A Parallel Algorithm to Compute the Greatest Common Divisor of Sparse Multivariate Polynomials.

Jiaxiong Hu and Michael Monagan. Department of Mathematics, Simon Fraser University, British Columbia, Canada.

Efficient algorithms for computing greatest common divisors (GCD) of multivariate polynomials have been developed over the last 40 years. Many of the general purpose computer algebra systems are using either Zippel's GCD Algorithm [4] or the EEZ-GCD [3] Algorithm or both. Both algorithms sequentially interpolate variables one at a time which limits parallel speedup. Since multicore processors are now widely available, parallel algorithms are desirable. In this poster, we present a first multivariate GCD computation algorithm over $\mathbb{Z}$ which is based on the Ben-Or/Tiwari interpolation [1]. By using Ben-Or/Tiwari interpolation, we reduce the number of points needed to interpolate the GCD and improve parallelism.

Main idea:

Our algorithm considers multivariate GCD problems with at least three variables. The key of the algorithm is to determine the first modular GCD image which tells the correct *form* of the true GCD, then we use Zippel's sparse interpolation with this form to compute more modular images and apply Chinese remaindering to reconstruct the true GCD over $\mathbb{Z}$.

The first modular image can obtained as follows. Suppose $a,\ b \in \mathbb{Z}[x_1,\ldots,x_n]$ are the input polynomials and let

$$g = \gcd(a,b) = \sum_{i=1}^{l} c_i M_i(x_1,x_2)$$

where l is the number of terms of $g(x_1,x_2)$, M_i is the ith monomial of $g(x_1,x_2)$ and $c_i \in \mathbb{Z}[x_3,...,x_n]$ is the ith coefficient of $g(x_1,x_2)$. Our algorithm projects a and b down to bivariate polynomials by evaluating $\{x_3,\ldots,x_n\}$ at specific point $\{e_3^k,\ldots,e_n^k\}$ which satisfies the requirement of the Ben-Or/Tiwari interpolation. Then we compute bivariate

$$g_k = \gcd(a(x_1,x_2,e_3^k,\ldots,e_n^k), b(x_1,x_2,e_3^k,\ldots,e_n^k)) \in \mathbb{Z}_p[x_1,x_2],$$

where p is a carefully chosen prime. We redo this for $k = 0,1,2,\ldots,m$ until m is large enough. Now all bivariate GCDs should have the same monomials but different coefficients. For each monomial $M_i(x_1,x_2)$ in the g_k, we form an integer sequence by collecting M_i's coefficient in g_k $(0 \le k \le m)$. Then the Ben-Or/Tiwari algorithm is applied to this sequence to interpolate the coefficient $c_i \in \mathbb{Z}_p[x_3,\ldots,x_n]$. For this to work we require $m \ge 2t$ where $t = \max_{i=1}^{l}(\#\text{ terms } c_i)$, where t is not known in advance. So we must try $t = 2,4,8,16,\ldots$ stopping when we have redundancy. Obviously all polynomial coefficients c_i can be recovered in parallel. Moreover, all g_k can be computed in parallel as well. In general, this approach is easy to parallelize.

Instead of using powers of primes as evaluation points. We pick a *smooth prime* p of the form $p-1 = \prod_{i=3}^{n} q_i$, where the q_i are relatively prime and $q_i > \deg_{x_i} g$. For example, if $\deg_{x_3} g <= 6, \deg_{x_4} g <= 6$ and $\deg_{x_5} g <= 6$, we could pick $p = 1 + q_3 \times q_4 \times q_5 = 1 + 7 \times 8 \times 11 = 617$.

A further problem is that all underlying bivariate GCDs are monic over $\mathbb{Z}_p$. The leading coefficient of the true GCD is required to scale all bivariate GCDs consistently. We use Wang's leading coefficient algorithm [5] to solve this problem. We compute and factor the gcd $h \in \mathbb{Z}[x_3,\ldots,x_n]$ of the leading coefficients of $a,b \in \mathbb{Z}[x_3,\ldots,x_n][x_1,x_2]$. This creates another sequential step in our algorithm. This is the main reason why we reduce to bivariate GCDs instead of univariate – we likely reduce the size of h. We also likely reduce t and hence the number of g_k needed. If $a(x_1,x_2)$ and $b(x_1,x_2)$ are dense (which they often are in practice) we lose nothing by doing this.

