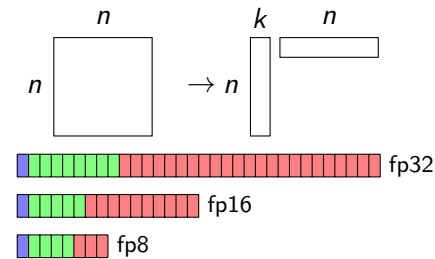
Nineteen dubious ways to compute low-rank approximations in mixed precision

Theo Mary
Sorbonne Université, CNRS, LIP6
theo.mary@lip6.fr
https://perso.lip6.fr/Theo.Mary/

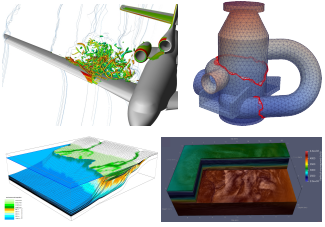
Talk at LIP, ENS Lyon
18 November 2024



Challenges of computing at exascale

Exascale applications:

- Large scale computations and datasets
- Complex requirements (speed, storage, energy, and accuracy constraints)
- Numerically sensitive/difficult



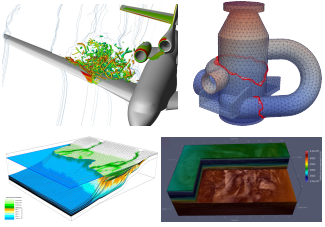
Exascale computers:

- Huge amounts of parallelism/concurrency
- High heterogeneity in the computing units: CPUs, GPUs, other accelerators
- Large gap between speed of computations and communications
- Expensive power consumption

Challenges of computing at exascale

Exascale applications:

- Large scale computations and datasets
- Complex requirements (speed, storage, energy, and accuracy constraints)
- Numerically sensitive/difficult



Exascale methods and software??

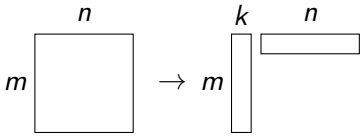
Exascale computers:

- Huge amounts of parallelism/concurrency
- High heterogeneity in the computing units: CPUs, GPUs, other accelerators
- Large gap between speed of computations and communications
- Expensive power consumption

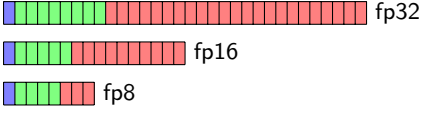
Approximate computing

Approximate computing: introduce *controlled* inexactness to reduce the computational costs and to exploit more efficiently the computer

- **Low-rank approximations:** compress full $m \times n$ matrix A into rank-k product $XY^T \Rightarrow$ reduced storage and cost of operating on A



- **Mixed precision arithmetic:** Combine several precisions with the goal of
  - Maximizing the use of low precisions to match their performance...
  - ...while strategically but parcimoniously using high precisions to preserve their accuracy



Mixed precision

Low-rank approximations

Mixed precision low-rank approximations

Mixed precision

Low-rank approximations

Mixed precision low-rank approximations

Lower precisions: an opportunity

		number of bits			
		signif. (t)	exp.	range	$u = 2^{-t}$
fp128	quadruple	113	15	$10^{\pm 4932}$	1×10^{-34}
fp64	double	53	11	$10^{\pm 308}$	1×10^{-16}
fp32	single	24	8	$10^{\pm 38}$	6×10^{-8}
fp16	half	11	5	$10^{\pm 5}$	5×10^{-4}
bfloat16		8	8	$10^{\pm 38}$	4×10^{-3}
fp8 (e4m3)	quarter	4	4	$10^{\pm 2}$	6×10^{-2}
fp8 (e5m2)		3	5	$10^{\pm 5}$	1×10^{-1}

- **Great benefits:**
  - Reduced storage, data movement, and communications
  - Increased speed thanks to increasing hardware support
  - Reduced energy consumption
- However, **low precision** $\equiv$ **low accuracy**

Lower precisions: a necessity?

