Accelerated Polynomial Evaluation and Differentation at Power Series in Multiple Double Precision

Jan Verschelde[†]

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

The 22nd IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC 2021)
21 May 2021, online

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

Outline

1. Problem Statement
   - power series and multiple double precision
   - algorithmic differentiation on five different GPUs
2. Accelerated Convolutions
   - polynomial evaluation and differentiation
   - data staging algorithms
3. Computational Results
   - three test polynomials
   - performance and scalability

problem statement

Input: one polynomial f in n variables and n power series $\mathbf{z}(t)$.

Output: $f(\mathbf{z}(t))$, and its gradient evaluated at $\mathbf{z}(t)$.

Why bother? It is a computationally intensive step in
S. Telen, M. Van Barel, and J. Verschelde:
A robust numerical path tracking algorithm for polynomial homotopy continuation. *SIAM J. Scientific Computing* 42(6):A3610–A3637, 2020.

Problem: large problems require multiprecision.

Solution: use GPU acceleration to compensate for the cost overhead caused by power series and multiprecision arithmetic.

algorithmic differentiation on five different GPUs

Run code generated by the CAMPARY software,
by M. Joldes, J.-M. Muller, V. Popescu, W. Tucker, in ICMS 2016.

on five different GPUs:

NVIDIA GPU	CUDA	#MP	#cores/MP	#cores	GHz
Tesla C2050	2.0	14	32	448	1.15
Kepler K20C	3.5	13	192	2496	0.71
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 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.

a selection of related work

- M.M. Maza and W. Pan. Fast polynomial multiplication on a GPU. *Journal of Physics: Conference Series*, 256, 2010. High Performance Computing Symposium (HPCS 2010), 5-9 June 2010, Victoria College, University of Toronto, Canada.
- J. Glabe and J.M. McCarthy. A GPU homotopy path tracker and end game for mechanism synthesis. In *the Proceedings of the 2020 USCToMM Symposium on Mechanical Systems and Robotics*, pages 206–215. Springer 2020.
- K. Isupov and V. Knyazkov. Multiple-precision matrix-vector multiplication on graphics processing units. *Program Systems: Theory and Applications* 11(3): 62–84, 2020.

accelerated convolutions

$\left(x_0 + x_1 t + \cdots + x_d t^d\right) \star \left(y_0 + y_1 t + \cdots + y_d t^d\right)$ has coefficients

$$z_k = \sum_{i=0}^{k} x_i \cdot y_{k-i}, \quad k = 0, 1, \ldots, d.$$

Thread k computes z_k.

To avoid thread divergence, pad with zeros, e.g. for $d = 3$:

X	x_0	x_1	x_2	x_3				
Y	0	0	0	0	y_0	y_1	y_2	y_3
Z	z_0	z_1	z_2	z_3				

The arrays X, Y, and Z are in shared memory.

pseudo code for a convolution kernel

Thread k computes $z_k = \sum_{i=0}^{k} x_i \cdot y_{k-i}, k = 0, 1, \ldots, d,$

after loading the coefficients into shared memory arrays

X	x_0	x_1	x_2	x_3				
Y	0	0	0	0	y_0	y_1	y_2	y_3
Z	z_0	z_1	z_2	z_3				

1. $X_k := x_k$
2. $Y_k := 0$
3. $Y_{d+k} := y_k$
4. $Z_k := X_0 \cdot Y_{d+k}$
5. for i from 1 to d do $Z_k := Z_k + X_i \cdot Y_{d+k-i}$
6. $z_k := Z_k$

evaluating and differentiating one monomial

Evaluating and differentiating $a x_1 x_2 x_3 x_4 x_5$ at a power series vector:

$$
\begin{array}{lll}
f_1 := a \star z_1 & b_1 := z_5 \star z_4 & \\
f_2 := f_1 \star z_2 & b_2 := b_1 \star z_3 & \\
f_3 := f_2 \star z_3 & b_3 := b_2 \star z_2 & c_1 := f_1 \star b_2 \\
f_4 := f_3 \star z_4 & b_3 := b_3 \star a & c_2 := f_2 \star b_1 \\
f_5 := f_4 \star z_5 & & c_3 := f_3 \star z_5
\end{array}
$$

- The gradient is in $(b_3, c_1, c_2, c_3, f_4)$ and the value is in f_5.
- Each $\star$ is a convolution between two truncated power series.
- Statements on the same line can be computed in parallel.

