Accurate Path Tracking

Jan Verschelde
joint work with Genady Yoffe

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

SIAM Conference on Algebraic Geometry, Minisymposium on
Subdivision Methods in Numerical Algebraic Geometry
Raleigh, 6-9 October 2011

Outline

1. Problem Statement
   - solving polynomial systems accurately
   - using quad doubles
   - path tracking on multicore computers
2. Exploratory Computations
   - cost overhead of arithmetic
   - polynomial system evaluation with many threads
   - multithreaded linear system solving
3. Preliminary Applications
   - since version 2.3.59 in PHCpack
   - talk by Genady Yoffe in MS51 on Sunday 10:30 in Riddick 451

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.

Efficiency and Accuracy

We assume we have the “right” homotopy, or we have no choice:
the family of systems is naturally given to us.

We want accurate answers, two opposing situations:

- Adaptive refinement: starting with machine arithmetic
  leads to meaningful results, e.g.: leading digits, magnitude.
- Problem is too ill-conditioned for machine arithmetic
  e.g.: huge degrees, condition numbers $> 10^{16}$.

Runs of millions of solution paths are routine,
but then often some end in failure and spoil the run.

Speedup and Quality Up

Selim G. Akl, Journal of Supercomputing, 29, 89-111, 2004

How much **faster** *if we can use p cores?*

Let T_p be the time on p cores, then

$$\text{speedup} = \frac{T_1}{T_p} \to p,$$

keeping the working precision fixed.

How much **better** *if we can use p cores?*

Take quality as the number of correct decimal places.

Let Q_p be the quality on p cores, then

$$\text{quality up} = \frac{Q_p}{Q_1} \to p,$$

keeping the time fixed.

accuracy $\sim$ precision

Taking a narrow view on quality:

$$\text{quality up} = \frac{Q_p}{Q_1} = \frac{\text{\# decimal places with } p \text{ cores}}{\text{\# decimal places with 1 core}}$$

Confusing working precision with accuracy is
okay if running Newton's method on well conditioned solution.

Can we keep the running time constant?

If we assume optimal (or constant) speedup
and Q_p/Q_1 is linear in p, then we can rescale.

Error-free Transformations

From §3 of *Verification methods: Rigorous results using floating-point arithmetic* by S.M. Rump, Acta Numerica 2010:

An algorithm by Knuth (1969) allows to

> *recover the error of a floating-point addition using only basic floating-point operations.*

Dekker (1971) extended this idea to other floating-point operations.

According to Rump, this leads to a rigorous compututations (e.g.: sum of floating-point numbers) using only floating-point computations.

Double Doubles

T.J. Dekker, Numer. Math. 18, 224-242, 1971

Let a and b be two machine doubles.

Denote by $\oplus$ and $\ominus$ respectively the $+$ and $-$ on doubles.

Then $a + b$ is stored as a pair (x, y) of two doubles:

$$x = a \oplus b \quad \text{and} \quad y = b \ominus (x \ominus a).$$

We can interpret y as the error caused by $\oplus$.

Similarities with other arithmetic in software, e.g.:

- accuracy: interval arithmetic
- efficiency: complex arithmetic

Important distinction: *multiple component* $\neq$ *multiple digit*.

Multiple digit arithmetic extend fraction with an integer sequence.

the QD library

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

Arbitrary precision takes an arbitrary amount of storage
$\Rightarrow$ dynamic memory allocation
$\Rightarrow$ threads requesting memory block progress of other threads

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
- Standalone code by Genady Yoffe: multithreaded routines in C (recently upgraded to C++, use of Standard Template Library) for polynomial evaluation and linear algebra, with pthreads.
- 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.

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`.

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

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.

the quality up factor

Compare `4.818s` (1 core in dd) with `8.076s` (8 cores in qd).
With 8 cores, doubling accuracy in less than double the time.

The speedup is close to optimal: how many cores would we need to reduce the calculation with quad doubles to `4.818s`?

$$\frac{8.076}{4.818} \times 8 = 13.410 \Rightarrow 14 \text{ cores}$$

Denote $y(p) = Q_p/Q_1$ and assume $y(p)$ is linear in p.

We have $y(1) = 1$ and $y(14) = 2$, so we interpolate:

$$y(p) - y(1) = \frac{y(14) - y(1)}{14 - 1}(p - 1).$$

and the quality up factor is $y(8) = 1 + \frac{7}{13} \approx 1.538$.

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`.

the quality up factor

8 cores reduced overhead to 17%, as `81.220/69.536 = 1.168`.

