Solving Polynomial Systems using PHCpack

Jan Verschelde

Department of Math, Stat & CS
University of Illinois at Chicago
Chicago, IL 60607-7045, USA

email: jan@math.uic.edu

URL: http://www.math.uic.edu/~jan

Geometry of Mechanism Science (GeMS’07)
University of Notre Dame, 1 - 3 March 2007
Mission

- solve polynomial systems arising in applications
  → turn applications into benchmarks
- develop convenient interfaces to PHCpack
  → blackbox, toolbox, PHCmaple, PHClab, C
- run on multiprocessor computers and clusters
  → solve systems with > 100,000 solutions
Polynomial Systems in Applications

What are we solving?

- polynomial systems: format of our input
  the study of its solutions = algebraic geometry
- applications: relevance to science & engineering
  our application field concerns mechanical design
- benchmarks: test performance of our methods
  one goal is to turn applications into benchmarks
About PHCpack

PHC = Polynomial Homotopy Continuation

- Version 1.0 archived as Algorithm 795 by ACM TOMS.
- Platform for numerical algebraic geometry.
- Pleasingly parallel implementations
  - Yusong Wang of Pieri homotopies (HPSEC’04);
  - Anton Leykin of monodromy factorization (HPSEC’05);
  - Yan Zhuang of polyhedral homotopies (HPSEC’06).
- Some current developments:
  - Yun Guan: PHClab, experiments with MPITB in Octave;
  - Kathy Piret: bindings with Python for use in SAGE.
## Some Benchmarks from Mechanical Design

<table>
<thead>
<tr>
<th>name</th>
<th>n</th>
<th>D</th>
<th>$B_Z$</th>
<th>$B_S$</th>
<th>V</th>
<th>#sols</th>
</tr>
</thead>
<tbody>
<tr>
<td>assur44</td>
<td>8</td>
<td>864</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>56</td>
</tr>
<tr>
<td>fbremb2</td>
<td>6</td>
<td>256</td>
<td>240</td>
<td>163</td>
<td>96</td>
<td>68</td>
</tr>
<tr>
<td>fbrfive12</td>
<td>12</td>
<td>4096</td>
<td>96</td>
<td>96</td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td>fbrfive4</td>
<td>4</td>
<td>256</td>
<td>96</td>
<td>194</td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td>fourbar</td>
<td>4</td>
<td>256</td>
<td>96</td>
<td>96</td>
<td>80</td>
<td>36</td>
</tr>
<tr>
<td>ipp</td>
<td>8</td>
<td>256</td>
<td>96</td>
<td>96</td>
<td>64</td>
<td>48</td>
</tr>
<tr>
<td>ipp2</td>
<td>11</td>
<td>1024</td>
<td>576</td>
<td>848</td>
<td>288</td>
<td>16</td>
</tr>
<tr>
<td>kin1</td>
<td>12</td>
<td>4608</td>
<td>320</td>
<td>896</td>
<td>192</td>
<td>48</td>
</tr>
<tr>
<td>kinema</td>
<td>9</td>
<td>64</td>
<td>240</td>
<td>64</td>
<td>64</td>
<td>40</td>
</tr>
<tr>
<td>puma</td>
<td>8</td>
<td>128</td>
<td>16</td>
<td>32</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>rbp1</td>
<td>6</td>
<td>486</td>
<td>160</td>
<td>160</td>
<td>160</td>
<td>40</td>
</tr>
<tr>
<td>rbpl24</td>
<td>9</td>
<td>576</td>
<td>80</td>
<td>80</td>
<td>80</td>
<td>40</td>
</tr>
<tr>
<td>stewgou40</td>
<td>9</td>
<td>4,096</td>
<td>2,560</td>
<td>2,560</td>
<td>1,536</td>
<td>40</td>
</tr>
<tr>
<td>rps10</td>
<td>10</td>
<td>262,144</td>
<td>18,432</td>
<td>9,216</td>
<td>1,024</td>
<td>1,024</td>
</tr>
</tbody>
</table>

Different formulations of the same problem matter for solving most problems are now of little use as challenging benchmarks...
Five-Point Path Synthesis

Design a 4-bar linkage = design trajectory of coupler point.

