Factoring Solution Sets of Polynomial Systems in Parallel

Anton Leykin[*] Jan Verschelde[†]

Abstract

We report on a £rst parallel implementation of a recent algorithm to factor positive dimensional solution sets of polynomial systems. As the algorithm uses homotopy continuation, we observe a good speedup of the path tracking jobs. However, for solution sets of high degree, the overhead of managing different homotopies and large lists of solutions exposes the limits of the master/servant parallel programming paradigm for this type of problem. A probabilistic complexity study suggests modi£cations to the method which will also improve the serial version of the original algorithm.

2000 Mathematics Subject Classi£cation. *Primary 65H10. Secondary 14Q99, 68W30.*

Key words and phrases. *Linear trace, monodromy, numerical algebraic geometry, numerical homotopy algorithms, numerical irreducible decomposition, parallel computation, path following, polynomial systems.*

1 Introduction

Systems of polynomial equations occur frequently in various £elds of science and engineering [22], such as the assembly of a mechanical device. In addition to isolated solutions, a system may have positive dimensional solution sets, for example, when the corresponding mechanism permits motion. Once the dimension and the degree of a solution set are known, the next important question concerns its irreducible decomposition. Does the solution set factor? Or is there only one irreducible factor? The number of irreducible factors is related to the number of possible ways one can assemble the pieces of the mechanical device.

One recent algorithm [17, 18] to decompose a pure dimensional solution set into irreducible components uses homotopy continuation methods [14]. See [1, 3, 9] for granularity issues of path following. The preliminary implementation of our factorization algorithm has factored solution sets of degrees up to one thousand. As path following methods are "embarrassingly parallel" (i.e.: the processors no longer have to communicate after the distribution of the solution paths), it would be embarrassing not to have a parallel implementation. However, many modern homotopies occur in sequences: some paths start at the end of other paths, e.g: Pieri homotopies, parallelized in [24]. The jobs we distribute are still path tracking jobs, but the homotopies change, and the jobs must follow a certain order of execution, thus no longer embarrassingly parallel.

The main contribution of this paper is the parallel implementation of a factorization algorithm, as an extension to PHCpack [23], a publicly available software package to solve polynomial systems. In addition to this parallel implementation, a probabilistic complexity study provides ideas to improve the ef£ciency of the algorithm, even on one single processor machine. Our computational experiments show that, while the path tracking jobs give a good speedup, to improve the total execution time of the factorization algorithm, alternatives to the master/servant parallel programming paradigm must be developed for this problem.

This paper is organized as follows. In the next section we give a formal de£nition of the problem, followed by the presentation of our algorithms in the third section. In section four, we present a probabilistic complexity study. Computational experiments with our £rst parallel implementation are given in section £ve. In section six we summarize our conclusions.

[*]Department of Mathematics, Statistics, and Computer Science, University of Illinois at Chicago, 851 South Morgan (M/C 249), Chicago, IL 60607-7045, USA. *Email:* leykin@math.uic.edu. *URL:* http://www.math.uic.edu/~leykin.

[†]Department of Mathematics, Statistics, and Computer Science, University of Illinois at Chicago, 851 South Morgan (M/C 249), Chicago, IL 60607-7045, USA. *Email:* jan@math.uic.edu or jan.verschelde@na-net.ornl.gov. *URL:* http://www.math.uic.edu/~jan. This material is based upon work supported by the National Science Foundation under Grant No. 0134611 and Grant No. 0410036.

2 Representing Varieties by Witness Sets

To deal numerically with “varieties” (the proper term for solution sets of polynomial systems), we desire (1) a memory ef£cient and (2) a well-conditioned representation. We achieve (1) by storing only the minimal data that is needed to sample new points from the variety, and (2) by taking random sections of the variety, thus avoiding singularities.

