Parallel Sparse Polynomial Interpolation over Finite Fields *

Seyed Mohammad Mahdi Javadi
School of Computing Science
Simon Fraser University
Burnaby, B.C. Canada.
sjavadi@cecm.sfu.ca.

Michael Monagan
Department of Mathematics
Simon Fraser University
Burnaby, B.C. Canada.
mmonagan@cecm.sfu.ca.

ABSTRACT

We present a probabilistic algorithm to interpolate a sparse multivariate polynomial over a finite field, represented with a black box. Our algorithm modifies the algorithm of Ben-Or and Tiwari from 1988 for interpolating polynomials over rings with characteristic zero to characteristic p by doing additional probes.

To interpolate a polynomial in n variables with t non-zero terms, Zippel's (1990) algorithm interpolates one variable at a time using $O(ndt)$ probes to the black box where d bounds the degree of the polynomial. Our new algorithm does $O(nt)$ probes. It interpolates each variable independently using $O(t)$ probes which allows us to parallelize the main loop giving an advantage over Zippel's algorithm.

We have implemented both Zippel's algorithm and the new algorithm in C and we have done a parallel implementation of our algorithm using Cilk [2]. In the paper we provide benchmarks comparing the number of probes our algorithm does with both Zippel's algorithm and Kaltofen and Lee's hybrid of the Zippel and Ben-Or/Tiwari algorithms.

Categories and Subject Descriptors:
I.1.2 [Symbolic and Algebraic Manipulation]: Algorithms

General Terms: Algorithms, Theory.

Keywords: sparse polynomial interpolation, parallel interpolation algorithms, Ben-Or Tiwari.

1. INTRODUCTION

Let p be a prime and let $f \in \mathbb{Z}_p[x_1, \ldots, x_n]$ be a multivariate polynomial with $t > 0$ non-zero terms which is represented with a *black box* $\mathbb{Z}_p^n \to \mathbb{Z}_p$. On input $(\alpha_1, \ldots, \alpha_n) \in \mathbb{Z}_p^n$, the black box evaluates and outputs $f(x_1 = \alpha_1, \ldots, x_n = \alpha_n)$. Given also a degree bound d on the degree of f, our goal is to interpolate the polynomial f with minimum number of evaluations (probes to the black box).

*This work was supported by NSERC of Canada and the MITACS NCE of Canada

PASCO 2010, 21–23 July 2010, Grenoble, France.

Sparse interpolation is a key part of many algorithms in computer algebra such as polynomial GCD computation [17, 7, 13] over $\mathbb{Z}$. Here one computes the GCD modulo p where p is chosen to be a machine size prime. We are interested in algorithms whose computational complexity is polynomial in t, n, d and $\log p$. In 1979 Richard Zippel presented the first such algorithm. Zippel's algorithm is probabilistic. It relies heavily on the assumption that if a polynomial is zero at a *random* evaluation point, then it is the zero polynomial with high probability. Zippel's algorithm interpolates f one variable at a time, sequentially. It makes $O(ndt)$ probes to the black box. In 1990, Zippel in [18] improved his 1979 algorithm to use evaluation points of the form $(\alpha_1^i, \ldots, \alpha_k^i) \in \mathbb{Z}_p^k$ so that the linear systems to be solved become transposed Vandermonde systems which can be solved in $O(t^2)$ time instead of $O(t^3)$ – see [8].

In 1988, Ben-Or and Tiwari [1] presented a deterministic algorithm for interpolating a multivariate polynomial with integer, rational, real or complex coefficients. Given a bound T on the number of terms t of the polynomial f, the algorithm evaluates the black box at powers of the first n primes; it evaluates at the points $(2^i, 3^i, 5^i, \ldots, p_n^i)$ for $0 \le i < 2T$. If $M_j(x_1, \ldots, x_n)$ are the monomials of the t non-zero terms of f, it then uses Berlekamp/Massey algorithm [12] from coding theory to find the monomial evaluations $M_j(2, 3, 5, \ldots, p_n)$ for $1 \le j \le t$ and then determines the degree of each monomial M_j in x_k by trial division of $M_j(2, 3, 5, \ldots, p_n)$ by p_k. This algorithm is not variable by variable. Instead, it interpolates the polynomial f with $2T$ probes to the black box which can all be computed in parallel. The major disadvantage of the Ben-Or/Tiwari algorithm is that the evaluation points are large ($O(T \log n)$ bits long – see [1]) and computations over $\mathbb{Q}$ encounter an expression swell which makes the algorithm very slow. This problem was addressed by Kaltofen *et al.* in [9] by running the algorithm modulo a power of a prime of sufficiently large size; the modulus must be greater than $\max_j M_j(2, 3, 5, \ldots, p_n)$.

In [6], Huang and Rao describe how to make the Ben-Or/Tiwari approach work over finite fields GF(q) with at least $4t(t-2)d^2 + 1$ elements. Their idea is to replace the primes $2, 3, 5, \ldots, p_n$ in Ben-Or/Tiwari by linear (hence irreducible) polynomials in GF(q)[y]. Their algorithm is Las Vegas and does $O(dt^2)$ probes. Although the authors discuss how to parallelize the algorithm, the factor of t^2 may limit this approach.

In 2000, Kaltofen et al. in [10, 11] design a hybrid algorithm, a hybrid of the Zippel and Ben-Or Tiwari algorithms, which they call a "racing algorithm". To reduce the number

of probes when interpolating the next variable in Zippel's algorithm, their algorithm runs a Newton interpolation and a univariate Ben-Or/Tiwari simultaneously, stopping when the first succeeds. However, this further sequentializes the algorithm. In Section 5, we compare the number of probes made by this algorithm to our new algorithm.

In 2009, Giesbrecht, Labahn and Lee in [4] present two new algorithms for sparse interpolation for polynomials with floating point coefficients. The first is a modification of the Ben-Or/Tiwari algorithm that uses $O(t)$ probes. In principle, this algorithm can be made to work over finite fields $GF(q)$ for applications where one can choose q. One needs $q-1$ to have n distinct prime factors all $> d$. One would also need $q-1$ to have no large prime factors so that the discrete logarithms needed could be done efficiently using the Pohlig-Hellman algorithm [14]. We have not explored the feasibility of this approach.

Our approach for sparse interpolation over $\mathbb{Z}_p$ is to use evaluation points of the form $(\alpha_1^i, \ldots, \alpha_n^i) \in \mathbb{Z}_p^n$ and modify the Ben-Or/Tiwari algorithm to do extra probes to determine the degrees of the variables in each monomial in f. We do a factor of at most $2n$ more evaluations in order to recover the monomials from their images. A motivation for our new algorithm is to use the Ben-Or/Tiwari approach in modular algorithms (e.g. GCD computations in characteristic 0 – see [7]) where the prime p is chosen to be a machine prime so that arithmetic in $\mathbb{Z}_p$ is efficient.

Our paper is organized as follows. In Section 2 we present an example showing the main flow and the key features of our algorithm. We then identify possible problems that can occur and how the new algorithm deals with them in Section 3. In Section 4 we present our new algorithm and analyze its sequential time complexity. Finally, in Section 5 we compare the C implementations of our algorithm and Zippel's algorithm with the racing algorithm of Kaltofen and Lee [11] on various sets of polynomials.

2. THE IDEA AND AN EXAMPLE

Let $f = \sum_{i=1}^{t} a_i M_i \in \mathbb{Z}_p[x_1, \ldots, x_n]$ be the polynomial represented with the black box with $a_i \in \mathbb{Z}_p \backslash \{0\}$. Here t is the number of non-zero terms in f. $M_i = x_1^{e_{i1}} \times x_2^{e_{i2}} \times \cdots \times x_n^{e_{in}}$ is the i'th monomial in f where $M_i \neq M_j$ for $i \neq j$. Let $d \geq \deg f$ be a bound on the degree of f so that $e_{ij} \leq d$ for all $1 \leq i, j \leq n$.

We demonstrate our algorithm on the following example. Here we use x, y and z for variables instead of x_1, x_2 and x_3.

