GPU Accelerated QR

MCS 572 Lecture 36
Introduction to Supercomputing
Jan Verschelde, 18 November 2024

GPU Accelerated QR

constructing orthogonal vectors

Lemma

Let $\mathbf{x}$ and $\mathbf{y}$ be vectors with the same norm: $||\mathbf{x}||_2 = ||\mathbf{y}||_2$, then $\mathbf{x} - \mathbf{y}$ and $\mathbf{x} + \mathbf{y}$ are perpendicular to each other.

Proof. The vector $\mathbf{x} - \mathbf{y}$ is perpendicular to $\mathbf{x} + \mathbf{y}$ if $(\mathbf{x} - \mathbf{y})^T(\mathbf{x} + \mathbf{y}) = 0$.

$$\begin{aligned}(\mathbf{x} - \mathbf{y})^T(\mathbf{x} + \mathbf{y}) &= \mathbf{x}^T\mathbf{x} + \mathbf{x}^T\mathbf{y} - \mathbf{y}^T\mathbf{x} - \mathbf{y}^T\mathbf{y} \\ &= ||\mathbf{x}||_2^2 - ||\mathbf{y}||_2^2 = 0.\end{aligned}$$

Q.E.D.

a projection operator

For a nonzero vector $\mathbf{z}$, consider $P = \dfrac{\mathbf{z}\mathbf{z}^T}{\mathbf{z}^T\mathbf{z}}$.

Observe:

- If $\mathbf{z}$ is a n-by-1 column vector,
  then $\mathbf{z}^T$ is a 1-by-n row vector, and $\mathbf{z}\mathbf{z}^T$ is n-by-n.
- $\mathbf{z}^T\mathbf{z} = ||\mathbf{z}||_2^2$.

P is a projection operator, with the property $P^2 = P$.

$$P^2 = \left(\frac{\mathbf{z}\mathbf{z}^T}{\mathbf{z}^T\mathbf{z}}\right)\left(\frac{\mathbf{z}\mathbf{z}^T}{\mathbf{z}^T\mathbf{z}}\right) = \frac{\mathbf{z}(\mathbf{z}^T\mathbf{z})\mathbf{z}^T}{(\mathbf{z}^T\mathbf{z})^2} = \frac{\mathbf{z}\mathbf{z}^T}{\mathbf{z}^T\mathbf{z}} = P$$

Householder reflectors

Definition

Given two nonzero vectors $\mathbf{x}$ and $\mathbf{y}$: $||\mathbf{x}||_2 = ||\mathbf{y}||_2$ and $\mathbf{z} = \mathbf{y} - \mathbf{x} \neq \mathbf{0}$.

Then $H = I - 2\dfrac{\mathbf{z}\mathbf{z}^T}{\mathbf{z}^T\mathbf{z}}$ is *a Householder reflector*.

H is an orthogonal matrix: $H^T H = I$.

Denote $P = \dfrac{\mathbf{z}\mathbf{z}^T}{\mathbf{z}^T\mathbf{z}}$, then $H = I - 2P$.

$$H^T H = (I - 2P)^T (I - 2P) = I - 4P + 4P^2 = I,$$

because $P^2 = P$.

Since P is symmetric, H is symmetric too.

the reflector property

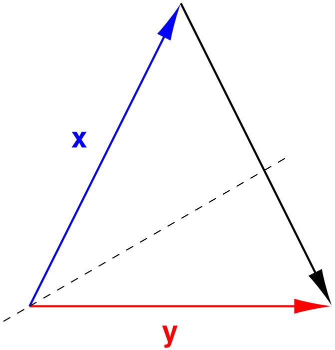
Given two nonzero vectors $\mathbf{x}$ and $\mathbf{y}$: $||\mathbf{x}||_2 = ||\mathbf{y}||_2$ and $\mathbf{z} = \mathbf{y} - \mathbf{x} \neq \mathbf{0}$.