An application in algebraic statistics [6] (see also [10]) considers the solution set de£ned by all adjacent minors of a general 2-by-9 matrix:

$$\left[\begin{array}{cccc} x_{1,1} & x_{1,2} & \cdots & x_{1,9} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,9} \end{array}\right] \qquad (1)$$

Writing all adjacent minors gives 8 quadrics $x_{1,k}x_{2,k+1} - x_{2,k}x_{1,k+1} = 0$, $k = 1, 2, \ldots, 8$, in 18 unknowns, whose solution set is a 10-dimensional surface of degree 256. The degree 256 is seen algebraically as the product of all degrees of the de£ning equations, yielding 2^8. Geometrically, if we intersect the solution set with 10 random hyperplanes (10 is the difference between the number of variables and the number of equations), we will obtain 256 isolated solutions. Computing these 256 intersection points with homotopy methods takes on a modern workstation only a couple of minutes.

The numerical representation of the 10-dimensional surface then consists in the original 8 quadrics, the 10 random hyperplanes, and the 256 isolated solutions cut out by these 10 hyperplanes. As the quadrics are nice and easy to evaluate, this representation is well conditioned and serves as start system in a homotopy to generate new samples from the surface. We formalize this concept in the following de£nition.

De£nition 2.1 A *witness set* for a k-dimensional solution set Z of degree d in $\mathbb{C}^n$ consists of

1. a system $f(\mathbf{x}) = \mathbf{0}$ of polynomial equations in n variables $\mathbf{x}$;
2. k hyperplanes $L(\mathbf{x}) = \mathbf{0}$, whose coef£cients are random; and
3. a list W_L of d solutions, $W_L = \{ \mathbf{x} \in \mathbb{C}^n \mid f(\mathbf{x}) = \mathbf{0}$ and $L(\mathbf{x}) = \mathbf{0} \}$.

Note that, instead of the multivariate polynomials in the system f in item (1), algorithms to evaluate and differentiate the polynomials in the system serve equally well.

By the random choice of the coef£cients of the k hyperplanes of the linear system $L(\mathbf{x}) = \mathbf{0}$, the k-dimensional solution set Z is reduced to a list W_L of isolated solutions. If Z occurs with no multiplicity (which we assume), all points in W_L are nice well-conditioned solutions of the system $f(\mathbf{x}) = \mathbf{0}$, augmented with the k linear equations of L.

Many varieties arising in practical applications are often de£ned by overdetermined polynomial systems, i.e.: having more equations than unknowns. As we still need to intersect with k hyperplanes to get to isolated solutions cutting away the k degrees of freedom, we proposed the addition of slack variables – in analogy with linear programming – and a cascade of homotopies in [16] to ef£ciently compute witness sets on all positive dimensional solution sets of a polynomial system. Witness sets are a key concept in “numerical algebraic geometry”, a new £eld proposed in [21]. The subject of this paper is to address the development of a parallel algorithm to solve one of the main problems in computational algebraic geometry.

The input of our factorization algorithms is a witness set, formalized in De£nition 2.1. The output is formalized in the following de£nition:

De£nition 2.2 A *numerical factorization* of a pure dimensional solution set is a partition of a witness set. Every set in the partition represents an irreducible factor of the solution set.

One speci£c application is the factorization of multivariate polynomials. For example, $x^2 + y^2 - 1$ is irreducible, whereas $x^2 - y^2$ can be written as the product of $x - y$ and $x + y$. The need for a polynomial time algorithm for the approximate multivariate polynomial factorization expressed in [12] received a lot of research attention [4, 5, 7, 8, 11, 15, 20].

3 Monodromy Breakup certi£ed by Linear Trace

Given a partitioned witness set, with linear traces we can certify whether the sets in the partition correspond to a factor with relatively little work. For example, consider a planar cubic curve de£ned by the equation $f(x, y) = 0$. After a generic coordinate change, we may assume y^3 appears in f with a nonzero coef£cient. Then we may view f as $f(x, y(x))$ and consider $f(x, y(x)) = (y - y_1(x))(y - y_2(x))(y - y_3(x))$. Expanding the £rst two factors gives

$$(y^2-(y_1(x)+y_2(x))y+y_1(x)y_2(x))(y-y_3(x)). \quad (2)$$

The y-values of points on the curve are thus considered as functions $y_1(x)$, $y_2(x)$, $y_3(x)$ depending on x. Suppose the witness set of the cubic curve is partitioned into $\{\{(x_0, y_1(x_0),(x_0, y_2(x_0))\}, \{(x_0, y_3(x_0)\}\}$. If this corresponds to the true factorization of f, then $t_1(x) = y_1(x) + y_2(x)$ must be linear in x, in order for the degree of the £rst factor of f in (2) to be two. With samples at x_0 and at x_1, we £nd coef£cients a and b after solving the following linear system:

$$\left\{ \begin{array}{l} t_1(x_0) = y_1(x_0) + y_2(x_0) = ax_0 + b \\ t_1(x_1) = y_1(x_1) + y_2(x_1) = ax_1 + b. \end{array} \right. \quad (3)$$

Then the test for the linearity of $y_1(x) + y_2(x)$ is executed by

$$t_1(x_2) = ax_2 + b = y_1(x_2) + y_2(x_2). \quad (4)$$

The equality in (4) on the value $t_1(x_2)$ compares what is predicted from samples at x_0 and x_1 with what is observed from summing the y-values from samples at x_2. If the test holds, then f has a quadratic factor.

The use of the linear trace is a key ingredient in many factorization algorithms, see for example [4, 7, 15, 18, 20]. While for low degree solution sets, a good approach (as done in [7]) is to enumerate all possible partitions and then to apply the linear trace test, for high degrees this approach becomes prohibitively expensive. We can predict a breakup by generating loops around singularities, exploiting the monodromy group action. We applied this idea in [17], see also [5] and [20]. Note that for this breakup to work, we do not need to know the precise location of the singularities.

Generating loops is naturally done by homotopies. Starting from a list of well-conditioned solutions in W_L, we compute more samples of a k-dimensional solution set, moving L to another set of k random hyperplanes $K(\mathbf{x}) = \mathbf{0}$, using the homotopy $h_{L,K,\alpha}(\mathbf{x}, t)$

$$= \left(\begin{array}{c} f(\mathbf{x}) \\ \alpha(1-t)L(\mathbf{x}) + tK(\mathbf{x}) \end{array} \right) = \mathbf{0}, \quad \alpha \in \mathbb{C}. \quad (5)$$

At $t = 0$, the solution paths start at the set W_L and end, at $t = 1$, giving a new set W_K. The constant α is chosen at random, to avoid singularities. A loop is then generated by picking another constant, say β, and return from W_K to W_L using the homotopy $h_{K,L,\beta}(\mathbf{x}, t)$

$$= \left(\begin{array}{c} f(\mathbf{x}) \\ \beta(1-t)L(\mathbf{x}) + tK(\mathbf{x}) \end{array} \right) = \mathbf{0}, \quad \beta \in \mathbb{C}. \quad (6)$$

At $t = 1$, we £nd the same set W_L back, but with possible permutations in the order of solutions. As a permutation may occur only among points on the same irreducible component, one loop typically reveals more about the factorization.

With monodromy loops we join points on the same irreducible components. To certify whether we have found enough points, we use the linear trace test, as a stop condition for the monodromy breakup algorithm, i.e.: the monodromy breakup algorithm stops when all irreducible components have passed the linear trace test. The linear trace test requires two new witness sets, with samples taken on hyperplane sections parallel to the original ones.

We give the sequential version of the factorization method in Algorithm 3.1 below, and indicate at the right the obvious parallelization strategy, using a master/servant scheme.

Obviously, the amount of computational work in Algorithm 3.1 is concentrated in the "track d paths" statements. Statements 2 and 3 are needed to use the linear trace as stop test. Algorithm 3.1 generates one loop with statements 4.2 and 4.4. Every path tracking statement occurs with a different homotopy.

Notice that the communication overhead is relatively minor. The polynomials in the homotopy are sent out only once, at the beginning of the algorithm. The data transmitted in the loop consists of the numbers de£ning the moving hyperplane and the corresponding solutions.

Since we choose generic hyperplane sections, every path tracking job takes roughly the same amount of work, so a static work load distribution will perform well. In our parallel implementation we have one master node distributing the d paths evenly among the available nodes. The master node is responsible for broadcasting the homotopy, collecting the results, computing the permutation, updating the partition, and the linear trace test.

