GPU Acceleration of Polynomial Homotopy Continuation

Jan Verschelde
joint work with Xiangcheng Yu

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

The SIAM conference on Parallel Processing for Scientific Computing
Université Pierre et Marie Curie, Paris, France, 12-15 April 2016

Outline

1. Polynomial Homotopy Continuation
   - compensating for the cost of extra precision
   - the problems with path tracking
2. Accelerated Path Tracking
   - monomial evaluation and differentiation
   - arithmetic circuits to evaluate and differentiate
   - tracking paths in Single Instruction Multiple Threads mode
3. Applications and Computational Results
   - hardware and software
   - speedup and quality up

polynomial homotopy continuation methods

$\mathbf{f}(\mathbf{x}) = \mathbf{0}$ is a polynomial system we want to solve,
$\mathbf{g}(\mathbf{x}) = \mathbf{0}$ is a start system ($\mathbf{g}$ is similar to $\mathbf{f}$) with known solutions.

A homotopy $\mathbf{h}(\mathbf{x}, t) = (1 - t)\mathbf{g}(\mathbf{x}) + t\,\mathbf{f}(\mathbf{x}) = \mathbf{0}$, $t \in [0, 1]$,
to solve $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ defines solution paths $\mathbf{x}(t)$: $\mathbf{h}(\mathbf{x}(t), t) \equiv \mathbf{0}$.

Numerical continuation methods track the paths $\mathbf{x}(t)$, from $t = 0$ to 1.

Problem statement: when solving large polynomial systems, the hardware double precision may not be sufficient for accurate solutions.

Our goal: accelerate computations with general purpose Graphics Processing Units (GPUs) to compensate for the overhead caused by double double and quad double arithmetic.

Our first results (jointly with Genady Yoffe) on this goal with multicore computers are in the PASCO 2010 proceedings; also at SIAM PP 2010, High Performance Symbolic Computing.

quad double precision

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

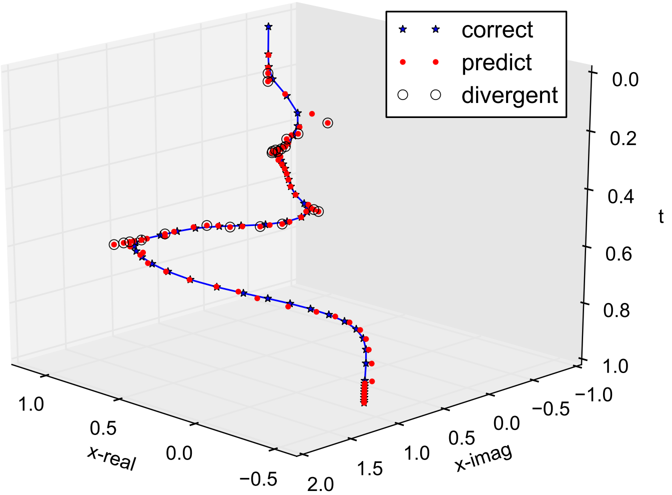
- Y. Hida, X.S. Li, and D.H. Bailey: **Algorithms for quad-double precision floating point arithmetic.** In the *15th IEEE Symposium on Computer Arithmetic*, pages 155–162. IEEE, 2001. Software at `http://crd.lbl.gov/~dhbailey/mpdist/qd-2.3.9.tar.gz`.

Predictable overhead: working with `double double` is of the same cost as working with complex numbers. Simple memory management.

The QD library has been ported to the GPU by

- M. Lu, B. He, and Q. Luo: **Supporting extended precision on graphics processors.** In the *Proceedings of the Sixth International Workshop on Data Management on New Hardware (DaMoN 2010)*, pages 19–26, 2010.
  Software at `http://code.google.com/p/gpuprec/`.

one coordinate of a solution path



Why is this difficult?

Tracking of one ***single*** path with the predictor-corrector method is a ***strictly sequential*** process.

Although we compute many points on a solution path, we cannot compute those points in parallel, independently from each other.

In order to move to the next point on the path,
the correction for the previous point must be completed.

