A Fast Parallel Sparse Polynomial GCD Algorithm

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

ABSTRACT

We present a parallel GCD algorithm for sparse multivariate polynomials with integer coefficients. The algorithm combines a Kronecker substitution with a Ben-Or/Tiwari sparse interpolation modulo a smooth prime to determine the support of the GCD. We have implemented our algorithm in Cilk C. We compare it with Maple and Magma's implementations of Zippel's GCD algorithm.

1. INTRODUCTION

Let A and B be two polynomials in $\mathbb{Z}[x_0, x_1, \ldots, x_n]$. In this paper we present a sparse modular GCD algorithm for computing $G = \gcd(A, B)$ the greatest common divisor of A and B. We will compare our algorithm with Zippel's sparse GCD algorithm from [25]. Zippel's algorithm is the main GCD algorithm currently used by Maple, Magma and Mathematica.

Let $A = G\bar{A} = \sum_{i=0}^{dA} a_i x_0^i$, $B = G\bar{B} = \sum_{i=0}^{dB} b_i x_0^i$ and $G = \sum_{i=0}^{dG} c_i x_0^i$ where a_i, b_i and c_i are in $\mathbb{Z}[x_1, \ldots, x_n]$. We will assume $\gcd(a_i) = 1$ and $\gcd(b_i) = 1$, that is, the contents have already been computed and divided out.

Let $\#A$ denote the number of terms in A and let $\mathrm{Supp}(A)$ denote the set of monomials appearing in A.

Let $\mathrm{LC}(A)$ denote the leading coefficient of A taken in x_0. Let $\Gamma = \gcd(LC(A), LC(B)) = \gcd(a_{dA}, b_{dB})$. Since $LC(G)|LC(A)$ and $\mathrm{LC}(G)|LC(B)$ it must be that $\mathrm{LC}(G)|\Gamma$ thus $\Gamma = LC(G)\Delta$ for some polynomial $\Delta \in \mathbb{Z}[x_1, \ldots, x_n]$.

Example 1. *If* $G = x_1x_0^2 + x_2x_0 + 3$, $\bar{A} = (x_2 - x_1)x_0 + x_2$ *and* $\bar{B} = (x_2 - x_1)x_0 + x_1 + 2$ *we have* $\#G = 3$, $\mathrm{LC}(G) = x_1$, $\Gamma = x_1(x_2 - x_1)$, $\Delta = x_2 - x_1$ *and* $\mathrm{Supp}(G) = \{x_1x_0^2, x_2x_0, 1\}$.

We provide an overview of the GCD algorithm. Let $H = \Delta \times G$ and $h_i = \Delta \times c_i$ so that $H = \sum_{i=0}^{dG} h_i x_0^i$. Our algorithm will compute H not G. After computing H it must then compute $\gcd(h_i)$ which is Δ and divide H by Δ to obtain G. We compute H modulo a sequence of primes $p_1, p_2, \ldots$, and recover the integer coefficients of H using Chinese remaindering. The use of Chinese remaindering is standard. Details may be found in [4, 8]. Let H_1 be the result of computing $H \mod p_1$. For the remaining primes we use the sparse interpolation approach of Zippel [25] which assumes $\mathrm{Supp}(H_1) = \mathrm{Supp}(H)$. From now on we focus on the computation of $H \mod p_1$.

To compute H mod p the algorithm will pick a sequence of points $\beta_1, \beta_2, \ldots$ from $\mathbb{Z}_p^n$, compute monic images

$$g_j = \gcd(A(x_0, \beta_j), B(x_0, \beta_j)) \in \mathbb{Z}_p[x_0]$$

of G, in parallel, then multiply g_j by the scalar $\Gamma(\beta_j) \in \mathbb{Z}_p$. Because the scaled image $\Gamma(\beta_j) \times g_j(x_0)$ is an image of a polynomial, H, we can use polynomial interpolation to interpolate each coefficient $h_i(x_1, \ldots, x_n)$ of H from the coefficients of the scaled images.

Let $t = \max_{i=0}^{dG} \#h_i$. The parameter t measures the sparsity of H. Let $d = \max_{i=1}^{n} \deg_{x_i} H$ and $D = \max_{i=0}^{dG} \deg h_i$. The cost of sparse polynomial interpolation algorithms is determined mainly by the number of points $\beta_1, \beta_2, \ldots$ needed and also the size of the prime p needed. These all depend on t, d and D. Table 1 below presents data for several sparse interpolation algorithms.

To get a sense for how large the prime needs to be for the different algorithms we include data in Table 1 for the following **benchmark problem:** Let $G, \bar{A}, \bar{B}$ have nine variables ($n = 8$), degree $d = 20$ in each variable, and total degree $D = 60$ (to better reflect real problems). Let G have 10,000 terms with $t = 1000$. Let $\bar{A}$ and $\bar{B}$ have 100 terms so that $A = G\bar{A}$ and $B = G\bar{B}$ have about one million terms.

	#points	size of p	benchmark
Zippel [1979]	$O(ndt)$	$p > 2nd^2t^2$	$= 6.4 \times 10^9$
BenOr/Tiwari [1988]	$O(t)$	$p > p_n^D$	$= 5.3 \times 10^{77}$
Monagan/Javadi [2010]	$O(nt)$	$p > nDt^2$	$= 4.8 \times 10^8$
Discrete Logs	$O(t)$	$p > (d+1)^n$	$= 3.7 \times 10^{10}$

Table 1: Some sparse interpolation algorithms

Notes: the figure $O(ndt)$ for Zippel's algorithm is for the worst case. The average case (for random inputs) is $O(dt)$ points. Also, Kaltofen and Lee showed in [14] how to modify Zippel's algorithm so that it will work for primes much smaller than $2nd^2t^2$.

The primary disadvantage of the Ben-Or/Tiwari algorithm is the size of the prime. In [12] Javadi and Monagan modify the Ben-Or/Tiwari algorithm to work for a prime of size $O(nDt^2)$ but using $O(nt)$ points.

The discrete logs method, first proposed by Murao and Fujise [19], is a modification of the Ben-Or/Tiwari algorithm which computes discrete logarithms in the cyclic group $\mathbb{Z}_p^*$.

ISSAC '16, July 19-22, 2016, Waterloo, ON, Canada

DOI: http://dx.doi.org/10.1145/2930889.2930903

We use this method. We give details for it in Section 1.2. The advantage over the Ben-Or/Tiwari algorithm is that the prime size is $O(n \log d)$ bits instead of $O(D \log n)$ bits.

In the GCD algorithm, not all evaluation points can be used. If $\gcd(\bar{A}(x_0, \beta_j), \bar{B}(x_0, \beta_j)) \neq 1$ then β_j is said to be *unlucky* and the image g_j cannot be used to interpolate H. In Zippel's algorithm, where the β_j are chosen at random from $\mathbb{Z}_p^n$, unlucky β_j, once identified, can simply be skipped. This is not the case for the evaluation point sequences used by the Ben-Or/Tiwari algorithm and the discrete logs method. In Section 1.4, we modify these point sequences to handle unlucky evaluation points.

Our modification for the discrete logarithm sequence increases the size of p which negates some of its advantage. This led us to consider using a Kronecker substitution on $x_1, x_2, \ldots, x_n$ to map the GCD computation into a bivariate computation in $\mathbb{Z}_p[x_0, y]$. Some Kronecker substitutions result in all evaluation points being unlucky so they cannot be used. We call these Kronecker substitutions *unlucky*. In Section 2 we show (Theorem 1) that there are only finitely many of them and how to detect them quickly so that a larger Kronecker substitution may be tried.

If a Kronecker substitution is not unlucky there can still be many unlucky evaluation points because the degree in y of the resulting polynomials is exponential in n. This prompted us to investigate the distribution of the unlucky evaluation points. Our next contribution (Theorem 2) is a result for the expected number of unlucky evaluations.

In Section 3 we assemble a Monte-Carlo GCD algorithm which chooses p and computes $H \bmod p$. We have implemented our algorithm in C and parallelized it using Cilk C. We did this initially for 31 bit primes then for 63 bit primes. The first timing results revealed that almost all the time (over 95%) was spent in evaluating $A(x_0, \beta_j)$ and $B(x_0, \beta_j)$. We describe an improvement for evaluation and how we parallelized it.

