Parallel Implementation of the Polyhedral Homotopy Method[*]

Jan Verschelde[†] Yan Zhuang[‡]

Abstract

Homotopy methods to solve polynomial systems are well suited for parallel computing because the solution paths defined by the homotopy can be tracked independently. For sparse polynomial systems, polyhedral methods give efficient homotopy algorithms. The polyhedral homotopy methods run in three stages: (1) compute the mixed volume; (2) solve a random coefficient start system; (3) track solution paths to solve the target system. This paper is about how to parallelize the second stage in PHCpack. We use a static workload distribution algorithm and achieve a good speedup on the cyclic n-roots benchmark systems. Dynamic workload balancing leads to reduced wall times on large polynomial systems which arise in mechanism design.

2000 Mathematics Subject Classification. *Primary 65H10. Secondary 14Q99, 68W30.*

Key words and phrases. *Continuation methods, load balancing, parallel computation, path following, polynomial systems, polyhedral homotopies.*

1 Introduction

Polynomial systems occur in a wide variety of application areas in science and engineering, see the case studies in [34, Chapter 9]. Homotopy continuation methods to solve polynomial systems (recently surveyed in [27] and [34]) are well suited for parallel implementation, as described in [1], [6, 7], [18], [31], and [32].

[*]Date: 26 May 2006.

[†]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.

[‡]Department of Mathematics, Statistics, and Computer Science, University of Illinois at Chicago, 851 South Morgan (M/C 249), Chicago, IL 60607-7045, USA. *Email:* yzhuan1@math.uic.edu. *URL:* http://www2.uic.edu/˜yzhuan1.

Almost all polynomial systems occurring in applications are *sparse*, i.e.: only relatively few monomials appear with nonzero coefficients. For sparse systems, homotopies based on the degrees typically track many paths diverging to infinity. For example, in one benchmark system (from mechanism design [36] [39]), a degree-based homotopy used in [38] by the POLSYS_GLP extension [37] of HOMPACK [44] [45] [46] leads to 9,216 paths, whereas polyhedral homotopies track the optimal number of 1,024 paths. The 1,024 for this system is the *mixed volume* (which we will define precisely in the next section). We view the mixed volume as the number of isolated solutions of a system with randomly chosen complex coefficients.

To solve a polynomial system using polyhedral homotopies, we distinguish three stages. First we compute the mixed volume, ignoring the particular coefficients of the polynomial system. In the second stage, we apply *polyhedral homotopies* to solve a system with the same sparse structure as the given system, but with random coefficients, using the results of the first stage. The third and final stage applies a plain linear homotopy to solve the given system using coefficient-parameter polynomial continuation [30] (related to the cheater's homotopy [29]).

In this paper, we primarily focus on the second stage, and consider a mixed-cell configuration as given. These mixed-cell configurations may be computed using PHCpack [40], or by MixedVol [14], or PHoM [17]. A parallel implementation of PHoM is described in [16]. While the third stage involves as many paths as the second stage, the computational cost can increase significantly because the polynomial system at the end of the homotopy is no longer generic. Conditions on genericity are given in [33].

This paper continues the development of parallel implementations of homotopy algorithms in PHCpack [40] – started in [43] with Pieri homotopies, followed by parallel decomposition methods in [24, 25]. Our first parallel implementation of polyhedral homotopies uses a static workload distribution. This static distribution (as we experienced in [43]) is favorable when all solution paths require the same computational cost, as could be expected from polyhedral homotopies solving generic polynomial systems. However, dynamic load balancing significantly improves the wall time for large mechanisms design problems.

Algorithm 1.1 Polyhedral homotopies to solve a generic system.

```
Input: $\triangle_\omega$, $G(\mathbf{x}) = \mathbf{0}$.                     mixed-cell configuration and generic system
Output: $G^{-1}(\mathbf{0})$.                                           all solutions to $G(\mathbf{x}) = \mathbf{0}$
for all $C \in \triangle_\omega$ do                                     enumerate all $C$ with inner normals $\mathbf{v}$
    create a polyhedral homotopy $\widehat{G}(\mathbf{y}, s) = \mathbf{0}$;            apply coordinate transformation using $\mathbf{v}$
    solve the start system $\widehat{G}|_C(\mathbf{y}, 0) = \mathbf{0}$;            $\widehat{G}|_C(\mathbf{y}, 0) = \mathbf{0}$ has $\mathrm{Vol}(C)$ solutions
    track $\mathbf{y}(s)$: $\widehat{G}(\mathbf{y}(s), s) = \mathbf{0}$, for $s$ from 0 to 1;   track as many solution paths as $\mathrm{Vol}(C)$
end for.
```