Example 1 *Let $f = 91yz^2 + 94x^2yz + 61x^2y^2z + 42z^5 + 1$ and $p = 101$. Given the number of terms $t = 5$, the number of variables $n = 3$, a degree bound $d = 5$ and the black box that computes f, we want to find f.*

The first step is to pick $n = 3$ generators $\alpha_1, \alpha_2, \alpha_3$ of $\mathbb{Z}_p^*$. We evaluate the black box at the points $\beta_0, \ldots, \beta_{2t-1}$ where $\beta_i = (\alpha_1^i, \alpha_2^i, \ldots, \alpha_n^i)$. Thus we make $2t$ probes to the black box. The reason to use generators instead of random values from $\mathbb{Z}_p$ is that it decreases the probability of two distinct monomials having the same evaluation. For our example, let the generators be $\alpha_1 = 66, \alpha_2 = 12$ and $\alpha_3 = 3$ and let v_i be the output of the black box on input β_i and let $V = (v_0, \ldots, v_{2t-1})$. In this example we obtain

$$V = (87, 78, 65, 41, 49, 38, 87, 29, 23, 86).$$

Now we use the Berlekamp/Massey algorithm [12] (See [10] for a more accessible reference). The input to this algorithm is a sequence of elements $b_0, b_1, \ldots, b_{2t-1}, \ldots$ where $b_i \in \mathbb{Z}_p$. The algorithm computes a linear generator *for the sequence, i.e. the univariate polynomial $\Lambda(z) = z^t - \lambda_{t-1}z^{t-1} - \cdots - \lambda_0$ such that*

$$b_{t+i} = \lambda_{t-1}b_{t+i-1} + \lambda_{t-2}b_{t+i-2} + \cdots + \lambda_0 b_i$$

for all $i \geq 0$. In our example the input is $V = (v_0, \ldots, v_{2t-1})$ and the output is

$$\Lambda_1(z) = z^5 + 28z^4 + 62z^3 + 54z^2 + 11z + 46.$$

The next step is to find the roots of $\Lambda_1(z)$. We know (see [1]) that this polynomial is the product of exactly $t = 5$ linear factors. The roots are $r_1 = 1, r_2 = 7, r_3 = 41, r_4 = 61$ and $r_5 = 64$. Ben-Or and Tiwari prove that for each $1 \leq i \leq t$, there exists $1 \leq j \leq t$ such that

$$m_i = M_i(\alpha_1, \ldots, \alpha_n) \equiv r_j \bmod p.$$

The main step now is to determine the degrees of each monomial in f in each variable. Consider the first variable x. Let α_{n+1} be a new random generator of $\mathbb{Z}_p^*$. In this example we choose $\alpha_4 = 34$. This time we choose the evaluation points $\beta'_0, \ldots, \beta'_{2t-1}$ where $\beta'_i = (\alpha_{n+1}^i, \alpha_2^i, \ldots, \alpha_n^i)$. Note that this time we are evaluating the first variable at powers of α_{n+1} instead of α_1. We evaluate the black box at these points and apply the Berlekamp/Massey algorithm on the sequence of the outputs to compute the linear generator for the new sequence

$$\Lambda_2 = z^5 + 45z^3 + 54z^2 + 60z + 42.$$

Let $\bar{r}_1, \ldots, \bar{r}_5$ be distinct roots of Λ_2.
We know that $M_i(\alpha_{n+1}, \alpha_2, \ldots, \alpha_n)$ is a root of Λ_2 for $1 \leq i \leq n$. On the other hand we have

$$\frac{M_i(\alpha_{n+1}, \alpha_2, \ldots, \alpha_n)}{M_i(\alpha_1, \alpha_2, \ldots, \alpha_n)} = \left(\frac{\alpha_{n+1}}{\alpha_1}\right)^{e_{i1}}. \quad (1)$$

Let $r_j = M_i(\alpha_1, \alpha_2, \ldots, \alpha_n)$ and $\bar{r}_k = M_i(\alpha_{n+1}, \alpha_2, \ldots, \alpha_n)$. From Equation 1 we have

$$\bar{r}_k = r_j \times \left(\frac{\alpha_{n+1}}{\alpha_1}\right)^{e_{i1}},$$

i.e. for every root r_j of Λ_1, $r_j \times (\frac{\alpha_{n+1}}{\alpha_1})^{e_{i1}}$ is a root of Λ_2 for some e_{i1} which is the degree of some monomial in f with respect to x. This gives us a way to compute the degree of each monomial M_i in the variable x. In this example we have $\frac{\alpha_{n+1}}{\alpha_1} = 25$. We start with the first root of Λ_1 and check if $r_1 \times (\frac{\alpha_{n+1}}{\alpha_1})^i$ is a root of Λ_2 for $0 \leq i \leq d$. For $r_1 = 1$ we have that $r_1 \times (\frac{\alpha_{n+1}}{\alpha_1})^0$ is a root of Λ_2 and for $0 < i \leq d$, $r_1 \times (\frac{\alpha_{n+1}}{\alpha_1})^i$ is not a root of Λ_2, hence we conclude that the degree of the first monomial of f in x is 0. We continue this to find the degrees of all the monomials in f in the variable x. We obtain

$$e_{11} = 0, e_{21} = 0, e_{31} = 0, e_{41} = 2, e_{51} = 2.$$

We proceed to the next variable y. This time we evaluate the black box at $\beta''_0, \ldots, \beta''_{2t-1}$ where $\beta''_i = (\alpha_1^i, \alpha_{n+1}^i, \alpha_3^i, \ldots, \alpha_n^i)$ and apply the Berlekamp/Massey algorithm on the sequence of the outputs to compute

$$\Lambda_3 = z^5 + 5z^4 + 27z^3 + 36z^2 + 93z + 40$$

the linear generator for the new sequence. Let $\tilde{r}_1, \ldots, \tilde{r}_5$ *be distinct roots of* Λ_3*. Again using the same approach as above, we find that the degrees of the monomials in the second variable* y *to be*

$$e_{12} = 0, e_{22} = 1, e_{32} = 0, e_{42} = 2, e_{52} = 1.$$

Finally we proceed to the last variable z*. This time we evaluate* z *at powers of* α_{n+1} *instead of* α_3 *and compute the following linear generator for the sequence of outputs obtained by probing the black box*

$$\Lambda_4 = z^5 + 27z^4 + 99z^3 + 18z^2 + 16z + 41.$$

We compute the degrees with the same technique and obtain

$$e_{13} = 0, e_{23} = 2, e_{33} = 5, e_{43} = 1, e_{53} = 1.$$

The reader may observe that determining the degrees of the monomials M_i *in each variable represent* n *independent tasks which can therefore be done in parallel. At this point we have computed all the monomials. Recall that* $M_i = x_1^{e_{i1}} \times x_2^{e_{i2}} \times \cdots \times x_n^{e_{in}}$ *hence we have*

$$M_1 = 1, M_2 = yz^2, M_3 = z^5, M_4 = x^2y^2z \text{ and } M_5 = x^2yz.$$

Now we need to compute the coefficients. We do this by solving one linear system of equations. We computed the roots of Λ_1 *and we have computed the monomials such that* $M_i(\alpha_1, \ldots, \alpha_n) = r_i$*. Recall that* v_i *is the output of the black box on the input* $\beta_i = (\alpha_1^i, \ldots, \alpha_n^i)$ *hence we have*

$$v_i = a_1 r_1^i + a_2 r_2^i + \cdots + a_t r_t^i$$

for $0 \leq i \leq 2t-1$*. Note that the system of equations obtained from the above set of equations is a Vandermonde system which can be solved in* $O(t^2)$ *time and* $O(t)$ *space (See [18]). After solving we obtain*

$$a_1 = 1, a_2 = 91, a_3 = 42, a_4 = 61 \text{ and } a_5 = 94$$

and hence $f = 1 + 91yz^2 + 42z^5 + 61x^2y^2z + 94x^2yz$ *is interpolated and we are done.*

3. PROBLEMS

The evaluation points $\alpha_1, \ldots, \alpha_n, \alpha_{n+1}$ must satisfy certain conditions for our new algorithm to work properly. Here we identify all problems.