<table>
<thead>
<tr>
<th>Input:</th>
<th>coordinates of points on coupler curve.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>lengths of the bars of the linkage.</td>
</tr>
</tbody>
</table>


Isotropic Coordinates

• A point \((a, b) \in \mathbb{R}^2\) is mapped to \(z = a + ib, \ i = \sqrt{-1}\).

• \((z, \bar{z}) = (a + ib, a - ib) \in \mathbb{C}^2\) are isotropic coordinates.

• Observe \(z \cdot \bar{z} = a^2 + b^2\).

• Rotation around \((0, 0)\) through angle \(\theta\) is multiplication by \(e^{i\theta}\). Multiply by \(e^{-i\theta}\) to invert the rotation.

• Abbreviate a rotation by \(\Theta = e^{i\theta}\), then its inverse \(\Theta^{-1} = \bar{\Theta}\), satisfying \(\Theta\bar{\Theta} = 1\).
The Loop Equations

Let $A = (a, \bar{a})$ and $B = (b, \bar{b})$ be the fixed base points.

Unknown are $(x, \bar{x})$ and $(y, \bar{y})$, coordinates of the other two points in the 4-bar linkage.

For given precision points $(p_j, \bar{p}_j)$, assuming $\theta_0 = 1$,

\[
\begin{align*}
(p_j + x\theta_j + a)(\bar{p}_j + \bar{x}\bar{\theta}_j + \bar{a}) &= (p_0 + x + a)(\bar{p}_0 + \bar{x} + \bar{a}) \\
(p_j + y\theta_j + b)(\bar{p}_j + \bar{y}\bar{\theta}_j + \bar{b}) &= (p_0 + y + b)(\bar{p}_0 + \bar{y} + \bar{b})
\end{align*}
\]

Since the angle $\theta_j$ corresponding to each $(p_j, \bar{p}_j)$ is unknown, five precision points are needed to determine the linkage uniquely.

Adding $\theta_j\bar{\theta}_j = 1$ to the system leads to 12 equations in 12 unknowns: $(x, \bar{x})$, $(y, \bar{y})$, and $(\theta_j, \bar{\theta}_j)$, for $j = 1, 2, 3, 4$. 
for k from 1 to 4 do
    eq[k] := theta[k]*Theta[k] - 1;
    eq[k+4] := (p[k] + x*theta[k] + a)*(P[k] + X*Theta[k] + A) - (p[0] + x + a)*(P[0] + X + A);
    eq[k+8] := (p[k] + y*theta[k] + b)*(P[k] + Y*Theta[k] + B) - (p[0] + y + b)*(P[0] + Y + B);
end do;
sys := [seq(eq[k],k=1..12)]; s := map(expand,sys);
for k from 1 to 4 do
    ss[k] := s[k];
    ss[k+4] := expand(algsubs(theta[k]*Theta[k]=1,s[k+4]));
    ss[k+8] := expand(algsubs(theta[k]*Theta[k]=1,s[k+8]));
end do;
points := seq([stats[random,uniform[-1,+1]](2)],k=1..7);
for k from 1 to 5 do
    p[k-1] := points[k][1] + points[k][2]*I; P[k-1] := points[k][1] - points[k][2]*I;
end do;
a := points[6][1] + points[6][2]*I; A := points[6][1] - points[6][2]*I;
b := points[7][1] + points[7][2]*I; B := points[7][1] - points[7][2]*I;
writeto("/tmp/fbrfive");
for k from 1 to 12 do lprint(expand(ss[k])); printf(";\n"); end do;
benchmarks

12
-.4091256991 * X * theta[1] - 1.061607555 * I * X + .3374636810 * X + .1524877812 * I * X
-.3374636810 * X - .1524877812 * I * X -.4091256991 * X * Theta[1] + 1.061607555 * I * X * Theta[1];
-.3374636810 * X - .1524877812 * I * X + .3705985316 * X * Theta[3] + .1.454067014 * I * X * Theta[3];
-.3374636810 * X - .1524877812 * I * X + .3188425748 * X * Theta[4] + .850446965 * I * X * Theta[4];
+ .9955481716 * y -.8208949212 * I * y - 1.742137552 * Y * Theta[1] + .3932004150 * I * Y * Theta[1];
+ .9955481716 * y -.8208949212 * I * y -.9318817788 * Y * Theta[2] -.4780708150 * I * Y * Theta[2];
+ .9955481716 * y -.8208949212 * I * y -.9624133210 * Y * Theta[3] -.7856598740 * I * Y * Theta[3];
+ .9955481716 * y -.8208949212 * I * y -.1.014169278 * Y * Theta[4] -.1820398250 * I * Y * Theta[4];
Output of phc -b on 5-Point Synthesis Problem