2 Polyhedral Homotopies

A sparse polynomial f in n variables $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is written as $f(\mathbf{x}) = \sum_{\mathbf{a} \in A} c_{\mathbf{a}} \mathbf{x}^{\mathbf{a}}$, with nonzero coefficients $c_{\mathbf{a}} \in \mathbb{C}^*$, $\mathbb{C}^* = \mathbb{C} \setminus \{0\}$, and $\mathbf{x}^{\mathbf{a}} = x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}$. The set A is called the *support* of f. We denote the volume of its convex hull by $\mathrm{Vol}(A)$. Kushnirenko showed in [23] that a system $F(\mathbf{x}) = \mathbf{0}$ whose polynomials all share the same support A can have no more than $\mathrm{Vol}(A)$ isolated solutions in $(\mathbb{C}^*)^n$. Moreover, if the coefficients $c_{\mathbf{a}}$ are generic, then $\mathrm{Vol}(A)$ is an exact root count. Kushnirenko's theorem was generalized to general polynomial systems by Bernshteǐn [2] and Khovanskiǐ [21].

Polyhedral homotopies (proposed in [20] and [42]) provide efficient algorithms to solve systems with random coefficients, denoted by $G(\mathbf{x}) = \mathbf{0}$. To solve a given system $F(\mathbf{x}) = \mathbf{0}$, we then follow the solution paths defined by the linear homotopy $H(\mathbf{x}, t) = (1-t)G(\mathbf{x}) + tF(\mathbf{x}) = \mathbf{0}$, for t varying between 0 and 1. As shown in [30] (see also [29]) all isolated solutions of $F(\mathbf{x}) = \mathbf{0}$ lie at the end of some solution path originating at a solution of $G(\mathbf{x}) = \mathbf{0}$. Polyhedral homotopies are induced by *lifting functions* $\omega : A \rightarrow \mathbb{R} : \mathbf{a} \mapsto \omega(\mathbf{a})$ applied to polynomials g of G in the above notation as $\widehat{g}(\mathbf{x}, t) = \sum_{\mathbf{a} \in A} c_{\mathbf{a}} \mathbf{x}^{\mathbf{a}} t^{\omega(\mathbf{a})}$. Obviously, for $t = 1$, we have a polynomial, but at $t = 0$, we need coordinate transformations to start a polyhedral homotopy.

The right coordinate transformations are obtained as follows. A *regular triangulation* $\triangle_\omega$ is a triangulation — i.e.: a collection of nonoverlapping simplices so that $\mathrm{Vol}(A) = \sum_{C \in \triangle_\omega} \mathrm{Vol}(C)$ — where every cell $C \in \triangle_\omega$ corresponds to a facet $\widehat{C}$ of the lower hull of the lifted support $\widehat{A} = \omega(A) = \{\ \widehat{\mathbf{a}} = (\mathbf{a}, \omega(\mathbf{a})) \mid \mathbf{a} \in A\ \}$. Every facet $\widehat{C}$ is characterized by an *inner normal* $\mathbf{v} \in \mathbb{R}^{n+1}$, $v_{n+1} > 0$, satisfying equalities $\langle \widehat{\mathbf{c}}_i, \mathbf{v} \rangle = \langle \widehat{\mathbf{c}}_j, \mathbf{v} \rangle$, for $\widehat{\mathbf{c}}_i, \widehat{\mathbf{c}}_j \in \widehat{C}$ and inequalities $\langle \widehat{\mathbf{a}}, \mathbf{v} \rangle > \langle \widehat{\mathbf{c}}, \mathbf{v} \rangle$, for $\widehat{\mathbf{c}} \in \widehat{C}$ and $\widehat{\mathbf{a}} \in \widehat{A} \backslash \widehat{C}$, where $\langle \cdot, \cdot \rangle$ is the usual inner product. Because the inner normal $\mathbf{v}$ satisfies these equalities and inequalities, the coordinate transformation replacing x_i by $y_i s^{v_i}$ and t by $s^{v_{n+1}}$ applied to $\widehat{g}(\mathbf{x}, t)$ results in the homotopy composed of polynomials $\widehat{g}(\mathbf{y}, s) = \widehat{g}|_C(\mathbf{y}, s) + \widehat{g}|_{A \backslash C}(\mathbf{y}, s)$, with $\widehat{g}|_C(\mathbf{y}, s) = \sum_{\mathbf{a} \in C} \mathbf{y}^{\mathbf{a}} s^{\langle \mathbf{a}, \mathbf{v} \rangle}$ and $\widehat{g}|_{A \backslash C}(\mathbf{y}, s) = \sum_{\mathbf{a} \in A \backslash C} \mathbf{y}^{\mathbf{a}} s^{\langle \mathbf{a}, \mathbf{v} \rangle}$. In particular: the equalities on $\mathbf{v}$ ensure that the power of s in $\widehat{g}|_C$ is the same, say p, and the inequalities on $\mathbf{v}$ ensure that all powers of s in $\widehat{g}|_{A \backslash C}$ are strictly larger than p. Therefore, we have a homotopy in s, starting at $s = 0$ at a system so sparse it is trivial to solve, and ending at $s = 1$ at the random coefficient system $G(\mathbf{x}) = \mathbf{0}$.