3.1 Distinct Monomials

The first condition is that for $i \neq j$

$$M_i(\alpha_1, \ldots, \alpha_n) \neq M_j(\alpha_1, \ldots, \alpha_n) \text{ in } \mathbb{Z}_p$$

so that $\deg(\Lambda_1(z)) = t$. Also, at the k'th step of the algorithm, when computing the degrees of the monomials in x_k, we must have

$$\forall\ 1 \leq i \neq j \leq t, \quad m_{i,k} \neq m_{j,k} \text{ in } \mathbb{Z}_p$$

where $m_{i,k} = M_i(\alpha_1, \ldots, \alpha_{k-1}, \alpha_{n+1}, \alpha_{k+1}, \ldots, \alpha_n)$ so that $\deg(\Lambda_{k+1}(z)) = t$. To reduce the probability of monomial evaluations colliding, we pick α_i to have order $> d$. The easiest way to do this is to use generators of $\mathbb{Z}_p^*$. There are $\phi(p-1)$ generators where ϕ is Euler's totient function. We now give an upper bound on the probability that no monomial evaluations collide when we use generators for evaluations.

Theorem 1 *Let* $\alpha_1, ..., \alpha_n$ *be generators from* $\mathbb{Z}_p$ *chosen at random and let* $m_i = M_i(\alpha_1, \ldots, \alpha_n)$*. Then the probability that two or more monomials evaluate to the same value (we get a collision) is*

$$\leq \binom{t}{2} \frac{d}{\phi(p-1)} < \frac{dt^2}{2\phi(p-1)}.$$

PROOF. Consider the polynomial

$$A = \prod_{1 \leq i < j \leq t} \left(M_i(x_1, \ldots, x_n) - M_j(x_1, \ldots, x_n) \right).$$

Observe that $A(\alpha_1, \ldots, \alpha_n) = 0$ iff two monomial evaluations collide. Recall that the Schwartz-Zippel lemma ([16, 17]) says that if $r_1, \ldots, r_n$ are chosen at random from any subset S of a field K and $F \in K[x_1, \ldots, x_n]$ is non-zero then

$$\text{Prob}(F(r_1, \ldots, r_n) = 0) \leq \frac{\deg f}{|S|}.$$

Our result follows from noting that $d \geq \deg f$ and thus $\deg A \leq \binom{t}{2} d$ and $|S| = \phi(p-1)$, the number of primitive elements in $\mathbb{Z}_p$. □

3.2 Root Clashing

Let $r_1, \ldots, r_t$ be the roots of $\Lambda_1(z)$ which is the output of the Berlekamp/Massey algorithm on the sequence of the outputs from the black box on the first set of evaluation points $\alpha_1, \ldots, \alpha_n$. Suppose at the k'th step, we want to compute the degrees of all the monomials in the variable x_k. As mentioned in the Example 1, the first step is to compute Λ_{k+1}. Then if $\deg_{x_k}(M_i) = e_{ik}$ we have $\bar{r}_i = r_i \times (\frac{\alpha_{n+1}}{\alpha_k})^{e_{ik}}$ is a root of Λ_{k+1}. If $r_i \times (\frac{\alpha_{n+1}}{\alpha_k})^{e'}$, $0 \leq e' \neq e_{ik} \leq d$ is also a root of Λ_{k+1} then we may not be able to uniquely identify the correct degree of the i'th monomial in the k'th variable x_k. We will illustrate this with an example.

Example 2 *Consider the polynomial given in Example 1. Suppose instead of choosing* $\alpha_4 = 34$*, we choose* $\alpha_4 = 72$ *which is another generator of* $\mathbb{Z}_p^*$*. Since* α_1, α_2 *and* α_3 *are the same as before,* Λ_1 *does not change and hence the roots of* Λ_1 *are* $r_1 = 1, r_2 = 7, r_3 = 41, r_4 = 61$ *and* $r_5 = 64$*. In the next step we substitute* $\alpha_4 = 72$ *for* α_1 *and compute* $\Lambda_2 = z^5 + 61z^4 + 39z^3 + 67z^2 + 37z + 98$*. We proceed to compute the degrees of the monomials in* x *but we find that*

$$r_4 \times (\frac{\alpha_4}{\alpha_1})^2 = 15 \text{ and } r_4 \times (\frac{\alpha_4}{\alpha_1})^4 = 7$$

are both roots of Λ_2 *and hence we can not decide the correct degree of the last monomial in* x*.*

Theorem 2 *If* $\deg \Lambda_1(z) = \deg \Lambda_{k+1}(z) = t$ *then the probability that we cannot uniquely compute the degrees of all* $M_i(x_1, \ldots, x_n)$ *in* x_k *is at most* $\frac{d^2t^2}{4\phi(p-1)}$*.*

PROOF. Let $S_i = \{r_j \times (\frac{\alpha_{n+1}}{\alpha_k})^i \mid 1 \leq j \leq t\}$ for $0 \leq i \leq d$. We assume that $r_i \neq r_j$ for all $1 \leq i \neq j \leq t$. We will not be able to uniquely identify the degree of the j'th monomial in x_k if there exists $\bar{d}$ such that $r_j \times (\frac{\alpha_{n+1}}{\alpha_k})^{\bar{d}} = \bar{r}_i$ is a root of $\Lambda_{k+1}(z)$ and $0 \leq d \neq e_{jk} \leq d$ where e_{jk} is $\deg_{x_k}(M_j)$. But we have $\bar{r}_i = r_i \times (\frac{\alpha_{n+1}}{\alpha_k})^{e_{ik}}$ thus $r_j \times (\frac{\alpha_{n+1}}{\alpha_k})^{\bar{d}} = r_i \times (\frac{\alpha_{n+1}}{\alpha_k})^{e_{ik}}$. Without loss of generality, assume $\tilde{d} = \bar{d} - e_{ik} > 0$. We have $r_i = r_j \times (\frac{\alpha_{n+1}}{\alpha_k})^{\tilde{d}}$ and hence

$r_i \in S_{\tilde{d}} \Rightarrow S_0 \cap S_{\tilde{d}} \neq \emptyset$. Hence we will not be able to compute the degrees in x_k if $S_0 \cap S_i \neq \emptyset$ for some $1 \leq i \leq d$. Let

$$g(x) = \prod_{1 \leq l \neq j \leq t} (r_j x^i - r_l \alpha_k^i).$$

We have $r_l = r_j \times (\frac{\alpha_{n+1}}{\alpha_k})^i \in S_0 \cap S_i$ iff $g(\alpha_{n+1}) = 0$. Using the Schwartz-Zippel lemma, the probability that $g(\alpha_{n+1}) = 0$ is at most $\frac{\deg g}{\phi(p-1)} = \frac{\binom{t}{2} i}{\phi(p-1)} < \frac{it^2}{2\phi(p-1)}$. If we sum this quantity for all $1 \leq i \leq d$ we obtain that the overall probability is at most $\frac{d^2 t^2}{4\phi(p-1)}$. $\square$

Using Theorem 2, the probability that we will not be able to uniquely identify the degrees of the monomials in *all* the variables is at most $\frac{nd^2t^2}{4\phi(p-1)}$, i.e. for p, s.t. $\phi(p-1) > \frac{nd^2t^2}{2}$ with probability at least half, the algorithm succeeds without dealing with any problem. We will now discuss our solution to this problem. Note that we assume the images of the monomials are distinct, i.e. $\forall\ 1 \leq i \neq j \leq t,\ m_{i,k} \neq m_{j,k}$. Suppose we have computed Λ_{k+1} and we want to compute the degrees of the monomials in x_k and let $R_1 = \{r_1, \ldots, r_t\}$ be the set of all the roots of Λ_1 and $R_k = \{\bar{r}_1, \ldots, \bar{r}_t\}$ be the set of all the distinct roots of Λ_{k+1}. Let

$$D_j = \{(i, r) \mid 0 \leq i \leq d,\ r = r_j \times (\frac{\alpha_{n+1}}{\alpha_1})^i \in R_k\}.$$

D_j contains the set of all possible degrees of the j'th monomial M_j in the k'th variable x_k. We know that $(e_{jk}, \bar{r}_j) \in D_j$ and hence $|D_j| \geq 1$. If $|D_j| = 1$ for all $1 \leq j \leq t$, then the degrees are unique and this step of the algorithm is complete. Let G_k be a balanced bipartite graph defined as follows. G_k has two independent sets of nodes U and V each of size t. Nodes in U and V represent elements in R_1 and R_k respectively, i.e. $u_i \in U$ and $v_j \in V$ are labeled with r_i and $\bar{r}_j$. We connect $u_i \in U$ to $v_j \in V$ with an edge of weight (degree) d_{ij} if and only if $(d_{ij}, \bar{r}_j) \in D_i$.

