Polynomial Homotopies on Multicore Workstations

Jan Verschelde Genady Yoffe

University of Illinois at Chicago
Department of Mathematics, Statistics, and Computer Science
http://www.math.uic.edu/~jan
jan@math.uic.edu gyoffe2@uic.edu

Parallel Symbolic Computation 2010 (PASCO 2010)
Grenoble, France, 21-23 July 2010.

Outline

1. Homotopy Continuation Methods
   - solving polynomial systems
   - pleasingly parallel computations
2. Multicore Workstations
   - applying tasking
   - MPI versus threads
   - results for polyhedral blackbox solver
3. Quality Up
   - compensating for the cost of higher precision arithmetic
   - tracking one path more accurately
   - multithreaded system evaluation and LU factorization

Solving Polynomial Systems

On input is a polynomial system $f(\mathbf{x}) = \mathbf{0}$.

A homotopy is a family of systems:

$$h(\mathbf{x}, t) = (1 - t)g(\mathbf{x}) + t\, f(\mathbf{x}) = \mathbf{0}.$$

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

- solutions are known or easier to solve; and
- all solutions of $g(\mathbf{x}) = \mathbf{0}$ 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:

- $h(\mathbf{x}, t) = (1-t)g(\mathbf{x}) + t\, f(\mathbf{x}) = \mathbf{0}$, from start to target.
- $h(\mathbf{x}, t) = f(\mathbf{c}_0(1-t) + \mathbf{c}_1 t, \mathbf{x}) = \mathbf{0}$, cheater's homotopy.
- $h(\mathbf{x}, t) = f(\mathbf{c}, \mathbf{x}(t)) = \mathbf{0}$, moving basis coordinates.

Homotopies are often used in combination, or in cascades.

1. **Tien-Yien Li.** Numerical solution of polynomial systems by homotopy continuation methods. In Volume XI of *Handbook of Numerical Analysis*, pages 209–304, 2003.
2. **Andrew J. Sommese and Charles W. Wampler.** *The Numerical Solution of Systems of Polynomials Arising in Engineering and Science.* World Scientific, 2005.

Software Systems

Starring in alphabetical order:

- **Bertini**, first released in Fall 2006, by D.J. Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler. MPI executables available.
- **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.
- **PHoMpara** by T. Gunji, S. Kim, K. Fujisawa, and M. Kojima (2006) is a parallel version of **PHoM** by T. Gunji, S. Kim, M. Kojima, A. Takeda, K. Fujisawa and T. Mizutani (2004).
- **POLSYS_GLP** is Algorithm 857 of ACM TOMS (2006) by H.-J. Su, J.M. McCarthy, M. Sosonkina, and L.T. Watson extends **HOMPACK90** by L.T. Watson, M. Sosonkina, R.C. Melville, A.P. Morgan, and H.F. Walker (1997) and **HOMPACK** by L.T. Watson, S.C. Billups, and A.P. Morgan (1987).

Anton Leykin is developing homotopy continuation in Macaulay2. `http://www.math.uic.edu/~leykin/NAG4M2/index.html`.