Given sufficiently many blocks of threads,
monomial evaluation and differentiation takes n steps for n variables.

accelerated algorithmic differentiation

$$
\begin{array}{rcccccc}
p & = & a_0 + & a_1 x_1 x_3 x_6 & + & a_2 x_1 x_2 x_5 x_6 & + & a_3 x_2 x_3 x_4 \\
 & & & f_{1,1} := a_1 \star z_1 & & f_{2,1} := a_2 \star z_1 & & f_{3,1} := a_3 \star z_2 \\
 & & & f_{1,2} := f_{1,1} \star z_3 & & f_{2,2} := f_{2,1} \star z_2 & & f_{3,2} := f_{3,1} \star z_3 \\
 & & & f_{1,3} := f_{1,2} \star z_6 & & f_{2,3} := f_{2,2} \star z_5 & & f_{3,3} := f_{3,2} \star z_4 \\
 & & & & & f_{2,4} := f_{2,3} \star z_6 & & \\
 & & & b_{1,1} := z_6 \star z_3 & & b_{2,1} := z_6 \star z_5 & & b_{3,1} := z_4 \star z_3 \\
 & & & b_{1,1} := b_{1,1} \star a_1 & & b_{2,2} := b_{2,1} \star z_2 & & b_{3,1} := b_{3,1} \star a_3 \\
 & & & & & b_{2,2} := b_{2,2} \star a_2 & & \\
 & & & c_{1,1} := f_{1,1} \star z_6 & & c_{2,1} := f_{2,1} \star b_{2,1} & & c_{3,1} := f_{3,1} \star z_4 \\
 & & & & & c_{2,2} := f_{2,2} \star z_6 & &
\end{array}
$$

The 21 convolutions are arranged below (on same line: parallel execution):

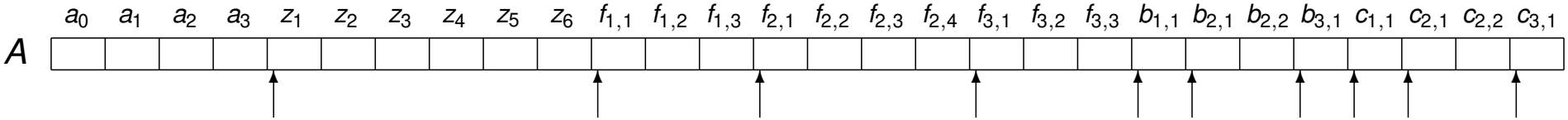
$$
\begin{array}{cccccccccc}
f_{1,1} & b_{1,1} & & f_{2,1} & b_{2,1} & & f_{3,1} & b_{3,1} & & \\
f_{1,2} & b_{1,1} & c_{1,1} & f_{2,2} & b_{2,2} & c_{2,1} & f_{3,2} & b_{3,1} & c_{3,1} & \\
f_{1,3} & & & f_{2,3} & b_{2,2} & c_{2,2} & f_{3,3} & & & \\
 & & & f_{2,4} & & & & & &
\end{array}
$$

staging of the data

The 21 convolutions

$$
\begin{array}{cccccccccc}
f_{1,1} & b_{1,1} & & f_{2,1} & b_{2,1} & & f_{3,1} & b_{3,1} & & \\
f_{1,2} & b_{1,1} & c_{1,1} & f_{2,2} & b_{2,2} & c_{2,1} & f_{3,2} & b_{3,1} & c_{3,1} & \\
f_{1,3} & & & f_{2,3} & b_{2,2} & c_{2,2} & f_{3,3} & & & \\
 & & & f_{2,4} & & & & & &
\end{array}
$$

are stored after the power series coefficients a_0, a_1, a_2, a_3,
and the input power series z_1, z_2, z_3, z_4, z_5, z_6 in the array A below:



- The arrows point at the start position of the input series and at the forward $f_{i,j}$, backward $b_{i,j}$, cross products $c_{i,j}$ for every monomial i.
- For complex and multiple double numbers, there are multiple data arrays, for real, imaginary parts, and for each double in a multiple double.