Peak performance (TFLOPS)

	Pascal 2016	Volta 2018	Ampere 2020	Hopper 2022	Blackwell 2025
fp64	5	8	20	67	40
fp32	10	16	20	67	80
tfloat32	--	--	160	495	2,200
fp16/bfloat16	20	125	320	990	4,500
fp8	--	--	--	2,000	9,000
fp4	--	--	--	--	18,000

NVIDIA Hopper (H100) GPU

fp64/fp16 speed ratio:

- Hopper (2022): 15×
- Blackwell (2025): 112×

Acta Numerica (2022), pp. 347–414
doi:10.1017/S0962492922000022

Mixed precision algorithms in numerical linear algebra

Nicholas J. Higham
Department of Mathematics, University of Manchester, Manchester, M13 9PL, UK
E-mail: nick.higham@manchester.ac.uk

Theo Mary
Sorbonne Université, CNRS, LIP6, Paris, F-75005, France
E-mail: theo.mary@lip6.fr

https://bit.ly/mixed-survey

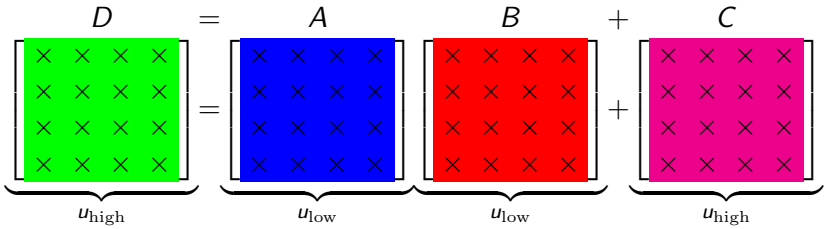
CONTENTS

Three approaches to mixed precision

- **Multiword arithmetic:** emulate high precision arithmetic using low precision computations
- **Adaptive precision:** dynamically adapt the precision at runtime based on the data at hand, switching to low precision only the part of the computations that can be
- **Iterative refinement:** perform the entire computation in low precision, then try to recover a high accuracy

NVIDIA GPU tensor cores

Tensor cores units available on NVIDIA GPUs carry out a mixed precision matrix multiply–accumulate ($u_{\text{high}} \equiv$ fp32 and $u_{\text{low}} \equiv$ fp16/fp8/fp4)



- Let $C = AB$, with $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times p}$, the computed $\widehat{C}$ satisfies

$$|\widehat{C} - C| \lesssim c_n|A||B|, \quad c_n = \begin{cases} nu_{\text{low}} & \text{(uniform low precision)} \\ 2u_{\text{low}} + nu_{\text{high}} & \text{(tensor cores)} \\ nu_{\text{high}} & \text{(uniform high precision)} \end{cases}$$

Blanchard, Higham, Lopez, M., Pranesh (2020)

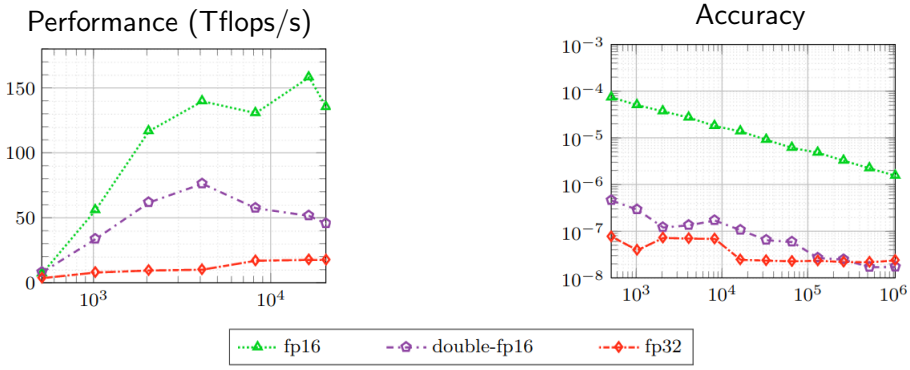
Multiword arithmetic

- Represent high precision number as the unevaluated sum of lower precision numbers
  - Double-double arithmetic: $x = \underbrace{x_1}_{\text{fp64}} + \underbrace{x_2}_{\text{fp64}} \rightarrow \approx$ quad precision accuracy
  - Double-fp16 arithmetic: $x = \underbrace{x_1}_{\text{fp16}} + \underbrace{x_2}_{\text{fp16}} \rightarrow \approx$ single precision accuracy