In Section 4 we compare our new algorithm with the C implementations of Zippel's algorithm in Maple and Magma. The timing results are very promising. For our benchmark problem, Maple takes 62,520 seconds, Magma dies with an internal error, and our new algorithm takes 4.47 seconds on 16 cores. We conclude by discussing some ideas for reducing the number of evaluation points and the size of p.

The proofs in the paper make use of the Schwartz-Zippel Lemma and properties of the Sylvester resultant. We state these results here for later use.

LEMMA 1. *Let F be a field and A and B be polynomials in $F[x_0, x_1, \ldots, x_n]$ with positive degree in x_0. Let $R = \text{res}_{x_0}(A, B)$ denote the Sylvester resultant of A and B. Then*

(i) R is a polynomial in $F[x_1, \ldots, x_n]$ and
(ii) $\deg R \leq \deg A \deg B$ (Bezout bound).

If $\alpha \in F^n$ satisfies $\deg_{x_0} A(x_0, \alpha) = \deg_{x_0}(A)$ and $\deg_{x_0} B(x_0, \alpha) = \deg_{x_0}(B)$ then
(iii) $\gcd(A(x_0, \alpha), B(x_0, \alpha)) \neq 1$
$\iff \text{res}_{x_0}(A(x_0, \alpha), B(x_0, \alpha)) = 0$ and
(iv) $\text{res}_{x_0}(A(x_0, \alpha), B(x_0, \alpha)) = R(\alpha)$.

Proofs may be found in Ch. 3 and Ch. 6 of [5]. Note that the degree condition on α means that the dimension of Sylvester's matrix for A and B in x_0 is the same as for $A(x_0, \alpha)$ and $B(x_0, \alpha)$ which proves (*iv*).

LEMMA 2. *(Schwartz-Zippel [22, 25]). Let F be a field and $f \in F[x_1, x_2, \ldots, x_n]$ be non-zero with total degree d and let $S \subset F$. If β is chosen at random from S^n then $\text{Prob}[f(\beta) = 0] \leq \frac{d}{|S|}$. In particular, if $F = \mathbb{Z}_p$ and $S = \mathbb{Z}_p$ then $\text{Prob}[f(\beta) = 0] \leq \frac{d}{p}$.*

1.1 Ben-Or Tiwari Sparse Interpolation

Let $C(x_1, \ldots, x_n) = \sum_{i=1}^t a_i M_i$ where $a_i \in \mathbb{Z}$ and M_i are monomials in $(x_1, \ldots, x_n)$. In our context, C represents one of the coefficients of $H = \Delta G$ we wish to interpolate. Let $D = \deg C$ and let $d = \max_{i=1}^n \deg_{x_i} C$ and let p_n denote the n'th prime. Let

$$v_j = C(2^j, 3^j, 5^j, \ldots, p_n^j) \quad \text{for } j = 0, 1, \ldots, 2t-1.$$

The Ben-Or/Tiwari sparse interpolation algorithm [3] interpolates $C(x_1, x_2, \ldots, x_n)$ from the $2t$ points v_j. Let $m_i = M_i(2, 3, 5, \ldots, p_n) \in \mathbb{Z}$ and let $\lambda(z) = \prod_{i=1}^t (z - m_i) \in \mathbb{Z}[z]$. The algorithm proceeds in 4 steps.

1 Compute $\lambda(z)$ from v_j using the Berlekamp-Massey algorithm [16] or the Euclidean algorithm [2, 24].

2 Compute the integer roots m_i of $\lambda(z)$.

3 Factor the integers m_i using trial division by $2, 3, \ldots, p_n$ from which we obtain M_i. For example, for $n = 3$, if $m_i = 45000 = 2^3 3^2 5^4$ then $M_i = {x_1}^3 {x_2}^2 {x_3}^4$.

4 Solve the following $t \times t$ linear system $Va = b$ for the unknown coefficients a_i in $C(x_1, \ldots, x_n)$.

$$\begin{bmatrix} 1 & 1 & \ldots & 1 \\ m_1 & m_2 & \ldots & m_t \\ {m_1}^2 & {m_2}^2 & \ldots & {m_t}^2 \\ \vdots & \vdots & \vdots & \vdots \\ {m_1}^{t-1} & {m_2}^{t-1} & \ldots & m_t^{t-1} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_t \end{bmatrix} = \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{t-1} \end{bmatrix} \tag{1}$$

The matrix V above is a transposed Vandermonde matrix. The linear system $Va = b$ can be solved in $O(t^2)$ arithmetic operations (see [26]). Note, the master polynomial $P(Z)$ in [26] is $\lambda(z)$.

Notice that the largest integer in $\lambda(z)$ is the constant term $\Pi_{i=1}^t m_i$ which is of size $O(tn \log D)$ bits. Moreover, in [13], Kaltofen, Lakshman and Wiley noticed a severe expression swell occurs if either the Berlekamp-Massey algorithm or the Euclidean algorithm is used to compute $\lambda(z)$ over $\mathbb{Q}$. For our purposes, because we want to interpolate H modulo a prime p, we run steps 1,2, and 4 modulo p. Provided $p > \max_{i=1}^t m_i \leq p_n^D$ the integers $m_i \bmod p$ remain unique. The roots of $\lambda(z) \in \mathbb{Z}_p[z]$ can be found using Rabin's algorithm [21] which has classical complexity $O(t^2 \log p)$.

In practice, t is not known in advance so the algorithm needs to be modified to also determine t. For p sufficiently large, if we compute $\lambda(z)$ after $j = 2, 4, 6, \ldots$ points, we will see $\deg \lambda(z) = 1, 2, 3, \ldots, t-1, t, t, t, \ldots$ with high probability. Thus we simply wait until the degree of $\lambda(z)$ does not change. This is first discussed by Kaltofen, Lee and Lobo in [14]. We will return to this in Section 3.1.

Let $M(t)$ denote the cost of multiplying two polynomials of degree t in $\mathbb{Z}_p[t]$. The fast Euclidean algorithm can be used to accelerate Step 1. It has complexity $O(M(t) \log t)$. See Ch. 11 of [7]. Computing the roots of $\lambda(z)$ in Step 2 can be done in $O(M(t) \log t \log p)$. See Ch 14 of [7]. Step 4 may be done in $O(M(t) \log t)$ using fast interpolation. See Ch 10 of [7].

1.2 Ben-Or/Tiwari with discrete logarithms

The discrete logarithm method modifies the Ben-Or/Tiwari algorithm so that the prime needed is a little larger than $(d+1)^n$ thus of size is $O(n \log d)$ bits instead of $O(D \log n)$. Murao and Fujise [19] were the first to use this method. Some practical aspects of it are discussed by van der Hoven and Lecerf in [11]. We explain how the method works.

To interpolate $C(x_1, \ldots, x_n)$ we first pick a prime p of the form $p = q_1 q_2 q_3 \ldots q_n + 1$ satisfying $q_i > \deg_{x_i} C$ and $\gcd(q_i, q_j) = 1$. Finding such primes is not difficult and we omit presenting an explicit algorithm here.

Next we pick a random primitive element $\alpha \in \mathbb{Z}_p$ which we can do using the partial factorization $p - 1 = q_1 q_2 \ldots q_n$ (see [23]). We set $\omega_i = \alpha^{(p-1)/q_i}$ so that $\omega_i^{q_i} = 1$ and replace the evaluation points $(2^j, 3^j, \ldots, p_n^j)$ with $(\omega_1^j, \omega_2^j, \ldots, \omega_n^j)$. After Step 1 we factor $\lambda(z)$ in $\mathbb{Z}_p[z]$ to determine the m_i. If $M_i = \prod_{k=1}^n x_k^{d_k}$ we have $m_i = \prod_{k=1}^n \omega_k^{d_k}$. To compute d_k in Step 3 we compute the discrete logarithm $x := \log_\alpha m_i$, that is, solve $\alpha^x \equiv m_i \pmod p$ for $0 \le x < p-1$. We have

$$x = \log_\alpha m_i = \log_\alpha \prod_{k=1}^n \omega_k^{d_k} = \sum_{k=1}^n d_k \frac{p-1}{q_k}. \qquad (2)$$

Taking (1) mod q_k we obtain $d_k = x[(p-1)/q_k]^{-1} \mod q_k$. Step 4 remains unchanged.

