Parallel Polynomial Evaluation

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 Parallel Processing for Scientific Computing
Session CP4 on Solvers in Applications
Savannah, 15-17 February 2012

Outline

1. Solving Polynomial Systems Accurately
   - compensating for the cost of quad double arithmetic
   - speedup and quality up
   - refining solutions of an academic benchmark

2. Parallel Polynomial Evaluation
   - two algorithms for sparse polynomials
   - distributed evaluation of all monomials
   - using algorithmic differentiation

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 (using MPI), but then often some end in failure and spoil the run.

Goal: use **multicore CPUs** to track ***one difficult*** solution path.

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.

Goal: determine thresholds on the dimensions and degrees of the system for a significant speedup on 8 cores.

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.
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`.

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).

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 arithmetic

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 $y(8) = 1 + \frac{7}{13} \approx 1.538$.

parallel polynomial evaluation

We consider sparse polynomials:

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

Sparse means: $\#A$ is $O(n)$ or $O(n^2)$, often $\#A \ll \deg(f)$.

Given $F = (f_1, f_2, \ldots, f_n)$ and some $\mathbf{z} \in \mathbb{C}^n$, compute $F(\mathbf{z})$ and the Jacobian matrix $J_F(\mathbf{z})$ of all partial derivatives.

Two parallel algorithms:

1. Collect all monomials with coefficients in F and J_F in the vectors M and C. Distribute M and C among threads.
2. For each monomial $\mathbf{x}^{\mathbf{a}}$ in F, evaluate $\mathbf{x}^{\mathbf{a}}$ and all its partial derivatives using Algorithmic Differentiation techniques.

related work

- 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.
- C. Bischof, N. Guertler, A. Kowartz, and A. Walther: **Parallel Reverse Mode Automatic Differentiation for OpenMP Programs with ADOL-C**. In *Advances in Automatic Differentiation*, pages 163–173, Springer 2008.

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 $y(8) = 1 + \frac{7}{9} \approx 1.778$.

Speelpenning product: reduce n^2 to $3n - 5$

A. Griewank & A. Walther, SIAM 2008

To evaluate $g(\mathbf{x}) = x_1x_2x_3x_4x_5$ and $\frac{\partial g}{\partial x_1}, \frac{\partial g}{\partial x_2}, \frac{\partial g}{\partial x_3}, \frac{\partial g}{\partial x_4}$, and $\frac{\partial g}{\partial x_5}$ at $\mathbf{z} = (z_1, z_2, z_3, z_4, z_5)$, we do

1. Store consecutive products in P ($4 \star$ operations):

$$P = (z_1 \mid z_1 \star z_2 \mid z_1z_2 \star z_3 \mid z_1z_2z_3 \star z_4 \mid z_1z_2z_3z_4 \star z_5).$$

2. Store products in reverse order in Q ($3 \star$ operations):

$$Q = (z_5 \mid z_4 \star z_5 \mid z_3 \star z_4z_5 \mid z_2 \star z_3z_4z_5).$$

3. We have $g(\mathbf{z}) = P_5$, $\frac{\partial g}{\partial x_5}(\mathbf{z}) = P_4$, and $\frac{\partial g}{\partial x_1}(\mathbf{z}) = Q_4$.

   Then $\frac{\partial g}{\partial x_2}(\mathbf{z}) = P_1 \star Q_3$, $\frac{\partial g}{\partial x_3}(\mathbf{z}) = P_2 \star Q_2$, and $\frac{\partial g}{\partial x_4}(\mathbf{z}) = P_3 \star Q_1$.

evaluation of monomials and their derivatives

Denote by k the number of variables appearing in a monomial $\mathbf{x}^{\mathbf{a}}$,
i.e.: $\mathbf{x}^{\mathbf{a}} = x_{i_1}^{a_{i_1}} x_{i_2}^{a_{i_2}} \cdots x_{i_k}^{a_{i_k}}$.

