Computing Power Series accurately with Graphics Processing Units (*preliminary report*)

Jan Verschelde†

University of Illinois at Chicago
Department of Mathematics, Statistics, and Computer Science
http://www.math.uic.edu/~jan
https://github.com/janverschelde
https://www.youtube.com/@janverschelde5226
janv@uic.edu

AMS Special Session on Polynomial Systems, Homotopy Continuation and Applications, 4 January 2023

†Supported by the National Science Foundation, grant DMS 1854513.

Outline

1. Introduction
   - problem statement
   - multiple double arithmetic
   - graphics processing units
2. Accelerated Newton
   - setup and scalability parameters
   - staggered computations
   - computational results

problem statement

The problem is to make Newton's method scalable,
that is: for thousands (or more) equations and variables,
to compute power series accurately.

On power series, the computational bottlenecks involve

1. evaluation and differentiation require convolutions, and
2. solution of a lower triangular block Toeplitz system.

The two main concerns of performance and convergence
impose regularity assumptions on the input.

Three parameters to scale:

1. n is the number of variables;
2. d is the order of the series, truncation degree; and
3. p is the precision.

a task graph for a triangular block Toeplitz system

$Q, R = \mathrm{qr}(A_0)$

$\mathbf{x}_0 = R^{-1} Q^H \mathbf{b}_0$

$\mathbf{b}_1 = \mathbf{b}_1 - A_1 \mathbf{x}_0$ $\quad$ $\mathbf{b}_2 = \mathbf{b}_2 - A_2 \mathbf{x}_0$ $\quad$ $\mathbf{b}_3 = \mathbf{b}_3 - A_3 \mathbf{x}_0$

$\mathbf{x}_1 = R^{-1} Q^H \mathbf{b}_1$

$\mathbf{b}_2 = \mathbf{b}_2 - A_1 \mathbf{x}_1$ $\quad$ $\mathbf{b}_3 = \mathbf{b}_3 - A_2 \mathbf{x}_1$

$\mathbf{x}_2 = R^{-1} Q^H \mathbf{b}_2$

$\mathbf{b}_3 = \mathbf{b}_3 - A_1 \mathbf{x}_2$

$\mathbf{x}_3 = R^{-1} Q^H \mathbf{b}_3$

$$
\begin{bmatrix}
A_0 & & & \\
A_1 & A_0 & & \\
A_2 & A_1 & A_0 & \\
A_3 & A_2 & A_1 & A_0
\end{bmatrix}
\begin{bmatrix}
\mathbf{x}_0 \\ \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3
\end{bmatrix}
=
\begin{bmatrix}
\mathbf{b}_0 \\ \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3
\end{bmatrix}
$$

linearizes $A(t)\mathbf{x}(t) = \mathbf{b}(t)$, with

$$
\begin{array}{rcl}
A(t) & = & A_0 + A_1 t + A_2 t^2 + A_3 t^3, \\
\mathbf{x}(t) & = & \mathbf{x}_0 + \mathbf{x}_1 t + \mathbf{x}_2 t^2 + \mathbf{x}_3 t^3, \quad \mathbf{b}(t) \; = \; \mathbf{b}_0 + \mathbf{b}_1 t + \mathbf{b}_2 t^2 + \mathbf{b}_3 t^3.
\end{array}
$$

multiple double arithmetic / error free transformations

From **An Algorithm for Analytic Continuation** by Peter Henrici, *J. SIAM Numer. Anal.*, Vol. 3, No. 1, pages 67–78, 1966:
"Some reflection shows that in order to get a convergent process *the early vectors $A_n^{(k)}$ (early with respect to k) must be computed more accurately than the late ones.*"

A multiple double is an unevaluated sum of doubles.