Lemma 1 *We can uniquely identify the degrees of all the monomials in x_k, if and only if the bipartite graph G_k has a unique* perfect matching.

The proof of this lemma is immediate by looking at the structure of the graph G_k. We illustrate with an example.

Example 3 *Let f be the polynomial given in Example 1 and suppose for some evaluation points $\alpha_1, \ldots, \alpha_4$ we obtain the graph G_1 as shown in Figure 1. This graph has a perfect matching, i.e. the set of edges $\{(r_i, \bar{r}_i) \mid 1 \leq i \leq 5\}$. If there was an edge connecting r_1 to $\bar{r}_2$ then the new graph would no longer have a unique perfect matching and we would fail to uniquely compute the degrees of monomials in x.*

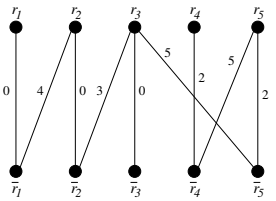

Figure 1: The bipartite graph G_1

We now give a solution for the case where G_k does not have a unique perfect matching for some $1 \leq k \leq n$. The solution involves $2t$ more probes to the black box. Suppose we choose a random element $\alpha_{n+2} \in \mathbb{Z}_p$ such that $\gamma = \frac{\alpha_{n+2}}{\alpha_{n+1}}$ is a generator of $\mathbb{Z}_p^*$ (or is of order greater than d). Let $\beta_i = (\alpha_1^i, \ldots, \alpha_{k-1}^i, \alpha_{n+2}^i, \alpha_{k+1}^i, \ldots, \alpha_n^i)$ and let v_i be the output of the black box on input β_i $(0 \leq i \leq 2t-1)$. On input $V = (\beta_0, \ldots, \beta_{2t-1})$, the Berlekamp/Massey algorithm computes a linear generator $\Lambda'_{k+1}(z)$ for V. Let $\{\tilde{r}_1, \ldots, \tilde{r}_t\}$ be the set of distinct roots of Λ'_{k+1}. Let G'_k be the balanced bipartite graph, obtained from Λ_1 and Λ'_{k+1}.

Definition 1. We define $\bar{G}_k$, the intersection of G'_k and G_k, as follows. $\bar{G}_k$ has the same nodes as G'_k and there is an edge between r_i and $\tilde{r}_j$ with weight (degree) d_{ij} if and only if r_i is connected to $\bar{r}_j$ in G_k and to $\tilde{r}_j$ in G'_k, both with the same degree d_{ij}.

Lemma 2 *Let $e_{ij} = \deg_{x_j}(M_i)$. The two nodes r_i and $\tilde{r}_i$ are connected in $\bar{G}_k$ with degree e_{ij}.*

We take advantage of the following theorem which implies we need at most one extra set of probes.

Theorem 3 *Let $\bar{G}_k = G_k \cap G'_k$. $\bar{G}_k$ has a* unique *perfect matching.*

Proof. Let U and V be the set of independent nodes in $\bar{G}_k$ such that $u_i \in U$ and $v_j \in V$ are labeled with r_i and $\tilde{r}_j$ respectively where $\tilde{r}_j$ is a root of Λ'_{k+1}. We will prove that each node in V has degree exactly 1 and hence there is a unique perfect matching. The proof is by contradiction. Suppose the degree of $v_j \in V$ is at least 2. With out loss of generality assume that r_1 and r_2 are both connected to $\tilde{r}_j$ with degrees d_{1j} and d_{2j} respectively (See Figure 2).

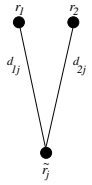

Figure 2: Node $\tilde{r}_j$ of graph $\bar{G}_k$

Using Definition 1 we have

$$\bar{r}_j = r_1 \times (\frac{\alpha_{n+1}}{\alpha_k})^{d_{1j}} = r_2 \times (\frac{\alpha_{n+1}}{\alpha_k})^{d_{2j}} \quad \text{and}$$

$$\tilde{r}_j = r_1 \times (\frac{\alpha_{n+2}}{\alpha_k})^{d_{1j}} = r_2 \times (\frac{\alpha_{n+2}}{\alpha_k})^{d_{2j}}.$$

Dividing the two sides of these equations results in

$$(\frac{\alpha_{n+2}}{\alpha_{n+1}})^{d_{1j}} = (\frac{\alpha_{n+2}}{\alpha_{n+1}})^{d_{2j}}.$$

Since we chose α_{n+2} such that $\frac{\alpha_{n+2}}{\alpha_{n+1}}$ has a sufficiently large order (greater than the degree bound d) we have $d_{1j} = d_{2j} \Rightarrow r_1 = r_2$. But this is a contradiction because both r_1 and r_2 are roots of Λ_1 which we assumed are distinct. $\square$

Lemma 2 and Theorem 3 prove that the intersection of G_k and G'_k will give us the correct degrees of all the monomials in the k'th variable x_k. We will illustrate with an example.

Example 4 *Let* $f = -10\,y^3 - 7\,x^2yz - 40\,yz^5 + 42\,y^3z^5 - 50\,x^7z^2 + 23\,x^5z^4 + 75\,x^7yz^2 - 92\,x^6y^3z + 6\,x^3y^5z^2 + 74\,xyz^8 + 4$ *and* $p = 101$. *We choose the first set of evaluation points to be* $\alpha_1 = 66, \alpha_2 = 11, \alpha_3 = 48$ *and* $\alpha_4 = 50$. *For the first variable* x *we will obtain the bipartite graph* G_1 *shown in Figure 3.*

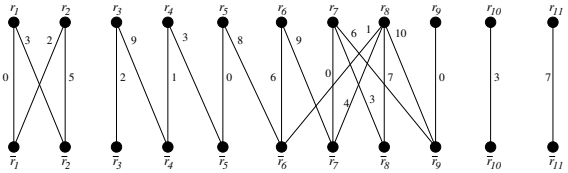

Figure 3: The bipartite graph G_1

This graph does not have a unique perfect matching, so we proceed to choose a new evaluation point $\alpha_5 = 89$. *This time we will get the bipartite graph* G_1' *shown in Figure 4.*

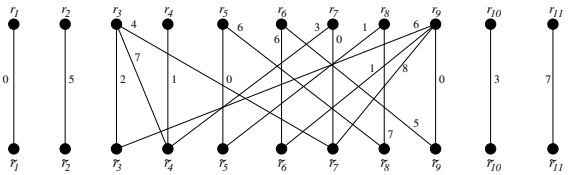

Figure 4: The bipartite graph G_1'

Again G_1' *does not have a unique perfect matching. We compute the intersection of* G_1 *and* G_1'*:* $\bar{G}_1 = G_1 \cap G_1'$. $\bar{G}_1$ *is shown in Figure 5.*

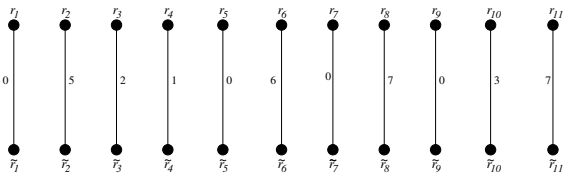

Figure 5: The bipartite graph $\bar{G}_1$

As stated by Theorem 3, $\bar{G}_1$ *has a unique perfect matching and the degree of every monomial in* x *is correctly computed.*

In this section we proved that if the prime p is sufficiently large ($\phi(p-1)$ must be approximately dt^2 for us to be able to get distinct images of monomials with reasonable probability), we will be able to compute the degrees of all the t monomials in each variable x_k using up to $4t$ evaluation points. If the graph G_k has a unique perfect matching, we will be able to compute the degrees in x_k with only $2t$ probes to the black box.

We conclude this section with the following lemma which we will later use in Section 4.