To solve systems $F(\mathbf{x}) = \mathbf{0}$, $F = (f_1, f_2, \ldots, f_n)$ with different supports $\mathcal{A} = (A_1, A_2, \ldots, A_n)$, A_i is the support of f_i, we use regular mixed-cell configurations. The *Cayley embedding* places the sets A_i of $\mathcal{A}$ on the vertices of the $(n-1)$-dimensional standard basis simplex S. In particular: $\mathcal{E}(\mathcal{A}) = \cup_{i=1}^n \mathcal{E}_i(A_i)$, where $\mathcal{E}_i(A) = \{\ (\mathbf{a}, \mathbf{e}_i) \mid \mathbf{a} \in A\ \}$, with $\mathbf{e}_i$ the ith vertex of S ($\mathbf{e}_1$ is $\mathbf{0}$ and $\mathbf{e}_i$ is also zero except for 1 at position $i-1$). Cells in a regular triangulation of $\mathcal{E}(\mathcal{A})$ which are spanned by exactly two points of every support A_i are called *mixed*. Extending our notation of $\triangle_\omega$ to denote the collection of mixed cells of any regular triangulation of $\mathcal{E}(\mathcal{A})$, we then call $\triangle_\omega$ a *regular mixed-cell configuration*, generalizing the volume to the *mixed volume* $\mathrm{Vol}(\mathcal{A}) = \sum_{C \in \triangle_\omega} \mathrm{Vol}(C)$. For the relation to sparse elimination theory, we refer to [9], [15], and [35], see also [19] for geometric applications. Algorithm 1.1 follows [20].

The execution of the coordinate transformation to create the polyhedral homotopy has a cost linear in n and the number of monomials in the system. As $\widehat{G}|_C(\mathbf{y}, 0) = \mathbf{0}$ has exactly $n+1$ distinct monomials, the cost of solving the start system is equivalent to solving one linear system. The computationally most intensive stage in Algorithm 1.1 is the path tracking, because tracking one solution path typically involves solving a couple of hundreds of linear systems in the Newton corrector.

In addition to leading to more solution paths, the difficulty with applying polyhedral homotopies is the height of the power of the continuation parameter, a problem first addressed in [41] by dynamic lifting, later in [13] by balancing, and by scaling in [22]. While solving large 12 dimensional systems from mechanical design (see section 5 below), we uncovered numerical instabilities when solving semi-mixed start systems. In [26] we describe a numerically stable algorithm to resolve this issue.

Algorithm 2.1 Sketch of a parallel version of polyhedral homotopies.

Input: $\triangle_\omega$, $G(\mathbf{x}) = \mathbf{0}$. *mixed-cell configuration and generic system*
Output: $G^{-1}(\mathbf{0})$. *all solutions to $G(\mathbf{x}) = \mathbf{0}$*

Manager		**Workers**	
read input file			
broadcast data	$\rightarrow$	receive data	*data = system and lifting*
distribute cells	$\rightarrow$	receive cells	*static workload distribution*
		track paths	*compute solutions*
collect solutions	$\leftarrow$	send solutions	
write to file			

3 Distribution of the Work Load

Polyhedral homotopies are applied to solve generic systems $G(\mathbf{x}) = \mathbf{0}$. The "generic" means in practice that the coefficients of G are random points on the complex unit circle. Because of this randomness, all solution paths are expected to follow the same uniform behavior, and the tracking will require the same amount of computational work. Therefore, reducing the communication overhead to a minimum, a static workload assignment seems the best choice. Using the manager-worker[1] paradigm, Algorithm 2.1 summarizes our parallel version of the polyhedral homotopies.

Because the polynomial systems we consider are sparse, the broadcast of the supports with their lifting will be dominated by the time to startup the communications. The distribution of the cells is more data intensive, but there are fewer cells than solutions, as the volume of every cell is at least one. So the initial communication cost (before the path tracking) does certainly not exceed the cost at the end to collect all solutions at the manager node. The tracking of the paths naturally overlaps the communications. A worker can start tracking paths, as soon as it has received one cell, and can send the end point of the path to the manager for writing to file. The manager node should not track any paths, so it remains available to handle the data flow, relieving the workers from needing too much memory.