The 2-norm of a vector of dimension 64 of random complex numbers on the unit circle equals 8. Observe the second double:

```
double double : 8.0000000000000E+00 - 6.4711246131411E-32
  quad double : 8.0000000000000E+00 + 3.2047541141939E-65
  octo double : 8.0000000000000E+00 - 9.7260991519831E-129
```

If the result fits exactly in a 64-bit floating-point number, then the second double is the roundoff error, or its accuracy.

cost overhead factors — software packages

The number of floating-point operations for multiple double arithmetic are cost overhead factors:

	$+$	$\star$	$/$	average
`double double`	20	32	70	37.7
`quad double`	89	336	893	439.3
`octo double`	269	1742	5126	2379.0

- QDlib by Y. Hida, X. S. Li, and D. H. Bailey. Algorithms for quad-double precision floating point arithmetic. In the *Proceedings of the 15th IEEE Symposium on Computer Arithmetic*, pages 155–162, 2001.
- CAMPARY by M. Joldes, J.-M. Muller, V. Popescu, and W. Tucker. CAMPARY: Cuda Multiple Precision Arithmetic Library and Applications. In *Mathematical Software – ICMS 2016, the 5th International Conference on Mathematical Software*, pages 232–240, Springer-Verlag, 2016.

related work and alternatives

- T. Nakayama and D. Takahashi. Implementation of multiple-precision floating-point arithmetic library for GPU computing. In *Proc. 23rd IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2011)*, pages 343–349. ACTA Press, 2011.
- N. Maho. MPLAPACK version 2.0.1. user manual `arXiv:2109.13406v2 [cs.MS] 12 Sep 2022`.
- K. Isupov and V. Knyazkov. Multiple-precision matrix-vector multiplication on graphics processing units. *Program Systems: Theory and Applications* 11(3): 62–84, 2020.
    - The double double arithmetic of CAMPARY performs best for the problem of matrix-vector multiplication.
    - Concerning quad double precision, *“the CAMPARY library is faster than our implementation; however as the precision increases the execution time of CAMPARY also increases significantly.”*

Graphics Processing Units (GPUs)

- Data parallel algorithms execute the same Single Instruction on Multiple Data elements (SIMD).
- Memory bandwidth of GPUs is typically ten times higher than the memory bandwidth of CPUs.

Definition (CGMA ratio)

The *Compute to Global Memory Access (CGMA) ratio* is the number of floating-point calculations performed by a kernel for each access to the global memory.

- Starting with complex double double and quad double arithmetic, computations become compute bound instead of memory bound.
- The NVIDIA K20C, P100, and V100 (also the A100 and H100) are capable of teraflop performance in double precision.

One *teraflops* is one trillion floating-point operations per second.

prior work

- S. Telen, M. Van Barel, and J. Verschelde. **Robust numerical tracking of one path of a polynomial homotopy on parallel shared memory computers.** In the *Proceedings of the 22nd International Workshop on Computer Algebra in Scientific Computing (CASC 2020)*, pages 563–582. Springer-Verlag, 2020.
- J. Verschelde. **Accelerated polynomial evaluation and differentiation at power series in multiple double precision.** In the *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 740–749. IEEE, 2021.
- J. Verschelde. **Least squares on GPUs in multiple double precision.** In the *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 828–837. IEEE, 2022.

monomial homotopies

Consider n variables $\mathbf{x}$, A is an n-by-n exponent matrix, and $\mathbf{b}(t)$ is a vector of n series of order $O(t^d)$:

$$\mathbf{x}^A = \mathbf{b}(t) \quad \text{is a } \textit{monomial homotopy}.$$

For example, $n = 3$, $\mathbf{x} = [x_1, x_2, x_3]$:

$$A = \left[\begin{array}{ccc} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{array} \right] \quad \left\{ \begin{array}{ll} x_1 & = b_1(t) \\ x_1 x_2 & = b_2(t) \\ x_1 x_2 x_3 & = b_3(t) \end{array} \right. \quad \begin{array}{l} x_1(t) = \exp(\alpha_1 t) + O(t^d) \\ x_2(t) = \exp(\alpha_2 t) + O(t^d) \\ x_3(t) = \exp(\alpha_3 t) + O(t^d) \end{array}$$

where

$$\exp(\alpha t) + O(t^4) = 1 + \alpha t + \frac{\alpha^2}{2!} t^2 + \frac{\alpha^3}{3!} t^3 + O(t^4),$$

with $\alpha \in [-1, -1 + \delta] \cup [1 - \delta, 1]$, $\delta > 0$, or $|\alpha| = 1$ for random $\alpha \in \mathbb{C}$.