For $p = q_1 q_2 \ldots q_n + 1$, a discrete logarithm can be computed in $O(\sum_{i=1}^n \sqrt{q_i})$ multiplications in $\mathbb{Z}_p$ using the Pohlig-Helman algorithm. See [20, 23]. Since the $q_i \sim d$ this leads to an $O(n\sqrt{d})$ cost. Kaltofen showed in [15] that this can be made polynomial in $\log d$ and n if one uses a Kronecker substitution to reduce multivariate interpolation to a univariate interpolation and uses a prime $p > (d+1)^n$ of the form $p = 2^k s + 1$ with s small.

1.3 Bad and Unlucky Evaluation Points

Let A and B be non constant polynomials in $\mathbb{Z}[x_0, \ldots, x_n]$ with $G = \gcd(A, B)$ and let $\bar{A} = A/G$ and $\bar{B} = B/G$. Let p be prime such that $LC(A)LC(B) \mod p \neq 0$.

Definition 1. *Let $\alpha \in \mathbb{Z}_p^n$ and let $\bar{g}_\alpha(x) = \gcd(\bar{A}(x, \alpha), \bar{B}(x, \alpha))$. We say α is* bad *if $LC(A)(\alpha) = 0$ or $LC(B)(\alpha) = 0$ and α is* unlucky *if $\deg \bar{g}_\alpha(x) > 0$.*

Example 2. *Let $G = (x_1 - 16)x_0 + 1$, $\bar{A} = x_0^2 + 1$ and $\bar{B} = x_0^2 + (x_1 - 1)(x_2 - 9)x_0 + 1$. Then $LC(A) = LC(B) = x_1 - 16$ so $\{(16, \beta) : \beta \in \mathbb{Z}_p\}$ are bad and $\{(1, \beta) : \beta \in \mathbb{Z}_p\}$ and $\{(\beta, 9) : \beta \in \mathbb{Z}_p\}$ are unlucky.*

The algorithm cannot reconstruct G using the image $g_\alpha(x) = \gcd(A(x, \alpha), B(x, \alpha))$ if α is unlucky. Brown's idea in [4] to detect unlucky α is based on the following Lemma.

Lemma 3. *Let α and g_α be as above and $h_\alpha = G(x, \alpha) \mod p$. If α is not bad then $h_\alpha | g_\alpha$ and $\deg_x g_\alpha \ge \deg_x G$.*

For a proof of Lemma 3 see Lemma 7.3 of [8]. Brown only uses α which are not bad and the images $g_\alpha(x)$ of least degree to interpolate G. The following Lemma implies if the prime p is large then unlucky evaluations points are rare.

Lemma 4. $\text{Prob}[\begin{array}{c}\alpha \text{ is bad} \\ \text{or unlucky}\end{array}] \le \frac{\deg A \deg B + \deg A + \deg B}{p - \deg A - \deg B}$.

Proof: $\text{Prob}[\alpha \text{ is bad}] = \text{Prob}[\ \text{LC}(A)(\alpha)\text{LC}(B)(\alpha) = 0\ ] \le \frac{\deg(\text{LC}(A))}{p} + \frac{\deg(\text{LC}(B))}{p} \le \frac{\deg A + \deg B}{p}$. To determine $\text{Prob}[\alpha \text{ is unlucky} \mid \alpha \text{ is not bad}]$ we have α is unlucky

$$\begin{aligned} &\iff \gcd(\bar{A}(x, \alpha), \bar{B}(x, \alpha)) \neq 1 \text{ (by definition)} \\ &\iff \text{res}_x(\bar{A}(x, \alpha), \bar{B}(x, \alpha)) = 0 \text{ (by Lemma 1)} \\ &\iff R(\alpha) = 0 \text{ where } R = \text{res}_{x_0}(\bar{A}, \bar{B}) \text{ (by Lemma 1).} \end{aligned}$$

Hence $\text{Prob}[\alpha \text{ is unlucky} \mid \alpha \text{ is not bad}] \le \frac{\deg R}{p - \deg A - \deg B}$ (by Schwartz-Zippel). Now the $\text{Prob}[\alpha \text{ is bad or unlucky}] \le \text{Prob}[\alpha \text{ is bad}] + \text{Prob}[\alpha \text{ is unlucky} \mid \alpha \text{ is not bad}] \le \frac{\deg A + \deg B}{p} + \frac{\deg R}{p - \deg A - \deg B} \le \frac{\deg R + \deg A + \deg B}{p - \deg A - \deg B}$ which by Lemma 1 is $\le \frac{\deg A \deg B + \deg A + \deg B}{p - \deg A - \deg B}$.

The following algorithm applies Lemma 3 to compute a lower bound d for $\deg_{x_i} G$. Note, later in the paper when we use Algorithm DegreeBound, if it happens that $d > \deg_{x_i} G$ (α is unlucky) then this won't affect the correctness of our algorithm, only the efficiency.

Algorithm DegreeBound(A,B,i)

Input: Non-zero $A, B \in \mathbb{Z}[x_0, x_1, \ldots, x_n]$ and an integer i satisfying $0 \le i \le n$.

Output: $d \ge \deg_{x_i}(G)$ where $G = \gcd(A, B)$.

1 Set $LA = \text{LC}(A, x_i)$ and $LB = \text{LC}(B, x_i)$. So $LA, LB \in \mathbb{Z}[x_0, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n]$.

2 Pick a prime $p \gg \deg A \deg B$ such that $LA \mod p \neq 0$ and $LB \mod p \neq 0$.

3 Pick $\alpha = (\alpha_0, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_n) \in \mathbb{Z}_p^n$ at random until $LA(\alpha)LB(\alpha) \neq 0$.

4 Compute $a = A(\alpha_0, \ldots, \alpha_{i-1}, x_i, \alpha_{i+1}, \ldots, \alpha_n)$ and $b = B(\alpha_0, \ldots, \alpha_{i-1}, x_i, \alpha_{i+1}, \ldots, \alpha_n)$.

5 Compute $g = \gcd(a, b)$ in $\mathbb{Z}_p[x_i]$ using the Euclidean algorithm and output $d = \deg_{x_i} g$.

1.4 Unlucky evaluations in Ben-Or/Tiwari

Consider again Example 2 where $G = (x_1 - 16)x_0 + 1$, $\bar{A} = x_0^2 + 1$ and $\bar{B} = x_0^2 + (x_1 - 1)(x_2 - 9)x_0 + 1$. For the Ben-Or/Tiwari points $\alpha_j = (2^j, 3^j)$ for $0 \le j < 2t$ observe that $\alpha_0 = (1, 1)$ and $\alpha_2 = (4, 9)$ are unlucky and $\alpha_4 = (16, 81)$ is bad. Since none of these points can be used to interpolate G we need to modify the Ben-Or/Tiwari point sequence. For the GCD problem, we want random evaluation points to avoid bad and unlucky points. The following fix works.

Pick the first $s > 0$ such that $2^s > p$ so that $(2^s, 3^s, \ldots, p_n^s) \mod p$ is not fixed and use $\alpha_j = (2^j, 3^j, \ldots, p_n^j)$ for $s \le j < s + 2t$. Steps 1,2 and 3 work as before. To solve the *shifted* transposed Vandermonde system $Wc = u$

$$\begin{bmatrix} m_1^s & m_2^s & \ldots & m_t^s \\ m_1^{s+1} & m_2^{s+1} & \ldots & m_t^{s+1} \\ \vdots & \vdots & \vdots & \vdots \\ m_1^{s+t-1} & m_2^{s+t-1} & \ldots & m_t^{s+t-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_t \end{bmatrix} = \begin{bmatrix} v_s \\ v_{s+1} \\ \vdots \\ v_{s+t-1} \end{bmatrix}$$

we first solve the transposed Vandermonde system $Vb = u$

$$\begin{bmatrix} 1 & 1 & \ldots & 1 \\ m_1 & m_2 & \ldots & m_t \\ \vdots & \vdots & \vdots & \vdots \\ m_1^{t-1} & m_2^{t-1} & \ldots & m_t^{t-1} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_t \end{bmatrix} = \begin{bmatrix} v_s \\ v_{s+1} \\ \vdots \\ v_{s+t-1} \end{bmatrix}$$

as before to obtain $b = V^{-1}u$. Observe that the matrix $W = VD$ where D is the t by t diagonal matrix with $D_{i,i} = m_i^s$. To solve $Wc = u$ we have

$$c = W^{-1}u = (VD)^{-1}u = D^{-1}(V^{-1}u) = D^{-1}b.$$

Thus $c_i = um_i^{-s}$ and we can solve $Wc = u$ in $O(t^2 + t \log s)$ multiplications.