We introduce the distribution of the cells with an example. Suppose there are 8 cells and 41 is the total number of paths. The manager will distribute these 41 paths evenly among three workers as the partition 41 = 14 + 14 + 13. In (1) we show how to manager assigns the cells (with respective volumes listed) and #paths to the workers.

The static workload distribution of the cells of the mixed-cell configuration $\triangle_\omega$ is formalized in Algorithm 3.1 and Algorithm 3.2, respectively executed by the manager and each worker. The data the manager sends to each worker is the triplet (C, k, l), with $C \in \triangle_\omega$ a cell, k and l are indices to the respective first and last start solutions in the polyhedral homotopy defined by the cell C.

[1]The terminology we use is less common than "master-slave", but avoids the unpleasant historical connotations.

The distribute function in Algorithm 3.1 determines the workload for each worker. In case the total number of paths is not divisible by the number of workers, workers with a lower number will receive some extra paths. This bias in the workload distribution is justified because workers with lower number receive their load earlier than the rest. Algorithm 3.2 details the actions performed by every worker. Because solving a start system involves only the solution of a linear system, each worker solves the start system and tracks only the assigned paths.

4 Dynamic workload distribution

For very sparse or generic problems, a static workload distribution seems the best choice. However, for polynomials which are not so sparse as the mechanisms design problems, dynamic load balancing is needed.

The dynamic workload distribution of the cells of the mixed-cell configuration $\triangle_\omega$ is formalized in Algorithm 4.1 and Algorithm 4.2, respectively executed by the manager and each worker. The data the manager sends to each worker is $(C, index)$, with $C \in \triangle_\omega$ a cell, $index$ is the index of the start solutions in the polyhedral homotopy defined by the cell C.

Algorithm 4.1 distributes the load according to the number of cells and number of processors. In case the number of cells is less than the number of processors, we will distribute the load path by path, otherwise we will distribute the load cell by cell except for the last cell which will be distributed path by path.

5 Computational Results

In this section we summarize our computational experiments on two typical benchmark applications. The first class of polynomial systems is a frequently used and widely known problem in computational algebraic geometry, the so-called cyclic n-roots problem. Our second example comes from mechanisms design.

manager	worker 1	worker 2	worker 3
Vol(cell 1) $= 5$	#paths(cell 1) : 5		
Vol(cell 2) $= 4$	#paths(cell 2) : 4		
Vol(cell 3) $= 4$	#paths(cell 3) : 4		
Vol(cell 4) $= 6$	#paths(cell 4) : 1	#paths(cell 4) : 5	
Vol(cell 5) $= 7$		#paths(cell 5) : 7	
Vol(cell 6) $= 3$		#paths(cell 6) : 2	#paths(cell 6) : 1
Vol(cell 7) $= 4$			#paths(cell 7) : 4
Vol(cell 8) $= 8$			#paths(cell 8) : 8
total #paths : 41	#paths to track : 14	#paths to track : 14	#paths to track : 13

(1)

5.1 Hardware and Software

We developed and tested our software using two Rocketcalc clusters, with four and eight processors respectively. Combined with the dual processor server machine, our configuration has a total of fourteen 2.4Ghz processors, connected by a 100Mbit Ethernet network.

We also ran our software on UIC's "argo", a cluster of sixty-eight PCs: including two master machines, sixty-four compute machines, and two file servers. The nodes are heterogeneous, some processors run at 2.8Ghz, others at 3.0Ghz. Compared to our Rocketcalc clusters, the faster network connections of argo led to an improved speedup.

Our software is an extension of PHCpack [40]. Our main programs are written in C, using the MPI library, and call the path tracking and homotopy facilities in PHCpack.

5.2 Experimental Results on the cyclic n-roots problem

The cyclic n-roots problem defines a family of polynomial systems widely used for benchmarking. The systems occur in Fourier analysis [3]. Progress on these systems is reported in [4], [5], [8], [10], [11], [12], and [28].

In Table 1 we list the computational results for our cluster configuration, for increasing values of the problem size n. As is typical for polynomial systems solving the number of solutions grows exponentially in the number of variables n in the system. Even for our relatively small cluster configuration, larger problems (in the range of several hundreds of thousands of solutions) are well within reach.