Lemma 3 *Let* G_k *be the bipartite graph for the k'th variable. Let* $u_{i_1} \to v_{j_1} \to u_{i_2} \to v_{j_2} \to \cdots \to v_{j_s} \to u_{i_1}$ *be a cycle in* G_k *where* $u_l \in U$ *is labeled with* r_l *(a root of* Λ_1*) and* $v_m \in V$ *is labeled with* $\bar{r}_m$ *(a root of* Λ_{k+1}*). Let* d_{lm} *be the weight (degree) of the edge between* u_l *and* v_m. *We have* $\sum_{m=1}^{s} d_{i_m j_m} - \sum_{m=1}^{s} d_{i_{m+1} j_m} = 0$.

Proof. It is easy to show that $r_{i_1} = (\frac{\alpha_{n+1}}{\alpha_k})^{\bar{d}} r_{i_s}$ where $\bar{d} = d_{i_1 j_1} - d_{i_2 j_1} + d_{i_2 j_2} - d_{i_3 j_2} + \cdots + d_{i_s j_{s-1}} - d_{i_s j_{s-1}}$. Also both u_{i_1} and u_{i_s} are connected to v_{j_s} in G_k hence we have $r_{i_1} = (\frac{\alpha_{n+1}}{\alpha_k})^{d_{i_1 j_s}} \bar{r}_{i_s}$ and $r_{i_s} = (\frac{\alpha_{n+1}}{\alpha_k})^{d_{i_s j_s}} \bar{r}_{i_s}$. These three equations yield to $r_{i_1} = (\frac{\alpha_{n+1}}{\alpha_k})^{\tilde{d}} r_{i_1}$ where $\tilde{d} = d_{i_1 j_1} - d_{i_2 j_1} + d_{i_2 j_2} - d_{i_3 j_2} + \cdots + d_{i_{s-1} j_{s-1}} - d_{i_s j_{s-1}} + d_{i_s j_s} - d_{i_1 j_s}$. But if $\frac{\alpha_{n+1}}{\alpha_k}$ is of sufficiently high order, $\tilde{d}$ must be zero thus $\sum_{m=1}^{s} d_{i_m j_m} - \sum_{m=1}^{s} d_{i_{m+1} j_m} = 0$. □

Example 5 *In* G_1' *shown in Figure 4, there is a cycle* $r_3 \to \tilde{r}_4 \to r_7 \to \tilde{r}_7 \to r_3$. *The weights (degrees) of the edges in this cycle are as* $7, 3, 0$ *and* 4. *We have* $7 - 3 + 0 - 4 = 0$.

4. THE ALGORITHM

Algorithm: Interpolation

Input: A *black box* $\mathbf{B} : \mathbb{Z}_p^n \to \mathbb{Z}_p$ that on input $\alpha_1, \ldots, \alpha_n \in \mathbb{Z}_p^n$ outputs $f(\alpha_1, \ldots, \alpha_n)$ where $f \in \mathbb{Z}_p[x_1, \ldots, x_n]$.
Input: A degree bound $d \geq \deg(f)$.
Input: A bound $T \geq t$ on the number of terms in f.
Output: The polynomial f or FAIL.

1: Choose $n+1$ generators $\alpha_1 \neq \ldots \neq \alpha_{n+1}$ of $\mathbb{Z}_p^*$ randomly.
2: **repeat** choose γ to be a random generator of $\mathbb{Z}_p^*$ and let $\alpha_{n+2} = \alpha_{n+1} \times \gamma$ **until** $\alpha_{n+2} \notin \{\alpha_1, \ldots, \alpha_{n+1}\}$.
3: Let $\beta_i = (\alpha_1^i, \ldots, \alpha_n^i)$ for $0 \leq i \leq 2T-1$.
4: **for** k from 1 to $n+1$ in **parallel do**
5: Compute $\Lambda_k(z)$:
6: Compute $v_i = \mathbf{B}(\beta_i)$ for $0 \leq i \leq 2t-1$ using α_{n+1}^i instead of α_{k-1}^i when $k > 1$.
7: Use the Berlekamp/Massey algorithm to compute a linear generator $\Lambda_k \in \mathbb{Z}_p[z]$ for the sequence $v_0, \ldots, v_{2t+1}$.
8: **end for**
9: Set $t = \max(\deg \Lambda_1(z), \ldots \Lambda_{n+1}(z))$. If the degree of the Λ's are not all equal to t then repeat steps 1 through step 8 once. If this does not yield equal degree Λ's then (p is likely too small so) **return** FAIL.
10: Compute $\{r_1, \ldots, r_t\}$ the set of distinct roots of $\Lambda_1(z)$.
11: **for** k from 1 to n in **parallel do**
12: Determine $\deg_{x_k}(M_i)$ for $1 \leq i \leq t$:
13: Construct the graph G_k as described in Section 3.
14: **if** G_k has a unique perfect matching **then**
15: Set $e_{ik} = d_{il}$ where d_{il} is the weight (degree) of the edge that matches the node r_i to $\bar{r}_l$ in the perfect matching.
16: **else**
17: Construct the graph G_k' as described in Section 3. Note, this requires $2t$ more probes to $\mathbf{B}$.
18: Find the intersection of G_k and G_k': $\bar{G}_k = G_k \cap G_k'$.
19: Set $e_{ik} = d_{il}$ where d_{il} is the weight (degree) of the edge that matches the node r_i to $\tilde{r}_l$ in the perfect matching of graph $\bar{G}_k$.
20: **end if**
21: **end for**
22: Let $S = \{a_1 r_1^i + a_2 r_2^i + \cdots + a_t r_t^i = v_i \mid 0 \leq i \leq 2t'-1\}$. Solve the linear system S for $(a_1, \ldots, a_t) \in \mathbb{Z}_p^t$.
23: Let $g = \sum_{i=1}^{t} a_i M_i$ where $M_i = \prod_{j=1}^{n} x_j^{e_{ij}}$.
24: Pick non-zero $a_1, \ldots, a_n$ from $\mathbb{Z}_p$ at random. If $\mathbf{B}(a_1, \ldots, a_n) \neq g(a_1, \ldots, a_n)$ then **return** FAIL.
25: **return** g.

Remark 1 The algorithm is probabilistic. If the degrees of the $\Lambda's$ are all equal to t then the algorithm will compute f with probability 1. If the degrees of the $\Lambda's$ are all equal but less than t then the algorithm cannot compute f; that is, $g \neq f$. The check in step 24 detects incorrect g with probability at least $1 - d/(p-1)$ (the Schwartz-Zippel lemma). Thus by doing one additional probe to the black box, we verify the output g with high probability. Kaltofen and Lee in [11] also use additional probes to verify the output this way.

Remark 2 For simplicity, our presentation of the algorithm assumes the term bound T is good. In applications where a good term bound is not available, one should first compute $\Lambda_1(z)$ using T, and then use $t = \deg \Lambda_1(z)$ when computing $\Lambda_2, \ldots, \Lambda_{n+1}$.

4.1 Complexity Analysis

We now discuss the sequential complexity of the algorithm assuming $t = T$. We need to consider the cost of probing the black box. Let $E(n, t, d)$ be the cost of one probe to the black box. If G_k has a unique perfect matching for $1 \leq k \leq n$ then we can correctly compute the degrees using only G_k. In this case the total number of probes is $2(n+1)t$ in the first loop. In the worst case where G_k does not have a unique perfect matching for all $1 \leq k \leq n$, we need to do additional $2nt$ probes to the black box in the second loop to construct all G'_k graphs. In this case the total number of probes to the black box is $2(n+1)t + 2nt = 2(2n+1)t$. Hence the total cost of probes to the black box is $O(ntE(n, t, d))$.
The $n+1$ calls to the Berlekamp/Massey algorithm in the first loop (as presented in [10]) cost $O(t^2)$ time each. The Vandermonde system of equations at Step 22 can be solved in $O(t^2)$ using the technique given in [18]. Note that as mentioned in [18], when inverting a $t \times t$ Vandemonde matrix defined by $k_1, \ldots, k_t$, one of the most expensive parts of this technique is to compute the master polynomial $M(z) = \prod_{i=1}^{t}(z - k_i)$. However, in our algorithm we can use the fact that $M(z) = \prod_{i=1}^{t}(z - r_i) = \Lambda_1(z)$.