- Multiword matrix multiplication in mixed precision:
  - Decompose $A \approx \sum_{i=1}^{p} A_i$, $B \approx \sum_{j=1}^{p} B_j$ where each A_i and B_j is stored in precision u_{low}
  - Compute $C = \sum_{i+j<p} A_i B_j$ in precision u_{high} ($p(p+1)/2$ products)
  - $|\widehat{C} - C| \lesssim (u_{\text{low}}^p + n u_{\text{high}})|A||B|$ Fasi, Higham, Lopez, M., Mikaitis (2023)

- Implementation on GPU tensor cores:
  - Can benefit from fp32 accumulation to compute C
  - Double-fp16 arithmetic $\Rightarrow 3\times$ more flops, but entirely in fp16 tensor core arithmetic

Double-fp16 arithmetic: $C \approx A_1B_1 + A_1B_2 + A_2B_1$ computed via 3 tensor core products



Iterative refinement for $Ax = b$

An algorithm to refine the solution: **iterative refinement** (IR)

Choose an initial x_0
while Not converged **do**
    $r_i = b - Ax_i$
    Solve $Ad_i \approx r_i$
    $x_{i+1} = x_i + d_i$
end while

Many variants over the years, depending on choice of precisions and solver for $Ad_i = r_i$

Iterative refinement for $Ax = b$

Factorize $A = LU$ **at precision $\mathbf{u_f} \gg \mathbf{u}$**
Solve $Ax_1 = b$ via $x_1 = U^{-1}(L^{-1}b)$ **at precision $\mathbf{u_f} \gg \mathbf{u}$**
repeat
  $r_i = b - Ax_i$ **at precision $\mathbf{u_r} \ll \mathbf{u}$**
  Solve $Ad_i = r_i$ via $d_i = U^{-1}(L^{-1}r_i)$ **at precision $\mathbf{u_f} \gg \mathbf{u}$**

  $x_{i+1} = x_i + d_i$ **at precision u**
until converged

- **LU-based IR**
  - Exploit LU factorization in low precision. Can solve systems to full fp64 accuracy with only an fp32 factorization, **as long as** $\kappa(A) \leq 10^7$ Langou et al. (2006)
  - Three-precision general analysis by Carson and Higham (2018)

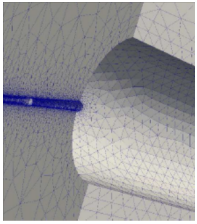
Iterative refinement for $Ax = b$

Factorize $A = LU$ **at precision $\mathbf{u_f} \gg \mathbf{u}$**
Solve $Ax_1 = b$ via $x_1 = U^{-1}(L^{-1}b)$ **at precision $\mathbf{u_f} \gg \mathbf{u}$**
repeat
  $r_i = b - Ax_i$ **at precision $\mathbf{u_r} \ll \mathbf{u}$**
  Solve $Ad_i = r_i$ with preconditioned GMRES
    at precision $\mathbf{u_g} \leq \mathbf{u}$ except matvecs **at precision $\mathbf{u_p} \leq \mathbf{u}^2$**
  $x_{i+1} = x_i + d_i$ **at precision u**
until converged

- **LU-based IR**
  - Exploit LU factorization in low precision. Can solve systems to full fp64 accuracy with only an fp32 factorization, **as long as $\kappa(A) \leq 10^7$** Langou et al. (2006)
  - Three-precision general analysis by Carson and Higham (2018)
- **GMRES-based IR**
  - Use GMRES to solve correction equation $Ad_i = r_i$, preconditioned by low-precision LU factors. Can handle up to $\kappa(A) \leq 10^{16}$ Carson and Higham (2017)
  - Five-precision general analysis by Amestoy, Buttari, Higham, L'Excellent, M., Vieublé (2024)