total degree : 4096
6-homogeneous Bezout number : 96
general linear-product Bezout number : 96
mixed volume : 36
solution 36 : start residual : 1.672E-15 #iterations : 1 success
  t : 1.00000000000000E+00  0.00000000000000E+00
  m : 1
the solution for t :
    theta[1] : 3.04923062675137E+00 -1.36666126486689E+01
    Theta[1] : 1.55514190369546E-02 6.97012611150467E-02
    theta[2] : 1.94158500874355E-01 -1.70861689159530E+00
...
    Y : 7.72626833143914E-01 -4.06259823401552E-01
Summary of Output of phc -b


===========================================================================

Frequency tables for correction, residual, condition, and distances:
FreqCorr :  0 0 0 0 0 0 0 0 0 0 0 0 0 1 8 17 0 10 : 36
FreqResi :  0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 33 : 36
FreqCond :  0 10 15 9 0 2 0 0 0 0 0 0 0 0 0 0 : 36
FreqDist :  36 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 : 36

Small correction terms and residuals counted to the right.
Well conditioned and distinct roots counted to the left.

---------------------------------------------------------------------
| root counts | start system | continuation | total time |
---------------------------------------------------------------------
| 0h 0m 4s910ms | 0h 0m 7s570ms | 0h 0m 8s 60ms | 0h 0m20s720ms |
---------------------------------------------------------------------
PHClab: A MATLAB/Octave Interface

with Yun Guan

- easier to use `phc` via automatic conversions of formats for polynomial systems and solutions
- adds scripting capabilities to PHCpack in a standard way
- solving many polynomial systems in parallel via the MPI toolbox for Octave:
  → this use of `phc` provides automatic quality control
Design of PHClab

• User only needs executable version of phc.

• PHClab function `set_path(location of phc)` is executed at the beginning of a MATLAB or Octave session.

• The scripts that implement the functions of PHClab call `phc` with the appropriate options and with redirected input and output from and to temporary files.
PHClab functions for isolated solutions

solve_system calls the blackbox solver
refine_sols refines the solutions by application of Newton’s method
mixed_volume computes the mixed volumes and solve a random system
track given target and start system + solutions, tracks paths
deflation applies deflation for isolated singularities

Note: mixed_volume calls translated code from


numerical irreducible decomposition in PHClab

- top down:
  - embed: adds extra hyperplanes and slack variables
  - cascade: performs a cascade of homotopies

- filtering and factoring:
  - phc_filter: filters junk on higher dimensional components
  - decompose: partitions a witness set along irreducible factors

- bottom up:
  - eqnbyeqn: an equation-by-equation solver
  - intersection: intersection of two witness sets
**scripts on Griffis-Duffy system**

**top down:**

S = read_system('gdplatB'); % read the system from file  
E = embed(S,1); % embed with one extra hyperplane  
sols = solve_system(E); % call the blackbox solver  
size(sols,2) % to see candidate witness #points  
[sw,R] = cascade(E,sols) % perform a cascade

**bottom up:**

p = read_system('gdplatBa') % read the system from file  
[sw,R] = eqnbyeqn(p) % call equation-by-equation solver

**process witness sets:**

dc = decompose(R{2},sw{2,1}) % decompose into irreducible factors
Using MPI Toolbox in Octave

• suppose one needs to solve hundreds of polynomial systems and a cluster computer is available

• requires dynamically linked LAM/MPI libraries and a custom compiled Octave

• use dynamic load balancing to distribute the systems among the worker nodes in the cluster

• used to solve 183 “origami equation systems” posed at problem session at IMA software workshop last fall

• with script in origamipar_v2, we run it typing
  mpirun -c 14 octave-2.1.64 -q --funcall origamipar_v2

Ways to compute in parallel

1. Batch jobs
   submit a job and wait for it to return

2. Interactive
   small number of processors on large supercomputer
   or on a cluster, asking for user input, interaction

3. Via scripting
   use MPI toolbox in Octave
Difficulties for Parallel PHCpack

- Optimal reuse of the existing software.
- Parallel code is just scheduling of jobs.
- Avoid storing all start solutions in main memory.
- Deal with large output sets.