To compute the roots of $\Lambda_1(z)$ at Step 10 of the algorithm, we use Rabin's Las Vegas algorithm [15]. If $f \in \mathbb{Z}_p[z]$ is a product of linear factors, Rabin's algorithm tries to split it into two factors of lower degree by computing the $\gcd((z-\beta)^{(p-1)/2} - 1, \Lambda_1(z))$ for randomly chosen $\beta \in \mathbb{Z}_p$. Since $\deg_z(\Lambda_1) = t$, the cost of finding the t roots of $\Lambda_1(z)$, assuming classical algorithms for polynomial arithmetic in $\mathbb{Z}_p[z]$ are used, is $O(t^2 \log p)$. See Algorithm 14.15 of [3].

We can compute the information needed to construct the bipartite graph G_k in $O(dt^2)$ time. This involves evaluating $\Lambda_{k+1}(z)$ at d points for each monomial and testing if it is zero or not. Also computing the intersection of G_k and G'_k can be done in $O(td \log d)$ time. This is because we know that each node in the intersection is of degree one (See proof of Theorem 3). Thus the overall time complexity is

$$O(t^2(\log(p) + nd) + ntE(n, t, d)).$$

Remark 3 The algorithm, as presented, corresponds to our parallel implementation in Cilk. Further parallelism in the algorithm could be exploited. For example, one could compute all probes to the black box **B** in step 6 and step 17 in parallel. When determining the degree of the monomials in step 13 and 17, one can parallelize the evaluations of $\Lambda_{k+1}(z)$. The most expensive sequential component is the computation of the roots of $\Lambda_1(z)$ in step 10 which has complexity $O(t^2 \log p)$. With asymptotically fast arithmetic this is $\tilde{O}(t \log p)$.

4.2 Optimizations

Let $D = \deg(f)$. If the prime p is large enough, i.e. $p > \frac{nD^2t^2}{4\epsilon}$ then with probability $1 - \epsilon$ the degree of every monomial in x_k can correctly be computed using only G_k and without needing any extra probes to the black box. In fact in this case, with high probability, every r_i will be matched with exactly only one $\bar{r}_j$ and hence every node in G_k would have degree one (e.g. see Figure 5). But if $d \gg D$, i.e. the degree bound d is not tight, the probability that we could identify the degrees uniquely drops significantly even though p is large enough. This is because the probability that *root clashing* (see Section 3) happens, linearly depends on d. In this case, with probability $1 - \epsilon$, the degree of M_i in x_k would be $\min\{d_{ij} \mid (d_{ij}, r_i) \in G_k\}$, i.e. the edge connected to r_i in G_k with minimum weight (degree) is our desired edge in the graph which will show up in the perfect matching. We apply the following theorem.

Theorem 4 *Let H_k be a graph obtained by eliminating all edges connected to r_i in G_k except the one with minimum weight (degree) for all $1 \leq i \leq t$. If the degree of every node in H_k is exactly one, then e_{ik} is equal to the weight of the edge connected to r_i in H_k.*

This theorem can be proved using Lemma 3 and the fact that there can not be any cycle in the graph H_k. We will give an example.

Example 6 *Let $f = 25y^2z + 90yz^2 + 93x^2y^2z + 60y^4z + 42z^5$. Here $t = 5, n = 3, d_{max} = 5$ and $p = 101$. We choose the following evaluation points $\alpha_1 = 85, \alpha_2 = 96, \alpha_3 = 58$ and $\alpha_4 = 99$. Suppose we want to construct G_2 in order to compute the degrees of the monomials in y. Suppose our degree bound is $d = 40$ which is not tight. The graph G_2 and H_2 are shown in Figures 6 and 7 respectively.*

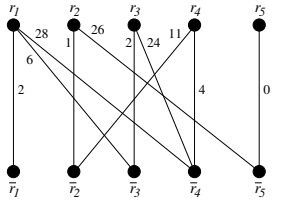

Figure 6: The bipartite graph G_2

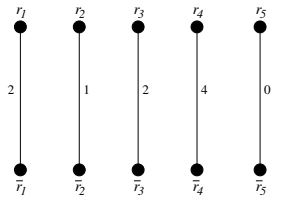

Figure 7: The bipartite graph H_2

The graph H_2 has the correct degrees of the monomials in variable y.

Theorem 4 suggests the following optimization. In the construction of the bipartite graph G_k, connect r_i to $\bar{r}_j$ with degree d_{ij} *only if* there is no $\bar{d} < d_{ij}$ such that $r_i \times (\frac{\alpha_{n+1}}{\alpha_k})^{\bar{d}}$ is a root of Λ_{k+1}, i.e. the degree of the node r_i in U is always one for all $1 \leq i \leq n$. If there is a perfect matching in this graph, this perfect matching is unique because this implies that the degree of each node $\bar{r}_j$ in V is also one (e.g. see Figure 7). If not, go back to and complete the graph G_k. This optimization makes our algorithm sensitive to the actual degree of $f(x_1, ..., x_n)$ in each variable.

The second optimization is to compute the degree of each monomial $M_i = x_1^{e_{i1}} x_2^{e_{i2}} \dots x_n^{e_{in}}$ in the last variable x_n without doing any more probes to the black box. Suppose we have computed the degree of M_i in x_k for $1 \le k < n$. We know that $M_i(\alpha_1, \dots, \alpha_n)$ is equal to r_i, a root of Λ_1. Hence $r_i = \alpha_1^{e_{i1}} \cdot \alpha_2^{e_{i2}} \cdot \dots \cdot \alpha_n^{e_{in}}$. Since we know the degrees e_{ij} for $1 \le j < n$ we can determine e_{in} by division by α_n. This reduces the total number of probes from $4(n+1)t$ to $4nt$.

5. BENCHMARKS

Here, we compare the performance of our new algorithm, Zippel's algorithm and the racing algorithm of Kaltofen and Lee from [11]. We have implemented Zippel's algorithm and our new algorithm in C. We have also implemented an interface to call the interpolation routines from Maple. The racing algorithm is implemented in Maple in the ProtoBox package by Lee [11]. Since this algorithm is not coded in C, we only report (see columns labelled PBox) the number of probes it makes to the black box.

We give benchmarks comparing the performances on five problem sets. The polynomials in the first four benchmarks were generated at random. The fifth set of polynomials are taken from [11]. We count the number of probes to the black box and measure the total CPU time (for our new algorithm and Zippel's algorithm only). All the timings given in this section are in CPU seconds and were obtained using Maple 13 on a 64 bit Intel Core i7 920 @ 2.66GHz, running Linux. This is a 4 core machine. For our algorithm, we report the real time for 1 core and (in parentheses) 4 cores.

The black box in our benchmarks computes a multivariate polynomial with coefficients in $\mathbb{Z}_p$ where $p = 3037000453$ is a 31.5 bit prime. In all benchmarks, the black box simply evaluates the polynomial at the given evaluation point. To evaluate efficiently we compute and cache the values of x_i^j mod p in a loop in $O(nd)$. Then we evaluate the t terms in $O(nt)$. Hence the cost of one black box probe is $O(nd+nt))$ arithmetic operations in $\mathbb{Z}_p$.

Benchmark #1

This set of problems consists of 13 multivariate polynomials in $n = 3$ variables. The i'th polynomial ($1 \le i \le 13$) is generated at random using the following Maple command:

```
> randpoly([x1,x2,x3], terms = 2^i, degree = 30) mod p;
```

The i'th polynomial will have about 2^i non-zero terms. Here $D = 30$ is the total degree hence the maximum number of terms in each polynomial is $t_{max} = \binom{n+D}{D} = 5456$. We run both the Zippel's algorithm and our new algorithm with degree bound $d = 30$. The timings and the number of probes are given in Table 1. In this table "DNF" means that the algorithm did not finish after 12 hours.

As i increases, the polynomial f becomes denser. For $i > 6$, f has more than $\sqrt{t_{max}}$ non-zero terms. This is indicated by a horizontal line in Table 1 and also in subsequent benchmarks. The line approximately separates sparse inputs from dense inputs. The last polynomial ($i = 13$) is 99.5% dense.

