Accelerating Polynomial Homotopy Continuation on a Graphics Processing Unit with Double Double and Quad Double Arithmetic

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 7th International Workshop on Parallel Symbolic Computation
the University of Bath, Bath, UK, 10-11 July 2015

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
   - accelerated predictor-corrector methods
3. Applications and Computational Results
   - hardware and software
   - matrix completion with Pieri homotopies
   - monodromy loops on cyclic *n*-roots

GPU Accelerated Path Tracking

1. Polynomial Homotopy Continuation
   - compensating for the cost of extra precision
   - the problems with path tracking
2. Accelerated Path Tracking
   - monomial evaluation and differentiation
   - accelerated predictor-corrector methods
3. Applications and Computational Results
   - hardware and software
   - matrix completion with Pieri homotopies
   - monodromy loops on cyclic n-roots

polynomial homotopy continuation methods

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

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

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: ccelerate 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.

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.

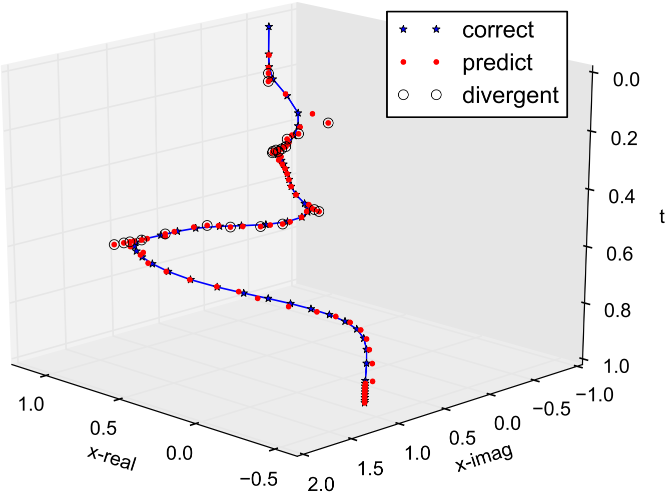
The QD library has been ported to the GPU by

- 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/`.

GPU Accelerated Path Tracking

1. Polynomial Homotopy Continuation
   - compensating for the cost of extra precision
   - the problems with path tracking
2. Accelerated Path Tracking
   - monomial evaluation and differentiation
   - accelerated predictor-corrector methods
3. Applications and Computational Results
   - hardware and software
   - matrix completion with Pieri homotopies
   - monodromy loops on cyclic n-roots

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.

GPU Accelerated Path Tracking

1. Polynomial Homotopy Continuation
   - compensating for the cost of extra precision
   - the problems with path tracking

2. Accelerated Path Tracking
   - monomial evaluation and differentiation
   - accelerated predictor-corrector methods

3. Applications and Computational Results
   - hardware and software
   - matrix completion with Pieri homotopies
   - monodromy loops on cyclic n-roots

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.

monomial evaluation and differentiation

Evaluating four monomials $x_0x_1x_2$, $x_3x_4x_5$, $x_2x_3x_4x_5$, $x_0x_1x_3x_4x_5$.
The tidx in the tables below stands for thread index.

tidx	0	1	2	3
m_{tidx}	$x_0x_1x_2$	$x_3x_4x_5$	$x_2x_3x_4x_5$	$x_0x_1x_3x_4x_5$
$\dfrac{\partial m_{tidx}}{\partial x_j}$	x_0 $x_0 \star x_1$	x_3 $x_3 \star x_4$	x_2 $x_2 \star x_3$ $x_2x_3 \star x_4$	x_1 $x_1 \star x_2$ $x_1x_2 \star x_3$ $x_1x_2x_3 \star x_4$

