

Parallel Implementation of a Subsystem-by-Subsystem Solver*

Yun Guan and Jan Verschelde
Department of Mathematics, Statistics, and Computer Science
University of Illinois at Chicago
851 South Morgan (M/C 249)
Chicago, IL 60607-7045, USA.

guan@math.uic.edu and jan@math.uic.edu

<http://www.math.uic.edu/~guan> and <http://www.math.uic.edu/~jan>

Abstract

Solving polynomial systems subsystem-by-subsystem means to solve a system of polynomial equations by first solving subsets of the system and then intersecting the results. The approach leads to numerical representations of all the solution components of a system. The focus of this paper is the development of a parallel implementation to solve large systems involving a recursive divide-and-conquer scheme. Because we concentrate our discussion on the distribution of the path tracking jobs, we have selected applications for which we have optimal homotopies, for which all paths converge to regular solutions.

p	time	min	max
16	22h47m	68,088	70,966
32	9h22m	33,329	34,113
64	4h44m	16,269	16,917
128	2h25m	7,199	8,639
256	1h16m	4,525	3,731

Table 1. Dynamic load balancing for tracking 1,048,576 solution paths. For p processors, we record the total time along with the minimum and maximum number of paths tracked by each worker node. Fluctuation in execution times occur because of different homotopy constants used in each run.

1 Introduction

A lot of effort has been spent recently on parallel implementations to solve polynomial systems numerically via homotopies. Starring in alphabetical order: Bertini [1], HOM4PS-2.0 [12], PHoMpara [7], and POLSYS_GLP [22]. We refer to [17] for a survey and introduction to homotopy continuation methods for solving polynomial systems. For an introduction to numerical algebraic geometry, we recommend the recent book [21].

As an illustration of a pleasingly parallel computation, we ran the classical total degree homotopies in PHCpack [23] on the polynomial systems of Katsura [10]. This family of systems consists of n quadrics and one linear equation. The number of complex solutions equals the Bézout bound 2^n . For $n = 20$, Table 1 lists execution times on Tungsten¹, using an increasing number of processors.

*This material is based upon work supported by the National Science Foundation under Grant No. 0105739, Grant No. 0134611, and Grant No. 0713018.

¹We thank the NCSA for granting access to Copper and Tungsten via TG-CCR060019N: Parallel Numerical Algebraic Geometry.

While solving this polynomial system remains still almost impossible by a single processor, even relatively modest cluster computers could handle it. A true supercomputer like Tungsten has no difficulty to complete a first run of all solution paths. Note that additional quality control, like re-running of paths at a finer resolution is not considered.

The computational results in Table 1 are obtained using a simple homotopy $h(x, t) = tf(\mathbf{x}) + (1 - t)g(\mathbf{x}) = \mathbf{0}$, where f is the system to be solved and each polynomial in g is of the form $x^{d_i} - c_i = 0$, for c_i some random complex number and d_i the degree of f_i . This type of homotopy does not yield any information about positive dimensional solution sets of a polynomial system.

The purpose of this paper is to outline the parallel implementation of a subsystem-by-subsystem solver, defined in [20], using the intrinsic version [19] of the diagonal homotopies [18]. The divide-and-conquer method has not been described elsewhere and is also of interest independently of parallel computation.

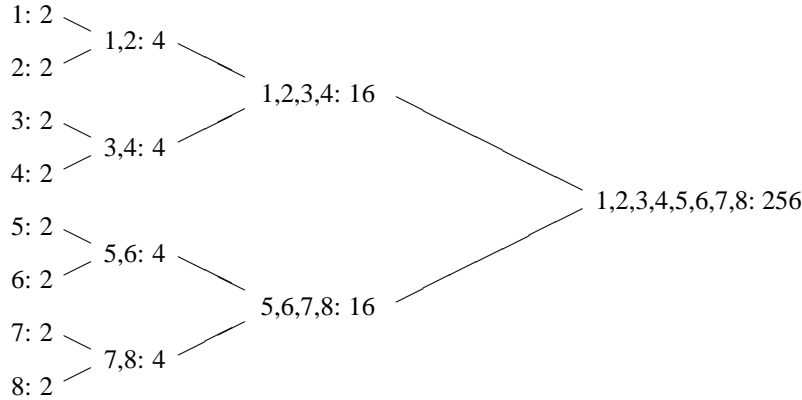


Figure 1. Schematic overview of solving a system of eight quadratics with an optimal homotopy.