Discrete logarithm method: Let ω be a primitive element in $\mathbb{Z}_p$ and let $\omega_i = \omega^{(p-1)/q_i}$ so that the ω_i are primitive q_i'th root of unity of relatively prime order. Suppose a monomial in c_i is $M = x^i y^j z^k$ where i,j,k are unknown and we have the value $m = M(w_1,w_2,w_3)$ over $\mathbb{Z}_p$. So

$$m = \omega_3^i \omega_4^j \omega_5^k = \omega^{i(p-1)/q_3} \omega^{j(p-1)/q_4} \omega^{k(p-1)/q_5}.$$

We compute $x = \log_w m$ by using Pohlig-Hellman which is easy because p is smooth. Now

$$x = i(p-1)/q_3 + j(p-1)/q_4 + k(p-1)/q_5 \mod (p-1) \quad (1)$$

To solve for i,j,k, we take (1) mod q_3 then q_4 then q_5. For example, (1) mod q_3, we obtain

$$x = i(p-1)/q_3 + 0 + 0 \mod q_3,$$

and we can solve this linear equation for i since $(p-1)/q_3 = q_4 q_5$ is relatively prime to q_3. Remarks: the requirement $q_i < \deg_{x_i} g$ means that two distinct monomials in c_i have distinct values at $\omega_3,\omega_4,\omega_5 \mod p$.

Example:

We compute the first modular GCD of $a = g \times \bar{a}$ and $b = g \times \bar{b} \in \mathbb{Z}[x,y,z,u]$, where

$$\bar{a} = zux + 1, \quad \bar{b} = zuy + 1 \quad \text{and} \quad g = (z+u)\,x^6y^5 + (z^4u^3)\,xy^2 + (u^5z^3+u)\,y,$$

with pure lexgraphic order $x > y > z > u$. We also suppose all bivariate GCDs to compute are over $\mathbb{Z}_p[x,y]$.

Leading coefficient: The GCD of the leading coefficients of $a(x,y)$ and $b(x,y)$ is $zu(z+u)$. Wang's algorithm heuristically determines that the leading coefficient of the true GCD is $C(z,u) = (z+u)$.

A smooth prime: We compute one univariate image of g in each variable (in parallel) and obtain $\deg_z g = 4$ and $\deg_u g = 5$. A smooth prime p for the discrete logarithm is 31 since $31-1 = 30 = 5 \times 6$ where $5 > \deg_z g$, $6 > \deg_u g$ and $\gcd(5,6) = 1$. A generator of $\mathbb{Z}_{31}$ is 17 which is randomly chosen. The evaluation point for (z,u) is

$$\mathbf{e}^k = (8^k \equiv 17^{k(31-1)/5} \mod 31,\ \ 26^k \equiv 17^{k(31-1)/6} \mod 31).$$

The Ben-Or/Tiwari: All computations below are over $\mathbb{Z}_{31}$.

First iteration: We compute g_0 and g_1 in parallel:

$$g_0 = C(\mathbf{e}^0)\gcd(a(x,y,\mathbf{e}^0), b(x,y,\mathbf{e}^0)) = 2x^6y^5 + 1\,xy^2 + 2\,y,$$
$$g_1 = C(\mathbf{e}^1)\gcd(a(x,y,\mathbf{e}^1), b(x,y,\mathbf{e}^1)) = 3x^6y^5 + 27\,xy^2 + 29\,y.$$

We apply Berlekamp/Massey Algorithm(BMA) to sequences $\{1,27\}$ and $\{2,29\}$ in parallel and obtain the connection polynomials $4v+1$ and $v+1$.

Second iteration: We compute g_2 and g_3 in parallel:

$$g_2 = C(\mathbf{e}^2)\gcd(a(x,y,\mathbf{e}^2), b(x,y,\mathbf{e}^2)) = 27x^6y^5 + 16\,xy^2 + 3\,y,$$
$$g_3 = C(\mathbf{e}^3)\gcd(a(x,y,\mathbf{e}^3), b(x,y,\mathbf{e}^3)) = 15x^6y^5 + 29\,xy^2 + 26\,y.$$

We apply BMA to $\{1,27,16,29\}$ and $\{2,29,3,26\}$ in parallel and obtain the connection polynomials $4v+1$ and $16v^2+2v+1$. The first connection polynomial remains the same and has degree 1, so there is 1 term in the coefficient of xy^2.