The Householder reflector $H = I - 2\dfrac{\mathbf{z}\mathbf{z}^T}{\mathbf{z}^T\mathbf{z}}$ has the property $H\mathbf{x} = \mathbf{y}$.

$$\begin{aligned}
H\mathbf{x} &= \left(I - 2\frac{\mathbf{z}\mathbf{z}^T}{\mathbf{z}^T\mathbf{z}} \right) \mathbf{x} \\
&= \mathbf{x} - 2\frac{\mathbf{z}\mathbf{z}^T\mathbf{x}}{\mathbf{z}^T\mathbf{z}} \\
&= \mathbf{y} - \mathbf{z} - \frac{\mathbf{z}\mathbf{z}^T\mathbf{x}}{\mathbf{z}^T\mathbf{z}} - \frac{\mathbf{z}\mathbf{z}^T(\mathbf{y} - \mathbf{z})}{\mathbf{z}^T\mathbf{z}} \\
&= \mathbf{y} - \frac{\mathbf{z}\mathbf{z}^T}{\mathbf{z}^T\mathbf{z}}\mathbf{z} - \frac{\mathbf{z}\mathbf{z}^T\mathbf{x}}{\mathbf{z}^T\mathbf{z}} - \frac{\mathbf{z}\mathbf{z}^T(\mathbf{y} - \mathbf{z})}{\mathbf{z}^T\mathbf{z}} \\
&= \mathbf{y} - \frac{\mathbf{z}\mathbf{z}^T(\mathbf{z} + \mathbf{x} + \mathbf{y} - \mathbf{z})}{\mathbf{z}^T\mathbf{z}} \\
&= \mathbf{y}, \text{ because } \mathbf{z}^T(\mathbf{x} + \mathbf{y}) = \mathbf{0}
\end{aligned}$$

a geometric interpretation

$\mathbf{y} = H\mathbf{x} = \mathbf{x} - 2P\mathbf{x}$



Reflection through the bisector brings **x** to **y**.

GPU Accelerated QR

QR factorization with Householder reflectors

Consider a 3-by-2 matrix $A = [\mathbf{a}_1 \ \mathbf{a}_2]$.

1. Compute H_1 so that $H_1 \mathbf{a}_1 = \left[\begin{array}{c} a_{1,1}/||\mathbf{a}_1||_2 \\ 0 \\ 0 \end{array} \right]$.
2. Let $\mathbf{b}_2 = H_1 \mathbf{a}_2$ and select $\mathbf{c} = \left[\begin{array}{c} b_2 \\ b_3 \end{array} \right]$.

   Compute H_2 so that $H_2 \mathbf{c} = \left[\begin{array}{c} c_1/||\mathbf{c}||_2 \\ 0 \end{array} \right]$.

Then $H_2 H_1 A$ is an upper triangular matrix R.

$$H_2 H_1 A = R \quad \Rightarrow \quad A = H_1 H_2 R.$$

Recall that H is orthogonal and symmetric: $H^{-1} = H^T = H$.

evolution of the matrix

$$\left[\begin{array}{c|ccc} \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} \\ \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} \\ \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} \\ \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} \end{array}\right] \qquad \left[\begin{array}{c|ccc} \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} \\ \hline 0 & \mathrm{x} & \mathrm{x} & \mathrm{x} \\ 0 & \mathrm{x} & \mathrm{x} & \mathrm{x} \\ 0 & \mathrm{x} & \mathrm{x} & \mathrm{x} \end{array}\right]$$

$$\left[\begin{array}{c|c|cc} \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} \\ \hline 0 & \mathrm{x} & \mathrm{x} & \mathrm{x} \\ 0 & \mathrm{x} & \mathrm{x} & \mathrm{x} \\ 0 & \mathrm{x} & \mathrm{x} & \mathrm{x} \end{array}\right] \qquad \left[\begin{array}{c|c|cc} \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} \\ \hline 0 & \mathrm{x} & \mathrm{x} & \mathrm{x} \\ \cline{2-4} 0 & 0 & \mathrm{x} & \mathrm{x} \\ 0 & 0 & \mathrm{x} & \mathrm{x} \end{array}\right]$$

$$\left[\begin{array}{c|c|c|c} \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} \\ \hline 0 & \mathrm{x} & \mathrm{x} & \mathrm{x} \\ \cline{2-4} 0 & 0 & \mathrm{x} & \mathrm{x} \\ 0 & 0 & \mathrm{x} & \mathrm{x} \end{array}\right] \qquad \left[\begin{array}{c|c|c|c} \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} \\ \hline 0 & \mathrm{x} & \mathrm{x} & \mathrm{x} \\ \cline{2-4} 0 & 0 & \mathrm{x} & \mathrm{x} \\ 0 & 0 & 0 & \mathrm{x} \end{array}\right]$$