To evaluate $\mathbf{x}^{\mathbf{a}}$ and its k derivatives at $\mathbf{z}$:

1. Evaluate the factor $x_{i_1}^{a_{i_1}-1} x_{i_2}^{a_{i_2}-1} \cdots x_{i_k}^{a_{i_k}-1}$ at $\mathbf{z}$.
   If the powers of the variables are precomputed,
   evaluation of the factor takes $k-1$ multiplications.
2. Evaluate $x_{i_1} x_{i_2} \cdots x_{i_k}$ and all its derivatives at $\mathbf{z}$
   in $3k-5$ multiplications.
3. Multiply the evaluated $x_{i_1} x_{i_2} \cdots x_{i_k}$ and all its derivatives by the
   evaluated factor, additionally, multiply the i_jth derivative by a_{i_j}.
   This takes $2k+1$ multiplications.

Evaluating of $\mathbf{x}^{\mathbf{a}}$ and all its derivatives takes $6k-5$ multiplications.

multithreaded monomial evaluation and derivatives

Three stages:

1. Compute all powers of all variables: x_i^j, for $i = 1, 2, \ldots, n$, and $j = 1, 2, \ldots, d_i$.

   Each thread computes powers of a subset of the variables.

2. Evaluation of all monomials in the system and computation of all monomials of the derivatives of the monomials.

   Each tread computes for a subset of the monomials in the system all monomials of the derivatives.

3. Multiplication of the coefficients with the evaluated monomials.

   Distribution of the coefficient vectors of the polynomials may lead to an uneven work load, therefore a dynamic assignment is better.

timing for tracking one path, dimension 20

Elapsed real, user, system time, and speedup for tracking one path in complex quad double arithmetic on a system of dimension 20, once with quadrics, and once with polynomials of degree 10:

Dim=20, 20 monomials of degree 2 in a polynomial

#threads	real	user	sys	speedup
1	0m37.853s	0m37.795s	0m0.037s	1
2	0m21.094s	0m42.011s	0m0.063s	1.794
4	0m12.804s	0m50.812s	0m0.061s	2.956
8	0m 8.721s	1m 8.646s	0m0.097s	4.340

Dim=20, 20 monomials of degree 10 in a polynomial

#threads	real	user	sys	speedup
1	7m17.758s	7m17.617s	0m0.123s	1
2	3m42.742s	7m24.813s	0m0.206s	1.965
4	1m53.972s	7m34.386s	0m0.150s	3.841
8	0m59.742s	7m53.469s	0m0.279s	7.327

timings for tracking one path, dimension 40

Elapsed real, user, system time, and speedup for tracking one path in complex quad double arithmetic on a system of dimension 40, once with quadrics, and once with polynomials of degree 20:

Dim=40, 40 monomials of degree 2 in a polynomial

#threads	real	user	sys	speedup
1	5m25.509s	5m25.240s	0m0.254s	1
2	2m54.098s	5m47.506s	0m0.186s	1.870
4	1m38.316s	6m31.580s	0m0.206s	3.312
8	1m 2.257s	8m11.130s	0m0.352s	5.226

Dim=40, 40 monomials of degree 20 in a polynomial

#threads	real	user	sys	speedup
1	244m55.691s	244m48.501s	0m 6.621s	1
2	123m 1.536s	245m53.987s	0m 3.838s	1.991
4	61m53.447s	247m14.921s	0m 4.181s	3.958
8	32m22.671s	256m27.142s	0m11.541s	7.567

conclusions

For systems of polynomials with monomials of high degree, the cost of polynomial evaluation dominates.

For sparse polynomial systems, load balancing monomial evaluation and their derivative evaluation gives good performance.

For lower degrees, our parallel Gaussian elimination does not speedup so well and this impacts the entire speedup for tracking one path.

our papers

Available at our web sites:

- J. Verschelde and G. Yoffe: **Polynomial Homotopies on Multicore Workstations**. In the *Proceedings of the 4th International Workshop on Parallel Symbolic Computation (PASCO 2010)*, pages 131–140, ACM 2010.
- J. Verschelde and G. Yoffe: **Quality Up in Polynomial Homotopy Continuation by Multithreaded Path Tracking**. `arXiv:1109.0545v1 [cs.DC] 2 Sep 2011`.
- J. Verschelde and G. Yoffe: **Evaluating polynomials in several variables and their derivatives on a GPU computing processor**. `arXiv:1201.0499v1 [cs.MS] 2 Jan 2012`.