Iterative refinement for $Ax = b$

- thmgaz matrix ($n = 5M$)
  - multi-physics (thermo-hydro-mechanics)
  - 2 MPI $\times$ 18 threads
  - MUMPS solver Amestoy, Buttari, L'Excellent, M. (2019)



(from code_aster)

	Facto. time (s)	Memory (GB)
Full-rank double	98	192
BLR ($\varepsilon = 10^{-8}$) double	81	131
Full-rank single + LU-IR	65	98
BLR ($\varepsilon = 10^{-8}$) single + LU-IR	59	67
BLR ($\varepsilon = 10^{-6}$) single + GMRES-IR	71	61

Amestoy, Buttari, Higham, L'Excellent, M., Vieublé (2023)

Adaptive precision algorithms

- Given an algorithm and a prescribed accuracy ϵ, adaptively select the minimal precision for each instruction depending on the data

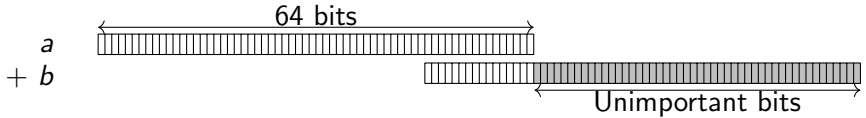
$\Rightarrow$ **First of all, why should the precisions vary?**

Adaptive precision algorithms

- Given an algorithm and a prescribed accuracy ϵ, adaptively select the minimal precision for each instruction depending on the data

$\Rightarrow$ **First of all, why should the precisions vary?**

- Because not all computations are equally "important"!
  Example:



$\Rightarrow$ **Opportunity for mixed precision:** adapt the precisions to the data at hand by storing and computing "less important" (which usually means smaller) data in lower precision

- Goal: compute $y = Ax$, where A is a sparse matrix, with a prescribed accuracy ε
- Given p available precisions $u_1 < \varepsilon < u_2 < \ldots < u_p$, define partition $A = \sum_{k=1}^{p} A^{(k)}$ where

$$a_{ij}^{(k)} = \begin{cases} \mathrm{fl}_k(a_{ij}) & \text{if } |a_{ij}| \in (\varepsilon\|A\|/u_k, \varepsilon\|A\|/u_{k+1}] \\ 0 & \text{otherwise} \end{cases}$$

$\Rightarrow$ the precision of each element is chosen inversely proportional to its magnitude

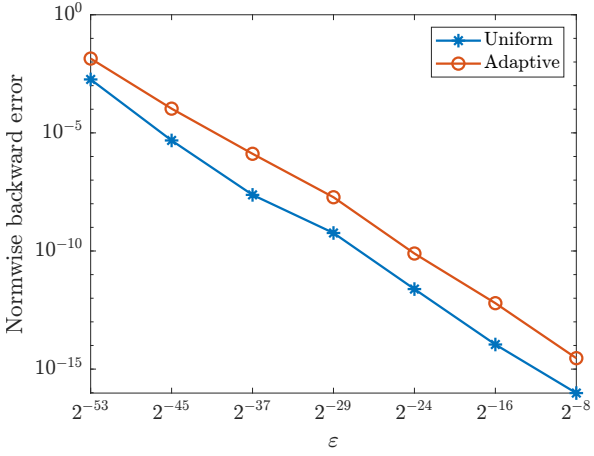
$$\begin{pmatrix} \times & & \times \\ \times & \times & \\ & \times & \times \end{pmatrix} = \begin{pmatrix} d & & \\ & d & \\ & & d \end{pmatrix} + \begin{pmatrix} & & s \\ & & \\ & s & \end{pmatrix} + \begin{pmatrix} h & & \\ & & \\ & & \end{pmatrix}$$