convolution jobs

One block of threads does one convolution.
It takes four kernel launches, to compute the 21 convolutions

$f_{1,1}$	$b_{1,1}$		$f_{2,1}$	$b_{2,1}$		$f_{3,1}$	$b_{3,1}$	
$f_{1,2}$	$b_{1,1}$	$c_{1,1}$	$f_{2,2}$	$b_{2,2}$	$c_{2,1}$	$f_{3,2}$	$b_{3,1}$	$c_{3,1}$
$f_{1,3}$			$f_{2,3}$	$b_{2,2}$	$c_{2,2}$	$f_{3,3}$		
			$f_{2,4}$					

respectively with 6, 9, 5, and 1 block of threads.

Given sufficiently many blocks of threads, all convolutions for the polynomial evaluation and differentiation take m steps, where m is the largest number of variables in one monomial.

A convolution job j is defined by the triplet $(t_1(j), t_2(j), t_3(j))$, where

- $t_1(j)$ and $t_2(j)$ are the locations in the array A for the input, and
- $t_3(j)$ is the location in A of the output of the convolution.

addition jobs

One block of threads does one addition of two power series.

An addition job j is defined by the pair $(t_1(j), t_2(j))$, where

- $t_1(j)$ is the location in the array A of the input, and
- $t_2(j)$ is the location in the array A of the update.

Jobs are defined recursively, following the tree summation algorithm.

The job index j corresponds to the block index in the kernel launch.

Given sufficiently many blocks of threads,
polynomial evaluation and differentiation takes $m + \lceil \log_2(N) \rceil$ *steps,*
where m *is the largest number of variables in one monomial*
and N *is the number of monomials.*

For $N \gg m$, an upper bound for the speedup is $dN/\log_2(N)$,
where d is the degree at which the series are truncated.

three test polynomials

For each polynomial, p_1, p_2, p_3,

- n is the total number of variables,
- m is the number of variables per monomial, and
- N is the number of monomials (not counting the constant term).

The last two columns list the number of convolution and addition jobs:

	n	m	N	#cnv	#add
p_1	16	4	1,820	16,380	9,084
p_2	128	64	128	24,192	8,192
p_3	128	2	8,128	24,256	24,256

Does the shape of the test polynomials influence the execution times?

performance of the five GPUs

In evaluating p_1 for degree $d = 152$ in deca double precision, the `cudaEventElapsedTime` measures the times of kernel launches. The last line is the wall clock time for all convolution and addition kernels. All units are milliseconds.

	C2050	K20C	P100	V100	RTX 2080
convolution	12947.26	11290.22	1060.03	634.29	10002.32
addition	10.72	11.13	1.37	0.77	5.01
sum	12957.98	11301.35	1061.40	635.05	10007.34
wall clock	12964.00	11309.00	1066.00	640.00	10024.00

- The $12964/640 \approx 20.26$ is for the V100 over the oldest C2050.
- Compare the ratio of the wall clock times for P100 over V100 $1066/640 \approx 1.67$ with the ratios of theoretical double peak performance of the V100 of the P100: $7.9/4.7 \approx 1.68$.

teraflop performance

In 1.066 second, the P100 did 16,380 convolutions, 9,084 additions.

- One convolution on series truncated at degree d requires $(d+1)^2$ multiplications and $d(d+1)$ additions.
- One addition of two series at degree d requires $d+1$ additions.

So we have $16,380(d+1)^2$ multiplications and
$16,380d(d+1) + 9,084(d+1)$ additions in deca double precision.

- One multiplication and one addition in deca double precision require respectively 3,089 and 397 double operations.
- The $16,380(d+1)^2$ evaluates to 1,184,444,368,380 and $16,380d(d+1) + 9,084(d+1)$ to 151,782,283,404 operations.