Table 2 shows the speedup on the cyclic 7-roots problem for an increasing number of workers. Because the manager does no path tracking, the time spent when using one worker corresponds to the sequential time. Moreover, we observe a very good distribution of the workload, as the difference in time between the workers is minor. With 13 workers we obtain an almost tenfold speedup, which is acceptable. For dynamic workload distribution we get 11.3 speedup which is better than static. The dynamic workload balancing algorithm on argo achieves a close to optimal speedup.

Problem	#Paths	CPU Time
cyclic 5-roots	70	0.13m
cyclic 6-roots	156	0.19m
cyclic 7-roots	924	0.30m
cyclic 8-roots	2,560	0.78m
cyclic 9-roots	11,016	3.64m
cyclic 10-roots	35,940	21.33m
cyclic 11-roots	184,756	2h 39m
cyclic 12-roots	500,352	24h 36m

Table 1. Wall time for start systems to solve the cyclic n-roots problems, using 13 workers, with static load distribution.

5.3 Mechanism Design: the Computation of Reachable Surfaces

One must solve polynomial systems to design a robot with five degrees of freedom, forming an RPS (links connected by Revolute, Prismatic, and Spherical joints) serial chain, see [36] and [39]. In [38] and [37] using results of [46], the POLSYS_GLP extension of HOMPACK [44] (see also [45] and [46]) used a linear-product start systems to solve these type of systems.

These systems are more challenging to polyhedral homotopies than the cyclic n-roots problems, because they are not so sparse. For example, nine equations of the first example we considered (equation (28) of [38]) have 201 monomials, of which 102 are vertices and belong to the random coefficient start system. This makes the evaluation of the polynomials more expensive. The mixed volume of this system equals 125,888, which is about half of the bound 247,968 based on the degrees of the system, used in [38]. The semi-mixed nature of the system (i.e.: nine equations share the same support) is exploited by the dynamic lifting algorithm of [41]. While the computation of the mixed volume is not the main cost, we also ran the program MixedVol [14] to confirm this number.

	Static versus Dynamic on our cluster				Dynamic on argo	
#workers	Static	Speedup	Dynamic	Speedup	Dynamic	Speedup
1	50.7021	–	53.0707	–	29.2389	–
2	24.5172	2.1	25.3852	2.1	15.5455	1.9
3	18.3850	2.8	17.6367	3.0	10.8063	2.7
4	14.6994	3.4	12.4157	4.2	7.9660	3.7
5	11.6913	4.3	10.3054	5.1	6.2054	4.7
6	10.3779	4.9	9.3411	5.7	5.0996	5.7
7	9.6877	5.2	8.4180	6.3	4.2603	6.9
8	7.8157	6.5	7.4337	7.1	3.8528	7.6
9	7.5133	6.8	6.8029	7.8	3.6010	8.1
10	6.9154	7.3	5.7883	9.2	3.2075	9.1
11	6.5668	7.7	5.3014	10.0	2.8427	10.3
12	6.4407	7.9	4.8232	11.0	2.5873	11.3
13	5.1462	9.8	4.6894	11.3	2.3224	12.6

Table 2. Wall time in seconds for an increasing number of workers to solve a start system for the cyclic 7-roots problem. First we list the wall time and speedup for the static and dynamic distribution on our cluster. The last 2 columns list wall time and speedup for the dynamic distribution on argo.

The first parallel implementation using static workload distribution finished the tracking of 125,888 solution paths (9,683 to 9,684 paths per processor) with a reported wall time of 57091.30 seconds using 13 processors within 16 hours (basically "overnight"). The dynamic workload balancing algorithm improves on this time by more than four hours, see Table 3 for computational results of this and other examples of this class.

References

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

[2] D. Bernshteǐn. The number of roots of a system of equations. *Functional Anal. Appl.*, 9(3):183–185, 1975. Translated from *Funktsional. Anal. i Prilozhen.*, 9(3):1–4,1975.

[3] G. Björck. Functions of modulus one on Z_n whose Fourier transforms have constant modulus, and "cyclic n-roots". In J. Byrnes and J. Byrnes, editors, *Recent Advances in Fourier Analysis and its Applications*, volume 315 of *NATO Adv. Sci. Inst. Ser. C: Math. Phys. Sci.*, pages 131–140. Kluwer, 1989.

[4] G. Björck and R. Fröberg. A faster way to count the solutions of inhomogeneous systems of algebraic equations, with applications to cyclic n-roots. *J. Symbolic Computation*, 12(3):329–336, 1991.

[5] G. Björck and R. Fröberg. Methods to "divide out" certain solutions from systems of algebraic equations, applied to find all cyclic 8-roots. In M. Gyllenberg and L. Persson, editors, *Analysis, Algebra and Computers in Math. research*, volume 564 of *Lecture Notes in Mathematics*, pages 57–70. Dekker, 1994.