order of series, accuracy and precision

$$\exp(t) = \sum_{k=0}^{d-1} \frac{t^k}{k!} + O(t^d)$$

k	$1/k!$	recommended precision
7	2.0e-004	double precision okay
15	7.7e-013	use double doubles
23	3.9e-023	use double doubles
31	1.2e-034	use quad doubles
47	3.9e-060	use octo doubles
63	5.0e-088	use octo doubles
95	9.7e-149	need hexa doubles
127	3.3e-214	need hexa doubles

scalability parameters

1. Going from one to two columns of monomials:

$$\mathbf{c}_1 \mathbf{x}^{A_1} + \mathbf{c}_2 \mathbf{x}^{A_2} = \mathbf{b}(t),$$

   for two n-vectors $\mathbf{c}_1$ and $\mathbf{c}_2$ and two exponent matrices A_1 and A_2.
2. For increasing dimensions: $n = 64, 128, 256, 512, 1024$.
3. For increasing orders d in $O(t^d)$, $d = 1, 2, 4, 8, 16, 32, 64$.
4. For increasing precision:
   double, double double, quad double, octo double.

Doubling columns, dimensions, orders, and precision,
how much of the overhead can be compensated by GPU acceleration?

different types of accelerated computations

1. convolutions for evaluation and differentiation
2. Householder QR
3. $Q^H\mathbf{b}$ computations
4. back substitutions to solve $R\mathbf{x} = Q^H\mathbf{b}$
5. updates $\mathbf{b} = \mathbf{b} - A\mathbf{x}$
6. residual computations $\|\mathbf{b} - A\mathbf{x}\|_1$

Which of the six types occupies most time?

staggered computations

Computing $\mathbf{x}(t) = \mathbf{x}_0 + \mathbf{x}_1 t + \mathbf{x}_2 t^2 + \cdots + \mathbf{x}_{d-1} t^{d-1}$, observe:

1. Start $\mathbf{x}_0$ with half its precision correct, otherwise Newton's method may not converge.

2. Increase d in the order $O(t^d)$ gradually, e.g.: the new d is $d + 1 + d/2$, hoping (at best) for quadratic convergence.

3. Once $\mathbf{x}_k$ is correct, the corresponding $\mathbf{b}_k = 0$, as $\mathbf{b}_k$ is obtained by evaluation, and then the update $\Delta\mathbf{x}_k$ should no longer be computed because

$$QR\Delta\mathbf{x}_k = \mathbf{b}_k = \mathbf{0} \quad \Rightarrow \quad \Delta\mathbf{x}_k = \mathbf{0}.$$

   This gives a criterion to stop the iterations.

graphics processing units

The code was developed for the “Volta” V100 NVIDIA GPU,
and tested on the “Pascal” P100 and RTX 2080 NVIDIA GPUs.

NVIDIA GPU	CUDA	#MP	#cores/MP	#cores	GHz
Pascal P100	6.0	56	64	3584	1.33
Volta V100	7.0	80	64	5120	1.91
GeForce RTX 2080	7.5	46	64	2944	1.10

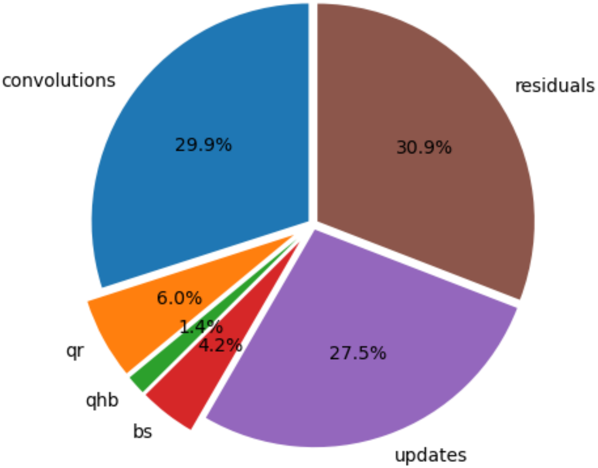
The double precision peak performance of the P100 is 4.7 TFLOPS.
At 7.9 TFLOPS, the V100 is 1.68 times faster than the P100.

To evaluate the algorithms, compare the ratios of the wall clock times on the P100 over V100 with the factor 1.68.