In total, in 1.066 second the P100 did 1,336,226,651,784 double float operations, reaching a performance of 1.25 TFLOPS.

scalability

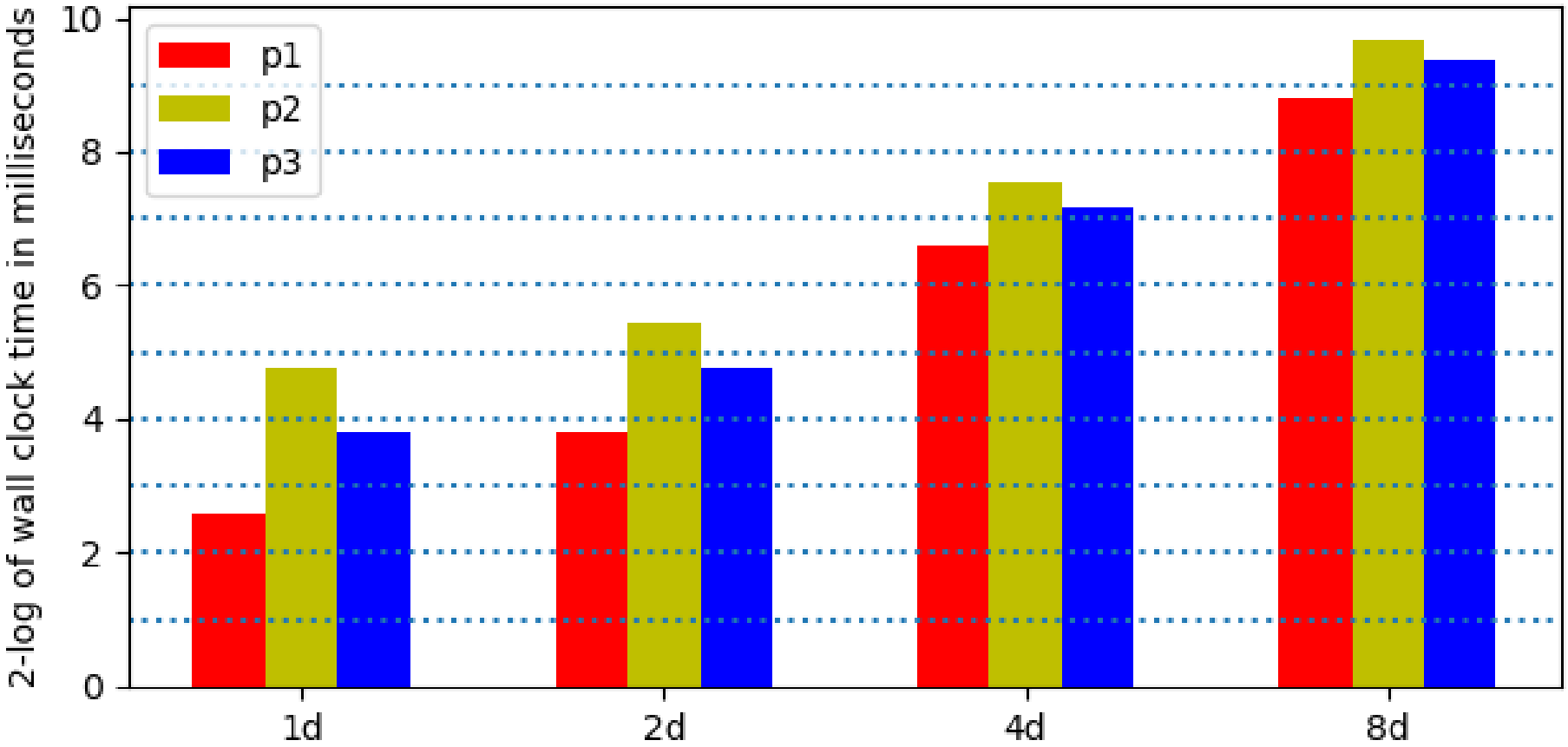
To examine the scalability, consider

1. the doubling of the precision, in double, double double, quad double and octo double precision, for fixed degree 191; and
2. the doubling of the number of coefficients of the series, from 32 to 64, and from 64 to 128.

For increasing precision, the problem becomes more compute bound.