Starting with [24], and continuing in [14, 13], [16] [25], we described our efforts to make PHCpack [23] run on parallel computers. With respect to those previous papers [24], [14, 13], [16], [25], this paper describes the parallel implementation of a new solver, to compute numerical representations of both *isolated* solutions and *positive dimensional* solution sets.

2 Witness Sets

In this section we explain our representations of solution sets (extrinsically and intrinsically). Furthermore we describe the memory efficient jumpstarting mechanism for intersecting solution sets.

As numerical representations for positive dimensional solution sets we use witness sets [21]. A solution set of dimension k is reduced to a collection of isolated points when intersected with k hyperplanes in general position. The number of isolated points obtained in this intersection equals the degree d of the solution set. A witness set records the dimension k and degree d of a solution set by storing a set of k general hyperplanes along with the d isolated solution in the intersection.

In the extrinsic representation the general linear spaces of dimension k used to intersect k -dimensional solution sets are determined as the intersection of $n - k$ hyperplanes and we add slack variables to have systems with as many equations as unknowns. In the intrinsic representation, a k -dimensional linear space is represented by an offset vector and k directions. This intrinsic representation allows us to use k parameters to describe a k -dimensional solution set.

The algorithm to intersect witness sets takes pairs of solutions, combining one solution of the first witness set with one solution of the second witness set. Even for relatively modest degrees – say a couple of thousands – the combination of all possible solutions at the start can lead to the reser-

vation of large chunks of memory space – with a million or more start solutions. To avoid this allocation of memory at the start of the computations, the implementation of diagonal homotopies has been redone – as was also needed for other homotopies in PHCpack (see [16]) – with a jumpstarting mechanism: start solutions are computed or read from file just in time, i.e.: only just before the path tracking starts.

Consider the intersection of a cubic and a quadratic surface, each represented respectively by 3 and 2 generic points. To intersect these surfaces, a lexicographic pairing of the points is formed, leading to (1,1), (1,2), (2,1), (2,2), (3,1), (3,2). For each pair we read one pair of solutions from separate files, one solution for each witness set. After the path tracking for the first two pairs is done, the input file with the witness set for the second surface is reset. The result of each path tracking job is immediately written to file. This jumpstarting mechanism lets us enumerate long lists of solutions using only as much memory as it takes to store the definition of the homotopy and one solution.

A comparison of witness sets with lifting fibers [11] appears in [20], see also [3]. Exploiting the similarities between these two data structures, we speculate that parallel implementations of those two (different) geometric solvers could proceed along the same lines.

3 Divide and Conquer

Suppose we have a system of eight quadratic equations. Consider Figure 1, as we assume the homotopy is *optimal*, i.e.: no solution paths diverge and the number of paths increases as the product of the degrees. The term *optimal homotopy* was coined in [9].

At first, there does not seem to be any benefit in arranging the path tracking as in the divide-and-conquer algorithm outlined as in Figure 1 because the number of solution paths exceeds 256, the total number of expected solutions. How-

ever, observe that in the intrinsic formulation of the witness sets, the number of independent variables stays very small at first: it takes only one variable to track a hypersurface. So only at the very last stage is the ambient dimension of the solution paths equal to the original number of variables. The largest proportion of the work is concentrated in the last stage. Even when no paths diverge, the benefit of this divide-and-conquer algorithm lies in having more stepping stones – we call them *witness stones* – in the resolution of large polynomial systems.

