Polynomial Homotopy Continuation on GPUs

Jan Verschelde and Xiangcheng Yu
Department of Mathematics, Statistics, and Computer Science
University of Illinois at Chicago
851 South Morgan (M/C 249)
Chicago, IL 60607, USA
emails: janv@uic.edu and xiangchengyu@outlook.com
30 August 2015

Abstract
The purpose of the software presentation is to announce a library to track many solution paths defined by a polynomial homotopy on a Graphics Processing Unit (GPU). Developed on NVIDIA graphics cards with CUDA SDKs, our code is released under the GNU GPL license. Via the C interface to PHCpack, we can call our GPU library from Python.

1 Polynomial Homotopy Continuation
Polynomial homotopy continuation methods apply predictor-corrector algorithms to track solution paths defined by a homotopy, a family of polynomial systems. As corrector, a path tracker applies Newton’s method, which requires the evaluation of the polynomials and all their derivatives. The evaluated and differentiated polynomials at the current approximation for the solution provide the input for a linear system that must be solved to update that approximation.

With this paper we want to announce the availability of a novel software library for tracking solution paths defined by a polynomial homotopy on graphics processing units. This coding project represents a successful application of the QD library [3, 5] which provides double double and quad double arithmetic to improve the accuracy of numerical computations. While related to interval arithmetic [6], the QD library seems to be a lesser known solution to increase the numerical robustness. Our original motivation, originating in [10], is to offset the cost of double double and quad double arithmetic by parallel computation.

The other contribution we want to highlight is the extension of the Python package phcpy [9] to call the GPU accelerated path trackers. Python is a popular scripting language in computational science and is frequently used to build web applications. In this interface, the input polynomials are given as strings of human readable polynomials in conventional symbolic form. The solutions on output are returned as strings and phcpy provides tools to parse those strings.

This document concerns release 2.4 of PHCpack [7, 8]. The code is available at https://github.com/janverschelde/PHCpack.

---

*This material is based upon work supported by the National Science Foundation under Grant ACI-1440534.
2 Graphics Processing Units

With Graphics Processing Units (GPU) we aim to accelerate the computations in order to achieve significant speedups over unaccelerated code executed on traditional processors. Graphics cards offer a significantly higher memory bandwidth. Their scalable execution model launches thousands of threads that perform the same sequences of instructions on different data.

In [11] we showed there is enough regularity in the evaluation and differentiation of a polynomial system, using the reverse mode of algorithmic differentiation [2], to achieve a sufficient amount of data parallelism and to keep thousands of threads occupied. As in [10], with parallelism we aimed to compensate for the cost overhead caused by the double double and quad double arithmetic, and in [11, 12, 13], we applied the CUDA version [5] of the QD library [3].

For double and double double arithmetic, the computations are memory bound and speedups are achieved via the higher memory bandwidth. Double digit speedups are obtained in complex double double and quad double arithmetic, when the problem is compute bound, when we reach a higher compute to global memory access ratio.

Recently, we improved the arithmetic circuits to evaluate and differentiate polynomials and upgraded our linear systems solvers for integration in code to track one single path of a high dimensional system [14] and to track many paths of smaller homotopies [15]. In tracking many paths, all tasks of the path tracker (predictor, corrector, adaptive step control) are performed on the device (GPU). For one single path in high dimensions, the adaptive step control can be done separately on the host (CPU).

While the computations in [11, 12] ran on randomly generated data, in [13, 14, 15] we applied our GPU code on benchmark problems, such as the cyclic $n$-roots problems, Pieri hypersurface intersection conditions in the numerical Schubert calculus, and the computation of all Nash equilibria in multiplayer games. When tracking many paths, the benefits of the accelerated code start to show for problems with ten of thousands of solutions. When tracking one single path, the threshold dimension to experience the benefits of acceleration is about one hundred.

3 PHCpack, PHClib, and phcpy

The C interface to PHCpack [7] was developed [4] to write relatively short distributed memory message passing programs in various parallel versions of polynomial homotopies. This interface, called PHClib, gives the C programmer access to the data stored by PHCpack, without duplication of data structures. PHClib defines the layer between PHCpack and the Python package phcpy [9]. By design, any upgrade to phcpy increases the functionality for the C and C++ programmers.

The main program that launches the accelerated path trackers starts with the definition of the polynomial homotopy and initializes the solution(s) at the start of the path(s). With PHClib, the start system can be generated and its start solutions computed. PHCpack provides condition number estimators at standard double, double double, quad double, and arbitrary precision. These estimators can be applied to determine the precision required to reach a prescribed accuracy.

Unlike the main executable program phc of PHCpack, which is statically linked, our GPU software is released as source code on github. The workstation that hosts our NVIDIA K20C GPUs is also a web server [1] to the solvers of PHCpack, at https://kepler.math.uic.edu. The path trackers that run in the cloud will also become accelerated.
4 GPU Acceleration in a Python session

To illustrate calling the library to accelerate the path trackers we use a script to track the 35,940 solution paths defined by an artificial-parameter homotopy between the cyclic 10-roots problem and a random coefficient start system for the cyclic 10-roots problem. The two most relevant lines in the script are below:

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

The first two arguments of the function call are the target and start system respectively, followed by a list of start solutions.

To measure the wall clock time, we apply the command `time`, when calling the script at the command prompt `$`, for example, as `$ time python runcyc10d.py > /tmp/output`. The output of `time` consists of three numbers. We see first the elapsed wall clock time (`real`), followed by the user CPU time (`user`), and then finally the system time (`sys`). Table 1 summarizes the runs.

<table>
<thead>
<tr>
<th>precision</th>
<th>real</th>
<th>user</th>
<th>sys</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>14.980s</td>
<td>11.683s</td>
<td>3.192s</td>
</tr>
<tr>
<td>double double</td>
<td>45.266s</td>
<td>35.897s</td>
<td>9.228s</td>
</tr>
<tr>
<td>quad double</td>
<td>6m 57.368s</td>
<td>5m 23.224s</td>
<td>1m 33.266s</td>
</tr>
</tbody>
</table>

Table 1: Running times in minutes (m) and seconds (s) for the tracking of 35,940 paths of the cyclic 10-roots problem, in double, double double, and quad double precision on an Intel Xeon E5-2670 processor, accelerated by the NVIDIA K20C.

References