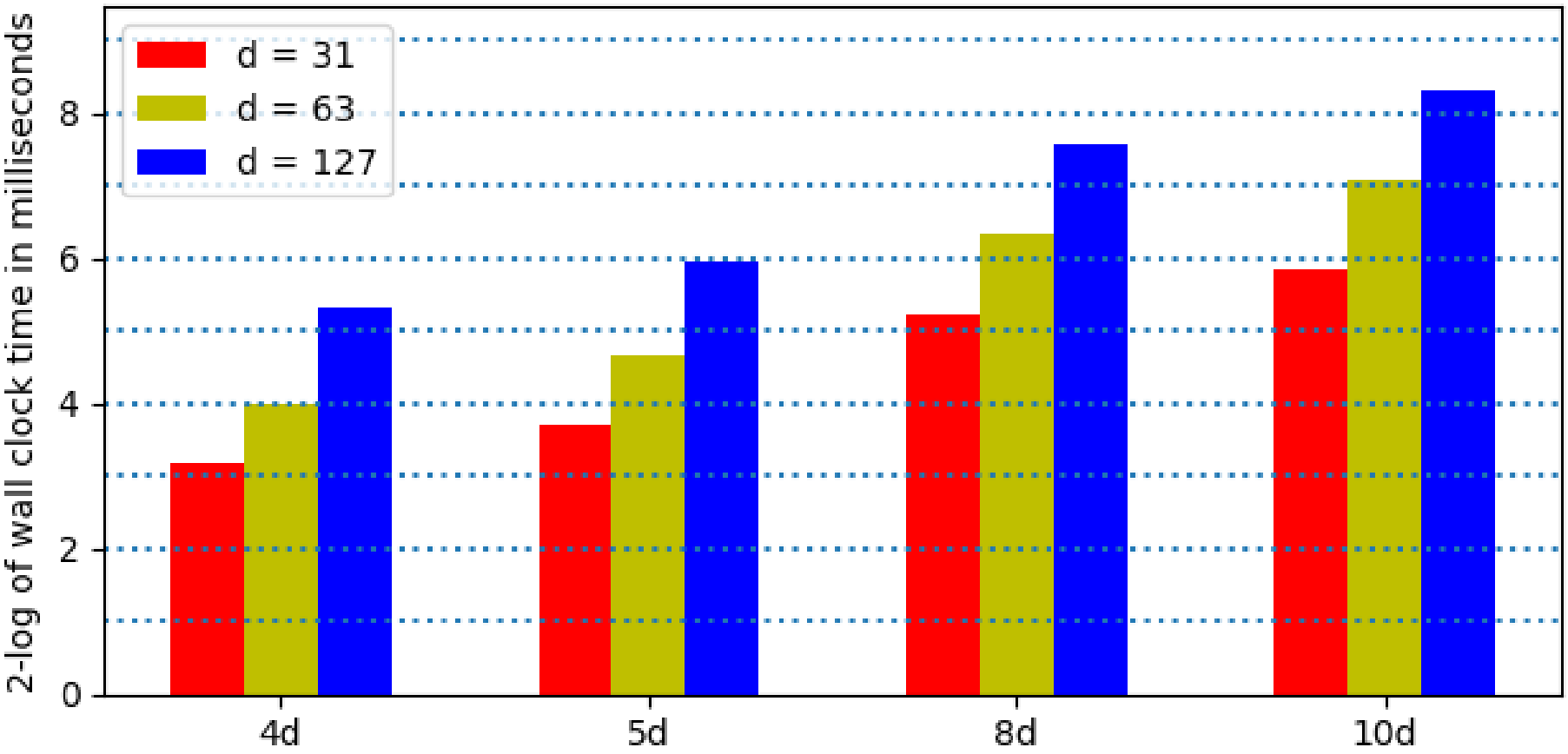
doubling the precision

The 2-logarithm of the wall clock times to evaluate and differentiate p_1, p_2, p_3 in double (1d), double double (2d), quad double (4d), and octo double (8d) precision, for power series truncated at degree 191:



doubling the number of coefficients of the series

The 2-logarithm of the wall clock times to evaluate and differentiate p_1 in quad double (4d), penta double (5d), octo double (8d), and deca double (10d) precision, for power series of degrees 31, 63, and 127:



conclusions

With GPU acceleration, one can compensate the cost overhead

- caused by power series arithmetic, and
- caused by multiple double precision arithmetic.

Data staging algorithms define the coordinates for the convolution and the addition jobs.

Speedup factors comparing the V100 and P100 are close to the ratio of the theoretical peak performances.

The source code belongs to PHCpack, available on github, released under the GNU GPL license.