In general, a witness stone is an intermediate witness set such as the witness set consisting of 16 witness points for equations 1 through 4 in Figure 1. In the case of large polynomial systems, the advantage comes in if any failure happens in the middle of solving process, we may use those witness stones to continue, instead of solving the whole system from the very beginning. If paths diverge because of a special structure, and if this structure is shared among different subsets of equations (e.g.: equations 1 through 4 share the same structure as equations 5 through 8), then one could use coefficient-parameter continuation to solve the second set of equations, starting at the witness set of the first set of equations.

3.1 data structures to schedule jobs

To implement the divide-and-conquer witness set intersection algorithm, we use the following data structures.

One job represents the calculation of one point in the witness set representation of the intersection of two witness sets. A witness set is determined by the equations (their number and range) and the degree (number of solutions on random hyperplanes). The essential information for one job is a pair of indices, indicating the pair of solutions that must be taking from the pair of witness sets in the intersection.

The state table keeps track of the completed jobs. The number of rows in the table equals the total number of equations and there are as many columns as there are stages in the algorithm. This table is triangular as it maps the tree of the divide-and-conquer algorithm.

Queues of jobs and idle workers are maintained by the manager node as first-in first-out queues to ensure a fair balancing of the computational load among all workers. The job queue is empty as long as there are idle workers. Only if the queue of idle workers is empty, will the job queue grow.

As the total number of solutions grows considerably and our implementation is scheduled to run on a cluster of nodes, where each node has only as much internal memory as a typical workstation, we avoid that the manager

node keeps all solutions in its memory. The witness sets are stored on file and read by the worker nodes. As soon as a worker node sends the end solution of a path to the manager, the manager appends it to the file that stores the result of the witness set intersection.

3.2 scheduling jobs

Denote by n the total number of equations in the system. The number of processors equals p . In the manager-worker paradigm, the manager node has the identity label 0, where the remaining $p - 1$ workers are labeled 1 through $p - 1$.

The initialization phase consists in the creation of witness sets for n hypersurfaces, defined by the n equations in the system. This phase involves no path tracking, the points in each witness set are the roots of an univariate polynomial equation of modest degree. While this phase is computationally light, its parallel implementation provides a test on the communication network. The algorithm is summarized as in Algorithm 1.

Algorithm 1: Initialization Phase

Input: A polynomial system with n equations.

Output: n files that contain each a witness set for an equation in the given system.

Manager		Workers
broadcast filename	→	receive filename
send data	→	receive data
receive data	←	send data

The manager distributes n equations evenly among the $p - 1$ workers, first broadcasting the name of the file with the equations, then followed by a sequence of numbers, terminated by 0. Each number in the sequence is an index to an equation. If $n < p$, then $p - 1 - n$ workers receive only 0 and remain idle.

Each worker receives through the broadcast the name of the file and opens it to read the equations. For every nonzero index, a univariate polynomial equation is solved and its solutions are written to file in the proper witness set output format.

The distribution of the jobs in the initialization phase ends when every worker has received 0. To synchronize the termination, the following happens:

Each worker sends the index of the equation for which it has computed a witness set back to the manager. The last number that is sent to the manager is 0.

The manager should receive from every worker the same number the same numbers it has sent to.