[6] A. Chakraborty, D.C.S. Allison, C.J. Ribbens, and L.T. Watson. Note on unit tangent vector computation for homotopy curve tracking on a hypercube. *Parallel Computing*, 17(12):1385–1395, 1991.

[7] A. Chakraborty, D.C.S. Allison, C.J. Ribbens, and L.T. 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.

[8] Y. Dai, S. Kim, and M. Kojima. Computing all nonsingular solutions of cyclic-n polynomial using polyhedral homotopy continuation methods. *J. Comput. Appl. Math.*, 152(1-2):83–97, 2003.

[9] I.Z. Emiris and J.F. Canny. Efficient incremental algorithms for the sparse resultant and the mixed volume. *J. Symbolic Computation*, 20(2):117–149, 1995.

[10] J.C. Faugère. A new efficient algorithm for computing Gröbner bases (F_4). *Journal of Pure and Applied Algebra*, 139(1-3):61–88, 1999. Proceedings of MEGA'98, 22–27 June 1998, Saint-Malo, France.

[11] T. Gao and T.Y. Li. Mixed volume computation via linear programming. *Taiwan J. of Math.*, 4:599–619, 2000.

[12] T. Gao and T.Y. Li. Mixed volume computation for semi-mixed systems. *Discrete Comput. Geom.*, 29(2):257–277, 2003.

[13] T. Gao, T.Y. Li, J. Verschelde, and M. Wu. Balancing the lifting values to improve the numerical stability of polyhedral homotopy continuation methods. *Appl. Math. Comput.*, 114:233–247, 2000.

[14] T. Gao, T.Y. Li, and M. Wu. Algorithm 846: MixedVol: A software package for mixed volume computation. *ACM Trans. Math. Softw.*, 31(4):555–560, 2005.

[15] I.M. Gel'fand, M.M. Kapranov, and A.V. Zelevinsky. *Discriminants, Resultants and Multidimensional Determinants*. Birkhäuser, Boston, 1994.

[16] T. Gunji, S. Kim, K. Fujisawa, and M. Kojima. PHoMpara – parallel implementation of the Polyhedral Homotopy continuation Method for polynomial systems. To appear in Computing.

[17] T. Gunji, S. Kim, M. Kojima, A. Takeda, K. Fujisawa, and T. Mizutani. PHoM – a polyhedral homotopy continuation method for polynomial systems. *Computing*, 73(4):55–77, 2004.

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

[19] B. Huber, J. Rambau, and F. Santos. The Cayley trick, lifting subdivisions and the Bohne-Dress theorem on zonotopal tilings. *J. Eur. Math. Soc.*, 2(2):179–198, 2000.

[20] B. Huber and B. Sturmfels. A polyhedral method for solving sparse polynomial systems. *Math. Comp.*, 64(212):1541–1555, 1995.

[21] A. Khovanskiĭ. Newton polyhedra and the genus of complete intersections. *Functional Anal. Appl.*, 12(1):38–46, 1978. Translated from *Funktsional. Anal. i Prilozhen.*, 12(1),51–61,1978.

[22] S. Kim and M. Kojima. Numerical stability of path tracing in polynomial homotopy continuation methods. *Computing*, 73:329–348, 2004.

[23] A. Kushnirenko. Newton Polytopes and the Bézout Theorem. *Functional Anal. Appl.*, 10(3):233–235, 1976. Translated from *Funktsional. Anal. i Prilozhen.*, 10(3),82–83,1976.

[24] 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).

[25] A. Leykin and J. Verschelde. Factoring solution sets of polynomial systems in parallel. In T. Skeie and C.-S. 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.

[26] A. Leykin, J. Verschelde, and Y. Zhuang. Parallel homotopy algorithms to solve polynomial systems. Submitted for publication.

[27] 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.

[28] T.Y. Li and X. Li. Finding mixed cells in the mixed volume computation. *Found. Comput. Math.*, 1(2):161–181, 2001.

[29] T.Y. Li, T. Sauer, and J. Yorke. The cheater's homotopy: an efficient procedure for solving systems of polynomial equations. *SIAM J. Numer. Anal.*, 26(5):1241–1251, 1989.

[30] A.P. Morgan and A.J. Sommese. Coefficient-parameter polynomial continuation. *Appl. Math. Comput.*, 29(2):123–160, 1989. Errata: *Appl. Math. Comput.* 51:207(1992).

[31] A.P. Morgan and L.T. Watson. A globally convergent parallel algorithm for zeros of polynomial systems. *Nonlinear Analysis*, 13(11):1339–1350, 1989.