- Compute $y^{(k)} = A^{(k)}x$ in precision u_k and $y = \sum_{k=1}^{p} y^{(k)}$ in precision u_1. The computed $\widehat{y}$ satisfies Graillat, Jézéquel, M., Molina (2024)

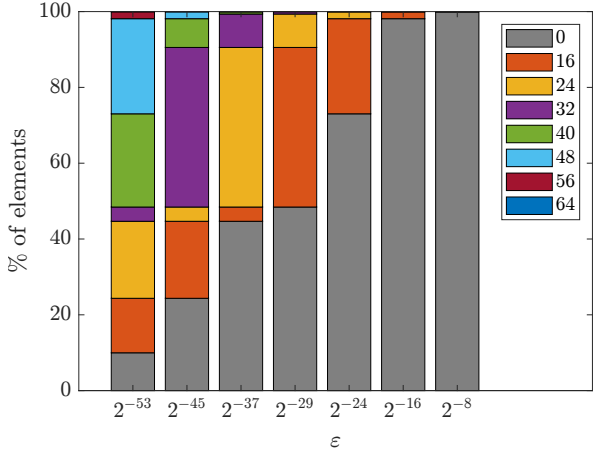
$$\widehat{y} = (A + \Delta A)x, \quad \|\Delta A\| \leq c\varepsilon\|A\|.$$

Adaptive precision SpMV

- 7 precisions: fp64, fp32, and 5 emulated formats with 56, 48, 40, 24, and 16 bits
- Long_Coup_dt6 matrix ($n \approx 1.5$M) Graillat, Jézéquel, M., Molina, Mukunoki (2024)

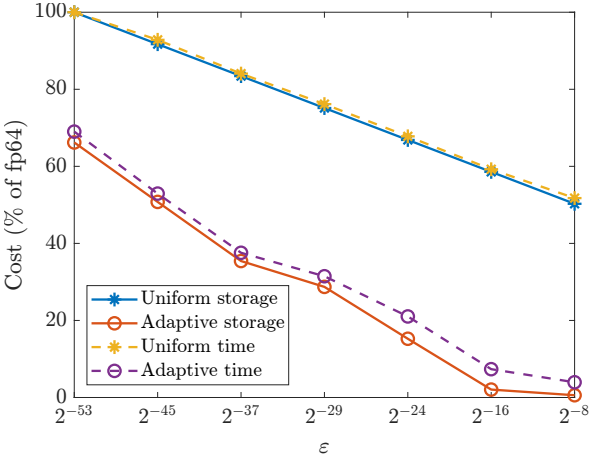


- 7 precisions: fp64, fp32, and 5 emulated formats with 56, 48, 40, 24, and 16 bits
- Long_Coup_dt6 matrix ($n \approx 1.5$M) Graillat, Jézéquel, M., Molina, Mukunoki (2024)



Adaptive precision SpMV

- 7 precisions: fp64, fp32, and 5 emulated formats with 56, 48, 40, 24, and 16 bits
- Long_Coup_dt6 matrix ($n \approx 1.5$M) Graillat, Jézéquel, M., Molina, Mukunoki (2024)