Third iteration: We compute g_k $(4 \le k \le 7)$ in parallel:

$$g_4 = C(\mathbf{e}^4)\gcd(a(x,y,\mathbf{e}^4), b(x,y,\mathbf{e}^4)) = 9x^6y^5 + 8\,xy^2 + 24\,y,$$
$$g_5 = C(\mathbf{e}^5)\gcd(a(x,y,\mathbf{e}^5), b(x,y,\mathbf{e}^5)) = 7x^6y^5 + 30\,xy^2 + 1\,y,$$
$$g_6 = C(\mathbf{e}^6)\gcd(a(x,y,\mathbf{e}^6), b(x,y,\mathbf{e}^6)) = 9x^6y^5 + 4\,xy^2 + 17\,y,$$
$$g_7 = C(\mathbf{e}^7)\gcd(a(x,y,\mathbf{e}^7), b(x,y,\mathbf{e}^7)) = 28x^6y^5 + 15\,xy^2 + 12\,y.$$

We run BMA with input $\{2,29,3,26,24,1,17,12\}$ and obtain $16v^2+2v+1$. Since the connection polynomial is unchanged and has degree 2, there are 2 terms in the coefficient of y. The next step of the Ben-Or/Tiwari algorithm is to compute the roots of $v+4$ and $v^2+2v+16$ (reversing the coefficients of connection polynomials). The roots of $v+4$ and $v^2+2v+16$ are 27 and $\{3,26\}$, which are the evaluated monomials in the coefficients of xy^2 and y respectively. By discrete logarithm method, 27 is corresponding to z^4u^3, 3 is corresponding to u^5z^3 and 26 is corresponding to u^1. The coefficient C_1 of z^4u^3 can be computed by equation $C_1 \times 27^0 = 1$. So $C_1 = 1$. The coefficients C_2, C_3 of u^5z^3, u can be obtained by linear system $C_2 \times 3^0 + C_3 \times 26^0 = 2$ and $C_2 \times 3^1 + C_3 \times 26^1 = 29$. So $C_2 = 1$ and $C_3 = 1$. Finally, we conclude that

$$G = (z+u)\,x^6y^5 + (z^4u^3)\,xy^2 + (u^5z^3+u)\,y \mod 31.$$

Maple implementation and benchmark:

We have implemented our algorithm in Maple without any parallelism and have compared it with Maple's default algorithm, an implementation of a Zippel based algorithm by de Kleine, Monagan and Wittkopf[2]. For most large problems, our algorithm outperforms Maple's. For example, for input polynomials having 40 variables and 4000 terms, our algorithm is almost 20 times faster. But it is an unfair game because the Maple's default algorithm is almost entirely coded in C. So a timing comparison makes no sense and is not provided here. We plan to do a parallel implementation of our algorithm in C by using Cilk in the future. However, Compared with Zippel's algorithm, it's obvious that our algorithm uses fewer evaluation points – $O(t)$ instead of $O((n-2)dt)$ and fewer trial divisions – $O(1)$ instead of $O(n)$. The numbers of evaluations (probes to the black box) to compute $\gcd(\bar{a}g, \bar{b}g)$ by both algorithms are provided below, where

$$\bar{a} = \text{randpoly}(X, degree = 10, terms = 20) + 1,$$
$$\bar{b} = \text{randpoly}(X, degree = 10, terms = 20) + 1,$$
$$g = \text{randpoly}(X, degree = \text{d}, terms = \text{T}) + 1.$$

# variables ($\lvert X\rvert$)	d	T	Zippel's algorithm	New algorithm
3	10	10	67	32
6	20	20	474	13
12	40	40	2038	14
12	40	100	3438	32
15	80	500	33653	50
20	50	200	17394	69
30	80	500	DNF	119
30	100	1000	DNF	176

References:

[1] M. Ben-Or, P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolate. *Proc. 20th annual ACM Symp Theory Comp*, 1988, 301–309.

[2] J. de Kleine, M. B. Monagan, A. D. Wittkopf. Algorithms for the Non-monic case of the Sparse Modular GCD Algorithm. *ISSAC'05*, ACM Press, 2005, 124–131.

[3] P. Wang. The EEZ-GCD Algorithm. *SIGSAM Bulletin*, 14, 1980, 50–60.

[4] R. E. Zippel. Probabilistic algorithms for sparse polynomials. *EUROSAM '79*, Springer-Verlag LNCS, 2, 1979, 216–226.

Maplesoft — Mathematics • Modeling • Simulation

cecm — centre for experimental & constructive mathematics