Referring again to Example 2, if we use the discrete logarithm evaluation points $\alpha_j = (\omega_1^j, \omega_2^j)$ for $0 \le j < 2t$ then $\alpha_0 = (1, 1)$ is unlucky and also, since $\omega_1^{q_1} = 1$, all $\alpha_{q_1}, \alpha_{2q_1}, \alpha_{3q_1}, \ldots$ are unlucky. Shifting the sequence to start at $j = 1$ and picking $q_i > 2t$ is problematic because for the GCD problem, t may be larger than $\max \#a_i, \#b_i$, or smaller; there is no way to know in advance. This difficulty led us to consider using a Kronecker substitution.

2. KRONECKER SUBSTITUTIONS

We propose to use a Kronecker substitution to map a multivariate polynomial GCD problem in $\mathbb{Z}[x_0, x_1, \ldots, x_n]$ into a bivariate GCD problem in $\mathbb{Z}[x, y]$. After making the Kronecker substitution, we need to interpolate $H(x, y) = \Delta(x, y)G(x, y)$ where $\deg_y H(x, y)$ will be exponential in n. To make discrete logarithms in $\mathbb{Z}_p$ feasible, we follow Kaltofen [15] and pick $p = 2^k s + 1 > \deg_y H(x, y)$ with s small.

Definition 2. *Let D be an integral domain and let f be a polynomial in $D[x_0, x_1, \ldots, x_n]$. Let $r \in \mathbb{Z}^{n-1}$ with $r_i > 0$. Let $K_r : D[x_0, x_1, \ldots, x_n] \to D[x, y]$ be the Kronecker substitution $K_r(f) = f(x, y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{n-1}})$.*

Let $d_i = \deg_{x_i} f$ be the partial degrees of f for $1 \le i \le n$. Observe that K_r is invertible if $r_i > d_i$ for $1 \le i \le n - 1$. Not all such Kronecker substitutions can be used, however, for the GCD problem. We consider an example.

Example 3. *Consider the following GCD problem*

$$G = x + y + z, \ \bar{A} = x^3 - yz, \ \bar{B} = x^2 - y^2$$

in $\mathbb{Z}[x, y, z]$. Since $\deg_y G = 1$ the Kronecker substitution $K_r(G) = G(x, y, y^2)$ is invertible. But $\gcd(K_r(\bar{A}), K_r(\bar{B})) = \gcd(\bar{A}(x, y, y^2), \bar{B}(x, y, y^2)) = \gcd(x^3 - y^3, x^2 - y^2) = x - y$. If we proceed to interpolate the $\gcd(K_r(A), K_r(B))$ we will obtain $(x - y)K_r(G)$ in expanded form from which and we cannot recover G.

We call such a Kronecker substitution unlucky. Theorem 1 below tells us that the number of unlucky Kronecker substitutions is finite. To detect them we will also avoid bad Kronecker substitutions in an analogous way Brown did to detect unlucky evaluation points.

Definition 3. *Let K_r be a Kronecker substitution. We say K_r is* bad *if $\deg_x K_r(A) < \deg_{x_0} A$ or $\deg_x K_r(B) < \deg_{x_0} B$ and K_r is* unlucky *if $\deg_x \gcd(K_r(\bar{A}), K_r(\bar{B})) > 0$.*

Lemma 5. *Let $f \in \mathbb{Z}[x_1, \ldots, x_n]$ be non-zero and $d_i \ge 0$ for $1 \le i \le n$. Let X be the number of Kronecker substitutions from the sequence $r_k = [d_1 + k, d_2 + k, \ldots, d_{n-1} + k]$ for $k = 1, 2, 3, \ldots$ for which $K_r(f) = 0$. Then $X \le (n - 1)\sqrt{2 \deg f}$.*

Proof: $K_r(f) = 0 \iff f(y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{n-1}}) = 0$
$\iff f \bmod \langle x_1 - y, x_2 - y^{r_1}, \ldots, x_n - y^{r_1 r_2 \cdots r_{n-1}} \rangle = 0$
$\iff f \bmod \langle x_2 - x_1^{r_1}, x_3 - x_2^{r_2}, \ldots, x_n - x_{n-1}^{r_{n-1}} \rangle = 0.$

Thus X is the number ideals $I = \langle x_2 - x_1^{r_1}, \ldots, x_n - x_{n-1}^{r_{n-1}} \rangle$ for which $f \bmod I = 0$ with $r_i = d_i + 1, d_i + 2, \ldots$. We prove that $X \le (n - 1)\sqrt{2 \deg f}$ by induction on n.

If $n = 1$ then I is empty so $f \bmod I = f$ and hence $X = 0$ and the Lemma holds. For $n = 2$ we have $f(x_1, x_2) \bmod \langle x_2 - x_1^{r_1} \rangle = 0 \implies x_2 - x_1^{r_1} | f$. Now X is maximal when $d_1 = 0$ and $r_1 = 1, 2, 3, \ldots$. We have

$$\sum_{r_1=1}^{X} r_1 \le \deg f \Longrightarrow X(X+1)/2 \le \deg f \Longrightarrow X < \sqrt{2 \deg f}.$$

For $n > 2$ we proceed as follows. Either $x_n - x_{n-1}^{r_{n-1}} | f$ or it doesn't. If not then the polynomial $S = f(x_1, \ldots, x_{n-1}, x_{n-1}^{r_{n-1}})$ is non-zero. For the sub-case $x_n - x_{n-1}^{r_{n-1}} | f$ we obtain at most $\sqrt{2 \deg f}$ such factors of f using the previous argument. For the case $S \ne 0$ we have

$$S \bmod I = 0 \iff S \bmod \langle x_2 - x_1^{r_1}, \ldots, x_{n-2} - x_{n-1}^{r_{n-2}} \rangle = 0$$

Notice that $\deg_{x_i} S = \deg_{x_i} f$ for $1 \le i \le n - 2$. Hence, by induction on n, $X < (n - 2)\sqrt{2 \deg f}$ for this case. Adding the number of unlucky Kronecker substitutions for both cases yields $X \le (n - 1)\sqrt{2 \deg f}$. □

Theorem 1. *Let $A, B \in \mathbb{Z}[x_0, x_1, \ldots, x_n]$ be non-zero, $G = \gcd(A, B)$, $\bar{A} = A/G$ and $B = \bar{B}/G$. Let $d_i \ge \deg_{x_i} G$. Let X be the number of bad and unlucky Kronecker substitutions K_{r_k} from the sequence $r_k = [d_1 + k, d_2 + k, \ldots, d_{n-1} + k]$ for $k = 1, 2, 3, \ldots$. Then*

$$X \le \sqrt{2}(n - 1) \left[\sqrt{\deg A} + \sqrt{\deg B} + \sqrt{\deg A \deg B} \right].$$

Proof Let $LA = \mathrm{LC}(A)$ and $LB = \mathrm{LC}(B)$ be the leading coefficients of A and B in x_0. Then K_r is bad $\iff K_r(LA) = 0$ or $K_r(LB) = 0$. Applying Lemma 5, the number of bad Kronecker substitutions is at most

$$(n-1)(\sqrt{2 \deg LA} + \sqrt{2 \deg LB}) \le (n-1)(\sqrt{2 \deg A} + \sqrt{2 \deg B}).$$

Now let $R = \mathrm{res}_{x_0}(\bar{A}, \bar{B})$. We will assume K_r is not bad.

$$\begin{aligned} K_r \text{ is unlucky} &\iff \deg_x(\gcd(K_r(\bar{A}), K_r(\bar{B})) > 0 \\ &\iff \mathrm{res}_x(K_r(\bar{A}), K_r(\bar{B})) = 0 \\ &\iff K_r(\mathrm{res}_x(\bar{A}, \bar{B})) = 0 \\ &\iff K_r(R) = 0 \ \ (K_r \text{ is not bad}). \end{aligned}$$

By Lemma 5, the number of unlucky Kronecker substitutions $\le (n-1)\sqrt{2 \deg R} \le (n-1)\sqrt{2 \deg A \deg B}$ by Lemma 1. Adding the two contributions proves the theorem. □

In algorithm PGCD below, we identify an unlucky substitution as follows. After computing the first two monic images $g_1(x)$ and $g_2(x)$ in step 9 if both $\deg_x g_1 > d_0$ and $\deg_x g_2 > d_0$ then with high probability K_r is unlucky so we try the next Kronecker substitution $r = [r_1 + 1, r_2 + 1, \ldots, r_{n-1} + 1]$.