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.

```
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

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

appeared in

- G. Björck: **Functions of modulus one on Z_p whose Fourier transforms have constant modulus**. In *Proceedings of the Alfred Haar Memorial Conference, Budapest*, pages 193–197, 1985.
- J. Backelin and R. Fröberg: **How we proved that there are exactly 924 cyclic 7-roots**. In ISSAC'91 proceedings, pages 101-111, ACM, 1991.

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).

#workers	real	user	sys	speedup
1	15.478s	15.457s	0.010s	1
2	7.790s	15.483s	0.010s	1.987
4	3.926s	15.445s	0.011s	3.942
8	1.992s	15.424s	0.015s	7.770

Since version 2.3.46 of PHCpack:

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

blackbox solver (`phc -b`) uses multitasking

3 stages to solve a polynomial system $f(\mathbf{x}) = \mathbf{0}$

1. Compute the mixed volume MV (aka the BKK bound) of the Newton polytopes spanned by the supports A of f via a **regular mixed-cell configuration** Δ_ω.
2. Given Δ_ω, solve a generic system $g(\mathbf{x}) = \mathbf{0}$, using polyhedral homotopies. Every cell $C \in \Delta_\omega$ defines one homotopy

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

   tracking as many paths as the mixed volume of the cell C, as s goes from 0 to 1.
3. Use $(1-t)g(\mathbf{x}) + tf(\mathbf{x}) = \mathbf{0}$ to solve $f(\mathbf{x}) = \mathbf{0}$.

Stages 2 and 3 are **computationally most intensive** ($1 \ll 2 < 3$),

e.g.: cyclic 10-roots (MV = 35940): stage 1: 21 secs, stage 2: 39 min.

A Static Distribution of the Workload

used in `mpi2cell_s` with Yan Zhuang

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 : 14	#paths : 14	#paths : 13

Since polyhedral homotopies solve a **generic** system $g(\mathbf{x}) = \mathbf{0}$,
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:

		#tasks, times in seconds			
n	MV	1	2	4	8
7	924	12	6	3	2
8	2560	58	29	15	8
9	11016	417	209	104	52
10	35940	2156	1068	534	270

Comparison with MPI (`mpi2cell_d`) on cyclic 10-roots:

- `mpirun -n 9`: total wall time = 270.5 seconds.
- on same random coefficient system and same tolerances: elapsed wall clock time is 233 seconds.

Quality Up

defined by S.G. Akl, 2004

Given more cores, more accurate results in same time?

A quad double is an unevaluated sum of 4 doubles, improves working precision from 2.2×10^{-16} to 2.4×10^{-63}.

- Y. Hida, X.S. Li, and D.H. Bailey: **Algorithms for quad-double precision floating point arithmetic.** In *15th IEEE Symposium on Computer Arithmetic* pages 155–162. IEEE, 2001. Software at `http://crd.lbl.gov/~dhbailey/mpdist/qd-2.3.9.tar.gz`.
- X. Li, J. Demmel, D. Bailey, G. Henry, Y. Hida, J. Iskandar, W. Kahan, S. Kang, A. Kapur, M. Martin, B. Thompson, T. Tung, and D. Yoo: **Design, implementation and testing of extended and mixed precision BLAS.** *ACM Trans. Math. Softw.*, 28(2):152–205, 2002.

Why `QD-2.3.9`? + simple memory management

Cost Overhead of Arithmetic

Solve 100-by-100 system 1000 times with LU factorization:

type of arithmetic	user CPU seconds
`double real`	2.026s
`double complex`	16.042s
`double double real`	20.192s
`double double complex`	140.352s
`quad double real`	173.769s
`quad double complex`	1281.934s

Fully optimized Ada code on one core of 3.2 Ghz Intel Xeon.

Overhead of complex arithmetic: `16.042/2.026 = 7.918`, `140.352/20.192 = 6.951`, `1281.934/173.769 = 7.377`.

Overhead of double double complex: `140.352/16.042 = 8.749`.

Overhead of quad double complex: `1281.934/140.352 = 9.134`.

Newton's Method with QD

Refining the 1,747 generating cyclic 10-roots is pleasingly parallel.

`double double complex`

#workers	real	user	sys	speedup
1	4.818s	4.790s	0.015s	1
2	2.493s	4.781s	0.013s	1.933
4	1.338s	4.783s	0.015s	3.601
8	0.764s	4.785s	0.016s	6.306

`quad double complex`

#workers	real	user	sys	speedup
1	58.593s	58.542s	0.037s	1
2	29.709s	58.548s	0.054s	1.972
4	15.249s	58.508s	0.053s	3.842
8	8.076s	58.557s	0.058s	7.255

For quality up: compare `4.818s` with `8.076s`.
With 8 cores, doubling accuracy in less than double the time.

Multitasking Newton's method

Often one path requires extra precision.