4 A Probabilistic Complexity Study

In order to test out various approaches to the parallelization of the monodromy breakup algorithm we developed a probabilistic model based on several crucial assumptions/postulates. Note that these assumptions are not meant to model the reality closely and we intentionally sacri£ce the quality of the model in favor of simplicity.

```
Algorithm 3.1 Monodromy Breakup certi£ed by Linear Trace: $\mathcal{P}$ = Breakup($W_L, d, N$)
  Input: $W_L$, $d$, $N$                                          witness set, degree, max #loops
  Output: $\mathcal{P}$                                              partitioned witness set

  0. initialize $\mathcal{P}$ with $d$ singletons;                               done by master node
  1. generate two slices $L'$ and $L''$ parallel to the given $L$;      broadcast data to all nodes
  2. track $d$ paths for witness set with $L'$;                  executed in parallel by servants
  3. track $d$ paths for witness set with $L''$;                 executed in parallel by servants
  4. for $k$ from 1 to $N$ do
       4.1 generate new slices $K$ and a random $\alpha$;           broadcast $K$ and $\alpha$ to all nodes
       4.2 track $d$ paths de£ned by (5);                        executed in parallel by servants
       4.3 generate a random $\beta$;                                   broadcast $\beta$ to all nodes
       4.4 track $d$ paths de£ned by (6);                        executed in parallel by servants
       4.5 compute the permutation and update $\mathcal{P}$;                done by master node
       4.6 if linear trace test certi£es $\mathcal{P}$
             then leave the loop;
           end if;
     end for.
```

The following setup is assumed:

• The model algorithm's goal is to certify the irreducibility of a solution component of degree d using s generic witness sets $W_1, W_2, \ldots, W_s$ that are the solutions of the following systems for $i = 1, 2, \ldots, s$:

$$\begin{cases} f(\mathbf{x}) &= \mathbf{0} \quad \text{(the original system)} \\ L_i(\mathbf{x}) &= \mathbf{0} \quad (d \text{ hyperplanes}) \end{cases} \tag{7}$$

• The (only) atomic operation the algorithm can perform is tracking a single path starting at a given point P of a given witness set W_i to discover where the other endpoint $Q \in W_j$, where W_j $(j \neq i)$ is another given component. Let us give it a name TRACK:

$$Q = \mathtt{TRACK}(P, W_i, W_j). \tag{8}$$

• We think of the witness sets W_i as vertices in the complete non-oriented graph, where the edge connecting W_i and W_j is assigned a nonnegative integer weight ν_{ij} equal to the number of calls of the form $\mathtt{TRACK}(-, W_i, W_j)$ made by the algorithm.

• We assume that the £rst task of the algorithm is to establish a 1-to-1 correspondence between the points of different witness sets.

After this is done, it makes sense to assume that the partitions of different witness sets agree with each other. Therefore, the algorithm records the current partition simply as a partition $C = \{C_1, C_2, \ldots, C_m\}$ of the set $\{1, 2, \ldots, d\}$ representing the witness points. At the beginning, C consists of d singletons $C_i = \{i\}$; at the end, C is supposed to have only one component.

• In practice, TRACK proceeds by picking a random $\gamma \in \mathbb{C}$, $|\gamma| = 1$ and tracking the path starting at $P \in W_i$ along the homotopy $h_{L_i,L_j,\gamma}(\mathbf{x}, t) = 0$ as in (5). The time this takes and the probability $prob(P, Q)$ of ending up at a particular $Q \in W_j$ are hard to determine or even predict analytically. In our model we assume that *every path takes 1 unit of time* to track.

Let Q be in the witness set W_j, but outside the component $C(P)$ of the current partition C that contains P. We also assume that *the probability is distributed uniformly* among all $d - |C(P)|$ possible endpoints Q outside the component $C(P)$, i.e. for £xed i, j:

$$prob_{(i,j)}(p, q_1) = prob_{(i,j)}(p, q_2), \tag{9}$$

where $p, q_1, q_2 \in \{1, ..., d\}$, $q_1, q_2 \notin C(p)$ where $C(p)$ is the current component of p and $prob_{(i,j)}(p, q)$ is the probability of going from the p-th point of W_i to the q-th point of W_j.