It is still not obvious that a Kronecker substitution that is not unlucky can be used because it can create a content in y of exponential degree. The following example shows how we recover $H = \Delta G$ when this happens.

Example 4. *Consider the following GCD problem*

$$G = wx^2 + zy, \ \bar{A} = ywx + z, \ \bar{B} = yzx + w$$

in $\mathbb{Z}[x, y, z, w]$. We have $\Gamma = wy$ and $\Delta = y$. For $K(f) = f(x, y, y^3, y^9)$ we have $\gcd(K(A), K(B)) = K(G) \gcd(y^{10}x + y^3, y^4 x + y^9) = (y^9 x^2 + y^4)y^3 = y^7(y^5 x^2 + 1)$.

One must not try to compute $\gcd(K(A),K(B))$ because the degree of the content of $\gcd(K(A),K(B))$ (y^7 in our example) can be exponential in n the number of variables and we cannot compute this efficiently using the Euclidean algorithm. The crucial observation is that if we compute **monic** images $g_j = \gcd(K(A)(x,\alpha^j),K(B)(x,\alpha^j))$ any content is divided out, and when we scale by $K(\Gamma)(\alpha^j)$ and interpolate y in $K(H)$ using sparse interpolation, we recover any content. We obtain $K(H) = K(\Delta)K(G) = y^{10}x^2 + y^5$, then invert K to obtain $H = (yw)x^2 + (y^2z)$.

2.1 Unlucky evaluation points

Even if the Kronecker substitution is not unlucky, after applying it to input polynomials A and B, because the degree in y may be very large, the number of bad and unlucky evaluation points may be very large.

EXAMPLE 5. *Consider the following GCD problem*
$G = x_0 + x_1^d + x_2^d + \cdots + x_n^d$,
$\bar{A} = x_0 + x_1 + \cdots + x_{n-1} + x_n$, *and*
$\bar{B} = x_0 + x_1 + \cdots + x_{n-1} + 1$.
Using $r = [d+1, d+1, \ldots, d+1]$ *we need* $p > (d+1)^n$. *But* $R = \mathrm{res}_{x_0}(\bar{A},\bar{B}) = 1-x_n$ *and* $K_r(R) = 1-y^{r_1r_2\cdots r_{n-1}} = 1-y^{(d+1)^{n-1}}$ *which means there could be as many as* $(d+1)^{n-1}$ *unlucky evaluation points, that is, one in* $d+1$.

To guarantee that we avoid unlucky evaluation points with high probability we would need to pick $p \gg \deg_y K_r(R)$ which could be much larger than what is needed to interpolate $K_r(H)$. But this upper bound based on the resultant is a worst case. This lead us to investigate what the expected number of unlucky evaluation points is. We ran an experiment. We computed all monic quadratic and cubic bivariate polynomials over small finite fields $\mathbb{F}_q$ of size $q = 2,3,4,5,7,8,11$ and counted the number of unlucky evaluation points to find the following result.

THEOREM 2. *Let* $\mathbb{F}_q$ *be a finite field with* q *elements and* $f = x^l + \sum_{i=0}^{l-1}(\sum_{j=0}^{d_i} a_{ij}y^j)x^i$ *and* $g = x^m + \sum_{i=0}^{m-1}(\sum_{j=0}^{e_i} b_{ij}y^j)x^i$ *with* $l \geq 1$, $m \geq 1$, *and* $a_{ij}, b_{ij} \in \mathbb{F}_q$. *Let* $X = |\{\alpha \in \mathbb{F}_q : \gcd(f(x,\alpha),g(x,\alpha)) \neq 1\}|$ *be a random variable over all choices* $a_{ij}, b_{ij} \in \mathbb{F}_q$. *So* $0 \leq X \leq q$ *and for* f *and* g *not coprime in* $\mathbb{F}_q[x,y]$ *we have* $X = q$. *If* $d_i \geq 0$ *and* $e_i \geq 0$ *then* $\mathrm{E}[X] = 1$.

Proof: Let $C(y) = \sum_{i=0}^d c_iy^i$ with $d \geq 0$ and $c_i \in \mathbb{F}_q$ and fix $\beta \in \mathbb{F}_q$. Consider the evaluation map $C_\beta : \mathbb{F}_q^{d+1} \to \mathbb{F}_q$ given by $C_\beta(c_0,\ldots,c_d) = \sum_{i=0}^d c_i\beta^i$. We claim that C is balanced, that is, C maps q^d inputs to each element of $\mathbb{F}_q$. It follows that $f(x,\beta)$ is also balanced, that is, over all choices for $a_{i,j}$ each monic polynomial in $\mathbb{F}_q[x]$ of degree n is obtained equally often. Similarly for $g(x,\beta)$.

Recall that two univariate polynomials a, b in $\mathbb{F}_q[x]$ with degree $\deg a > 0$ and $\deg b > 0$ are coprime with probability $1 - 1/q$ (see Ch 11 of Mullen and Panario [18]). This is also true under the restriction that they are monic. Therefore $f(x,\beta)$ and $g(x,\beta)$ are coprime with probability $1 - 1/q$. Since we have q choices for β we obtain

$$E[X] = \sum_{\beta\in\mathbb{F}_q} \mathrm{Prob}[\gcd(A(x,\beta),B(x,\beta)) \neq 1] = q(1-(1-\frac{1}{q})) = 1.$$

Proof of claim. Since $B = \{1, y-\beta, (y-\beta)^2, \ldots, (y-\beta)^d\}$ is a basis for polynomials of degree d we can write each $C(y) = \sum_{i=0}^d c_iy^i$ as $C(y) = u_0 + \sum_{i=1}^d u_i(y-\beta)^i$ for a unique choice of $u_0, u_1, \ldots, u_d \in \mathbb{F}_q$. Since $C(\beta) = u_0$ it follows that all q^d choices for $u_1, \ldots, u_d$ result in $C(\beta) = u_0$ hence C is balanced. □

That $E[X] = 1$ was a surprise to us. We thought $E[X]$ would have a logarithmic dependence on $\deg f$ and $\deg g$. In light of Theorem 2, when picking $p > \deg_y(K_r(H))$ we will ignore the unlucky evaluation points, and, should the algorithm encounter unlucky evaluations, restart the algorithm with a larger prime.

3. GCD ALGORITHM

Algorithm PGCD(A,B,Γ)

Input $A = a_lx_0^l + ... + a_0$, and $B = b_mx_0^m + ... + b_0$ with $a_i, b_i \in \mathbb{Z}[x_1,...x_n]$ and $\Gamma \in \mathbb{Z}[x_1,\ldots,x_n]$ satisfying $\gcd(a_i) = 1$ (A is primitive) and $\gcd(b_i) = 1$ (B is primitive) and $\Gamma = \gcd(a_l, b_m) = LC(G) \times \Delta$ where $G = \gcd(A,B)$.

Output A prime p and polynomial $H \in \mathbb{Z}_p[x_0,x_1,\ldots,x_n]$ satisfying $H = \Delta \times G \bmod p$ with probability at least $1 - \frac{\deg A \deg B + \min(\deg A, \deg B)}{p - \deg A - \deg B}$.

1 Compute $d_i = DegreeBound(A,B,i)$ for $0 \leq i \leq n-1$. Set $r_i = 1 + \min(\deg_{x_i} A, \deg_{x_i} B, d_i + \deg_{x_i}\Gamma)$ for $1 \leq i \leq n-1$.

Kronecker-substitution:

2 Let $Y = (y, y^{r_1}, y^{r_1r_2}, \ldots, y^{r_1r_2\cdots r_{n-1}})$ be the Kronecker substitution. Set $K(A) = A(x,Y)$, $K(B) = B(x,Y)$ and $K(\Gamma) = \Gamma(Y)$.

3 If $\deg_x K(A) < \deg_{x_0} A$ or $\deg_x K(B) < \deg_{x_0} B$ then this Kronecker substitution is bad. Set $r_i = r_i + 1$ for $1 \leq i \leq n-1$ and **goto** Kronecker-substitution.