[32] W. Pelz and L.T. Watson. Message length effects for solving polynomial systems on a hypercube. *Parallel Computing*, 10(2):161–176, 1989.

[33] J.M. Rojas. Toric intersection theory for affine root counting. *Journal of Pure and Applied Algebra*, 136(1):67–100, 1999.

[34] A.J. Sommese and C.W. Wampler II. *The Numerical solution of systems of polynomials arising in engineering and science*. World Scientific, 2005.

[35] B. Sturmfels. On the Newton polytope of the resultant. *Journal of Algebraic Combinatorics*, 3(2):207–236, 1994.

[36] H.-J. Su and J.M. McCarthy. Kinematic synthesis of RPS serial chains. In the Proceedings of the ASME Design Engineering Technical Conferences (CDROM), Chicago, IL, Sep 2-6, 2003.

[37] H.-J. Su, J.M. McCarthy, M. Sosonkina, and L.T. Watson. Algorithm 8xx: POLSYS_GLP: A parallel general linear product homotopy code for solving polynomial systems of equations. To appear in the Algorithms section of *ACM Trans. Math. Softw*.

[38] H.-J. Su, J.M. McCarthy, and L.T. Watson. Generalized linear product homotopy algorithms and the computation of reachable surfaces. *ASME Journal of Information and Computer Sciences in Engineering*, 4(3):226–234, 2004.

[39] H.-J. Su, C.W. Wampler, and J.M. McCarthy. Geometric design of cylindric PRS serial chains. *ASME Journal of Mechanical Design*, 126(2):269–277, 2004.

[40] 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`.

[41] J. Verschelde, K. Gatermann, and R. Cools. Mixed-volume computation by dynamic lifting applied to polynomial system solving. *Discrete Comput. Geom.*, 16(1):69–112, 1996.

[42] J. Verschelde, P. Verlinden, and R. Cools. Homotopies exploiting Newton polytopes for solving sparse polynomial systems. *SIAM J. Numer. Anal.*, 31(3):915–930, 1994.

[43] 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.

[44] L.T. Watson, S.C. Billups, and A.P. Morgan. Algorithm 652: HOMPACK: a suite of codes for globally convergent homotopy algorithms. *ACM Trans. Math. Softw.*, 13(3):281–310, 1987.

[45] L.T. Watson, M. Sosonkina, R.C. Melville, A.P. Morgan, and H.F. Walker. HOMPACK90: A suite of Fortran 90 codes for globally convergent homotopy algorithms. *ACM Trans. Math. Softw.*, 23(4):514–549, 1997.

[46] S.M. Wise, A.J. Sommese, and L.T. Watson. Algorithm 801: POLSYS_PLP: a partitioned linear product homotopy code for solving polynomial systems of equations. *ACM Trans. Math. Softw.*, 26(1):176–200, 2000.

Algorithm 3.1 Static distribution of cells executed by the manager.

```
Input: $\triangle_\omega$, $V$, $p$.                          mixed-cell configuration, volume, #processors
Output: $G^{-1}(\mathbf{0})$.                                  all solutions to $G(\mathbf{x}) = \mathbf{0}$
for $i$ from 1 to $p-1$ do                                     the manager is processor 0
    load[$i$] := distribute($V, p, i$);                        #paths for $i$-th worker
    send load[$i$] to $i$;                                     message to $i$-th worker
end for;                                                       $\lfloor V/(p-1) \rfloor \leq$ load[$i$] $\leq \lceil V/(p-1) \rceil$
$i := 1$;                                                      start at worker 1
for each $C \in \triangle_\omega$ do
    #paths := $\mathrm{Vol}(C)$;                               $C$ defines $\mathrm{Vol}(C)$ many paths
    $j := 1$;                                                  first index of start solution
    while #paths $> 0$ do
        if #paths $\leq$ load[$i$] then                        $i$-th worker can handle $C$
            send $(C, j, j+$#paths$-1)$ to $i$;                message to $i$-th worker
            load[$i$] := load[$i$] $-$ #paths;                 reduce load for $i$-th worker
            #paths := $0$;                                     done with cell $C$
        else                                                   too many paths for $i$-th worker
            send $(C, j, j+$load[$i$]$-1)$ to $i$;             message to $i$-th worker
            $j := j$ + load[$i$];                              update index of start solution
            #paths := #paths $-$ load[$i$];                    reduce #paths to track
            load[$i$] := $0$;                                  finished with $i$-th worker
            $i := i + 1$;                                      move to the next worker
        end if;
    end while;
end for.
```

