-8	1	526.12 s			
Pari/GP 2.3.3	1	642.74 s		707.61 s	
Singular 3-1-0	1	744.00 s		1048.00 s	
Maple 13	1	5849.48 s		9343.68 s	

[1] Stephen C. Johnson. Sparse Polynomial Arithmetic. ACM SIGSAM Bulletin, Volume 8, Issue 3 (1974) 63–71.

[2] Michael Monagan, Roman Pearce. Parallel Sparse Polynomial Multiplication Using Heaps. *Proceedings of ISSAC 2009.*

MITACS   NSERC CRSNG   Maplesoft Mathematics • Modeling • Simulation   cecm