• Another crucial assumption on the probabilities distribution is imposed to account for the fact that, in reality, choosing a pair of witness sets and tracking paths exclusively between them is not the best strategy: the probability of landing in the same component as you started grows with the number of tracked paths.

Hence, for $q \notin C(p)$, we have to make $prob_{(i,j)}(p, q)$ dependent on ν_{ij}. The formula that we used for our test runs is

$$prob_{(i,j)}(p,q) = \frac{1}{d(1 + r\ln(1+\nu_{ij}))}, \qquad (10)$$

where r is a positive real parameter.

Considering the rules of the game above, we propose the following serial algorithm that conducts the needed computational experiment and returns the time t consumed by the model.

Algorithm 4.1 *M1(serial)* t = M1S(d, s)

```
Input: $d$ is the degree of the irreducible solution
          component in question;
       $s$ is the number of witness sets $W_1, \dots, W_s$
          used in the algorithm.
Output: $t$ is the time it takes to certify the irreducibility.

$C := \{\{1\}, \dots, \{d\}\}$;
  { To establish a 1-to-1 correspondence, track $d$ paths
  from $W_1$ to $W_i$, $(i = 2, \dots, s)$. }
$t := d(s-1)$; { This takes $d(s-1)$ units of time. }
while $|C| > 1$ do { while the partition is non-trivial ...}
    pick a point $p$ such that the size of its partition
      component $C(p)$ is minimal;
    pick an edge $(i,j)$ such that $\nu_{ij}$ is minimal;
    generate a random number $r \in [0,1]$;
    if $r < (d - |C(p)|) \cdot prob_{(i,j)}(p,q),\ q \notin C(p)$
    { ... the probability of getting outside $C(p)$ }
      then take a random $q \notin C(p)$;
           { assume $q :=$ TRACK$(p, W_i, W_j)$ }
           merge $C(p)$ and $C(q)$ in the partition $C$;
    end if;
    $t := t + 1$;
end while.
```

Given the assumptions on the probability distribution, this algorithm minimizes the expected time of certifying irreducibility.

The pseudo-paths considered at steps where the partition C remains unchanged are called a *rejected* paths and in the implementation of the model we have a counter $n_{rejected}$ to record the number of these.

Example 4.2 We performed a test run of `M1S` with $d = 200$, $s = 2, \dots, 10$. (Also, we have deliberately set $r = 2.5$ in (10) for this example). Table 1 displays the resulting time t (= total #paths) and the percentage of rejected paths.

s	2	...	6	7	8	...
n_{paths}	4146	...	3365	3333	4121	...
% rejected	90	...	64	58	61	...

Table 1. Maple sequential simulation results.

In Table 1 we see that the rejection percentage decreases with s. However, the cost of initialization of every witness set does not let us improve the running time in£nitely.

Therefore, in the future (practical) implementations it would make sense to develop a heuristic that would adjust the number of used witness sets dynamically.

The algorithm `M1P`,

$$t = \texttt{M1P}(d, s, \mathrm{NP}), \qquad (11)$$

where NP is the number of processors, is a parallel version of `M1S` that uses the following simple parallelization strategy.

At every loop it devotes all NP processors to tracking random paths starting at the component(s) of minimal size. There is no synchronization issue, since all loops take equal amount of time to compute by the assumption. The information gathered at the end of each loop is processed – in negligible time – by merging components of the partition C in a natural way. Although there are no idling processors by construction, some of the resources are still wasted; it may happen that several processors £nd connecting paths between the same two components simultaneously. The number of the redundant passes is stored in the counter $n_{duplicate}$.

Example 4.3 Continuing Example 4.2 we made a test run for `M1P` using the same parameters with the number of processors $\mathrm{NP} = 1, 11, 21, 31, 41, 51$, see Table 2.

Why the speedups are not linear could be explained, for instance, on the results for $s = 7$, the value of s that seems to be optimal for this example, see Table 3.

From Table 3 it is evident that the number of duplicate paths grows with NP and this triggers the growth in the percentage of rejections as well.

