ACM Communications in Computer Algebra, Vol 47, No. 3, Issue 185, September 2013

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

Jiaxiong Hu and Michael Monagan
Department of Mathematics, Simon Fraser University
Burnaby, Canada, V5A 1S6
jha107@sfu.ca and mmonagan@cecm.sfu.ca

Extended Abstract

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 [5] or the EEZ-GCD [4] Algorithm or both. Both algorithms sequentially interpolate variables one at a time which limits parallel speedup. Since multi-core 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.

Our algorithm considers multivariate GCD problems with at least three variables. The structure of the algorithm is similar to Zippel's GCD Algorithm except the way we determine the first modular image which determines all the monomials. Once this correct *form* is obtained with those monomials, we use Zippel's sparse interpolation with this form to compute more modular images and apply Chinese remaindering to reconstruct the true GCD over $\mathbb{Z}$.

Our algorithm determines the first modular image 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 $g(x_1, x_2)$ and 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 $g(x_1, x_2)$. The 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 $t = \max_{i=1}^{l}(\#\text{ terms } c_i)$. Obviously all polynomial coefficients $c_i(x_3, ..., x_n)$ can be recovered in parallel. Moreover, the bivariate GCDs can be computed in parallel as well. In general, this approach is easy to parallelize.

Compared with Zippel's algorithm, our algorithm uses fewer evaluation points – $O(t)$ instead of $O((n-2)dt)$ and fewer trial divisions – $O(1)$ instead of $O(n)$. One disadvantage of our algorithm is that we do not know t. We must try $t = 2, 4, 8, 16, \ldots$ stopping when we have redundancy.

A problem with the original Ben-Or/Tiwari algorithm is the intermediate expression swell that occurs using $e_3^k, e_3^k, e_4^k, ... = 2^k, 3^k, 5^k, ...$ and computing over $\mathbb{Q}$. A modular version of the algorithm was first developed by Kaltofen, Lakshman and Wiley in [3]. Their algorithm uses a small prime q with a lifting technique to determine the monomials in the c_i. One lifts until $q^k > p_{n-2}^d$ where p_n denotes the n'th prime and $d \geq \max(\deg c_i)$. Instead, we adapt Giesbrecht, Labahn and Lee's method in [2]. We construct a smooth prime p so that we can efficiently compute discrete logarithms in $\mathbb{Z}_p$. The prime p is slightly larger than $\prod_{i=3}^n d_i$ where $d_i = \deg_{x_i} g$ thus of size $O(n \log d)$. To determine the d_i accurately we compute one univariate image of g in each variable (in parallel).

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 [4] to solve this problem. We compute and factor the gcd h 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 bivariate GCDs 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.

We have implemented our algorithm in Maple. For most large problems, it outperforms Maple's default multivariate GCD procedure, which is a Zippel based algorithm and almost entirely coded in C. For example, for input polynomials having 5 variables and 500 terms, our algorithm is almost 2 times faster than Maple's default procedure; with input polynomials having 40 variables and 4000 terms, our algorithm is almost 20 times faster. We have not yet attempted a parallel implmentation but plan to do so using Cilk. We expect that such an implementation will be much faster.

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] M. Giesbrecht, G. Labahn, W-S. Lee: Symbolic-numeric sparse interpolation of multivariate polynomials. *ISSAC'06*, 2006.

[3] E. Kaltofen, Y.N. Lakshman, J-M. Wiley: Modular rational sparse multivariate polynomial interpolation. *Watanabe and Nagata*, 1990, 135–139.

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

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