This difficulty requires

- a fine granularity in the parallel algorithm; and
- a sufficiently high enough threshold on the dimension.

some related work in algorithmic differentiation

- M. Grabner, T. Pock, T. Gross, and B. Kainz. Automatic differentiation for GPU-accelerated 2D/3D registration. In *Advances in Automatic Differentiation*, pages 259–269. Springer, 2008.
- G. Kozikowski and B.J. Kubica. Interval arithmetic and automatic differentiation on GPU using OpenCL. In *PARA 2012*, LNCS 7782, pages 489-503, Springer 2013.

some related work in polynomial system solving

- R.A. Klopotek and J. Porter-Sobieraj. Solving systems of polynomial equations on a GPU. In *Preprints of the Federated Conference on Computer Science and Information Systems, September 9-12, 2012, Wroclaw, Poland*, pages 567–572, 2012.
- M.M. Maza and W. Pan. Solving bivariate polynomial systems on a GPU. *ACM Communications in Computer Algebra*, 45(2):127–128, 2011.

some related work in numerical linear algebra

- D. Mukunoki and D. Takashashi. Implementation and evaluation of triple precision BLAS subroutines on GPUs. In *Proceedings of PDSEC 2012*, pages 1372–1380. IEEE Computer Society, 2012.

accelerated predictor-corrector methods

A path tracker has three ingredients:

1. The predictor applies an extrapolation method for the next point. Each coordinate is predicted independently, linear cost in n.
2. The corrector applies a couple of steps with Newton's method. Denote by $J_{\mathbf{f}}$ the matrix of all partial derivatives of $\mathbf{f}$,

$$J_{\mathbf{f}}(\mathbf{x})\Delta\mathbf{x} = -\mathbf{f}(\mathbf{x}), \quad \mathbf{x} := \mathbf{x} + \Delta\mathbf{x}.$$

3. The adaptive step length control sets the value for the step size.

When tracking one path, the step length control can be done on the host, as only some doubles are needed in the transfer.

- The device computes $\|\Delta\mathbf{x}\|$ and $\|\mathbf{f}(\mathbf{x})\|$; and then sends $\|\Delta\mathbf{x}\|$ and $\|\mathbf{f}(\mathbf{x})\|$ to the host.
- The host computes a new value for the step size Δt; and sends Δt to device.

polynomial evaluation and differentiation

We distinguish three stages:

1. Common factors and tables of power products:

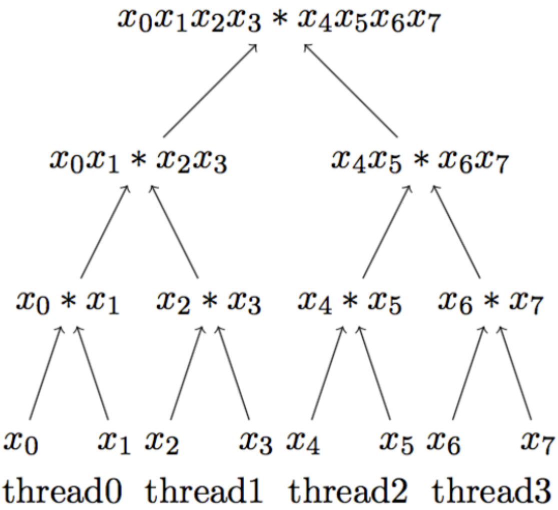
$$x_1^{d_1} x_2^{d_2} \cdots x_n^{d_n} = x_{i_1} x_{i_2} \cdots x_{i_k} \times x_{j_1}^{e_{j_1}} x_{j_2}^{e_{j_2}} \cdots x_{j_\ell}^{e_{j_\ell}}$$

   The factor $x_{j_1}^{e_{j_1}} x_{j_2}^{e_{j_2}} \cdots x_{j_\ell}^{e_{j_\ell}}$ is common to all partial derivatives.
   The factors are evaluated as products of pure powers of the variables, computed in shared memory by each block of threads.

2. Evaluation and differentiation of products of variables:
   Computing the gradient of $x_1 x_2 \cdots x_n$ with the reverse mode of algorithmic differentiation requires $3n - 5$ multiplications.