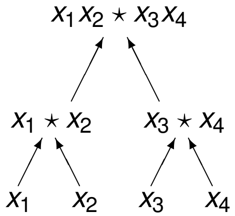
the forward calculation, from the top to bottom row

tidx	0	1	2	3
m_{tidx}	$x_0x_1x_2$	$x_3x_4x_5$	$x_2x_3x_4x_5$	$x_0x_1x_3x_4x_5$
$\dfrac{\partial m_{tidx}}{\partial x_j}$	$x_1 \star x_2$ $x_0 \star x_2$ x_0x_1	$x_3 \star x_4x_5$ $x_3 \star x_5$ x_3x_4	$x_3 \star x_4x_5$ $x_2 \star (x_4 \star x_5)$ $x_2x_3 \star x_5$ $x_2x_3x_4$	$x_2 \star x_3x_4x_5$ $x_1 \star (x_3 \star x_4x_5)$ $x_1x_2 \star (x_4 \star x_5)$ $x_1x_2x_3 \star (x_5)$ $x_1x_2x_3x_4$

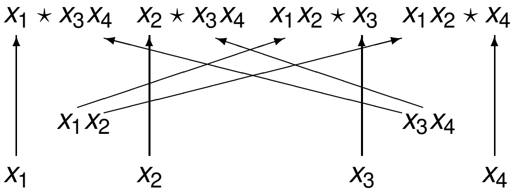
backward and cross products, from bottom to top row

arithmetic circuits in tree mode

First to evaluate the product:

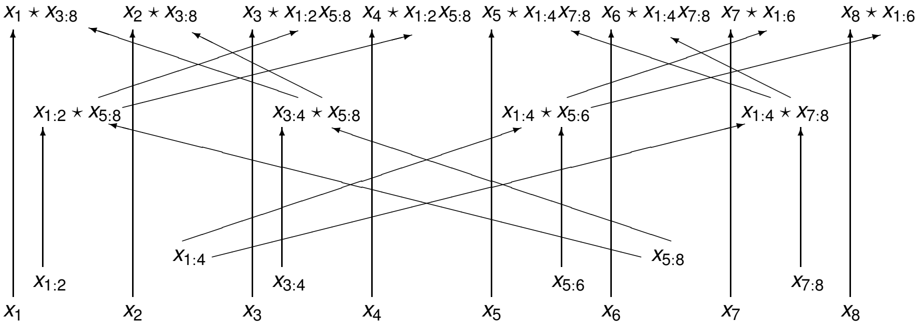


and (storing $x_1 x_2$ and $x_3 x_4$) then to compute the gradient:



computing the gradient of $x_1 x_2 \cdots x_8$

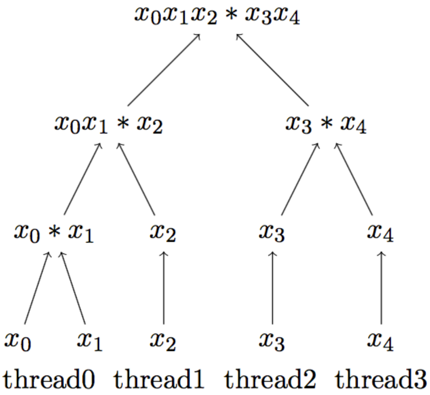
Denote by $x_{i:j}$ the product $x_i \star \cdots \star x_k \star \cdots \star x_j$ for all k between i and j.



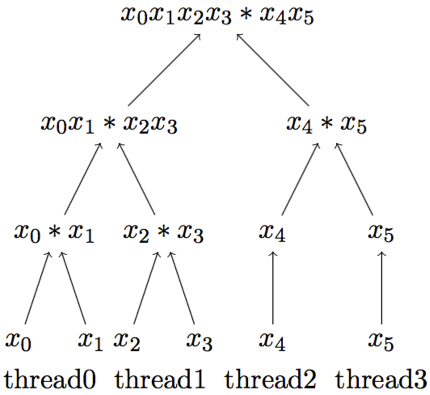
The computation of the gradient of $x_1 x_2 \cdots x_n$ requires