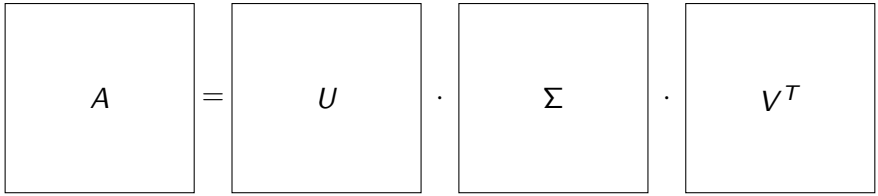


Mixed precision

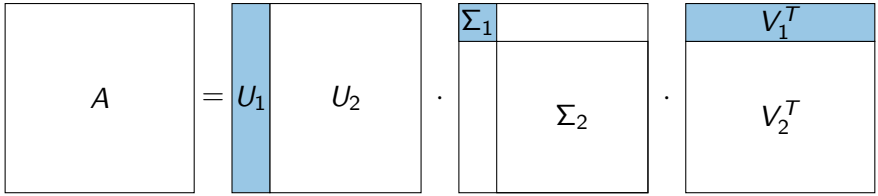
Low-rank approximations

Mixed precision low-rank approximations

Truncated SVD

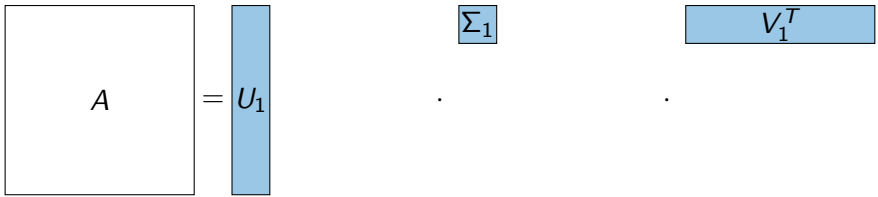


Truncated SVD



$A = U_1 \Sigma_1 V_1^T + U_2 \Sigma_2 V_2^T \quad \text{with} \quad \Sigma_1(k,k) = \sigma_k > \varepsilon,\ \Sigma_2(1,1) = \sigma_{k+1} \leq \varepsilon$

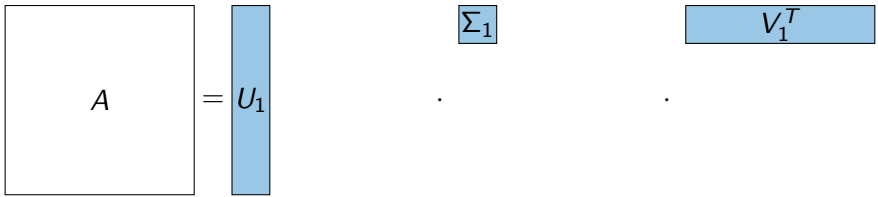
Truncated SVD



$A = U_1\Sigma_1 V_1^T + U_2\Sigma_2 V_2^T \quad \text{with} \quad \Sigma_1(k,k) = \sigma_k > \varepsilon,\ \Sigma_2(1,1) = \sigma_{k+1} \leq \varepsilon$

If $\tilde{A} = U_1\Sigma_1 V_1^T \quad \text{then} \quad \|A - \tilde{A}\|_2 = \|U_2\Sigma_2 V_2^T\|_2 = \sigma_{k+1} \leq \varepsilon$

Truncated SVD

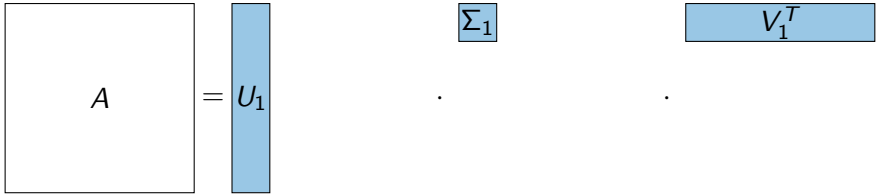


$A = U_1\Sigma_1 V_1^T + U_2\Sigma_2 V_2^T \quad \text{with} \quad \Sigma_1(k,k) = \sigma_k > \varepsilon,\ \Sigma_2(1,1) = \sigma_{k+1} \leq \varepsilon$

If $\tilde{A} = U_1\Sigma_1 V_1^T \quad \text{then} \quad \|A - \tilde{A}\|_2 = \|U_2\Sigma_2 V_2^T\|_2 = \sigma_{k+1} \leq \varepsilon$

If $k < mn/(m+n)$, $\tilde{A}$ requires less storage than $A \Rightarrow$ low-rank matrix.