NP\s	2	5	6	7	8	9
1	1.0	1.0	1.0	1.0	1.0	1.0
11	11.1	11.9	9.53	9.95	11.1	10.7
21	18.4	21.2	16.7	16.5	19.1	19.0
31	24.8	28.4	21.9	21.5	26.6	26.1
41	30.6	32.2	29.5	26.9	34.3	32.1
51	34.2	39.0	30.6	29.2	37.5	38.1

Table 2. Speedups achieved with `M1P`, for NP processors.

NP	n_{paths}	% rejected	% duplicate
1	3333	58	0
11	3686	62	0.14
21	4245	67	0.54
31	4827	70	1.1
41	5095	70	2.3
51	5841	74	1.9

Table 3. Speedups with $s = 7$.

5 Implementation and Computation

Algorithm 3.1 was implemented using a master/servant method, as indicated in the algorithm. Extensive experiments on one medium sized benchmark system show the qualities of this £rst parallel factorization algorithm and also point at the defects of this straightforward parallel approach.

5.1 A state machine interface to PHCpack

PHCpack [23] is legacy software, developed – and still under development – for more than a decade. Three recent papers document new extensions to the software: tools to deal with positive dimensional solution sets [19]; an interface to Maple [13]; and a parallel version of the Pieri homotopies [24].

The parallel main program is written in C and uses the publicly available version of MPI. This program does not have any data structures for multivariate polynomials or lists of solutions. Yet, it is in complete control of the computations and has full access to all data, managed by PHCpack (written in Ada). The interface to PHCpack operates very much like a vending machine, whose operations fall in four categories: (1) put in data; (2) select options; (3) push go; and (4) collect results.

Our equipment consists of one workstation with two dual 2.4Ghz processors, running Linux, and serving two Rocketcalc clusters: one with four and an other with eight 2.4Ghz processors. So we have a total of 14 processors: one master node and 13 computing nodes.

5.2 Benchmarking and performance results

As the benchmark problem, we took a medium sized polynomial system: the well known cyclic 8-roots problem [2]. This system has a solution curve of degree 144, which breaks up into 16 irreducible factors. There are 8 factors of degree 16 and 8 quadratic factors. Table 4 shows the ¤uctuation between the number of loops needed to factor this curve.

Since seven is the average number of loops to factor this system, we retained only the timings from different runs on different number of processors which all took seven loops, see Table 5 and Figure 1.

Looking at the max row in Table 5, we see a decrease in a factor of about ten times when going from 2 to 14 processors, i.e.: in going from 144.3 to 14.1 seconds. The constant time of about 7 seconds spent by the master node becomes more and more substantial. The data is visualized in Figure 1.

6 Conclusions

We have presented a £rst parallel implementation of a factorization method using monodromy and linear traces, employing the conventional master/servant computation model. On selected examples of medium sized degrees around one hundred, we have experienced a good performance of this model. For higher degrees, the workload of the master node prevents a good speedup.

Our probabilistic model outlined the future modi£cations to the algorithm aimed at boosting the ef£ciency of both serial and parallel implementations. Our experimental complexity study examined the trade off between generating more slices and using more loops between the same two slices within the assumption that using the same slices over and over again lowers the probability of gaining new information. As polynomial systems often arise in families, information on the probabilities may be gained from computing smaller systems in the family, or by sample runs.

#L	4	5	6	7	7	7	7	7	8	9
min	6.0	7.8	9.2	10.1	10.3	10.9	10.9	10.7	11.8	12.3
max	9.9	11.5	12.8	15.4	15.1	14.7	14.1	14.5	16.3	16.9
total	11.7	14.9	16.9	19.2	19.3	19.5	19.7	20.3	21.9	23.4

Table 4. Results of 10 runs on 14 processors. Each time we recorded the number of loops #L, the minimal and maximal time (in seconds) spent by the path tracking nodes and the total amount of time.

