Polynomial Homotopy Continuation on Graphics Processing Units

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

Graduate Computational Algebraic Geometry Seminar
8 September 2016

Polynomial Homotopy Continuation on GPUs

1. Quality Up by GPU Acceleration
   - graphics processing units
   - polynomial homotopy continuation
   - double double and quad double precision

2. GPU Accelerated Path Trackers
   - evaluation and differentiation
   - arithmetic circuits
   - granularity issues

3. Software
   - from standalone programs to production code
   - navigating the code

Polynomial Homotopy Continuation on GPUs

1. Quality Up by GPU Acceleration
   - graphics processing units
   - polynomial homotopy continuation
   - double double and quad double precision

2. GPU Accelerated Path Trackers
   - evaluation and differentiation
   - arithmetic circuits
   - granularity issues

3. Software
   - from standalone programs to production code
   - navigating the code

graphics processing units

NVIDIA Tesla K20 “Kepler” C-class Accelerator

- 2,496 CUDA cores, 2,496 = 13 SM $\times$ 192 cores/SM
- 5GB Memory at 208 GB/sec peak bandwidth
- peak performance: 1.17 TFLOPS double precision

NVIDIA Tesla P100 16GB “Pascal” Accelerator

- 3,586 CUDA cores, 3,586 = 56 SM $\times$ 64 cores/SM
- 16GB Memory at 720GB/sec peak bandwidth
- peak performance: 4.7 TFLOPS double precision

Programming model: Single Instruction Multiple Data (SIMD).

- Data parallelism: blocks of threads read from memory, execute the same instruction(s), write to memory.
- Massively parallel: need 10,000 threads for full occupancy.

polynomial homotopy continuation

$\mathbf{f}(\mathbf{x}) = \mathbf{0}$ 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.

Newton's method is the most computationally intensive stage:

1. Evaluation and differentiation of all polynomials in the system.
2. Solve a linear system for the update to the approximate solution.

Bootstrapping to solve a start system $\mathbf{g}(\mathbf{x}) = \mathbf{0}$:

- Random coefficients of $\mathbf{g}$ imply that all solutions are regular.
- Polyhedral homotopies deform $\mathbf{g}$ to 2-nomial systems.

double double and quad double precision

Larger problems give rise to larger condition numbers.

- Standard hardware double precision is no longer sufficient to obtain accurate and reliable results.
- Higher precision arithmetic as defined by software package causes a large overhead both on time and memory.

With GPU acceleration we can compensate for the overhead for the higher precision arithmetic and achieve ***quality up***.

A double double is a sequence of two doubles

- a high part with the most significant bits;
- a low part with the least significant bits.

We double the precision with predictable overhead:

- twice the amount of storage, compared to a double;
- comparable overhead to complex arithmetic (factor of 5 to 8).

software for GPU accelerated path tracking

Double double and quad double arithmetic:

- QDlib by Y. Hida, X.S. Li, and D.H. Bailey, 2001.
- GQD, a CUDA library version of QDlib,
  by M. Lu, B. He, and Q. Luo, 2010.

Proof-of-concept implementation on random data, with Genady Yoffe:

- Evaluation and differentiation in PDSEC 2012.
- Modified Gram-Schmidt orthogonalization in PDSEC 2013.

Generalized to benchmark polynomial systems, with Xiangcheng Yu.

- Newton's method, in HPCC 2014.
- Tracking one path of large systems, in PASCO 2015.
- Tracking all paths of benchmark problems, in HPCC 2015.
- Wrapped in Python (phcpy), in ACM Comm. Comp. Alg., 2015.

problem statement

Conclusions of the GPU accelerated path trackers:

- double double real arithmetic is memory bound,
  working with complex double doubles is compute bound.
- to occupy the GPU well, scale the problems:
  - need at least 10,000 paths, e.g.: cyclic 10-roots; or
  - need at least 10,000 monomials, e.g.: cyclic 100-roots.

A blackbox solver to solve a polynomial system $\mathbf{f}(\mathbf{x}) = \mathbf{0}$:

1. compute the mixed volume of the Newton polytopes;
2. polyhedral homotopies solve a random coefficient system,
   $\mathbf{g}(\mathbf{x}) = \mathbf{0}$ has the same Newton polytopes as $\mathbf{f}(\mathbf{x}) = \mathbf{0}$
3. track paths defined by $(1 - t)\mathbf{g}(\mathbf{x}) + t\mathbf{f}(\mathbf{x}) = \mathbf{0}$, $t \in [0, 1]$.

`phc -b -t` applies pipelining to multithreaded path trackers.

Problem: no GPU accelerated path trackers for polyhedral homotopies.

Polynomial Homotopy Continuation on GPUs

1. Quality Up by GPU Acceleration
   - graphics processing units
   - polynomial homotopy continuation
   - double double and quad double precision

2. GPU Accelerated Path Trackers
   - evaluation and differentiation
   - arithmetic circuits
   - granularity issues

3. Software
   - from standalone programs to production code
   - navigating the code

evaluating polyhedral homotopies

Multi-index notation, consider

- n variables $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, and
- n exponents $\mathbf{a} = (a_1, a_2, \ldots, a_n) \in \mathbb{Z}^n$,

then $\mathbf{x}^\mathbf{a} = x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}$.