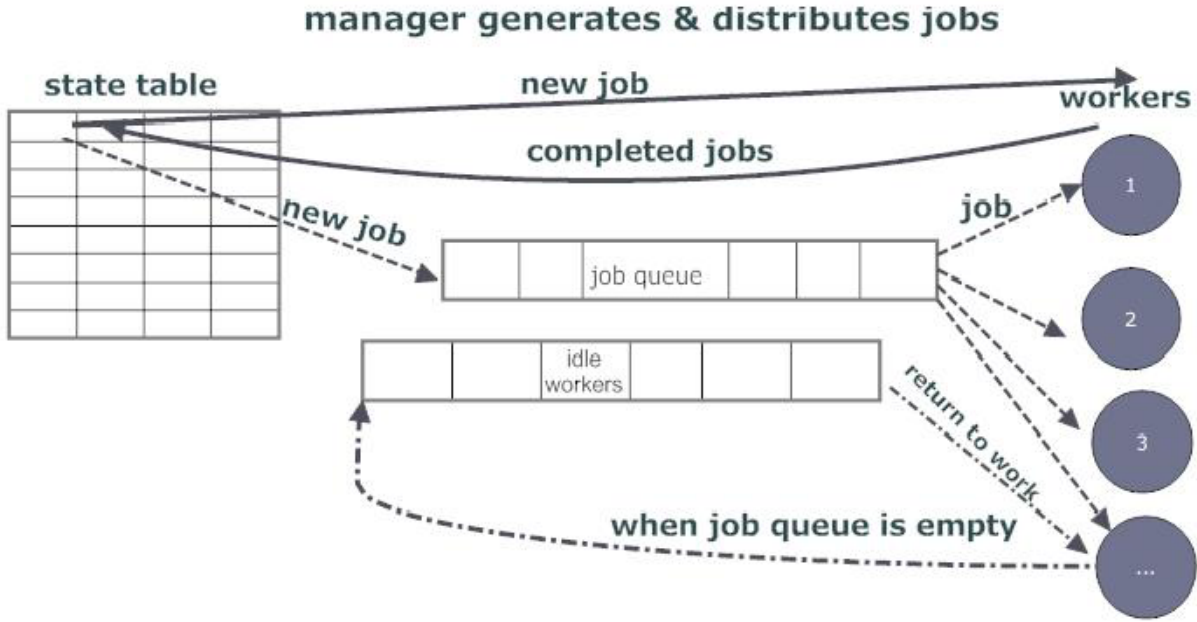


Figure 2. Overview of main loop.

The initialization phase ends if the manager has received 0 from every worker. The outcome of this phase consists in n files each containing a witness set for an equation.

The main loop in the algorithm runs in $\lceil \log_2(n) \rceil$ stages, homotopies in stage k involve 2^k equations. The manager node maintains the state table, the job queue, and the queue of idle workers. These data structures are used by the manager to distributed path tracking jobs among the workers. Figure 2 presents the actions in the main loop.

We describe the actions of worker and manager below:

A worker is either idle or busy tracking one path. The information to track a path is contained in a job. After receiving a job from the manager, the worker forms the right type of homotopy, reads the pair of solutions from file, and does the tracking of one path. After completion, the solution is sent back to the manager.

The manager receives a completed job from a worker node and manages the job scheduling then as follows. If the job queue is nonempty – in which case there are no idle workers – then the worker who completed the job will be immediately given the next job from the job queue. If the job queue is empty and the completed job does not lead to new jobs, then the label of the worker who completed the job will be added to the queue of idle workers. In both cases, the state table is updated with the completed job information and the computed solution is appended to the file that stores a new witness set.

To intersect a pair of witness sets there must be sufficient points in both sets. It could be that solutions for the first witness set accumulate at a steady pace, whereas the solutions for the second witness set are arriving much slowly. So the job queue becomes empty not only towards the end of the computations, but also at the beginning of a new stage.

At the start of the main loop, there are $n/2$ witness sets to be computed, requiring as many paths as the sum of product of the degrees of all consecutive pairs of equations in the system. In Figure 1, this first stage leads to 16 jobs to be distributed evenly among the workers. In general if at the start, there are N jobs to distribute, we have the following state of the queues. If $p - 1 \leq N$, then no workers will be idle and the job queue will contain the remaining $N - p - 1$ jobs. If $p - 1 > N$, then the job queue will be empty and there will be $p - 1 - N$ idle workers at the start.