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

collaborating threads

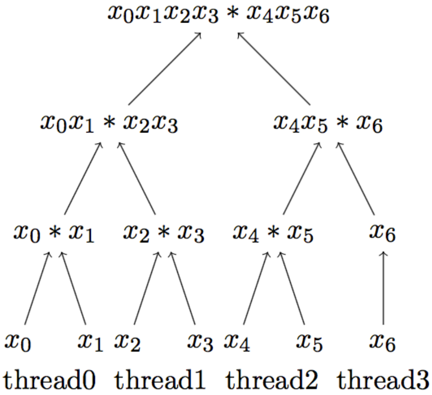


(a) Evaluate $x_0x_1x_2x_3x_4$

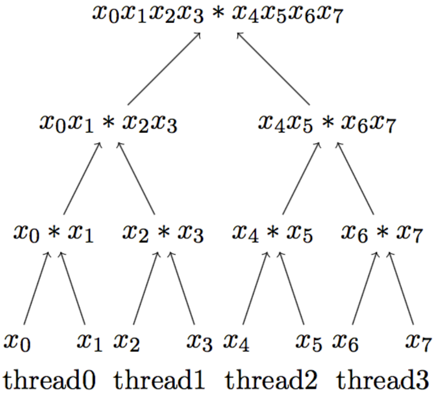


(b) Evaluate $x_0x_1x_2x_3x_4x_5$

collaborating threads – continued



(c) Evaluate $x_0x_1x_2x_3x_4x_5x_6$



(d) Evaluate $x_0x_1x_2x_3x_4x_5x_6x_7$

GPU Accelerated Path Tracking

1. Polynomial Homotopy Continuation
   - compensating for the cost of extra precision
   - the problems with path tracking
2. Accelerated Path Tracking
   - monomial evaluation and differentiation
   - accelerated predictor-corrector methods
3. Applications and Computational Results
   - hardware and software
   - matrix completion with Pieri homotopies
   - monodromy loops on cyclic n-roots

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.

accelerated tracking of one single path

Input: *Inst*, Polynomial Instructions
W, GPU Workspace
P, parameters for path tracker
Output: *success* or *fail*
$W.\mathbf{x}$, solution for $t = 1$ if *success*

$t = 0$
$\Delta t = P.max\Delta t$
$\#successes = 0$
$\#steps = 0$
while $t < 1$ do
  if ($\#steps > P.max\#steps$) then return *fail*
  $t = \min(1, t + \Delta t)$
  copy t from host to GPU
  launch kernel `predict`($W.\mathbf{x}$_*array*, $W.t$_*array*)
  newton_success = `GPU_Newton`(*Inst*,*W*,*P*)

adaptive step control

if (*newton_success*) then
    update pointer of $W.\mathbf{x}$ in $W.\mathbf{x}$_*array*
    $\#$*successes* = $\#$*successes* + 1
    if ($\#$*successes* > 2) then
        $\Delta t = \min(2\Delta t, P.max\Delta t)$
else
    $\#successes = 0$
    $\Delta t = \Delta t/2$
$\#steps = \#steps + 1$
return *success*

accelerated Newton's method

Input: *Inst*, Polynomial Instructions
W, GPU Workspace
P, parameters for Newton's method
Output: *success* or *fail*
updated *W*.**x**

last_max_eq_val = *P*.*max_eq_val*
for *k* from 1 to *P*.*max_iteration* do
  `GPU_Eval`(*Inst*,*W*)
  launch kernel `Max_Array`(*W*.*eq_val*, *max_eq_val*)
    with single block
  copy *max_eq_val* from GPU to host
  if (*max_eq_val* $>$ *last_max_eq_val*) then return *fail*
  if (*max_eq_val* $<$ *P*.*tolerance*) then return *success*

accelerated Newton's method continued

```
GPU_Modified_Gram_Schmidt(W)
launch kernel Max_Array(W.Δx, max_Δx)
   with single block
copy max_Δx from GPU to host
launch kernel Update_x(W.x,W.Δx)
if (max_Δx < P.tolerance) then return success
last_max_eq_val = max_eq_val
return fail
```

A right-looking algorithm to implement the modified Gram-Schmidt method provides the most thread parallelism.