Computations in Newton's method consists of

1. evaluate the system and the Jacobian matrix;
2. solve a linear system to update the solution.

Questions:

1. how large systems must be to allow speedup?
2. synchronization issues with LU factorization?

Polynomial System Evaluation

Need to evaluate system and its Jacobian matrix. Running example: 30 polynomials, each with 30 monomials of degree 30 in 30 variables leads to 930 polynomials, with 11,540 distinct monomials.
We represent a sparse polynomial

$$f(\mathbf{x}) = \sum_{\mathbf{a} \in A} c_{\mathbf{a}} \mathbf{x}^{\mathbf{a}}, \quad c_{\mathbf{a}} \in \mathbb{C} \setminus \{0\}, \; \mathbf{x}^{\mathbf{a}} = x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n},$$

collecting the exponents in the support A in a matrix E, as

$$F(\mathbf{x}) = \sum_{i=1}^{m} c_i \mathbf{x}^{E[k_i,:]}, \quad c_i = c_{\mathbf{a}}, \; \mathbf{a} = E[k_i,:]$$

where k is an m-vector linking exponents to rows in E: $E[k_i,:]$ denotes all elements on the k_ith row of E. Storing all values of the monomials in a vector V, evaluating F (and f) is equivalent to an inner product:

$$F(\mathbf{x}) = \sum_{i=1}^{m} c_i V_{k_i}, \quad V = \mathbf{x}^E.$$

Polynomial System Evaluation with Threads

Two jobs:

1. evaluate $V = \mathbf{x}^E$, all monomials in the system;
2. use V in inner products with coefficients.

Our running example: evaluating 11,540 monomials of degree 30 requires about 346,200 multiplications.
Since evaluation of monomials dominates inner products, we do not interlace monomial evaluation with inner products.

Static work assignment: if p threads are labeled as $0, 1, \ldots, p-1$, then ith entry of V is computed by thread t for which $i \mod p = i$.

Synchronization of jobs is done by p boolean flags.
Flag i is true if thread i is busy.
First thread increases job counter only when no busy threads.
Threads go to next job only if job counter is increased.

Speedup and Quality Up for Evaluation

930 polynomials of 30 monomials of degree 30 in 30 variables:

double double complex

#tasks	real	user	sys	speedup
1	1m 9.536s	1m 9.359s	0.252s	1
2	0m 37.691s	1m 10.126s	0.417s	1.845
4	0m 21.634s	1m 10.466s	0.753s	3.214
8	0m 14.930s	1m 12.120s	1.711s	4.657

quad double complex

#tasks	real	user	sys	speedup
1	9m 19.085s	9m 18.552s	0.563s	1
2	4m 43.005s	9m 19.402s	0.679s	1.976
4	2m 24.669s	9m 20.635s	1.023s	3.865
8	1m 21.220s	9m 26.120s	2.809s	6.884

Speedup improves with quad doubles. Quality up: with 8 cores overhead reduced to 17%, as `81.220/69.536 = 1.168`.

Multithreaded LU factorization

Routines in PHCpack to solve linear systems are based on ZGEFA and ZGESL of LINPACK.

The multithreaded version of LU factorization does pivoting, synchronizing jobs with busy flags
and a column counter updated by first thread.

For good computational results for our first multithreaded implementation, the dimension needs to be around 80.

Because LU is $O(n^3)$, backsubstitution is $O(n^2)$, and $n \gg p$, multithreaded LU still dominates the total cost.

Speedup and Quality up for Multithreaded LU

1000 times LU factorization of 80-by-80 matrix:

double double complex

#tasks	real	user	sys	speedup
1	1m 8.173s	1m 8.074s	0.131s	1
2	0m 36.712s	1m 13.061s	0.249s	1.857
4	0m 21.565s	1m 25.035s	0.455s	3.161
8	0m 20.986s	1m 42.156s	2.270s	3.248

quad double complex

#tasks	real	user	sys	speedup
1	10m 12.216s	10m 11.900s	0.311s	1
2	5m 12.753s	10m 24.774s	0.477s	1.958
4	2m 42.653s	10m 48.795s	0.699s	3.764
8	1m 33.234s	12m 17.653s	1.930s	6.566

Acceptable speedups with quad doubles. Quality up: with 8 cores, less than twice the time to double accuracy.

Conclusions

1. Threads offer more convenient programming model than MPI.
   To use five threads in blackbox solver, type at command prompt
   `phc -b -t5 input output`
   is also more user friendly than requiring availability of MPI.

2. Speedups of pleasingly parallel homotopies with threads
   very well suited for multicore workstations.

3. Quality up: cores compensate for multiprecision arithmetic.
   - multiprecision for homotopies as common as complex arithmetic
   - cost overhead of arithmetic keeps dimensions for speedup modest
   - good results with preliminary parallel algorithms