Pick-a-Prime:

4 Pick a new prime $p > 2\Pi_{i=1}^n r_i$ of the form $p = 2^kq + 1$ with q small.

5 If $\deg_x(K(A) \bmod p) < \deg_x A$ or $\deg_x(K(B) \bmod p) < \deg_x B$ then the prime is bad so **goto** Pick-a-Prime.

6 Set shift $s = 1$ and $j = 0$ and compute a random generator α for $\mathbb{Z}_p^*$.

Next-image:

7 Set $j = j + 1$. If $j = p - 1$ then we've run out of evaluation points. This could happen if one of the coefficients h_i of H is dense. **Goto** Pick-a-Prime and increase the length of p by 10 bits.

8 Compute $a_i = K(A)(x,\alpha^j)$ and $b_i = K(B)(x,\alpha^j)$. If $\deg_x a_i < \deg_{x_0} A$ or $\deg_x b_i < \deg_{x_0} B$ then α_j is bad so set $s = j$ and **goto** Next-image.

9 Compute $g_j = \gcd(a_i, b_i)$ in $\mathbb{Z}_p[x]$ using the Euclidean algorithm.

10 **Case** $\deg g_j < d_0$: (the degree bound d_0 is wrong) Set $d_0 = \deg g_j$, $s = j$ and **goto** Next-image.

11 **Case** $\deg g_j > d_0$: (α^j is unlucky)

11a If this happens for $j = s$ and $j = s + 1$ then the Kronecker substitution is very probably unlucky so set $r_i = r_i + 1$ for $1 \leq i \leq n - 1$ and **goto** Kronecker-substitution.

11b If this is the 2nd unlucky evaluation then **goto** Pick-a-Prime and double the length of p. Otherwise set $s = j$ and **goto** Next-image.

12 **Case** $\deg g_j = d_0$: (we have a new image)

12a Scale the image: Set $g_j = K(\Gamma)(\alpha^j)g_j$. If $s - j$ is even then **goto** compute-next-image – we need at least two new images for the next step.

12b Run the Berlekamp-Massey algorithm on the coefficients of the images $g_s, g_{s+1}, \ldots, g_{s+j}$ to obtain $\lambda_i(z)$ for $0 \leq i \leq d_0$. If any $\lambda_i(z)$ changed from the previous step **goto** Next-image.

12c Compute the roots of each $\lambda_i(z)$. If any $\lambda_i(z)$ has fewer than $\deg \lambda_i(z)$ distinct roots **goto** Next-image.

12d Complete the sparse interpolation to obtain polynomials $h_i(y) \in \mathbb{Z}_p[y]$. Note, s is the shift used for the shifted transposed Vandermonde systems. Set $H(x, y) := \sum_{i=0}^{d_0} h_i(y)x^i$ which we hope is equal to $\Delta(Y)G(x, Y)$.

12e Invert the Kronecker substitution to obtain H. If $\deg_{x_i} H > \min(\deg_{x_i} A, \deg_{x_i} B, d_i + \deg_{x_i} \Gamma)$ for any $1 \leq i \leq n$ then $H \neq \Delta G$ so **goto** Next-image.

12f **Probabilistic check:** Pick $\beta \in \mathbb{Z}_p^n$ at random until $\deg A(x_0, \beta) = \deg_{x_0} A$ and $\deg B(x_0, \beta) = \deg_{x_0} B$. Compute $g_\beta = \gcd(A(x_0, \beta), B(x_0, \beta))$. If $H(x_0, \beta) = \Gamma(\beta)g_\beta$ then **output** (p, H). Otherwise either t_i is wrong for some i or $d_0 > \deg_{x_0} G$ or β is unlucky. In all cases continue **goto** Next-image.

To prove the claim on the output (p, H) let $H = \sum_{i=0}^{d_0} h_i x_0^i$ and let $G = \sum_{i=0}^{dG} c_i x_0^i$. We will bound the probability that algorithm PGCD outputs $H \neq \Delta G \mod p$. Notice that if PGCD outputs H it must be that $\deg_{x_0} H = d_0 = \deg_{x_0} g_\beta$. Now either $d_0 > dG$ or $d_0 = dG$. If $d_0 > dG$ then H is wrong. Now $d_0 > dG \Rightarrow \beta$ is unlucky thus $\text{Prob}[d_0 > dG] \leq \text{Prob}[\beta \text{ is unlucky}]$ which is at most $\frac{\deg A \deg B}{p - \deg A - \deg B}$. If $d_0 = dG$ then H is output iff $h_i(\beta) = \Delta(\beta)c_i(\beta) \mod p$ for $0 \leq i \leq d_0$. Let $f_i = h_i - \Delta c_i \mod p$. $H \neq \Delta G$ implies $f_i \neq 0$ for at least one i, say $i = j$. The Schwartz-Zippel lemma implies $\text{Prob}[f_j(\beta) = 0] \leq \frac{\deg f_i}{p - \deg A - \deg B}$. Now the degree condition on $\deg_{x_i} H$ means the total degree $\deg f_i \leq \min(\deg A, \deg B)$ thus $\text{Prob}[f_j(\beta) = 0] \leq \frac{\min(\deg A, \deg B)}{p - \deg A - \deg B}$. Adding both probabilities $\text{Prob}[H \neq \Delta G \mod p] \leq \frac{\min(\deg A, \deg B)}{p - \deg A - \deg B} + \frac{\deg A \deg B}{p - \deg A - \deg B}$ and the result follows.

We are not able to say what the expected running time of the algorithm is. If we were to choose $p > At^B$ for suitably chosen constants A and B, then such an analysis should be possible. But since we do not have a bound for t other than $t < (d + 1)^n$, this would lead to a significantly larger prime.

3.1 Determining t

Algorithm PGCD assumes in step 12b that if none of the $\lambda_i(z)$ changed then $(j - s + 1)/2 = t$ but it could be that $(j - s + 1)/2 < t$. Let $V_r = (v_0, v_1, \ldots, v_{2r-1})$ be a sequence where $r \geq 1$. The Berlekamp-Massey algorithm (BMA) with input V_r computes a feedback polynomial $c(z)$ which is the reciprocal of $\lambda(z)$ if $r = t$. In PGCD, we determine the t by computing $c(z)$s on the input sequence V_r for $r = 1, 2, 3, \ldots$. If a $c(z)$ remains unchanged from the input V_k to the input V_{k+1}, then we conclude that this $c(z)$ is *stable* which implies that the last two consecutive discrepancies are both zero, see [16, 14] for a definition of the discrepancy. However, it is possible that the degree of $c(z)$ on the input V_{k+2} might increase again. In [14], Kaltofen, Lee and Lobo proved (Theorem 3) that the BMA encounters the first zero discrepancy after $2t$ points with probability at least

$$1 - \frac{t(t + 1)(2t + 1)\deg(C)}{6|S|}$$

where S is the set of all possible evaluation points. Here is an example where we encounter a zero discrepancy before $2t$ points. Consider

$$f(y) = y^7 + 60y^6 + 40y^5 + 48y^4 + 23y^3 + 45y^2 + 75y + 55$$

over $\mathbb{Z}_{101}$ with generator $\alpha = 93$. Since f has 8 terms, 16 points are required to determine the correct $\lambda(z)$ and two more for confirmation. We compute $f(\alpha^j)$ for $0 \leq j \leq 17$ and obtain $V_9 = (44, 95, 5, 51, 2, 72, 47, 44, 21, 59, 53, 29, 71, 39, 2, 27, 100, 20)$. We run the BMA on input V_r for $1 \leq r \leq 9$ and obtain feedback polynomials in the following table.

r	Output $c(z)$
1	$69z + 1$
2	$24z^2 + 59z + 1$
3	$24z^2 + 59z + 1$
4	$24z^2 + 59z + 1$
5	$70z^7 + 42z^6 + 6z^3 + 64z^2 + 34z + 1$
6	$70z^7 + 42z^6 + 25z^5 + 87z^4 + 16z^3 + 20z^2 + 34z + 1$
7	$z^7 + 67z^6 + 95z^5 + 2z^4 + 16z^3 + 20z^2 + 34z + 1$
8	$31z^8 + 61z^7 + 91z^6 + 84z^5 + 15z^4 + 7z^3 + 35z^2 + 79z + 1$
9	$31z^8 + 61z^7 + 91z^6 + 84z^5 + 15z^4 + 7z^3 + 35z^2 + 79z + 1$

The ninth call of the BMA confirms that the feedback polynomial returned by the eighth call is the desired one. But, by our design, the algorithm terminates at the third call because the feedback polynomial remains unchanged from the second call. It also remains unchanged for V_4. In this case, $\lambda(z) = z^2c(1/z) = z^2 + 59z + 24$ has roots 56 and 87 which correspond to monomials y^4 and y^{20} since $\alpha^4 = 56$ and $\alpha^{20} = 87$.