For every kernel, the number of arithmetical operations is accumulated. The total number of double precision operations is computed using the cost overhead multipliers.

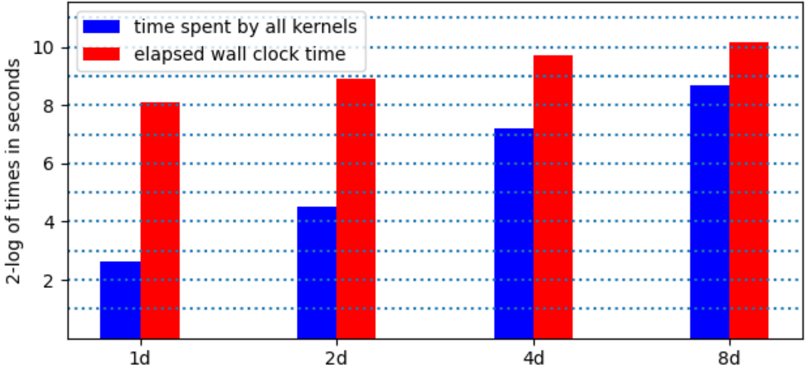
$n = 1024$, $d = 64$, 24 steps in octo double precision

On one column of monomials, triangular exponent matrix of ones.
Six different types of accelerated computations, on the V100.



doubling precisions, wall clock and kernel times

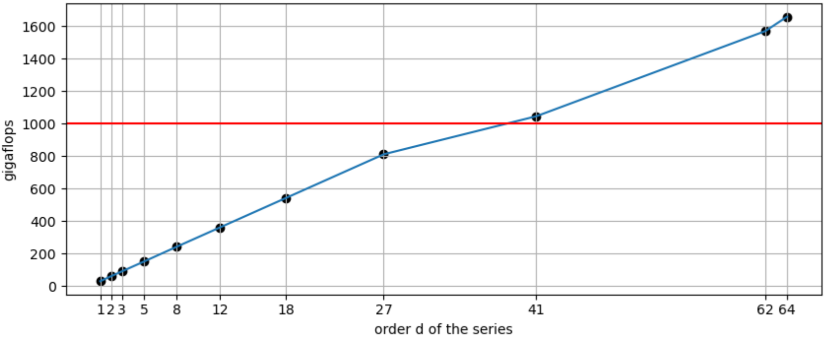
On one column of monomials, triangular exponent matrix of ones, $n = 1024$, $d = 64$, 24 steps, in 4 different precisions, on the V100:



Doubling the precision less than doubles the wall clock time and increases the time spent by all kernels.

teraflop performance of convolutions

On one column of monomials, triangular exponent matrix of ones, $n = 1024$, performance of the evaluation and differentiation, in octo double precision, for increasing orders of the series:

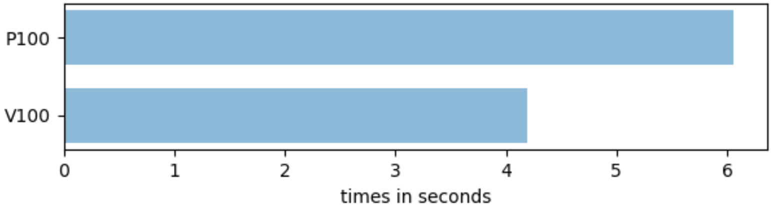


After degree 40, teraflop performance is observed on the V100.

comparing GPUs

Householder QR on quad double matrix, $n = 1024 = 16 \times 64$.

Performance on V100: 1.687 Teraflops, on P100: 1.169 Teraflops.



Ratio of milliseconds P100 over V100: $6059.121/4197.730 \approx 1.44$.
The theoretical peak performance ratio is 1.68.

RTX 2080 takes 82.553 seconds, with 87.556 Gigaflops performance.

Conclusions

Bringing HPC into PHC . . .

- Octo double precision suffices for series of order 64.
- Teraflop performance of the evaluation and differentiation is already attained at order 40.
  The convolutions to evaluate and differentiate at power series remain a significant portion of all computational work.
- Doubling precisions less than doubles the wall clock times because the computations are then compute bound and thus well suited for acceleration by graphics processing units.

. . . *is still a work in progress.*