Algorithm 3.2 Static distribution of cells executed by all the workers.

```
Input: $G(\mathbf{x}) = \mathbf{0}$.                           generic system
Output: a subset of $G^{-1}(\mathbf{0})$.                      as many solutions as load of the worker
count := $0$;                                                  initialize the counter
receive load from manager;                                     #paths the worker has to track
while count $\leq$ load do                                     as long as not finished
    receive $(C, k, l)$;                                       receive cell and indices to start solutions
    create polyhedral homotopy $\widehat{G}_C(\mathbf{y}, s) = \mathbf{0}$;   perform coordinate transformation
    solve the start system $\widehat{G}_C(\mathbf{y}, 0) = \mathbf{0}$;       one linear system to solve
    track paths from solutions $k$ to $l$;                     track $l - k + 1$ paths
    send solutions to manager;                                 message to manager reporting results
    count := count + $l - k + 1$;                              update the counter
end while.
```

Surface	Bounds on #Solutions			dynamic load distribution	
	Total degree	LPD bound	Mixvol	our cluster	time on argo
elliptic cylinder	2,097,152	247,968	125,888	11h 33m	6h 12m
circular torus	2,097,152	868,352	474,112	7h 17m	4h 3m
general torus	4,194,304	448,702	226,512	14h 15m	6h 36m

Table 3. Wall time for mechanism design problems on our cluster and argo. Compared to the LPD bound of [38], polyhedral homotopies cut the #paths about in half. The second example is easier (despite the larger #paths) because of increased sparsity, and thus lower evaluation cost.

Algorithm 4.1 Dynamic distribution of cells executed by the manager.

```
**Input:** $\triangle_\omega, V, p$.                          *mixed-cell configuration, volume, #processors*
**Output:** $G^{-1}(\mathbf{0})$.                                           *all solutions to $G(\mathbf{x}) = \mathbf{0}$*
if $\#\triangle_\omega \leq p$ then                                                 *distribute path by path*
    for $i$ from $1$ to $p-1$ do                                          *the manager is processor 0*
        compute $(C, index)$;                                  *determine which path of which cell*
        send $(C, index)$ to $i$;                                          *message to $i$-th worker*
    end for;
    for $k$ from $p$ to $V + p - 1$ do                                   *distribute the rest of the paths*
        compute $(C, index)$ from $k$;                         *determine which path of which cell*
        receive($i$,requestTag);                                         *$i$-th worker requests a job*
        send $k$ to $i$;                                    *$i$-th worker can expect a job or terminate*
        if $k \leq V$ then
            send $(C, index)$ to $i$;                                         *send job to $i$-th worker*
        end if;
    end for;
else                                                                          *distribute cell by cell*
    for $i$ from $1$ to $p-1$ do                                          *the manager is processor 0*
        send $(C_i, -1)$ to $i$;                                           *message to $i$-th worker*
    end for;
    for $k$ from $p$ to $\#\triangle_\omega - 1 + V_l + p - 1$ do                 *$V_l$ is #paths of the last cell*
        receive($i$,requestTag);                                         *$i$-th worker requests a job*
        if $k > \#\triangle_\omega + V_l$ then                                          *all cells processed*
            $k := V + 1$;                                                      *$k$ signals termination*
        end if;
        send $k$ to $i$;                                        *$i$-th worker terminates according to k*
        if $k \leq \#\triangle_\omega$ then
            send $(C_k, -1)$ to $i$;                                      *send $i$-th worker a new cell*
        else if $k \leq \#\triangle_\omega + V_l$ then
            send $(C_l, k - \#\triangle_\omega + 1)$ to $i$;                     *send $i$-th worker a new path*
        end if;
    end for;
end if.
```

Algorithm 4.2 Dynamic distribution of cells executed by all the workers.

```
**Input:** $G(\mathbf{x}) = \mathbf{0}$, $V$.                                          *generic system, total #paths*
**Output:** a subset of $G^{-1}(\mathbf{0})$.                          *as many solutions as load of the worker*
do
    receive $k$;                                                            *terminate according to $k$*
    if $k > V$ then exit loop; end if;                                                    *terminate loop*
    receive $(C, index)$;                                 *receive cell and indices to start solutions*
    create polyhedral homotopy $\widehat{G}_C(\mathbf{y}, s) = \mathbf{0}$;            *perform coordinate transformation*
    solve the start system $\widehat{G}_C(\mathbf{y}, 0) = \mathbf{0}$;                      *one linear system to solve*
    track paths according to $index$;                      *track $\mathrm{Vol}(C)$ paths or just one path*
    send solutions to manager;                              *message to manager reporting results*
end do.
```