Multitasking Polynomial Homotopy Continuation in PHCpack

Jan Verschelde

University of Illinois at Chicago
Department of Mathematics, Statistics, and Computer Science
http://www.math.uic.edu/~jan
jan@math.uic.edu

ACA 2009 Session on High-Performance Computer Algebra
ÉTS, Montréal, Canada, 25-28 June 2009.
Solving Polynomial Systems

On input is a polynomial system $f(x) = 0$.

A homotopy is a family of systems:

$$h(x, t) = (1 - t)g(x) + tf(x) = 0.$$ 

At $t = 1$, we have the system $f(x) = 0$ we want to solve.
At $t = 0$, we have a good system $g(x) = 0$:

- solutions are known or easier to solve; and
- all solutions of $g(x) = 0$ are regular.

Tracking all solution paths is pleasingly parallel, although not every path requires the same amount of work.
Homotopy Continuation Methods

Types of homotopies $h$:

- $h(x, t) = (1 - t)g(x) + tf(x) = 0$, from start to target.
- $h(x, t) = f(c_0(1 - t) + c_1 t, x) = 0$, cheater’s homotopy.
- $h(x, t) = f(c, x(t)) = 0$, moving basis coordinates.

Homotopies are often used in combination, or in cascades.


Software Systems

Starring in alphabetical order:


- **HOM4PS-2.0para** by T.Y. Li and C.H. Tsai (2009) is a parallel version of **HOM4PS-2.0** by T.L. Lee, T.Y. Li, and C.H. Tsai (2007); extends **HOM4PS** by T. Gao and T.Y. Li.


Anton Leykin is developing homotopy continuation in Macaulay2.

Parallel PHCpack
parallel implementation of polynomial homotopy continuation methods

PHC = Polynomial Homotopy Continuation

- Version 1.0 archived as Algorithm 795 by ACM TOMS (1999)
- Pleasingly parallel implementations
  + Yusong Wang of Pieri homotopies (HPSEC’04)
  + Anton Leykin of monodromy factorization (HPSEC’05)
  + Yan Zhuang of polyhedral homotopies (HPSEC’06)
  + Yun Guan of diagonal homotopies (HPCS’08)
- Interactive Parallel Computing:
  + Yun Guan: PHClab, experiments with MPITB in Octave
  + Kathy Piret: bindings with Python, use of sockets

Release v2.3.42 extends phcpy and a preliminary PHCwulf.py.
Hardware and Software

Running on a modern workstation (*not* a supercomputer):

- **Hardware**: Mac Pro with 2 Quad-Core Intel Xeons at 3.2 Ghz
  - Total Number of Cores: 8
  - 1.6 GHz Bus Speed
  - 12 MB L2 Cache per processor, 8 GB Memory

- PHCpack is written in Ada, compiled with gnu-ada compiler
  - gcc version 4.3.4 20090511 for GNAT GPL 2009 (20090511)
  - Target: x86_64-apple-darwin9.6.0
  - Thread model: posix

Also compiled for Linux and Windows (win32 thread model).
Starting Worker Tasks

procedure Workers is instantiated with a Job procedure, executing code based on the id number.

```plaintext
procedure Workers ( n : in natural ) is
task type Worker ( id,n : natural );
task body Worker is
begin
    Job(id,n);
end Worker;
procedure Launch_Workers ( i,n : in natural ) is
w : Worker(i,n);
begin
    if i < n
        then Launch_Workers(i+1,n);
    end if;
end Launch_Workers;
begin
    Launch_Workers(1,n);
end Workers;
```
MPI versus Threads

- MPI = Message Passing Interface
  The manager/worker paradigm:
  ▶ worker nodes perform path tracking jobs,
  ▶ manager maintains job queue, serves workers.
  Manager must be available to serve jobs.

- Threads are lightweight processes
  Collaborative workers launched by master thread:
  ▶ communication overhead replaced by memory sharing,
  ▶ job queue updated in critical section using locks.

- With MPI, we worry about communication overhead.
  With threads, memory (de)allocation must be in critical sections.
Load Balancing and Granularity Issues

We assume: \# solution paths \gg \# cores.

Granularity Issues:
- coarse: one job = track one solution path
- fine: polynomial evaluation, linear algebra

Dynamic load balancing:
- not all jobs 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
First Preliminary Results

Using version 2.3.45 of PHCpack:

```
$ time phc -p -t8 < /tmp/input8
```

#worker tasks = number following the -t

running a cheater’s homotopy on cyclic 7-roots (924 paths).

<table>
<thead>
<tr>
<th>#workers</th>
<th>real</th>
<th>user</th>
<th>sys</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15.478s</td>
<td>15.457s</td>
<td>0.010s</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>7.790s</td>
<td>15.483s</td>
<td>0.010s</td>
<td>1.987</td>
</tr>
<tr>
<td>4</td>
<td>3.926s</td>
<td>15.445s</td>
<td>0.011s</td>
<td>3.942</td>
</tr>
<tr>
<td>8</td>
<td>1.992s</td>
<td>15.424s</td>
<td>0.015s</td>
<td>7.770</td>
</tr>
</tbody>
</table>

Since version 2.3.46 of PHCpack:

```
$ phc -b -t8
```

blackbox solver (phc -b) uses multitasking
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

used in `mpi2cell_s` with Yan Zhuang

<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 6) : 1</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 8) : 8</td>
</tr>
<tr>
<td>Vol(cell 4) = 6</td>
<td>#paths(cell 4) : 1</td>
<td></td>
<td></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...
Running Polyhedral Homotopies

Running polyhedral homotopies on a random coefficient system, distributing mixed cells, for the cyclic \( n \)-roots problems.

Tracking MV (MV = mixed volume) many solution paths:

<table>
<thead>
<tr>
<th>( n )</th>
<th>MV</th>
<th>#tasks</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>924</td>
<td>12</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2560</td>
<td>58</td>
<td>29</td>
<td>15</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>11016</td>
<td>417</td>
<td>209</td>
<td>104</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>35940</td>
<td>2156</td>
<td>1068</td>
<td>534</td>
<td>270</td>
<td></td>
</tr>
</tbody>
</table>

Comparison with MPI (\texttt{mpi2cell\_d}) on cyclic 10-roots:

- \texttt{mpirun -n 9}: total wall time = 270.5 seconds.
- on same random coefficient system and same tolerances: elapsed wall clock time is 233 seconds.
Conclusions and Future Plans

Conclusions:
- multitasking is convenient programming model
- effective speedups on modern workstation

Some future applications:
- multi-tiered parallel implementations
- investigation of granularity issues
- attention for quality up (not only speedup)
- parallel versions of Littlewood-Richardson homotopies