computing the Householder vector $\mathbf{v}$, $P = I - \beta \mathbf{v}\mathbf{v}^T$

```
"""
    house(x::Array{Float64,1})

returns the Householder vector and the beta.
"""
function house(x::Array{Float64,1})
    n = length(x)
    y = x[2:n]
    sigma = y'*y
    v = [1; y]
    if(sigma == 0)
        beta = 0
    else
        mu = sqrt(x[1]^2 + sigma)
        if x[1] <= 0
            v[1] = x[1] - mu
        else
            v[1] = -sigma/(x[1] + mu)
        end
        beta = 2*v[1]^2/(sigma + v[1]^2)
        v = v/v[1]
    end
    return v, beta
end
```

the Householder QR

```
"""
    houseQR(A::Array{Float64,2})

returns the Q and R of the QR.
"""
function houseQR(A::Array{Float64,2})
    m = size(A,1)
    n = size(A,2)
    R = deepcopy(A)
    Q = Matrix{Float64}(I,m,m)
    for k = 1:n
        v, b = house(R[k:m,k])
        R[k:m,k:n] = R[k:m,k:n] - b*v*v'*R[k:m,k:n]
        Q[1:m,k:m] = Q[1:m,k:m] - b*Q[1:m,k:m]*v*v'
    end
    return Q, R
end
```

GPU Accelerated QR

the blocked Householder QR

Consider a $3m$-by-$3n$ tiled matrix, $m \geq n$:

$$A = \left[\begin{array}{c|c|c} & \multicolumn{2}{c}{A_{1,2}} \\ \cline{2-3} A_{1,1} & A_{2,2} & A_{2,3} \\ \cline{3-3} & & A_{3,3} \end{array} \right],$$

$A_{1,1}$ is $3m$-by-n,
$A_{1,2}$ is m-by-$2n$, $A_{2,3}$ is m-by-n,
$A_{2,2}$ is $2m$-by-n, $A_{3,3}$ is m-by-n.

The Householder transformations are accumulated in an orthogonal $3m$-by-$3m$ matrix Q.

The upper triangular reduction R of A is written in the matrix A.

Two main references:

- C. Bishof and C.F. Van Loan:
  The WY representation for products of Householder matrices.
  SIAM J. Sci. Stat. Comput. 8(1):s1–s13, 1987.
- *Matrix Computations* by G.H. Golub and C.F. Van Loan, 1996.

evolution of Q and R, I is the identity matrix

$$
\begin{array}{rcl}
A,\ Q & = & \left[\begin{array}{c|c|c} & \multicolumn{2}{c}{A_{1,2}} \\ \cline{2-3} A_{1,1} & A_{2,2} & A_{2,3} \\ \cline{3-3} & & A_{3,3} \end{array}\right], \quad \left[\begin{array}{c|c|c} I & & \\ & I & \\ & & I \end{array}\right] \\
 & \rightarrow & \left[\begin{array}{c|c|c} & \multicolumn{2}{c}{R_{1,2}} \\ \cline{2-3} R_{1,1} & A_{2,2} & A_{2,3} \\ \cline{3-3} & & A_{3,3} \end{array}\right], \quad \left[\begin{array}{c|c|c} Q_1 & I & \\ & & I \end{array}\right] \\
 & \rightarrow & \left[\begin{array}{c|c|c} & \multicolumn{2}{c}{R_{1,2}} \\ \cline{2-3} R_{1,1} & R_{2,2} & R_{2,3} \\ \cline{3-3} & & A_{3,3} \end{array}\right], \quad \left[\begin{array}{c|c|c} Q_1 & Q_2 & \\ & & I \end{array}\right] \\
 & \rightarrow & \left[\begin{array}{c|c|c} & \multicolumn{2}{c}{R_{1,2}} \\ \cline{2-3} R_{1,1} & R_{2,2} & R_{2,3} \\ \cline{3-3} & & R_{3,3} \end{array}\right], \quad \left[\begin{array}{c|c|c} Q_1 & Q_2 & Q_3 \end{array}\right].
\end{array}
$$

Householder Reflectors

The Householder reflector P is represented by a vector $\mathbf{v}$:

$$P = I - \beta \mathbf{v}\mathbf{v}^T, \quad \beta = 2/\mathbf{v}^T\mathbf{v}, \quad P\mathbf{x} = \|\mathbf{x}\|_2 \mathbf{e}_1,$$

where $\mathbf{x}$ is the current column and $\mathbf{e}_1 = (1, 0, \ldots, 0)^T$.

The Householder matrices are aggregated in an orthogonal matrix

$$P_{WY} = I + WY^T,$$

where Y stores the Householder vectors in a trapezoidal shape.
W is defined by the Householder vectors and the corresponding βs.