A polynomial $f(\mathbf{x})$ is supported on the set A:

$$f(\mathbf{x}) = \sum_{\mathbf{a} \in A} c_\mathbf{a} \mathbf{x}^\mathbf{a}, \quad c_\mathbf{a} \in \mathbb{C} \setminus \{0\}.$$

A polynomial in a polyhedral homotopy has the form

$$h(\mathbf{x}, t) = c_{\mathbf{b}_1} \mathbf{x}^{\mathbf{b}_1} + c_{\mathbf{b}_2} \mathbf{x}^{\mathbf{b}_2} + \sum_{\mathbf{a} \in A \setminus \{\mathbf{b}_1, \mathbf{b}_2\}} c_\mathbf{a} \mathbf{x}^\mathbf{a} t^{e_\mathbf{a}}, \quad t^{e_\mathbf{a}} > 0.$$

At $t = 0$, $h(\mathbf{x}, 0)$ is a 2-nomial: $c_{\mathbf{b}_1} \mathbf{x}^{\mathbf{b}_1} + c_{\mathbf{b}_2} \mathbf{x}^{\mathbf{b}_2}$.

polynomial evaluation and differentiation

We distinguish three stages:

1. Common factors and tables of power products:

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

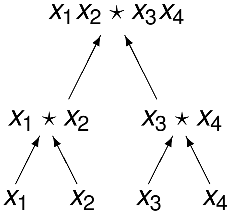
   The factor $x_{j_1}^{e_{j_1}} x_{j_2}^{e_{j_2}} \cdots x_{j_\ell}^{e_{j_\ell}}$ is common to all partial derivatives.
   The factors are evaluated as products of pure powers of the variables, computed in shared memory by each block of threads.

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.

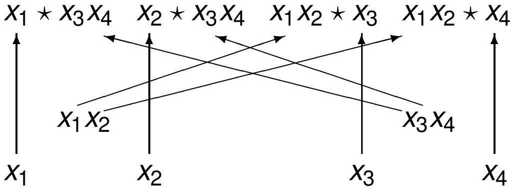
3. Coefficient multiplication and term summation.
   Summation jobs are ordered by the number of terms so each warp has the same amount of terms to sum.

arithmetic circuits

First to evaluate the product:

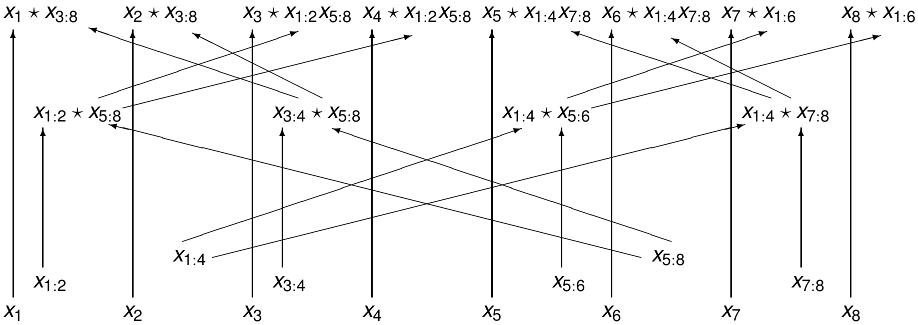


and then to compute the gradient:



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.

granularity issues

We distinguish between evaluation and differentiation

- of many products in few variables, $n \leq 32$; and
- of few products in many variables, $n \gg 32$.

Many monomials in few variables:

- Every thread has one product to compute.
- The number of threads which can multiply products in parallel depends on the size of the shared memory.

Few monomials in many variables:

- Many threads collaborate to compute one product.
- The multiplication is executed with memory coalescing, with the prefix sum algorithm, organized in a tree.

Polynomial Homotopy Continuation on GPUs

1. Quality Up by GPU Acceleration
   - graphics processing units
   - polynomial homotopy continuation
   - double double and quad double precision
2. GPU Accelerated Path Trackers
   - evaluation and differentiation
   - arithmetic circuits
   - granularity issues
3. Software
   - from standalone programs to production code
   - navigating the code

current state

All the CUDA code is

- released under the GNU GPL licence;
- under version control at github;
- installed and ready to run on `kepler.math.uic.edu`.

Reproducibility of published results not a short term problem.

Languages and compilers:

- needs the NVIDIA CUDA `nvcc` compiler;
- C++ for the code on the host;
- the gnu-ada compiler for the connection to PHCpack.

Goal: turn into production software that anybody can use.

Make it useful with GPU accelerated polyhedral homotopies!

navigating the code

Every computation is defined at least twice:

1. on the host, in plain C++,
2. on the device, kernels written in CUDA.

The code for the host is intended mainly to verify correctness, not for high performance.

Building executables:

- one central makefile in the `src/Objects` folder;
- standalone test programs for all parts of the code.