An ambitious Swap of Letters:

PHC = Polynomial Homotopy Continuation

HPC = High Performance Computing

towards High Performance Continuation
Other Parallel Homotopy Solvers

T. Gunji, S. Kim, K. Fujisawa, and M. Kojima:

Roughly, there are three stages when solving a polynomial system using polynomial homotopy continuation:

- **I**
  - **compute root count(s)**
- **II**
  - **solve start system**
- **III**
  - **track paths to target**

Usually stage III is most time consuming.

But if millions of start solutions, memory gets too full...
Applying Program Inversion to Homotopy Solver

\[ h(x, t) = \gamma(1 - t)g(x) + tf(x) = 0, \quad \gamma \in \mathbb{C}, \quad t \in [0, 1]. \]

Input: \( g(x) = 0; \)
for \( k \) from 1 to \( \#g^{-1}(0) \) do
    compute \( y_k: g(y_k) = 0; \)
end for;
output: \( g^{-1}(0). \)

solve start system

Input: \( g^{-1}(0), h(x, t) = 0; \)
for \( k \) from 1 to \( \#g^{-1}(0) \) do
    path starts at \( y_k \in g^{-1}(0); \)
end for;
output: \( f^{-1}(0). \)

track paths to target
Applying Program Inversion to Homotopy Solver

\[ h(x, t) = \gamma(1 - t)g(x) + tf(x) = 0, \quad \gamma \in \mathbb{C}, \quad t \in [0, 1]. \]

Input: \( h(x) = 0; \)
for \( k \) from 1 to \( \#g^{-1}(0) \) do
    path starts at \( y_k \in g^{-1}(0); \)
end for;
output: \( f^{-1}(0). \)

get next start solution

Input: \( g(x) = 0, \ k; \)
compute \( y_k: g(y_k) = 0; \)
or read \( y_k \) from file;
or type in values for \( y_k; \)
output: \( y_k \in g^{-1}(0). \)
Jumpstarting Homotopies with phc

[phcpack@idefix bin]$ phc -q
Welcome to PHC (Polynomial Homotopy Continuation) V2.3.24 18 Feb 2007
Tracking Solution Paths with incremental read/write of solutions.

MENU for type of start system or homotopy:
1. start system is based on total degree;
2. a linear-product start system will be given;
3. start system and start solutions are provided;
4. polyhedral continuation on a generic system;
5. diagonal homotopy to intersect algebraic sets;
6. descend one level down in a cascade of homotopies.
Type 1, 2, 3, 4, 5, or 6 to select type:
Use PHCpack as a C Library

- Most of the code in PHCpack is in Ada, compiles with gcc.
- The parallel path trackers follow manager-worker protocol.
- The main parallel program is written in C, using MPI. Also all routines which handle job scheduling are written in C.
- The C interface uses PHCpack as a state machine:
  1. Feed data into machine and select methods;
  2. Compute with given data and selected methods;
  3. Extract the results from the machine.

The C user is unaware of the data structures and algorithms.
Jumpstarting Parallel Homotopies

[phcpack@idefix bin]$ mpirun -np 14 mpi2track

MENU for type of start system or homotopy:
  1. start system is based on total degree;
  2. a linear-product start system will be given;
  3. start system and start solutions are provided;
  4. the homotopy is a cascade to go one level down;
  5. start extrinsic diagonal homotopy to intersect 2 sets.
Type 1, 2, 3, 4, or 5 to select type of start system:
Jumpstarting Homotopies

Problem: huge #paths (e.g.: > 100,000),

undesirable to store all start solutions in main memory.

Solution: (assume manager/worker protocol)

1. The manager reads start solution from file “just in time”
   whenever a worker needs another path tracking job.

2. For total degree and linear-product start systems,
   it is simple to compute the solutions whenever needed.

3. As soon as worker reports the end of a solution path
   back to the manager, the solution is written to file.
Indexing Start Solutions

\[
\begin{align*}
\begin{cases}
x_1^4 - 1 &= 0 \\
x_2^5 - 1 &= 0 \\
x_3^3 - 1 &= 0
\end{cases}
\end{align*}
\]

The start system \( \begin{cases}
x_1^4 - 1 &= 0 \\
x_2^5 - 1 &= 0 \\
x_3^3 - 1 &= 0
\end{cases} \) has \( 4 \times 5 \times 3 = 60 \) solutions.