The example shows that we may encounter a stable feedback polynomial too early. Furthermore, the recovered monomials may have degree higher than the degree of the input polynomial $f(y)$. Algorithm PGCD must check H for monomials of too high degree in step 12e for the degree argument in the proof of the claim to be valid.

3.2 Evaluation

Let $A, B \in \mathbb{Z}_p[x_0, x_1, \ldots, x_n]$, $s = \#A + \#B$, and $d = \max_{i=1}^n d_i$ where $d_i = \max(\deg_{x_i} A, \deg_{x_i} B)$. If we use a Kronecker substitution $K(A) = A(x, y, y^{r_1}, \ldots, y^{r_1 r_2 \cdots r_{n-1}})$ with $r_i = d_i + 1$, then $\deg_y K(A) < (d + 1)^n$. Thus we can evaluate the s monomials in $K(A)(x, y)$ and $K(B)(x, y)$ at $y = \alpha^k$ in $O(sn \log d)$ multiplications. Instead we first compute $\beta_1 = \alpha^k$ and $\beta_{i+1} = \beta_i^{r_i}$ for $i = 1, 3, \ldots, n - 2$ then precompute n tables of powers $1, \beta_i, \beta_i^2, \ldots, \beta_i^{d_i}$ for $1 \leq i \leq n$ using at most nd multiplications. Now, for each term in

A and B of the form $cx_0^{e_0}x_1^{e_1}\dots x_n^{e_n}$ we compute $c \times \beta_1^{e_1} \times \dots \times \beta_n^{e_n}$ using the tables in n multiplications. Hence we can evaluate $K(A)(x, \alpha^k)$ and $K(B)(x, \alpha^k)$ in at most $nd + ns$ multiplications. Thus for T evaluation points $\alpha, \alpha^2, \dots, \alpha^T$, the evaluation cost is $O(ndT + nsT)$ multiplications.

When we first implemented algorithm PGCD we noticed that often well over 95% of the time was spent evaluating the input polynomials A and B at the points α^k. This happens when $\#G \ll \#A + \#B$. The following method uses the fact that for a monomial $M_i(x_1, x_2, \dots, x_n)$

$$M_i(\beta_1^k, \beta_2^k, \dots, \beta_n^k) \;=\; M_i(\beta_1, \beta_2, \dots, \beta_n)^k$$

to reduce the total evaluation cost from $O(ndT + nsT)$ multiplications to $O(nd + ns + sT)$. Note, no sorting on x_0 is needed in step 4b if the monomials in the input A are are sorted on x_0.

Algorithm Evaluate.

Input $A = \sum_{i=1}^{m} c_i x_0^{e_i} M_i(x_1, \dots, x_n) \in \mathbb{Z}_p[x_0, \dots, x_n]$, $T > 0$, $\beta_1, \beta_2, \dots, \beta_n \in \mathbb{Z}_p$, and integers $d_1, d_2, \dots, d_n$ with $d_i \geq \deg_{x_i} A$.

Output $A(x_0, \beta_1^k, \dots, \beta_n^k)$ for $1 \leq k \leq T$.

1 Create the vector $C = [c_1, c_2, \dots, c_m] \in \mathbb{Z}_p^m$.
2 Compute $[\beta_i^j : j = 0, 1, \dots, d_i]$ for $1 \leq i \leq n$.
3 Compute $\Gamma = [M_i(\beta_1, \beta_2, \dots, \beta_n) : 1 \leq i \leq m]$.
4 For $k = 1, 2, \dots, T$ do
  4a Compute the vector $C := [C_i \times \Gamma_i$ for $1 \leq i \leq m]$.
  4b Assemble $\sum_{i=1}^{m} C_i x_0^{e_i} = A(x_0, \beta_1^k, \dots, \beta_n^k)$.

Even with this improvement evaluation still takes most of the time so we must parallelize it. Each evaluation of A could be parallelized in blocks of size m/N for N cores. In Cilk C, this is only effective, however, if the blocks are large enough (at least 50,000) so that the time for each block is much larger than the time it takes Cilk to create a task. For this reason, it is necessary to also parallelize on k. To parallelize on k for N cores, we multiply the previous N values of C in parallel by the vector

$$\Gamma_N = [M_i(\beta_1, \beta_2, \dots, \beta_n)^N : 1 \leq i \leq m]$$

Because most of the time is still in evaluation, we are presently implementing the asymptotically fast method of van der Hoven and Lecerf [10] and attempting to parallelize it. For our evaluation problem it has complexity $O(nd + ns + s \log^2 T)$ which is better than our $O(nd + ns + sT)$ method for large T.

4. BENCHMARKS

We have implemented algorithm PGCD for 31, 63 and 127 bit primes in Cilk C. For 127 bit primes we use the 128 bit signed integer type `__int128_t` supported by the gcc compiler. We parallelized evaluation (see Section 3.2) and we interpolate the coefficients $h_i(y)$ in parallel in step 12e. To assess how good it is, we have compared it with the serial implementations of Zippel's algorithm in Maple 2015 and Magma 2.21. For Maple we were able to determine the time spent computing G modulo the first prime only in Zippel's algorithm. It is over 90% of the total GCD time. For Magma we could not do this so the Magma timings are for the entire GCD computation over $\mathbb{Z}$.

All timings were made on the gaby server in the CECM at Simon Fraser University. This machine has two Intel Xeon E-2660 8 core CPUs running at 3.0 GHz on one core and 2.2 GHz on 8 cores. Thus the maximum parallel speedup is a factor of $16 \times 2.2/3.0 = 11.7$.

For our first benchmark (see Table 2) we created polynomials $G, \bar{A}$ and $\bar{B}$ in 6 variables ($n = 5$) and 9 variables ($n = 8$) of degree at most d in each variable. We generated $100d$ random terms for G and 100 random terms for $\bar{A}$ and $\bar{B}$. The integer coefficients of $G, \bar{A}, \bar{B}$ were generated at random from $[0, 2^{31} - 1]$. The monomials in $G, \bar{A}$ and $\bar{B}$ were generated using random exponents from $[0, d-1]$ for each variable. For G we included monomials $1, x_0^d, x_1^d, \dots, x_6^d$ so that G is monic in all variables and $\Gamma = 1$. Our GCD code used the 62 bit prime $p = 29 \times 2^{57} + 1$. Maple used the 32 bit prime $2^{32} - 5$ for the first image in Zippel's algorithm.

			New GCD algorithm		Zippel's algorithm	
n	d	t	1 core (eval)	16 cores	Maple	Magma
5	10	114	0.62s (68%)	0.091s (6.8x)	48.04s	6.97s
5	20	122	1.32s (69%)	0.155s (8.5x)	185.70s	318.22s
5	50	121	3.48s (69%)	0.326s (10.7x)	1525.80s	$> 10^4 s$
5	100	102	7.08s (69%)	0.657s (10.8x)	6018.23s	$> 10^4 s$
5	200	125	14.64s (71%)	1.287s (11.4x)	NA	NA
5	500	135	38.79s (71%)	3.397s (11.4x)	NA	NA
8	5	89	0.27s (61%)	0.065s (4.2x)	30.87s	2.39s
8	10	110	0.63s (65%)	0.098s (6.4x)	138.41s	6.15s
8	20	114	1.35s (66%)	0.163s (8.3x)	664.33s	63.49s
8	50	113	3.52s (66%)	0.336s (10.5x)	6390.22s	1226.77s
8	100	121	7.43s (68%)	0.645s (11.5x)	NA	NA

Table 2: Timings (seconds) for GCD problems.

In Table 2 column d is the maximum degree of the terms of $G, \bar{A}, \bar{B}$ in each variable, column t is the maximum number of terms of the coefficients of G and column eval is the %age of the time spent evaluating the inputs, that is computing $K(A)(x_0, \alpha^j)$ and $K(B)(x_0, \alpha^j)$ for $j = 1, 2, \dots, T$. The parallel speedup on 16 cores is shown in parens.