With this WY representation, the updates to Q and R are

$$\begin{aligned} Q &= Q + Q \star W \star Y^T, \\ R &= R + Y \star W^T \star C, \end{aligned}$$

which involve many matrix-matrix products.

GPU Accelerated QR

1. Householder Reflectors
   - projectors and reflectors
   - the Householder QR
   - the WY representation
2. Multiple Doubles on Graphics Processing Units
   - definition, motivation, and software
   - cost overhead factors
3. Accelerated Blocked Householder QR
   - accumulating the Householder reflectors
   - experimental results for least squares solving

multiple doubles / error free transformations

A multiple double is an unevaluated sum of nonoverlapping 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 : 8.00000000000000E+00 - 6.47112461314111E-32
  quad double : 8.00000000000000E+00 + 3.20475411419393E-65
  octo double : 8.00000000000000E+00 - 9.72609915198313E-129
```

If the result fits a 64-bit double,
then the second double represents the accuracy of the result.

Advantages and disadvantages:

+ predictable overhead, cost of double double $\approx$ complex double
+ exploits hardware double arithmetic
− not true multiprecision, fixed multiples of bits in fractions
− infinitesimal computations not possible because of fixed exponent

motivation: power series arithmetic

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

Assuming the quadratic convergence of Newton's method:

k	$1/k!$	recommended precision	`eps`
7	`2.0e-004`	double precision okay	`2.2e-16`
15	`7.7e-013`	use double doubles	`4.9e-32`
23	`3.9e-023`	use double doubles	
31	`1.2e-034`	use quad doubles	`6.1e-64`
47	`3.9e-060`	use octo doubles	`4.6e-128`
63	`5.0e-088`	use octo doubles	
95	`9.7e-149`	need hexa doubles	`5.3e-256`
127	`3.3e-214`	need hexa doubles	

`eps` is the multiple double precision

software packages

- 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.
- GQD 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.
- 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.

GPU Accelerated QR

1. Householder Reflectors
   - projectors and reflectors
   - the Householder QR
   - the WY representation
2. Multiple Doubles on Graphics Processing Units
   - definition, motivation, and software
   - cost overhead factors
3. Accelerated Blocked Householder QR
   - accumulating the Householder reflectors
   - experimental results for least squares solving

cost overhead factors — Graphics Processing Units

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

Teraflop performance compensates the overhead of quad doubles.

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.

customized software

The code for the arithmetical operations generated by the CAMPARY software was customized for each precision.

- Instead of representing a quad double by an array of four doubles, all arithmetical operations work on four separate variables, one for each double.
  By this customization an array of quad doubles is stored as four separate arrays of doubles and a matrix of quad doubles is represented by four matrices of doubles.
- The `double2` and `double4` types of the CUDA SDK work for double double and quad double, but not for the more general multiple double arithmetic.
- QDlib provides definitions for the square roots and various other useful functions for double double and quad double arithmetic. Those definitions are extended to octo double precision.

2-norm of a vector of complex numbers

Let $\mathbf{z} = (z_1, z_2, \ldots, z_n)$ be a vector of complex numbers.

$$\|\mathbf{z}\|_2 = \sqrt{\mathbf{z}^H \mathbf{z}} = \sqrt{\overline{z_1} z_1 + \overline{z_2} z_2 + \cdots + \overline{z_n} z_n}$$