Get 25th solution via decomposition: \( 24 = 1(5 \times 3) + 3(3) + 0 \).

Verify via lexicographic enumeration:

000 → 001 → 002 → 010 → 011 → 012 → 020 → 021 → 022 → 030 → 031 → 032 → 040 → 041 → 042
100 → 101 → 102 → 110 → 111 → 112 → 120 → 121 → 122 → 130 → 131 → 132 → 140 → 141 → 142
200 → 201 → 202 → 210 → 211 → 212 → 220 → 221 → 222 → 230 → 231 → 232 → 240 → 241 → 242
300 → 301 → 302 → 310 → 311 → 312 → 320 → 321 → 322 → 330 → 331 → 332 → 340 → 341 → 342
a problem from electromagnetics

posed by Shigetoshi Katsura to PoSSo in 1994:
  a family of \( n - 1 \) quadrics and one linear equation;
  #solutions is \( 2^{n-1} \) (= Bézout bound).

\( n = 21 \): 32 hours and 44 minutes to track \( 2^{20} \) paths by 13 workers at 2.4Ghz, producing output file of 1.3Gb.

    tracking about 546 paths/minute.

verification of output:
  1. parsing 1.3Gb file into memory takes 400Mb and 4 minutes;
  2. data compression to quadtree of 58Mb takes 7 seconds.
Using Linear-Product Start Systems Efficiently

- Store start systems in their linear-product product form, e.g.:

\[
g(x) = \begin{cases} 
\left(x_1 + c_{11}\right) \times \left(x_2 + c_{12}x_3 + c_{13}\right) \times \left(x_2 + c_{14}x_3 + c_{15}\right) = 0 \\
\left(x_2 + c_{21}\right) \times \left(x_1 + c_{22}x_3 + c_{23}\right) \times \left(x_1 + c_{24}x_3 + c_{25}\right) = 0 \\
\left(x_3 + c_{31}\right) \times \left(x_1 + c_{32}x_2 + c_{33}\right) \times \left(x_1 + c_{34}x_2 + c_{35}\right) = 0
\end{cases}
\]

- Lexicographic enumeration of start solutions, 
  → as many candidates as the total degree.

- Store results of incremental LU factorization. 
  → prune in the tree of combinations.
Nine-Point Path Synthesis


Find all four-bar linkages whose coupler curve passes through nine precision points.

Formulation into Polynomial System

The 9-point problem was translated into

- a system of 4 quadrics and 8 quartics in 12 unknowns.
- Its total degree equals $2^4 \cdot 4^8 = 2^{20}$.
- A 2-homogeneous Bézout number equals 286,720.
- Exploiting a 2-way symmetry leads to 143,360 solution paths.

At that time – early nineties – this was the largest polynomial system solved using numerical continuation methods.
Timings – Past and Present

Back Then: Tracking 143,360 solution paths in 12 variables took 331.9 hours of CPU time (about two weeks) on an IBM 3081 at the University of Notre Dame.

1,442 four-bar linkages were found

Computing various instances of the parameters with coefficient-parameter polynomial continuation requires only 1,442 paths to track. The number of real meaningful linkages ranged between 21 and 120.

Present: Using a personal cluster computer of 13 workers and one manager at 2.4 Ghz, running Linux, tracking 286,720 paths of a formulation in 20 variables takes about 14.1 hours.
The Theorems of Bernshteĭn

Theorem A: The number of roots of a generic system equals the mixed volume of its Newton polytopes.

Theorem B: Solutions at infinity are solutions of systems supported on faces of the Newton polytopes.


Structure of proofs: First show Theorem B, looking at power series expansions of diverging paths defined by a linear homotopy starting at a generic system. Then show Theorem A, using Theorem B with a homotopy defined by lifting the polytopes.
Some References on Polyhedral Methods


3 stages to solve a polynomial system $f(x) = 0$

1. Compute the mixed volume (aka the BKK bound) of the Newton polytopes spanned by the supports $A$ of $f$ via a regular mixed-cell configuration $\Delta_\omega$.