GPU Accelerated Path Tracking

1. Polynomial Homotopy Continuation
   - compensating for the cost of extra precision
   - the problems with path tracking
2. Accelerated Path Tracking
   - monomial evaluation and differentiation
   - accelerated predictor-corrector methods
3. Applications and Computational Results
   - hardware and software
   - matrix completion with Pieri homotopies
   - monodromy loops on cyclic n-roots

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.
The benchmark data were prepared with `phcpy`,
the Python interface to PHCpack.
The scripts `backelin.py` and `pierisystems.py` are in the `examples` folder of the `phcpy` distribution.

GPU Accelerated Path Tracking

1. Polynomial Homotopy Continuation
   - compensating for the cost of extra precision
   - the problems with path tracking
2. Accelerated Path Tracking
   - monomial evaluation and differentiation
   - accelerated predictor-corrector methods
3. Applications and Computational Results
   - hardware and software
   - matrix completion with Pieri homotopies
   - monodromy loops on cyclic n-roots

matrix completion with Pieri homotopies

The polynomial equations arise from minor expansions on

$$\det(A|X) = 0, \quad A \in \mathbb{C}^{n \times m},$$

and where X is an n-by-p matrix ($m + p = n$) of unknowns.

For example, a 2-plane in complex 4-space (or equivalently, a line in projective 3-space) is represented as

$$X = \begin{bmatrix} 1 & 0 \\ x_{2,1} & 1 \\ x_{2,2} & x_{3,2} \\ 0 & x_{4,2} \end{bmatrix}.$$

To determine for the four unknowns in X we need four equations, which via expansion results in four quadratic equations.

Pieri homotopies

Sequences of homotopies build up the solution:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & x_{3,2} \\ 0 & 0 \end{bmatrix}, \quad \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & x_{3,2} \\ 0 & x_{4,2} \end{bmatrix}, \quad \begin{bmatrix} 1 & 0 \\ x_{2,1} & 1 \\ 0 & x_{3,2} \\ 0 & x_{4,2} \end{bmatrix}, \quad \begin{bmatrix} 1 & 0 \\ x_{2,1} & 1 \\ x_{3,1} & x_{3,2} \\ 0 & x_{4,2} \end{bmatrix}.$$

Pieri homotopies are defined as, for k from 1 to $m \times p$:

$$\mathbf{h}(\mathbf{x}, t) = \left\{ \begin{array}{l} \det(A^{(i)}|X) = 0, \ i = 1, 2, \ldots, k-1, \\ t \det(A^{(k)}|X) + (1-t)\det(S_X|X) = 0. \end{array} \right.$$

Tracking one single path, we start a a linear system and gradually, the polynomials increase in degree and the cost of evaluation and differentiation dominates.

Pieri homotopies with double double arithmetic

n : dimension
m : number of predictor-corrector stages

Tracking one path of a 3-plane in 35-space,
with complex double double arithmetic, times are in seconds:

n	m	cpu	gpu	S	n	m	cpu	gpu	S
32	25	0.05	0.05	1.1	68	239	50.0	3.2	15.8
36	36	0.40	0.21	1.9	72	89	41.8	1.8	23.2
40	19	0.39	0.15	2.6	76	246	136.0	4.6	29.8
44	23	0.61	0.19	3.2	80	226	158.0	5.5	28.7
48	53	1.69	0.45	3.8	84	69	59.2	2.0	29.6
52	79	2.97	0.71	4.2	88	36	37.3	1.2	30.2
56	28	1.37	0.29	4.7	92	90	98.0	3.2	30.3
60	111	5.70	1.13	5.1	96	64	67.2	2.2	30.8
64	63	3.36	0.62	5.4					