- For small n, only one block of threads is launched.
- The prefix sum algorithm proceeds in $\log_2(n)$ steps.
- For large n, subsums are accumulated for each block.

Specific for the accelerated multiple double implementation:

- The size of shared memory needs to be set for each precision.
- $\sqrt{\ }$ applies Newton's method for staggered precisions.
- The use of registers increases as the precision increases ...

GPU Accelerated QR

1. Householder Reflectors
   - projectors and reflectors
   - the Householder QR
   - the WY representation
2. Multiple Doubles on Graphics Processing Units
   - definition, motivation, and software
   - cost overhead factors
3. Accelerated Blocked Householder QR
   - accumulating the Householder reflectors
   - experimental results for least squares solving

accumulating the Householder reflectors

Using the WY representation, the updates to Q and R are

$$\begin{aligned} Q &= Q + Q \star W \star Y^T, \\ R &= R + Y \star W^T \star C, \end{aligned}$$

which involve many matrix-matrix products.

Early GPU implementations:

- Baboulin, Dongarra, and Tomov, TR UT-CS-08-200, 2008
- Kerr, Campbell, and Richards, GPGPU'09 conference
- Volkov and Demmel, Conference on Supercomputing, 2008

four stages, several kernels

Algorithm 2: BLOCKED ACCELERATED HOUSEHOLDER QR.

Input : N is the number of tiles,
n is the size of each tile,
M is the number of rows, $M \geq Nn$,
A is an M-by-Nn matrix.

Output : Q is an orthogonal M-by-M matrix,
R is an M-by-Nn matrix, $A = QR$.

For k from 1 to N do

1. Compute Householder vectors for one tile, reduce $R_{k,k}$.
2. Define Y, compute W and $Y \star W^T$.
3. Add $Q \star YW^T$ to update Q.
4. If $k < N$, add $YW^T \star C$ to update R.

Householder QR is cubic in the dimension

The code is written for multiple double precision.

At `github.com/janverschelde/PHCpack`, GPL-3.0 License.

Thanks to the high Compute to Global Memory Access ratios of multiple double precision, entries of a matrix can be loaded directly into the registers of a kernel (bypassing shared memory).

The computation cost is proportional to M^3, for $M = Nn$.
The hope is to reduce the cost by a factor of M.

data staging

A matrix A of multiple doubles is stored as $[A_1, A_2, \ldots, A_m]$,

- A_1 holds the most significant doubles of A,
- A_m holds the least significant doubles of A.

Similarly, $\mathbf{b}$ is an array of m arrays $[\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_m]$,
sorted in the order of significance.

In complex data, real and imaginary parts are stored separately.

The main advantages of this representation are twofold:

1. facilitates staggered application of multiple double arithmetic,
2. benefits efficient memory coalescing,
   as adjacent threads in one block of threads read/write adjacent data in memory, avoiding bank conflicts.

GPU Accelerated QR

1. Householder Reflectors
   - projectors and reflectors
   - the Householder QR
   - the WY representation
2. Multiple Doubles on Graphics Processing Units
   - definition, motivation, and software
   - cost overhead factors
3. Accelerated Blocked Householder QR
   - accumulating the Householder reflectors
   - experimental results for least squares solving

experimental setup

About the input matrices:

- Random numbers are generated for the input matrices.
- Condition numbers of random triangular matrices almost surely grow exponentially [Viswanath & Trefethen, 1998].
- In the standalone tests, the upper triangular matrices are the Us of an LU factorization of a random matrix, computed by the host.

Two input parameters are set for every run:

- The size of each tile is the number of threads in a block.