The speedup is close to optimal: how many cores would we need to reduce the calculation with quad doubles to `69.536s`?

$$\frac{81.220}{69.536} \times 8 = 9.344 \Rightarrow 10 \text{ cores}$$

Denote $y(p) = Q_p/Q_1$ and assume $y(p)$ is linear in p.

We have $y(1) = 1$ and $y(10) = 2$, so we interpolate:

$$y(p) - y(1) = \frac{y(10) - y(1)}{10 - 1}(p - 1).$$

and the quality up factor is $y(8) = 1 + \frac{7}{9} \approx 1.778$.

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.

the quality up factor

Compare `68.173s` (1 core in dd) with `93.234s` (8 cores in qd).
With 8 cores, less than twice the time to double accuracy.

The speedup is close to optimal: how many cores would we need to reduce the calculation with quad doubles to `68.173s`?

$$\frac{93.234}{68.173} \times 8 = 10.941 \Rightarrow 11 \text{ cores}$$

Denote $y(p) = Q_p/Q_1$ and assume $y(p)$ is linear in p.

We have $y(1) = 1$ and $y(11) = 2$, so we interpolate:

$$y(p) - y(1) = \frac{y(11) - y(1)}{11 - 1}(p - 1).$$

and the quality up factor is $y(8) = 1 + \frac{7}{10} = 1.7$.

projected speedup for tracking one path

Running standalone C code by Genady Yoffe.

- 40 variables;
- unified support of 200 monomials;
- degree of monomials: 40 on average, 80 is maximum;
- 1000 Newton iterations.

#threads	polynomial evaluation	row reduction	back substitution	total time	speedup
1	35.732s	4.849s	0.197s	40.778s	1
2	17.932s	3.113s	0.100s	21.145s	1.928
4	9.248s	1.824s	0.062s	11.134s	3.662
8	4.775s	1.349s	0.053s	6.177s	6.602

what have we learned?

Results published in the PASCO 2010 proceedings.
(PASCO = Parallel Symbolic Computation)

1. Cost overhead of double double $\sim$ complex arithmetic.
2. Effective for multithreading: we get quality up.
3. On running a multithreaded Newton's method.
   Experimentally we determined threshold dimensions,
   i.e. lower bounds on n for which good speedup.
   - large degrees: system evaluation dominates
   - low degrees: linear algebra becomes more important

   Both evaluation and linear solving ought be multithreaded.

since version 2.3.59 in PHCpack

PHC = Polynomial Homotopy Continuation

- Version 1.0 archived as Algorithm 795 by ACM TOMS (1999).
- Implements many homotopy algorithms of numerical algebraic geometry: witness cascades, monodromy loops, diagonal homotopies.
- Since v2.3.13, contains a translation of `MixedVol`
  ACM TOMS Algorithm 846 by T. Gao, T.Y. Li, and M. Wu.

  Mixed volumes give a generically sharp bound on the number of isolated solutions of a polynomial system.
- Since version 2.3.59 path tracking with quad doubles.

option phc -t8

-t#: use # cores (since v2.3.45)
Path tracking on one core:

```
$ time phc -p < gcn6_input1
...
real 0m5.423s
user 0m5.393s
sys 0m0.010s
```

Using 8 cores:

```
$ time phc -p -t8 < gcn6_input8
...
real 0m0.716s
user 0m5.298s
sys 0m0.010s
```

Speedup: `5.423/0.716 = 7.574`.

Multiprecision Path Tracking

In *Lectures on Algebraic Statistics*, Mathias Dtron, Bernd Sturmfels, and Seth Sullivant, §7.4, on page 159, conjecture #solutions of the maximum likelihood degree of Gaussian cycles. Verification done on some cases jointly with Elizabeth Gross and Sonja Petrovic.

Preliminary results on tracking 75 paths in $\mathbb{C}^{21}$:

	tolerance	cpu time	factor
double precision	10^{-9}	5s	1.0
double double precision	10^{-24}	3m 46s	45.2
quad double precision	10^{-56}	5h 25m 54s	3910.8

The tolerance for the Newton corrector is quite severe.

The factor in the last column indicates the number of cores needed (assuming optimal speedup) for the quality up.

Improvements

- higher order predictors needed
- techniques of algorithmic differentiation
- mixing precision levels: prediction with standard doubles
- increasing levels of precision when running Newton's method
- talk by Genady Yoffe in MS51 on Sunday 10:30 in Riddick 451