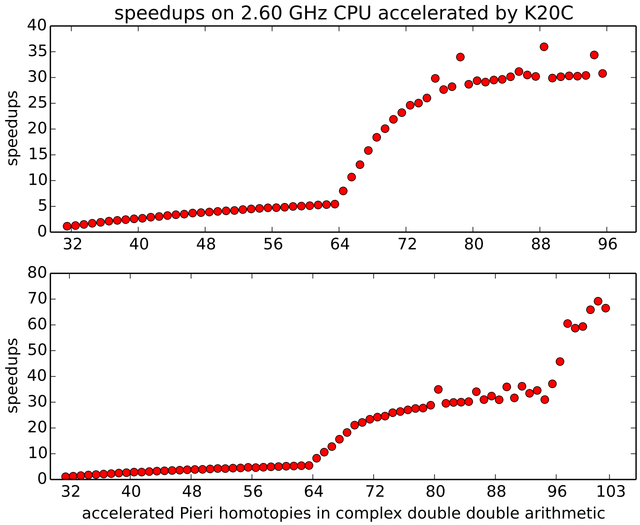
tracking a 4-plane in 36-space with double doubles

n : dimension
m : number of predictor-corrector stages

Tracking one path of a 4-plane in 36-space,
with complex double double arithmetic, times are in seconds:

n	m	cpu	gpu	S	n	m	cpu	gpu	S
32	16	0.03	0.03	1.1	68	51	11.3	0.7	15.6
36	18	0.20	0.11	1.9	72	39	19.4	0.8	23.4
40	12	0.25	0.09	2.6	76	63	38.8	1.5	26.4
44	10	0.28	0.08	3.2	80	157	115.0	4.0	28.8
48	16	0.54	0.14	3.8	84	19	17.3	0.6	30.0
52	40	1.52	0.36	4.3	88	574	431.2	13.3	32.4
56	36	1.71	0.36	4.7	92	250	219.8	6.1	36.2
60	16	0.95	0.19	5.0	96	190	230.8	6.2	37.1
64	75	4.54	0.84	5.4	100	242	1367.2	23.0	59.4
65	83	7.93	0.96	8.3	101	809	4655.7	70.7	65.9
66	195	27.20	2.56	10.6	102	1016	5231.7	75.6	69.2
67	154	26.43	2.07	12.8	103	375	2923.2	44.0	66.5

the speedups on the Pieri homotopies



GPU Accelerated Path Tracking

1. Polynomial Homotopy Continuation
   - compensating for the cost of extra precision
   - the problems with path tracking
2. Accelerated Path Tracking
   - monomial evaluation and differentiation
   - accelerated predictor-corrector methods
3. Applications and Computational Results
   - hardware and software
   - matrix completion with Pieri homotopies
   - monodromy loops on cyclic n-roots

loops of solution paths

Consider a sequence of homotopies:

$$\mathbf{h}_\alpha(\mathbf{x},t) = \begin{cases} \mathbf{f}(\mathbf{x}) = \mathbf{0} \\ \alpha(1-t)\mathbf{L}(\mathbf{x}) + t\mathbf{K}(\mathbf{x}) = \mathbf{0} \end{cases} \quad (1)$$

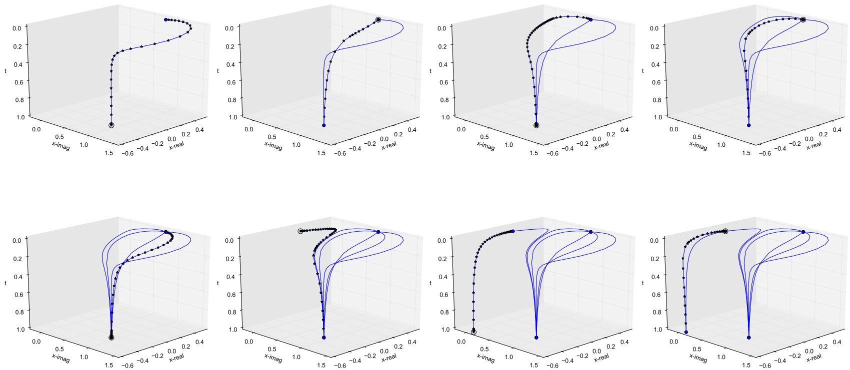
$$\mathbf{h}_\beta(\mathbf{x},t) = \begin{cases} \mathbf{f}(\mathbf{x}) = \mathbf{0} \\ \beta(1-t)\mathbf{K}(\mathbf{x}) + t\mathbf{L}(\mathbf{x}) = \mathbf{0} \end{cases} \quad (2)$$

where

- $\mathbf{K}(\mathbf{x}) = \mathbf{0}$ is as $\mathbf{L}(\mathbf{x}) = \mathbf{0}$ another set of linear equations with different random coefficients; and
- α and β are different random complex constants.