  The tile size is a multiple of 32.
- The number of tiles equals the number of blocks.
  As the V100 has 80 streaming multiprocessors,
  the number of tiles is at least 80.

5 GPUs on 8×128, double double, milliseconds, Gigaflops

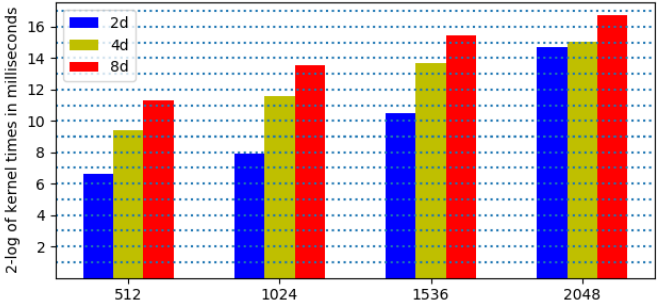
stage in	Linux on the host				Windows
Algorithm 2	C2050	K20C	P100	V100	RTX 2080
β, v	35.5	43.8	21.4	16.2	26.2
$\beta R^T \star v$	418.8	897.8	89.6	76.6	389.7
update R	107.0	107.6	23.0	15.2	47.5
compute W	1357.8	1631.8	349.2	222.4	1298.4
$Y \star W^T$	100.0	50.3	9.7	6.6	153.5
$Q \star WY^T$	790.9	423.9	77.2	52.1	1228.8
$YWT \star C$	6068.5	2345.2	141.2	61.6	822.6
$Q + QWY$	2.4	1.6	0.4	0.4	0.7
$R + YWTC$	7.4	4.2	0.7	0.5	0.8
all kernels	8888.3	5506.1	712.4	451.5	3968.2
wall clock	9083.0	5682.0	826.0	568.0	4700.0
kernel flops	115.8	187.0	1445.3	2280.4	259.5
wall flops	113.4	181.2	1247.2	1812.7	219.1

2-logarithms of times on the V100 in 3 precisions

For increasing dimensions,
we expect the time to be multiplied by 8 when the dimension doubles.

Increasing precision, multipliers are 11.7 (2d to 4d) and 5.4 (4d to 8d).

$512 = 4 \times 128$, $1024 = 8 \times 128$, $1536 = 12 \times 128$, $2048 = 16 \times 128$

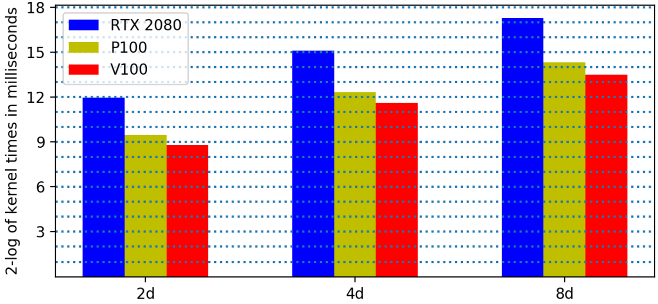


The performance drops at 2048 in double double precision.

2-logarithms of times on 3 GPUs in 3 precisions

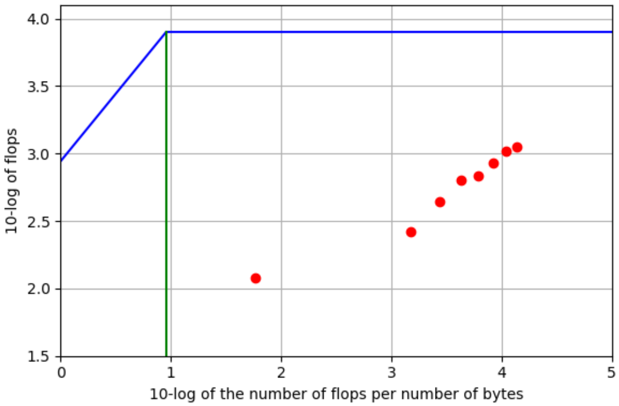
Increasing precision, multipliers are 11.7 (2d to 4d) and 5.4 (4d to 8d).

On 8 tiles for size 128:



The observed cost overhead factors are less than the multipliers.
The heights of the bars follow a regular pattern.

back substitution in quad double precision on the V100



Arithmetic intensity (1) and kernel time flops (2), for dimensions $80 \times n$:

n	32	64	96	128	160	192	224	256
(1)	58.71	1500	2740	4308	6203	8427	10980	13860
(2)	119.1	263.9	440.7	633.8	679.0	852.9	1036.0	1113.6

least squares on the V100 in 4 precisions, 8×128

BS = Back Substitution, times in milliseconds, flops unit is Gigaflops.

stage	1d	2d	4d	8d
QR kernel time	157.9	451.1	3020.6	11924.5
QR wall time	204.0	566.0	3203.0	12244.0
BS kernel time	2.0	4.0	28.0	114.5
BS wall time	4.0	7.0	35.0	127.0
QR kernel flops	303.4	2282.2	3369.8	4041.4
QR wall flops	235.1	1819.6	3177.8	3936.1
BS kernel flops	8.1	89.8	127.9	149.1
BS wall flops	4.2	49.8	102.9	134.5
total kernel flops	299.6	2262.9	3340.0	4004.4
total wall flops	230.8	1797.3	3144.7	3897.0

Teraflop performance is attained already in double double precision.

additional references

- J.R. Shewchuk. **Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates.** *Discrete & Computational Geometry* 18, 305–363, 1997.

- `https://docs.juliahub.com/MultiFloats`
  `MultiFloats.jl` is a Julia package by David K. Zhang which outperforms the `BigFloat` multiprecision arithmetic.

- N. Maho. **MPLAPACK version 2.0.1. user manual.** `arXiv:2109.13406v2 [cs.MS] 12 Sep 2022`.
  Offers support for quad double linear algebra operations and for arbitrary multiprecision linear system solvers.

- 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.

summary and exercises

The blocked Householder is rich in matrix-matrix multiplications, well suited for GPU acceleration. Teraflop performance compensates for cost overhead of multiple double precision.

Exercises:

1. Provide a justification for the formulas in the WY representation: $Q = Q + Q \star W \star Y^T$ and $R = R + Y \star W^T \star C$.
2. Use the `MultiFloats.jl` to build a multiple double version of the Julia `houseQR` function.
3. Describe a high level parallel Julia implementation by referring to the proper BLAS operations in the algorithm.