

Re: sparse polynomial arithmetic

Source: <http://sci.tech-archive.net/Archive/sci.math.symbolic/2008-04/msg00047.html>

- *From:* parisse@xxxxxxxxxxxxxxxx
 - *Date:* Fri, 04 Apr 2008 07:04:20 +0200
-

I don't think sparseness is the main problem, but memory certainly is (512M RAM is not enough).

I disagree. Many approaches work fine on dense problems but suffer or die on sparse problems. The problem is sorting the terms. For example, if we multiply two dense polynomials with n terms and add the partial products one at a time into a sum, the algorithm is $O(n^2)$. But if the polynomials are sparse this is $O(n^3)$. See Stephen C. Johnson, "Sparse Polynomial Arithmetic" ACM SIGSAM Bulletin (1974). Division is exactly the same way, and one can speculate on the effect that naively coded division had on implementations of Buchberger's algorithm for all those years. Also related, pseudo division does an order of magnitude too many coefficient multiplications if implemented naively. These issues do not appear with dense univariate polynomials, but if you take those algorithms and run them on sparse multivariate polynomials, you get a degradation of performance that is an order of magnitude.

You misunderstood, my point was about giac implementation. Giac algorithm for multivariate polynomials is a mix of sparse and dense algorithm. It is dense with respect to the first variable and sparse with respect to other variables. For each degree of the first variable, a thread is launched to compute the corresponding coefficient, each coefficient being computed using sparse polynomial multiplication (and a hash_map for sorting). Therefore giac * can benefit from multiple processors. It's probably harder to make something similar with heap multiplication, any idea?

I estimate that your example 3 would require around 500M in giac just to hold the answer. I could run it up to exponent 10 (2096600 terms, 22.5s) but after that it swaps too much. I'll try with more memory, I estimate it

Re: sparse polynomial arithmetic

will be around 100s.

Example 3 has a nasty surprise:

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

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

multiply $p := f*g$;

divide $q := p/f$;

This is a $6188 \times 6188 = 5821335$ multiplication. The computation is inherently $O(6188^2)$ or $O(6188^2 \log 6188)$ if sorting is included. I believe you use a hash function so for you the overhead of sorting is $O(1)$. So, it should be $(46376/6188)^2 = 56$ times faster than the Fateman problem. I ran it with floats and giac took 31.01 sec (versus 322.47 above). You lost a factor of 5.4. How ?

Perhaps because sorting is no more $O(1)$ with this size of hash_map. Or it's because of different timings for memory access as you explained.

.