One loop consists in tracking one path defined by $\mathbf{h}_\alpha(\mathbf{x},t) = \mathbf{0}$ and $\mathbf{h}_\beta(\mathbf{x},t) = \mathbf{0}$. In both cases t goes from 0 to 1.

visualization of monodromy on a quadratic set



the cyclic *n*-roots problem

Denoted the system by $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, $\mathbf{f} = (f_1, f_2, \ldots, f_n)$, with

$$\begin{aligned}
f_1 &= x_0 + x_1 + \cdots + x_{n-1}, \\
f_2 &= x_0 x_1 + x_1 x_2 + \cdots + x_{n-2} x_{n-1} + x_{n-1} x_0, \\
f_i &= \sum_{j=0}^{n-1} \prod_{k=j}^{j+i-1} x_{k \bmod n}, i = 3, 4, \ldots, n-1, \\
f_n &= x_0 x_1 x_2 \cdots x_{n-1} - 1.
\end{aligned}$$

Observe the increase of the degrees: $\deg(f_k) = k$.

High degrees are a likely cause of large roundoff errors.

a tropical version of Backelin's Lemma, CASC 2013

Lemma (Tropical Version of Backelin's Lemma (Adrovic-V.))

For $n = m^2\ell$, *where* $\ell \in \mathbb{N} \setminus \{0\}$ *and* ℓ *is no multiple of* k^2, *for* $k \geq 2$, *there is an* $(m-1)$*-dimensional set of cyclic n-roots, represented exactly as*

$$
\begin{array}{rcl}
x_{km+0} & = & u^k t_0 \\
x_{km+1} & = & u^k t_0 t_1 \\
x_{km+2} & = & u^k t_0 t_1 t_2 \\
& \vdots & \\
x_{km+m-2} & = & u^k t_0 t_1 t_2 \cdots t_{m-2} \\
x_{km+m-1} & = & \gamma u^k t_0^{-m+1} t_1^{-m+2} \cdots t_{m-3}^{-2} t_{m-2}^{-1}
\end{array}
$$

for $k = 0, 1, 2, \ldots, m-1$, *free parameters* $t_0, t_1, \ldots, t_{m-2}$, *constants* $u = e^{\frac{i2\pi}{m\ell}}$, $\gamma = e^{\frac{i\pi\beta}{m\ell}}$, *with* $\beta = (\alpha \mod 2)$, *and* $\alpha = m(m\ell - 1)$.

positive dimensional cyclic n-roots

Backelin's Lemma states that there are cyclic n-roots of dimension $m - 1$ for $n = \ell m^2$, where ℓ is no multiple of k^2, for $k \geq 2$.

To compute the degree of the sets, we add to $\mathbf{f}$ as many linear equations $\mathbf{L}$ (with random complex coefficients) as the dimension of the set and count the number of solutions of the system $\mathbf{f}(\mathbf{x}) = \mathbf{0}$:

$$\begin{cases} \mathbf{f}(\mathbf{x}) = \mathbf{0} \\ \mathbf{L}(\mathbf{x}) = \mathbf{0}. \end{cases}$$

The degree $d = m$ for $n = m^2$ and this result extends for $n = \ell m^2$.