The data in Table 1 shows that for sparse polynomials $1 \le i \le 6$, our new algorithm does a lot fewer probes to the black box compared to Zippel's algorithm. It also does fewer probes than the racing algorithm (PBox). However, as the polynomials get denser, Zippel's algorithm has a better performance. For a completely dense polynomial with t non-zero terms, Zippel's algorithm only does $O(t)$ probes to the black box while the new algorithm does $O(nt)$ probes.

i	t	New Algorithm		Zippel		PBox
		Time	Probes	Time	Probes	Probes
1	2	0.00 (0.00)	12	0.00	217	20
2	4	0.00 (0.00)	24	0.00	341	39
3	8	0.00 (0.00)	48	0.00	558	79
4	16	0.00 (0.00)	96	0.01	868	156
5	32	0.00 (0.00)	192	0.01	1519	282
6	64	0.01 (0.01)	384	0.03	2573	517
7	128	0.03 (0.02)	768	0.08	4402	962
8	253	0.11 (0.06)	1518	0.21	6417	1737
9	512	0.44 (0.24)	3072	0.55	9734	3119
10	1015	1.66 (0.88)	6090	1.16	12400	5627
11	2041	6.50 (3.44)	12246	2.43	15128	DNF
12	4081	25.3 (13.4)	24486	4.56	16182	DNF
13	5430	44.3 (23.3)	32580	5.93	16430	DNF

Table 1: benchmark #1: $n = 3$ and $D = 30$

To show how effective the first optimization described in Section 4.2 is, we run both our algorithm and Zippel's algorithm on the same set of polynomials but with a bad degree bound $d = 100$. The timings and the number of probes are given in Table 2. One can see that our algorithm is unaffected by the bad degree bound; the number of probes and CPU timings are the same.

i	t	New Algorithm		Zippel's Algorithm	
		Time	Probes	Time	Probes
1	2	0.00 (0.00)	12	0.01	707
2	4	0.00 (0.00)	24	0.01	1111
3	8	0.00 (0.00)	48	0.02	1818
4	16	0.00 (0.00)	96	0.03	2828
5	32	0.00 (0.00)	192	0.07	4949
6	64	0.01 (0.01)	384	0.14	8383
7	128	0.04 (0.02)	768	0.36	14342
8	253	0.12 (0.07)	1518	0.79	20907
9	512	0.45 (0.24)	3072	1.97	31714
10	1015	1.67 (0.89)	6090	3.97	40400
11	2041	6.50 (3.45)	12246	8.18	49288
12	4081	25.3 (13.4)	24486	15.16	52722
13	5430	44.1 (23.4)	32580	19.62	53530

Table 2: benchmark #1: bad degree bound $d = 100$

Benchmark #2

In this set of benchmarks the i'th polynomial is in $n = 3$ variables and is generated at random in Maple using

```
> randpoly([x1,x2,x3], terms = 2^i, degree = 100) mod p;
```

This set of polynomials differs from the first benchmark in that the total degree of each polynomial is set to be 100 in the second set. We run both the Zippel's algorithm and our new algorithm with degree bound $d = 100$. The timings and the number of probes are given in Table 3. Comparing this table to the data in Table 1 shows that the number of probes to the black box in our new algorithm does not depend on the degree of the target polynomial.

i	t	New Algorithm		Zippel		PBox
		Time	Probes	Time	Probes	Probes
1	2	0.00 (0.00)	12	0.01	707	19
2	4	0.00 (0.00)	24	0.01	1111	45
3	8	0.00 (0.00)	48	0.02	1919	89
4	16	0.00 (0.00)	96	0.04	3434	167
5	31	0.00 (0.00)	186	0.08	6161	320
6	64	0.02 (0.01)	384	0.19	10504	623
7	127	0.05 (0.02)	762	0.49	18887	1149
8	253	0.17 (0.09)	1518	1.38	32219	2137
9	511	0.66 (0.34)	3066	4.36	56863	4103
10	1017	2.54 (1.31)	6102	13.99	98677	7836
11	2037	9.83 (5.09)	12222	43.23	166650	DNF
12	4076	38.7 (19.9)	24456	121.68	262802	DNF
13	8147	152. (78.3)	48882	282.83	359863	DNF

Table 3: benchmark #2: $n = 3$ and $D = 100$

i	t	New Algorithm		Zippel		PBox
		Time	Probes	Time	Probes	Probes
1	2	0.00 (0.00)	44	0.03	1736	67
2	4	0.00 (0.00)	96	0.04	3038	121
3	8	0.00 (0.00)	192	0.08	5053	250
4	15	0.00 (0.00)	360	0.20	10230	470
5	32	0.02 (0.01)	768	0.54	18879	962
6	63	0.04 (0.02)	1512	1.79	36735	1856
7	127	0.15 (0.05)	3048	6.10	69595	3647
8	255	0.54 (0.17)	6120	22.17	134664	7055
9	507	2.01 (0.60)	12168	83.44	259594	13440
10	1019	7.87 (2.33)	24456	316.23	498945	26077
11	2041	31.0 (9.16)	48984	1195.13	952351	DNF
12	4074	122.3 (35.9)	97776	4575.83	1841795	DNF
13	8139	484.6 (141.)	195336	>10000	-	DNF

Table 5: benchmark #4: $n = 12$ and $D = 30$

Benchmarks #3 *and* #4

These sets of problems consist of 14 random multivariate polynomials in $n = 6$ variables and $n = 12$ variables all of total degree $D = 30$. The i'th polynomial will have about 2^i non-zero terms. We run both the Zippel's algorithm and our new algorithm with degree bound $d = 30$. The timings and the number of probes are given in Tables 4 and 5.

i	t	New Algorithm		Zippel		PBox
		Time	Probes	Time	Probes	Probes
1	2	0.00 (0.00)	24	0.01	496	37
2	3	0.00 (0.00)	36	0.01	651	59
3	8	0.00 (0.00)	96	0.01	1364	140
4	16	0.00 (0.00)	192	0.02	2511	284
5	31	0.00 (0.00)	372	0.05	4340	521
6	64	0.02 (0.01)	768	0.15	8060	995
7	127	0.06 (0.03)	1524	0.44	14601	1871
8	255	0.21 (0.09)	3060	1.51	27652	3615
9	511	0.81 (0.35)	6132	5.19	50530	6692
10	1016	3.10 (1.33)	12192	17.94	90985	12591
11	2037	12.2 (5.21)	24444	65.35	168299	DNF
12	4083	48.1 (20.5)	48996	230.60	301320	DNF
13	8151	189. (80.1)	97812	803.26	532549	DNF

Table 4: benchmark #3: $n = 6$ and $D = 30$

To assess the parallel implementation of our algorithm, Table 6 reports timings for benchmark #4 for our algorithm running on 1, 2 and 4 cores showing the speedup we obtain using 2 and 4 cores. We report (in column roots) the time spent computing the roots in step 10 of $\Lambda_1(z)$ using our implementation of Rabin's algorithm which uses classical polynomial arithmetic, and (in column solve) the time solving the linear system for the coefficients in step 22 and (in column probes) the total time spent probing the black box. The data shows that computing the roots will become a bottleneck for our parallel implementation for more cores. Thus for 2 cores and 4 cores we report two timings. The first (in column time 1) is for our parallel algorithm as presented. For the second (faster) time (in column time 2) we have parallelized the second and subsequent steps of the root finding algorithm which yields a modest speedup. The data can be interpreted as follows. For $i = 13$ the sequential time is 484.6s. Of this, 34.7s was spent computing the roots of $\Lambda_1(z)$ and 5.02s was spent solving for the coefficients. Thus the algorithm has a sequential component of $34.7 + 5.02 = 39.7$s and so the maximum possible speedup on 4 cores is a factor of $484.6/((484.6 - 39.7)/4 + 39.7) = 3.21$ compared with the observed speedup factor of $484.6/152.5 = 3.18$. One way to remove this bottleneck would be to use a fast multiplication and division algorithm for $\mathbb{Z}_p[z]$.