NP	2	3	4	5	6	7	8	9	10	11	12	13	14
min	–	68.7	47.4	31.5	25.8	21.5	20.0	18.0	14.8	12.1	11.7	11.2	10.9
max	144.3	69.2	48.6	33.6	28.0	25.3	22.0	20.1	18.8	17.6	16.2	14.7	14.1
total	150.9	77.1	56.5	41.4	35.7	32.5	29.1	27.3	25.9	22.3	21.7	20.2	19.7

Table 5. Execution times for number of processors NP, from 2 to 14, using seven loops. We list the total time and the maximal and minimal computational times (expressed in seconds) for the processors. The difference between total and max is about 7 seconds, which is the time spent by the master node.

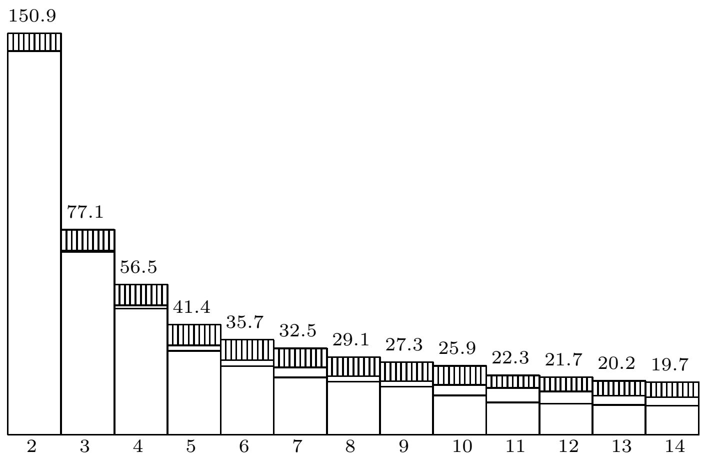

Figure 1. Plot of the data in Table 5. The shaded top area is the time spent by the master node. The lowest horizontal bar indicates the minimal time spent by the path tracking nodes.

Finally, we observed that the monodromy breakup algorithm makes the most gains in the £rst loops, often halving the number of irreducible factors in one step, but then spends several loops to £nd the connections for factors of small degree. A hybrid strategy, combining alternative methods [7, 15] based on the linear trace will further increase the performance of the factorization algorithm.

References

[1] D. Allison, A. Chakraborty, and L. Watson. Granularity issues for solving polynomial systems via globally convergent algorithms on a hypercube. *J. of Supercomputing*, 3:5–20, 1989.

[2] G. Björck and R. Fröberg. Methods to "divide out" certain solutions from systems of algebraic equations, applied to £nd all cyclic 8-roots. In M. Gyllenberg and L. Persson, editors, *Analysis, Algebra and Com-*

puters in Math. research, volume 564 of *Lecture Notes in Mathematics*, pages 57–70. Dekker, 1994.

[3] A. Chakraborty, D. Allison, C. Ribbens, and L. Watson. The parallel complexity of embedding algorithms for the solution of systems of nonlinear equations. *IEEE Transactions on Parallel and Distributed Systems*, 4(4):458–465, 1993.

[4] G. ChÁeze and A. Galligo. Four lectures on polynomial absolute factorization. In A. Dickenstein and I. Emiris, editors, *Solving Polynomial Equations: Foundations, Algorithms, and Applications*, volume 14 of *Algorithms and Computation in Mathematics*, pages 339–392. Springer–Verlag, 2005.

[5] R. Corless, A. Galligo, I. Kotsireas, and S. Watt. A geometric-numeric algorithm for factoring multivariate polynomials. In T. Mora, editor, *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation (ISSAC 2002)*, pages 37–45. ACM, 2002.

[6] P. Diaconis, D. Eisenbud, and B. Sturmfels. Lattice walks and primary decomposition. In B. Sagan and R. Stanley, editors, *Mathematical Essays in Honor of Gian-Carlo Rota*, volume 161 of *Progress in Mathematics*, pages 173–193. Birkhäuser, 1998.

[7] A. Galligo and D. Rupprecht. Irreducible decomposition of curves. *J. Symbolic Computation*, 33(5):661–677, 2002.

[8] X.-S. Gao, E. Kaltofen, J. May, Z. Yang, and L. Zhi. Approximate factorization of multivariate polynomials via differential equations. In J. Gutierrez, editor, *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation (ISSAC 2004)*, pages 167–174. ACM, 2004.