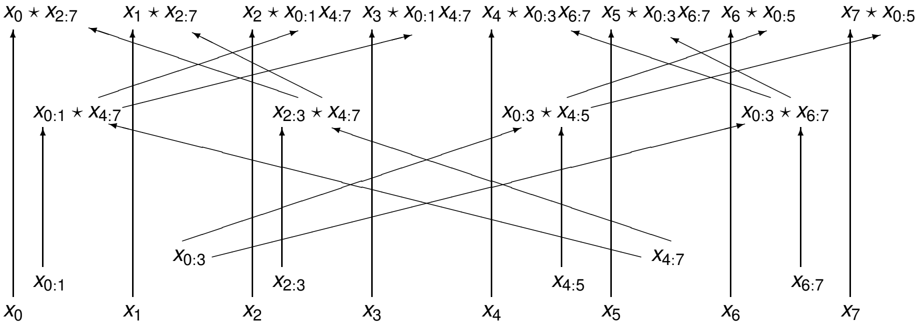
3. Coefficient multiplication and term summation.
   Summation jobs are ordered by the number of terms so each warp has the same amount of terms to sum.

collaborating threads – a parallel scan



an arithmetic circuit for the gradient of $x_0 x_1 \cdots x_7$

Denote by $x_{i:j}$ the product $x_i \star \cdots \star x_k \star \cdots \star x_j$ for all k between i and j.



The computation of the gradient of $x_0 x_1 \cdots x_{n-1}$ requires

- $2n - 4$ multiplications, and
- $n - 1$ extra memory locations.

SIMT Path Tracking (SIMT = Single Instruction Multiple Threads)

We run the same arithmetic circuit at different data:

- threads are evaluating and differentiation the same polynomials,
- at different approximations for the solutions along the path.

path0	path1	path2
`predict`	`predict`	`predict`
`evaluate` `correct`	`evaluate` `correct`	`evaluate` `correct`
`evaluate` `correct`		`evaluate` `correct`
		`evaluate` `correct`

The three stages in a predictor-corrector algorithm are:

1. `predict`: apply extrapolation to predict the next approximation,
2. `evaluate`: evaluate and differentiate the polynomials in the system,
3. `correct`: solve a linear system in Newton's method.

ordering the jobs to track paths

In order for approximations to reach the required accuracy,
some paths need two or three steps in Newton's method.

The labels to the jobs correspond to the indices of each path:

path0	path1	path2
`predict`	`predict`	`predict`
`evaluate` `correct`	`evaluate` `correct`	`evaluate` `correct`
`evaluate` `correct`		`evaluate` `correct`
		`evaluate` `correct`

job0	job1	job2
`predict0`	`predict1`	`predict2`
`evaluate0` `correct0`	`evaluate1` `correct1`	`evaluate2` `correct2`
`evaluate0` `correct0`	`evaluate2` `correct2`	
`evaluate2` `correct2`		

Every path has its own current value of the continuation parameter t.

The adaptive step size control is executed on the device.

The length of the total execution time is bounded from below
by the time required for the most difficult path.

hardware and software

Our NVIDIA Tesla K20C, has 2496 cores with a clock speed of 706 MHz, is hosted by a Red Hat Enterprise Linux workstation of Microway, with Intel Xeon E5-2670 processors at 2.6 GHz.

We implemented the path tracker with the gcc compiler and version 6.5 of the CUDA Toolkit, compiled with the optimization flag `-O2`.

The code is free and open source, at github.
`https://github.com/janverschelde/PHCpack`

The benchmark data were prepared with `phcpy`,
the Python interface to PHCpack.

The GPU accelerated path trackers are accessible to the Python programmer via `phcpy`.

applications: the polynomial systems

Summarizing the characteristics:

name	dim	#paths	#monomials
`cyclic10`	10	34,940	92
`nash8`	8	14,833	1,040
`pieri44`	16	24,024	3,936

Application areas:

- `cyclic10`: study of biunimodular vectors,
- `nash8`: totally mixed Nash equilibria in a game,
- `pieri44`: a problem from enumerative geometry.