Let us now consider the formation of new jobs in greater detail. When the manager node receives the information of a completed job, it consults the state table to find where the end solution will be used in the computation of the witness set intersection in the next stage. Recall that every job consists of a pair of labels to witness points. The new pairs consist of the result of the newly completed job and all other solutions of the other witness set that have been computed thus far. Note that the order of computation of the points in a witness set intersection does no longer occur lexicographically.

3.3 quality control

The automatic control of the quality of the computed results consists roughly in the following tasks:

1. gathering diagnostics, such as residuals, estimates for condition numbers and distance between the computed roots;
2. rerunning solution paths at a finer resolution and/or eventually with extended multiprecision arithmetic;
3. computing certificates for diverging solution paths and deflating solution paths converging to multiple roots;
4. using deflation to deal with multiple solution components.

These tasks, listed in increasing order of complexity could be done off line, by another sequential or parallel program, or online by the manager node. The distribution of the path tracking jobs is relatively light compared to the path tracking jobs themselves so that we observed that in many runs the manager node remains idle for most of the time.

4 Software and Applications

4.1 extensions to PHCpack

Our implementation of a subsystem-by-subsystem solver is an extension to PHCpack [23]. Just as the other parallel homotopies described in [16], the main program and job scheduling routines are written in C, using MPI library, and call the homotopy constructors and path trackers available in PHCpack.

4.2 polynomial systems

In this preliminary phase of the implementation, we selected two families of polynomial systems whose solution sets have degrees equal to what the plain application of Bézout’s theorem predicts. For those systems, the homotopies based on the total degree are optimal in the sense that no solution path diverges.

katsura ([5], [10]) This family has its origin in a magnetism problem from physics and is widely used as a computer algebra benchmark. Each system in the family consists of n quadrics and one linear equation. For $k = 1, 2, \dots, n$, the quadrics are

$$\sum_{j=-n}^n u_{j,n} u_{k-j,n} - u_k, n = 0, \quad (1)$$

and the one linear equation is

$$\sum_{j=-n}^n u_{j,n} - 1 = 0. \quad (2)$$

For $j < 0$: we set $u_j = u_{-j}$. The solution set consists of 2^n isolated points.

Maple 11 lists `katsura5` in its help system, just type `?updates,Maple11,enhancedpackages` in a Maple 11 worksheet.

minors ([2], [8]) In an application from algebraic statistics, all adjacent minors of a general matrix are considered. Starting from a general 2-by- n matrix of variables, we obtain a system of $n - 1$ quadratic equations in $2n$ variables. The solution set has dimension $n + 1$ and degree 2^{n-1} . For $k = 1, 2, \dots, n$, the equations have the form

$$x_{1,k} x_{2,k+1} - x_{2,k} x_{1,k+1} = 0. \quad (3)$$

As this system has a positive dimensional solution set, it used to be out of reach for classical homotopy methods. The book [21] describes progress in numerical algebraic geometry.

For the purpose of parallel computing, the two families provide systems of varying size. For both families, we selected the eight dimensional versions: `katsura8` and `adjmin28`. We like to emphasize that our solver gives two qualitatively different answers: either a collection of isolated points for `katsura8`, or a positive dimensional solution set for `adjmin28`.

4.3 computational results

Timings for solving `katsura8` on a 2.4Ghz Rocketcalc personal cluster are summarized in Table 2. With 2 nodes ($p = 2$), we have one manager and one worker. This first time is used as the equivalent “sequential time”. As we double the number of workers $p = 2 \rightarrow 3 \rightarrow 5 \rightarrow 9$, the execution times decrease as $459s \rightarrow 277s \rightarrow 140s \rightarrow 85s$.

Timings may fluctuate between runs, because different random numbers give different constants in the homotopies and also paths of varying length. A constant trend is the concentration of the time in the later phases, see Table 3. Because there is only one homotopy in that last stage, dynamic load balancing in the last stage, outweighs performance losses due to synchronizing the stages.