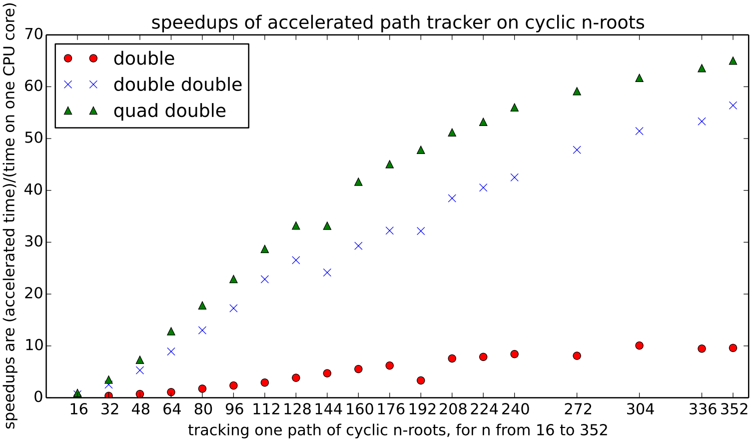
n	16	32	48	64	80	96	128	144	160	176
d	4	4	4	8	4	4	8	12	4	4

n	192	208	240	256	272	288	304	320	336	352
d	8	4	4	16	4	12	4	8	4	4

times in seconds on tracking one cyclic n-roots path

	complex double					complex double double					complex quad double				
n	s	m	cpu	gpu	S	s	m	cpu	gpu	S	s	m	cpu	gpu	S
16	1	32	0.00	0.03	0.14	1	20	0.04	0.06	0.65	1	32	0.48	0.52	0.92
32	1	100	0.06	0.16	0.35	1	79	1.03	0.41	2.53	1	100	12.66	3.62	3.50
48	1	103	0.17	0.24	0.72	1	78	3.23	0.61	5.29	1	103	39.46	5.39	7.32
64	0	987	4.48	4.15	1.08	1	225	22.94	2.57	8.92	1	987	229.99	17.93	12.83
80	1	99	0.73	0.42	1.74	1	75	14.96	1.15	13.01	1	99	180.37	10.13	17.81
96	1	95	1.23	0.52	2.36	1	69	23.17	1.34	17.26	1	95	289.38	12.64	22.90
112	1	171	3.42	1.17	2.92	1	121	68.07	2.98	22.86	1	171	813.91	28.36	28.70
128	1	162	5.66	1.47	3.85	1	123	102.94	3.88	26.54	1	162	1253.82	37.75	33.21
144	0	214	12.58	2.67	4.72	0	1500	1487.86	61.59	24.16	1	214	15898.67	479.18	33.18
160	1	68	4.84	0.87	5.53	1	49	83.11	2.84	29.31	1	68	998.43	23.96	41.67
176	1	160	15.65	2.52	6.21	1	118	259.80	8.06	32.24	1	160	3179.81	70.58	45.05
192	0	246	30.92	9.27	3.34	1	150	419.16	13.03	32.16	1	246	5054.70	105.69	47.83
208	1	231	39.51	5.22	7.57	1	168	628.46	16.33	38.48	1	231	7529.02	147.09	51.19
224	1	96	19.39	2.46	7.88	1	71	319.27	7.88	40.54	1	96	3925.33	73.76	53.22
240	1	140	34.04	4.04	8.42	1	96	531.01	12.49	42.50	1	140	6714.01	119.86	56.01
256	0	0	0.00	1.00	0.00	0	0	0.00	1.00	0.00	0	0	0.00	1.00	0.00
272	1	160	58.19	7.19	8.09	1	118	914.24	19.12	47.82	1	160	10829.36	183.12	59.14
288	0	0	0.00	1.00	0.00	0	0	0.00	1.00	0.00	0	0	0.00	1.00	0.00
304	1	142	81.04	8.05	10.07	1	103	1176.29	22.87	51.44	1	142	13992.60	226.78	61.70
320	0	0	0.00	1.00	0.00	0	0	0.00	1.00	0.00	0	0	0.00	1.00	0.00
336	1	157	105.30	11.12	9.47	1	114	1772.97	33.26	53.31	1	157	20807.27	327.25	63.58
352	1	121	93.89	9.78	9.60	1	90	1621.15	28.75	56.39	1	121	18881.13	290.36	65.03

speedups on one path of cyclic *n*-roots



In double precision, the dimensions are too small for good speedups. Double digits speedups in double double and quad double precision are achieved once $n = 64$.

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.

For *many* solution paths:
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`