Our second benchmark (see Table 3) is for 9 variables where the degree of $G, \bar{A}, \bar{B}$ is at most 20 in each variable. The terms are generated at random as before but restricted to have total degree at most 60. The middle row is our benchmark problem from Section 1.

		New GCD algorithm		Zippel's algorithm	
$\#G$	$\#A$	1 core (eval)	16 cores	Maple	Magma
10^3	10^5	0.66s (68%)	0.100s (6.6x)	341.9s	63.55s
10^3	10^6	5.66s (90%)	0.717s (9.4x)	5553.5s	FAIL
10^4	10^6	48.44s (87%)	4.474s (10.2x)	62520.1s	FAIL
10^3	10^7	52.102 (92%)	4.591s (11.3x)	NA	NA
10^4	10^7	428.96s (98%)	37.43s (11.5x)	NA	NA

Table 3: Timings (seconds) for 9 variable GCDs

Tables 2 and 3 show that most of the time is in evaluation. They show a parallel speedup approaching the maximum of 11.7 on this machine. There was a parallel bottleneck in how we computed the $\lambda_i(z)$ polynomials that limited parallel speedup to 10 on these benchmarks. For N cores, after generating a new batch of N images we used the Euclidean algorithm for Step 12b which is quadratic in the number of images j computed so far. To address this we now use an incremental version of the Berlekamp-Massey algorithm which is $O(Nj)$.

In comparing the new algorithm with Maple's implementation of Zippel's algorithm, for $n = 8, d = 50$ in Table 2 we achieve a speedup of a factor of $1815 = 6390.22/3.52$ on 1 core. Since Zippel's algorithm uses $O(dt)$ points and our Ben-Or/Tiwari algorithm uses $2t + O(1)$ points, we get a factor of $O(d)$ speedup because of this.

Our improved evaluation gives us a another factor of n speedup over Maple's implementation of Zippel's algorithm. Another factor is the cost of multiplication in $\mathbb{Z}_p$. The reader should realize that the running time of algorithm PGCD is proportional to the cost of multiplication in $\mathbb{Z}_p$. Maple is using `% p` to divide in C which generates a hardware division instruction which is much more expensive than a multiplication. We are using Roman Pearce's implementation of Möller and Granlund [17] which reduces division by p to two multiplications plus other cheap operations.

5. CONCLUSION AND FINAL REMARKS

We have shown that a Kronecker substitution can be used to reduce a multivariate GCD computation to bivariate by using a discrete logs Ben-Or/Tiwari point sequence. Our parallel implementation is fast and practical. Several questions remain. The Ben-Or/Tiwari method requires $2t+O(1)$ points. Can we use fewer points? Can we do anything when $\#\Delta > 1$ which increases t? For polynomials in more variables or higher degree algorithm PGCD may need a prime p larger than 127 bits. Can we do anything to reduce the size of the prime needed?

Algorithm PGCD interpolates H from univariate images in $\mathbb{Z}_p[x_0]$. If instead we interpolate H from bivariate images in $\mathbb{Z}_p[x_0, x_1]$, this will likely reduce both t and $\#\Delta$. For our benchmark problem this reduces t by a factor of 9 and the cost of the bivariate GCD computations in $\mathbb{Z}_p[x_0, x_1]$, if computed with Brown's dense GCD algorithm [4], would remain negligible compared with the cost of evaluating A and B. Although we have not implemented this we estimate a speedup of a factor of 6 on 16 cores.

We cite the methods of Garg and Schost [6], Giesbrecht and Roche [9] and Arnold, Giesbrecht and Roche [1] which can use a smaller prime and would also use fewer than $2t + O(1)$ evaluations. These methods compute $a_i = K_r(A)(x, y)$, $b_i = K_r(B)(x, y)$ and $g_i = \gcd(a_i, b_i)$ all mod $\langle p, y^{q_i} - 1\rangle$ for several primes q_i and recover the exponents of y in $K_r(H)$ using Chinese remaindering. The algorithms differ in the size of q_i and how they avoid or recover from exponent collisions. It is not clear whether this approach can work for the GCD problem as these methods assume a division free evaluation but computing g_i requires division and $y = 1$ may be bad or unlucky. They also require $q_i \gg t$ which means computing g_i will be expensive for large t.

In contrast, the earlier method of Murao and Fujise in [19], which also uses Chinese remaindering on the exponents, should work. Another approach is to try to compress the Kronecker substitution. We are considering the idea suggested by van der Hoven in [11].

6. REFERENCES

[1] A. Arnold, M. Giesbrecht and D. Roche. Faster Sparse Multivariate Polynomial Interpolation of Straight-Line Programs. `arXiv:1412.4088`[cs.SC], December 2014.

[2] N. B. Atti, G. M. Diaz-Toca, and H. Lombardi. The Berlekamp-Massey algorithm revisited. AAECC **17** pp. 75–82, 2006.

[3] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proc. of STOC '20*, ACM Press, pp. 301–309, 1988.

[4] W. S. Brown. On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors. *J. ACM* **18**:478–504, 1971.

[5] D. Cox, J. Little, D. O'Shea. *Ideals, Varities and Algorithms.* Springer-Verlag, 1991.

[6] Sanchit Garg and Eric Schost. Interpolation of polynomials given by straight-line programs. *J. Theor. Comp. Sci.*, **410**:2659–2662, June 2009.

[7] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra.* Cambridge University Press, UK, 1999.

[8] K. O. Geddes, S. R. Czapor, and G. Labahn. *Algorithms for Computer Algebra.* Kluwer, 1992.

[9] Mark Giesbrecht and Daniel S. Roche. Diversification improves interpolation. In *Proc. ISSAC 2011*, ACM Press, pp. 123–130, 2011.

[10] Joris van der Hoven and Grégoire Lecerf. On the bit complexity of sparse polynomial multiplication. *J. Symb. Cmpt.* **50**:227–254, 2013.

[11] Joris van der Hoven and Grégoire Lecerf. Sparse polynomial interpolation in practice. *CCA* **48**:187–191, September 2015.

[12] Mahdi Javadi and Michael Monagan. Parallel Sparse Polynomial Interpolation over Finite Fields. In *Proc. of PASCO 2010*, ACM Press, pp. 160–168, 2010.

[13] E. Kaltofen, Y.N. Lakshman and J-M. Wiley. Modular Rational Sparse Multivariate Interpolation Algorithm. In *Proc. ISSAC 1990*, pp. 135-139, ACM Press, 1990.

[14] E. Kaltofen, W. Lee, and A. Lobo. Early Termination in Ben-Or/Tiwari Sparse Interpolation and a Hybrid of Zippel's algorithm. In *Proc. ISSAC 2000*, ACM Press, pp. 192−201, 2000.

[15] E. Kaltofen. Fifteen years after DSC and WLSS2. In *Proc. of PASCO 2010*, ACM Press, pp. 10–17, 2010.

[16] J. L. Massey. Shift-register synthesis and BCH decoding. *IEEE Trans. on Information Theory*, 15:122–127, 1969.

[17] Niels Möller and Torbjorn Granlund. Improved division by invariant integers. *IEEE Trans. on Computers*, **60**:165–175, 2011.

[18] Gary Mullen and Daniel Panario. *Handbook of Finite Fields.* CRC Press, 2013.

[19] Hirokazu Murao and Tetsuro Fujise. Modular Algorithm for Sparse Multivariate Polynomial Interpolation and its Parallel Implementation. *J. Symb. Cmpt.* **21**:377–396, 1996.

[20] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Trans. on Information Theory*, **24**:106–110, 1978.

[21] Michael Rabin. Probabilistic algorithms in finite fields. *SIAM J. Comput*, **9**:273–280, 1979.

[22] Jack Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, **27**:701–717, 1980.

[23] Douglas Stinson. *Cryptography, Theory and Practice*, Chapman and Hall, 2006.

[24] Y. Sugiyama, M. Kashara, S. Hirashawa and T. Namekawa. A Method for Solving Key Equation for Decoding Goppa Codes. *Information and Control* **27**:87–99, 1975.

[25] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Proc. of EUROSAM '79*, pp. 216–226. Springer-Verlag, 1979.

[26] Richard Zippel. Interpolating Polynomials from their Values. *J. Symb Cmpt.* **9**:375–403, 1990.