Timings to solve `adjmin28` are summarized in Table 4. Doubling the number of workers, for p from 2 to 3, to 5, and then to 9, leads to the following drop in execution time: $2,179s \rightarrow 1,255s \rightarrow 619s \rightarrow 363s$. Doubling the number of workers, the execution time drops roughly in half.

p	time	max	min
2	459s	1,408	1,408
3	277s	787	621
4	175s	514	391
5	140s	375	289
6	104s	307	240
7	98s	251	207
8	86s	218	173
9	85s	193	147
10	81s	167	132
11	72s	152	124
12	68s	147	110

Table 2. Solving `katsura8` with p nodes. We list the total wall time (in seconds) and the maximal and minimal number of paths tracked by the workers in the dynamical load balancing scheme.

p	time	max	min
2	2,179s	1,408	1,408
3	1,255s	771	637
4	887s	497	423
5	619s	375	310
6	477s	299	247
7	417s	244	218
8	392s	223	180
9	363s	194	160
10	359s	181	143
11	331s	151	128
12	320s	135	119

Table 4. Solving `admin28` with p nodes. We list the total wall time (in seconds) and the maximal and minimal number of paths tracked by the workers in the dynamical load balancing scheme.

5 Conclusions and Discussion

Early experimental computational results show that the parallel diagonal homotopy allows jumpstarting for efficient memory management. Dynamic load balancing leads to an acceptable speedup. The overhead preventing an optimal speedup is due to the synchronization at every stage.

Choosing systems for which we have optimal homotopies we could concentrate on the scheduling issues and defer the treatment of solution paths diverging to singularities or to infinity. While MPI is sufficient to implement a dynamic load balancer in a manager/worker protocol, treating singularities requires different and computationally intensive algorithms, like deflation [15].

In view of increased availability of multi-processor and multi-core machines, we plan to use sockets in a client/server protocol. The server (called it `Job`) maintains the job queues. Clients request homotopies and start solutions from `Job` and report back to `Job` when finished. Other type of clients contact `Job` to do the quality control.

stage	time
1	10.3s
2	32.1s
3	462.0s

Table 3. The breakup of the timings for solving `katsura8` on 2 nodes (one manager and one worker), along the three stages. The last stage is the most time consuming.

With MPI on few nodes, the manager node is often idle, whereas on many nodes, the manager might become overloaded. In client/server computing, the operating system becomes co-responsible for the load balancing and allows user interaction. This would complement PHClab [6] for which we used the MPI Toolbox for Octave [4] and lead to a better interactive parallel computation.

References

- [1] D.J. Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler. Bertini: Software for numerical algebraic geometry. Available at <http://www.nd.edu/~sommese/bertini>.
- [2] P. Diaconis, D. Eisenbud, and B. Sturmfels. Lattice walks and primary decomposition. *Progress in Mathematics*, 161:173–193, 1998.
- [3] C. Durvye and G. Lecerf. A concise proof of the Kronecker polynomial system solver from scratch. Manuscript, June 2006.
- [4] J. Fernández, M. Anguita, E. Ros, and J.L. Bernier. SCE Toolboxes for the development of high-level parallel applications. In *Proceedings of ICCS 2006*, volume 3992 of *Lecture Notes in Computer Science*, pages 518–525. Springer-Verlag, 2006.
- [5] L. Gonzalez-Vega, F. Rouillier, and M.-F. Roy. Chapter 2. Symbolic recipes for polynomial system solving. In *Some Tapas of Computer Algebra*, volume 4 of *Algorithms and Computation in Mathematics*, pages 34–65. Springer-Verlag, 1999.