Benchmark #5

In this benchmark, we compare our new algorithm and the racing algorithm on seven target polynomials (below) from [11, p. 393]. Note, f_6 is dense. The number of probes for each algorithm is reported in Table 7.

$$f_1(x_1,\ldots,x_9) = x_1^2x_3^3x_4x_6x_8x_9^2 + x_1x_2x_3x_4^2x_5^2x_8x_9 +$$
$$x_2x_3x_4x_5^2x_8x_9 + x_1x_3^3x_4^2x_5^2x_6^2x_7x_8^2 + x_2x_3x_4x_5^2x_6x_7x_8^2$$
$$f_2(x_1,\ldots,x_{10}) = x_1x_2^2x_4^2x_8x_9^2x_{10}^2 + x_2^2x_4x_5^2x_6x_7x_9x_{10}^2 +$$
$$x_1^2x_2x_3x_5^2x_7^2x_9^2 + x_1x_3^2x_4^2x_7^2x_9^2 + x_1^2x_3x_4x_7^2x_8^2$$
$$f_3(x_1,\ldots,x_9) = 9x_2^3x_3^3x_5^2x_6^2x_8^3x_9^3 + 9x_1^3x_2^2x_3^3x_5^2x_7^2x_8^2x_9^3 +$$
$$x_1^4x_3^4x_4^2x_5^4x_6^4x_7x_8^5x_9 + 10x_1^4x_2x_3^4x_4^4x_5^4x_7x_8^3x_9 + 12x_2^3x_3^3x_4^3x_6^3x_7^2x_8^3$$
$$f_4(x_1,\ldots,x_9) = 9x_1^2x_3x_4x_6^3x_7^2x_8x_{10}^4 + 17x_1^3x_2x_5^2x_6^2x_7x_8^3x_9^4x_{10}^3 +$$
$$3x_1^3x_2^2x_6^3x_{10}^2 + 17x_2^2x_3^4x_4^2x_7^4x_8^3x_9x_{10}^3 + 10x_1x_3x_5^2x_6^2x_7^4x_8^4$$
$$f_5(x_1,\ldots,x_{50}) = \textstyle\sum_{i=1}^{i=50} x_i^{50}$$
$$f_6(x_1,\ldots,x_5) = \textstyle\sum_{i=1}^{i=5} (x_1 + x_2 + x_3 + x_4 + x_5)^i$$
$$f_7(x_1,x_2,x_3) = x_1^{20} + 2x_2 + 2x_2^2 + 2x_2^3 + 2x_2^4 + 3x_3^{20}$$

i	n	d	$\#f_i$	New Algorithm	ProtoBox
1	9	3	5	90	126
2	10	2	5	100	124
3	9	3	5	90	133
4	9	4	5	100	133
5	50	50	50	5000	251
6	5	5	251	2510	881
7	3	20	6	36	41

Table 7: benchmark #5.

		1 core				2 cores			4 cores		
i	t	time	roots	solve	probe	time 1	time 2	(speedup)	time 1	time 2	(speedup)
6	63	0.04	0.01	0.00	0.04	0.03	0.02		0.02	0.02	
7	127	0.15	0.02	0.00	0.15	0.08	0.08	(1.87x)	0.06	0.05	(3x)
8	255	0.54	0.05	0.00	0.41	0.30	0.28	(1.93x)	0.18	0.17	(3.18x)
9	507	2.02	0.18	0.02	1.48	1.11	1.06	(1.91x)	0.67	0.60	(3.37x)
10	1019	7.94	0.65	0.08	5.76	4.35	4.17	(1.90x)	2.58	2.33	(3.41x)
11	2041	31.3	2.47	0.32	22.7	17.1	16.3	(1.92x)	9.94	9.16	(3.42x)
12	4074	122.3	9.24	1.26	90.0	67.1	64.7	(1.89x)	38.9	35.9	(3.41x)
13	8139	484.6	34.7	5.02	357.3	264.9	255.8	(1.89x)	152.5	141.5	(3.42x)

Table 6: Parallel speedup timing data for benchmark #4 for the new algorithm.

6. CONCLUSION

Our sparse interpolation algorithm is a modification of the Ben-Or/Tiwari algorithm [1] for polynomials over finite fields. It does a factor of between n and $2n$ more probes where n is the number of variables. Our benchmarks show that for sparse polynomials, it usually does fewer probes to the black box than Zippel's algorithm and the racing algorithm of Kaltofen and Lee. Unlike Zippel's algorithm and the racing algorithm, our algorithm does not interpolate each variable sequentially and thus can more easily be parallelized. Our parallel implementation using Cilk, which parallelized only the main loops, demonstrates a good speedup. The downside of our algorithm is that it is clearly worse than Zippel's algorithm and the racing algorithm for dense polynomials. This disadvantage is partly compensated for by the increased parallelism.

Although we presented our algorithm for interpolating over $\mathbb{Z}_p$, it also works over any finite field $GF(q)$. Furthermore, if p (or q) is too small, one can work inside a suitable extension field. We conclude with some remarks about the choice of p in applications where one may choose p.

Theorem 1 says that monomial collisions are likely when $\frac{dt^2}{2\phi(p-1)} > \frac{1}{2}$, that is when $\phi(p-1) < dt^2$. In our benchmarks we used the 31.5 bit prime 3037000453. This is the biggest prime that we can use when programming in C on a 64 bit machine using signed 64 bit machine integers. Using this prime, if $d = 30$, monomial collisions will likely occur when $t > 5,808$ which means 31.5 bit primes are too small for large applications.

It is not difficult to choose p so that $p - 1 = 2q$ with q also prime. The largest such 31.5 bit prime is 3037000427. Solving $\phi(p-1) < dt^2$ for t with $d = 30$ using this prime gives $t > 7,114$. This choice of prime also makes it easy to find generators. However, for $p-1 = 2q$, since -1 is the only element of order 2, any value from the interval $[2, p-2]$ will have order q or $2q$. If we use elements of order q as well as generators in our algorithm, then the probability that two monomials collide is less than $dt^2/(2p-6)$ (using $|S| = p-3$ in the proof of Theorem 1). Solving $p - 3 < dt^2$ for t using $d = 30$ and $p = 3037000427$ yields $t > 10,061$.

The 31.5 bit prime limitation is not a limitation of the hardware, but of the C programming language. On a 64 bit machine, one can use 63 bit primes if one programs multiplication in assembler. We are presently implementing this.

7. ACKNOWLEDGMENTS

We would like to thank Wen-shin Lee for making the source code of the ProtoBox Maple package available to us.

8. REFERENCES

[1] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proc. of the twentieth annual ACM symposium on Theory of computing*, pages 301–309. ACM, 1988.

[2] Cilk 5.4.6 Reference Manual, `http://supertech.csail.mit.edu/cilk/manual-5.4.6.pdf`. Supercomputing Technologies Group, MIT Lab for Computer Science. `http://supertech.lcs.mit.edu/cilk`.

[3] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2003.

[4] M. Giesbrecht, G. Labahn and W. Lee. Symbolic-numeric sparse interpolation of multivariate polynomials. *J. Symb. Comput.*, 44:943–959, 2009.

[5] D. Grigoriev, M. Karpinski, and M. Singer. Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields. *SIAM J. Comput.* 19:1059–1063, 1990.

[6] M. A. Huang and A. J. Rao. Interpolation of sparse multivariate polynomials over large finite fields with applications. *Journal of Algorithms*, 33:204–228, 1999.

[7] S. M. M. Javadi and M. Monagan. A sparse modular gcd algorithm for polynomials over algebraic function fields. In *Proc. of ISSAC '07*, pages 187–194. ACM, 2007.

[8] E. Kaltofen and Y. N. Lakshman. Improved sparse multivariate polynomial interpolation algorithms. In *Proc. of ISSAC '88*, pages 467–474. Springer-Verlag, 1989.

[9] E. Kaltofen, Y. N. Lakshman, and J.-M. Wiley. Modular rational sparse multivariate polynomial interpolation. In *Proc. of ISSAC '90*, pages 135–139. ACM, 1990.

[10] E. Kaltofen, W. Lee, and A. A. Lobo. Early termination in Ben-Or/Tiwari sparse interpolation and a hybrid of Zippel's algorithm. In *Proc. of ISSAC '00*, pages 192–201. ACM, 2000.

[11] E. Kaltofen and W. Lee . Early termination in sparse interpolation algorithms. *J. Symb. Comput.*, 36(3-4):365–400, 2003.

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

[13] M. Monagan J. de Kleine and A. Wittkopf. Algorithms for the non-monic case of the sparse modular gcd algorithm. In *Proc. of ISSAC '05*, pages 124–131. ACM, 2005.

[14] 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.

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

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

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

[18] Richard Zippel. Interpolating polynomials from their values. *J. Symb. Comput.*, 9(3):375–403, 1990.