2. Given $\Delta_\omega$, solve a generic system $g(x) = 0$, using polyhedral homotopies. Every cell $C \in \Delta_\omega$ defines one homotopy

$$h_C(x, s) = \sum_{a \in C} c_a x^a + \sum_{a \in A \setminus C} c_a x^a s^{\nu_a}, \quad \nu_a > 0,$$

tracking as many paths as the mixed volume of the cell $C$, as $s$ goes from 0 to 1.

3. Use $(1 - t)g(x) + tf(x) = 0$ to solve $f(x) = 0$.

Stages 2 and 3 are computationally most intensive (1 $\ll$ 2 $<$ 3).
**A Static Distribution of the Workload**

<table>
<thead>
<tr>
<th>manager</th>
<th>worker 1</th>
<th>worker 2</th>
<th>worker 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vol(cell 1) = 5</td>
<td>#paths(cell 1) : 5</td>
<td>#paths(cell 4) : 5</td>
<td>#paths(cell 1) : 5</td>
</tr>
<tr>
<td>Vol(cell 2) = 4</td>
<td>#paths(cell 2) : 4</td>
<td>#paths(cell 5) : 7</td>
<td>#paths(cell 7) : 4</td>
</tr>
<tr>
<td>Vol(cell 3) = 4</td>
<td>#paths(cell 3) : 4</td>
<td>#paths(cell 6) : 2</td>
<td>#paths(cell 6) : 1</td>
</tr>
<tr>
<td>Vol(cell 4) = 6</td>
<td>#paths(cell 4) : 1</td>
<td>#paths(cell 4) : 5</td>
<td>#paths(cell 8) : 8</td>
</tr>
<tr>
<td>Vol(cell 5) = 7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vol(cell 6) = 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vol(cell 7) = 4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vol(cell 8) = 8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>total #paths : 41</td>
<td>#paths : 14</td>
<td>#paths : 14</td>
<td>#paths : 13</td>
</tr>
</tbody>
</table>

Since polyhedral homotopies solve a **generic** system $g(x) = 0$, we expect every path to take the same amount of work...
An academic Benchmark: cyclic $n$-roots

The system

$$f(x) = \begin{cases} 
  f_i = \sum_{j=0}^{n-1} \prod_{k=1}^{i} x_{(k+j) \mod n} = 0, & i = 1, 2, \ldots, n - 1 \\
  f_n = x_0 x_1 x_2 \cdots x_{n-1} - 1 = 0 
\end{cases}$$

appeared in


very sparse, well suited for polyhedral methods
# Results on the cyclic $n$-roots problem

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

Wall time for start systems to solve the cyclic $n$-roots problems, using 13 workers, with static load distribution.
## Dynamic versus Static Workload Distribution

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

Wall time in seconds to solve a start system for the cyclic 7-roots problem.
Design of Serial Chains


Results on Design of Serial Chains

Bézout vs Bernshteĭn

<table>
<thead>
<tr>
<th>Surface</th>
<th>Bounds on #Solutions</th>
<th>Wall Time</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$D$</td>
<td>$B$</td>
<td>$V$</td>
<td>our cluster</td>
</tr>
<tr>
<td>elliptic cylinder</td>
<td>2,097,152</td>
<td>247,968</td>
<td>125,888</td>
<td>11h 33m</td>
</tr>
<tr>
<td>circular torus</td>
<td>2,097,152</td>
<td>868,352</td>
<td>474,112</td>
<td>7h 17m</td>
</tr>
<tr>
<td>general torus</td>
<td>4,194,304</td>
<td>448,702</td>
<td>226,512</td>
<td>14h 15m</td>
</tr>
</tbody>
</table>

$D = \text{total degree}; \ B = \text{generalized Bézout bound}; \ V = \text{mixed volume}$

Wall time for mechanism design problems on our cluster and argo.

- Compared to the linear-product bound, 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.
Concluding Remarks

- To solve large polynomial systems in parallel we had to rethink the design of the original program.
- Scheduling of path tracking jobs leads to an almost optimal speedup, using dynamic load balancing.
- Still much work left to develop tools to process and certify the results, we need to consider also “quality up”.

Software, source code and executables, available at

http://www.math.uic.edu/~jan/download.html