- [6] Y. Guan and J. Verschelde. PHClab: A MATLAB/Octave interface to PHCpack. To appear in IMA Volume 148: Software for Algebraic Geometry, Springer-Verlag, 2008.
- [7] T. Gunji, S. Kim, K. Fujisawa, and M. Kojima. PHoMpara – parallel implementation of the Polyhedral Homotopy continuation Method for polynomial systems. *Computing*, 77(4):387–411, 2006.
- [8] S. Hosten and J. Shapiro. Primary decomposition of lattice basis ideals. *Journal of Symbolic Computation*, 29(4 and 5):625–639, 2000.
- [9] B. Huber, F. Sottile, and B. Sturmfels. Numerical Schubert calculus. *J. Symbolic Computation*, 26(6):767–788, 1998.
- [10] S. Katsura. Spin glass problem by the method of integral equation of the effective field. In M.D. Coutinho-Filho and S.M. Resende, editors, *New Trends in Magnetism*, pages 110–121. World Scientific, 1990.
- [11] G. Lecerf. Computing the equidimensional decomposition of an algebraic closed set by means of lifting fibers. *J. Complexity*, 19(4):564–596, 2003.
- [12] T.L. Lee, T.Y. Li, and C.H. Tsai. HOM4PS-2.0, a software package for solving polynomial systems by the polyhedral homotopy continuation method. Preprint and software are available via <http://www.math.msu.edu/~li>.
- [13] A. Leykin and J. Verschelde. Decomposing solution sets of polynomial systems: a new parallel monodromy breakup algorithm. Accepted for publication in *The International Journal of Computational Science and Engineering* (special issue dedicated to HPSEC’05).
- [14] A. Leykin and J. Verschelde. Factoring solution sets of polynomial systems in parallel. In Tor Skeie and Chu-Sing Yang, editors, *Proceedings of the 2005 International Conference on Parallel Processing Workshops, 14-17 June 2005, Oslo, Norway. High Performance Scientific and Engineering Computing*, pages 173–180. IEEE Computer Society, 2005.
- [15] A. Leykin, J. Verschelde, and A. Zhao. Newton’s method with deflation for isolated singularities of polynomial systems. *Theoretical Computer Science*, 359(1-3):111–122, 2006.
- [16] A. Leykin, J. Verschelde, and Y. Zhuang. Parallel homotopy algorithms to solve polynomial systems. In N. Takayama and A. Iglesias, editors, *Proceedings of ICMS 2006*, volume 4151 of *Lecture Notes in Computer Science*, pages 225–234. Springer-Verlag, 2006.
- [17] T.Y. 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.
- [18] A.J. Sommese, J. Verschelde, and C.W. Wampler. Homotopies for intersecting solution components of polynomial systems. *SIAM J. Numer. Anal.*, 42(4):552–1571, 2004.
- [19] A.J. Sommese, J. Verschelde, and C.W. Wampler. An intrinsic homotopy for intersecting algebraic varieties. *J. Complexity*, 21(4):593–608, 2005. Festschrift for the 70th Birthday of Arnold Schönhage, edited by T. Lickteig and L.M. Pardo.
- [20] A.J. Sommese, J. Verschelde, and C.W. Wampler. Solving polynomial systems equation by equation. In *Algorithms in Algebraic Geometry*, volume 146 of *The IMA Volumes in Mathematics and Its Applications*, pages 133–152. Springer-Verlag, 2008.
- [21] A.J. Sommese and C.W. Wampler. *The Numerical solution of systems of polynomials arising in engineering and science*. World Scientific, 2005.
- [22] H.-J. Su, J.M. McCarthy, M. Sosonkina, and L.T. Watson. Algorithm 857: POLSYS_GLP: A parallel general linear product homotopy code for solving polynomial systems of equations. *ACM Trans. Math. Softw.*, 32(4):561–579, 2006.
- [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.
- [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 Scientific and Engineering Computing*, pages 222–229. IEEE Computer Society, 2004.
- [25] J. Verschelde and Y. Zhuang. Parallel implementation of the polyhedral homotopy method. In Timothy Mark Pinkston and Fusun Ozguner, editors, *Proceedings of the 2006 International Conference on Parallel Processing Workshops, 14-18 August 2006, Columbus, Ohio. High Performance Scientific and Engineering Computing*, pages 481–488. IEEE Computer Society, 2006.