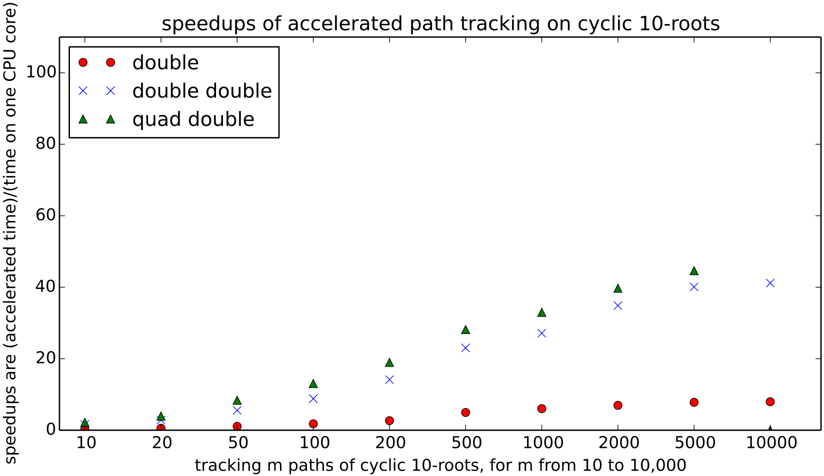
tracking paths for cyclic 10-roots

Times in seconds and speedups for tracking a number of paths of the cyclic 10-roots system.

	complex double			complex double double		
#paths	CPU	GPU	speedup	CPU	GPU	speedup
10	0.040	0.128	0.31	0.563	0.344	1.63
20	0.075	0.139	0.54	1.082	0.386	2.80
50	0.158	0.147	1.07	2.248	0.404	5.56
100	0.277	0.155	1.79	3.706	0.421	8.81
200	0.482	0.181	2.67	6.480	0.458	14.15
500	1.239	0.250	4.96	16.802	0.729	23.05
1000	2.609	0.432	6.03	35.683	1.315	27.14
2000	5.341	0.768	6.96	83.601	2.397	34.87
5000	**13.358**	1.711	7.81	210.287	5.246	40.09
10000	26.562	3.334	7.97	414.332	**10.063**	41.18

Quality Up!

visualization of the speedups



GPU Acceleration with Python

To compute *all* isolated solutions of the the cyclic 10-roots problem (`cyc10`), we need to track 35,940 solution paths.

The Python scripting interface to PHCpack is `phcpy`:

```
from phcpy.trackers import gpu_double_track
cyc10sols = gpu_double_track(cyc10, cyc10q, \
    cyc10qsols, verbose=0)
```

To time the execution, at the command prompt we type

```
$ time python runcyc10d.py > /tmp/output
```

.

precision	`real`	`user`	`sys`
double	14.980s	11.683s	3.192s
double double	45.266s	35.897s	9.228s
quad double	6m 57.368s	5m 23.224s	1m 33.266s

conclusions

We can compensate for the cost of double double arithmetic when tracking one solution path with GPU acceleration.

Double double and quad double arithmetic (using QD):

- ***memory bound*** for double and (real) double double arithmetic,
- ***compute bound*** for complex double doubles and quad doubles.

Double digit speedups $\Rightarrow$ *double the precision, compute twice as fast.*

We achieve not only speedup, but also *quality up*, and in some hard cases double precision is insufficient for a successful path tracking.

our papers

- **Solving polynomial systems in the cloud with polynomial homotopy continuation.** (with N. Bliss, J. Sommars, and X. Yu). *Proceedings of CASC 2015.* `arXiv:1506.02618`
- **Evaluating polynomials in several variables and their derivatives on a GPU computing processor.** (with G. Yoffe). *Proceedings of PDSEC 2012.* `arXiv:1201.0499`
- **Orthogonalization on a general purpose graphics processing unit with double double and quad double arithmetic.** (with G. Yoffe). *Proceedings of PDSEC 2013.* `arXiv:1210.0800`
- **Acceleration of Newton's method for large systems of polynomial equations in double double and quad double arithmetic.** (with X. Yu). *Proceedings of HPCC 2014.* `arXiv:1402.2626`
- **Accelerating polynomial homotopy continuation on a graphics processing unit with double double and quad double arithmetic.** (with X. Yu). *Proceedings of PASCO 2015.* `arXiv:1501.06625`
- **Tracking many solution paths of a polynomial homotopy on a graphics processing unit with double double and quad double arithmetic.** (with X. Yu). *Proceedings of HPCC 2015.* `arXiv:1505.00383`