[9] S. Harimoto and L. Watson. The granularity of homotopy algorithms for polynomial systems of equations. In G. Rodrigue, editor, *Parallel processing for scienti£c computing*, pages 115–120. SIAM, 1989.

[10] S. Hoşten and J. Shapiro. Primary decomposition of lattice basis ideals. *Journal of Symbolic Computation*, 29(4&5):625–639, 2000.

[11] Y. Huang, W. Wu, H. Stetter, and L. Zhi. Pseudofactors of multivariate polynomials. In C. Traverso, editor, *Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation (ISSAC 2000)*, pages 161–168. ACM, 2000.

[12] E. Kaltofen. Challenges of symbolic computation: my favorite open problems. *J. Symbolic Computation*, 29(6):891–919, 2000.

[13] A. Leykin and J. Verschelde. PHCmaple: A Maple interface to the numerical homotopy algorithms in PHCpack. In Q.-N. Tran, editor, *Proceedings of the Tenth International Conference on Applications of Computer Algebra (ACA'2004)*, pages 139–147, 2004.

[14] T. Li. Numerical solution of polynomial systems by homotopy continuation methods. In F. Cucker, editor, *Handbook of Numerical Analysis. Volume XI. Special Volume: Foundations of Computational Mathematics*, pages 209–304. North-Holland, 2003.

[15] T. Sasaki. Approximate multivariate polynomial factorization based on zero-sum relations. In B. Mourrain, editor, *Proceedings of the 2001 International Symposium on Symbolic and Algebraic Computation (ISSAC 2001)*, pages 284–291. ACM, 2001.

[16] A. Sommese and J. Verschelde. Numerical homotopies to compute generic points on positive dimensional algebraic sets. *J. of Complexity*, 16(3):572–602, 2000.

[17] A. Sommese, J. Verschelde, and C. Wampler. Using monodromy to decompose solution sets of polynomial systems into irreducible components. In C. Ciliberto, F. Hirzebruch, R. Miranda, and M. Teicher, editors, *Application of Algebraic Geometry to Coding Theory, Physics and Computation*, pages 297–315. Kluwer Academic Publishers, 2001. Proceedings of a NATO Conference, February 25 - March 1, 2001, Eilat, Israel.

[18] A. Sommese, J. Verschelde, and C. Wampler. Symmetric functions applied to decomposing solution sets of polynomial systems. *SIAM J. Numer. Anal.*, 40(6):2026–2046, 2002.

[19] A. Sommese, J. Verschelde, and C. Wampler. Numerical irreducible decomposition using PHCpack. In M. Joswig and N. Takayama, editors, *Algebra, Geometry, and Software Systems*, pages 109–130. Springer–Verlag, 2003.

[20] A. Sommese, J. Verschelde, and C. Wampler. Numerical factorization of multivariate complex polynomials. *Theoretical Computer Science*, 315(2-3):651–669, 2004. Special Issue on Algebraic and Numerical Algorithms edited by I.Z. Emiris, B. Mourrain, and V.Y. Pan.

[21] A. Sommese and C. Wampler. Numerical algebraic geometry. In J. Renegar, M. Shub, and S. Smale, editors, *The Mathematics of Numerical Analysis*, volume 32 of *Lectures in Applied Mathematics*, pages 749–763. AMS, 1996. Proceedings of the AMS-SIAM Summer Seminar in Applied Mathematics. Park City, Utah, July 17-August 11, 1995, Park City, Utah.

[22] A. Sommese and C. Wampler. *The Numerical solution of systems of polynomials arising in engineering and science*. World Scienti£c, 2005.

[23] J. Verschelde. Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation. *ACM Trans. Math. Softw.*, 25(2):251–276, 1999. Software available at `http://www.math.uic.edu/˜jan`.

[24] J. Verschelde and Y. Wang. Computing feedback laws for linear systems with a parallel Pieri homotopy. In Y. Yang, editor, *Proceedings of the 2004 International Conference on Parallel Processing Workshops, 15-18 August 2004, Montreal, Quebec, Canada. High Performance Scienti£c and Engineering Computing*, pages 222–229. IEEE Computer Society, 2004.