Truncated SVD



$A = U_1\Sigma_1 V_1^T + U_2\Sigma_2 V_2^T \quad$ with $\quad \Sigma_1(k,k) = \sigma_k > \varepsilon,\ \Sigma_2(1,1) = \sigma_{k+1} \leq \varepsilon$

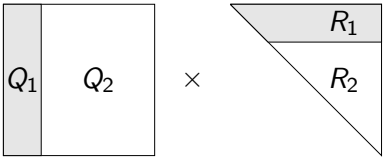
If $\tilde{A} = U_1\Sigma_1 V_1^T \quad$ then $\quad \|A - \tilde{A}\|_2 = \|U_2\Sigma_2 V_2^T\|_2 = \sigma_{k+1} \leq \varepsilon$

If $k < mn/(m+n)$, $\tilde{A}$ requires less storage than $A \Rightarrow$ low-rank matrix.

SVD cost: $O(mn\min(m,n))$ flops $\Rightarrow$ too expensive for large matrices. Other (suboptimal) methods are used in practice.

QR factorization

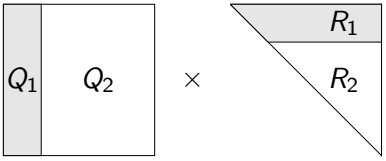
- Low-rank approximations can be computed through a Householder QR factorization $QR = A$:



$$\|A - Q_1 R_1\| \leq \|R_2\|$$

QR factorization

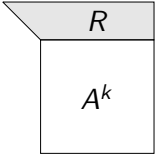
- Low-rank approximations can be computed through a Householder QR factorization $QR = A$:



$$\|A - Q_1 R_1\| \leq \|R_2\|$$

- The QR factorization need not be computed entirely but can actually be truncated

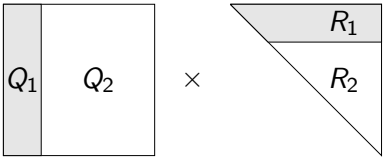
$$Q^{k\,T} A = H^k \dots H^1 A =$$



Because $\|A^k\|$ is monotonically decreasing for $k = 1, \dots, n$, the factorization can be interrupted as soon as $\|A^k\| \leq \varepsilon$.

QR factorization

- Low-rank approximations can be computed through a Householder QR factorization $QR = A$:



$$\|A - Q_1 R_1\| \leq \|R_2\|$$

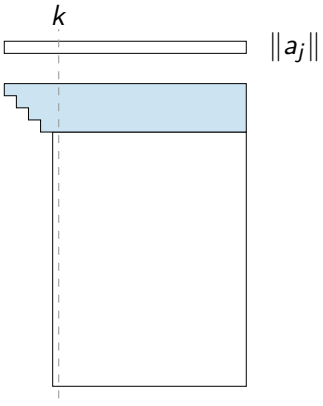
- The QR factorization need not be computed entirely but can actually be truncated

$$Q^k{}^T A = H^k \dots H^1 A = \begin{array}{|c|}\hline R \\ \hline A^k \\ \hline \end{array}$$

Because $\|A^k\|$ is monotonically decreasing for $k = 1, \dots, n$, the factorization can be interrupted as soon as $\|A^k\| \leq \varepsilon$.

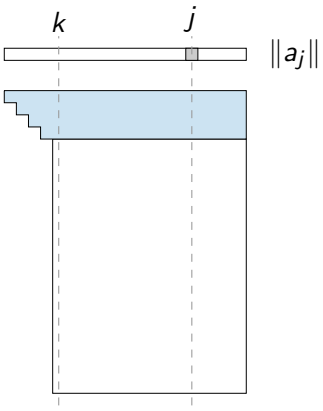
- If column pivoting is used, the QR factorization is rank-revealing and $\|A^k\|$ decays faster

QR with column pivoting (QRCP)



At step $k = 1, \ldots, n$

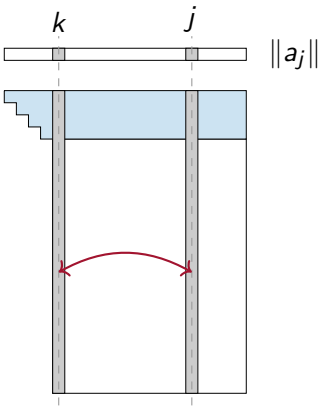
QR with column pivoting (QRCP)



At step $k = 1, \ldots, n$

1. select column j of largest norm

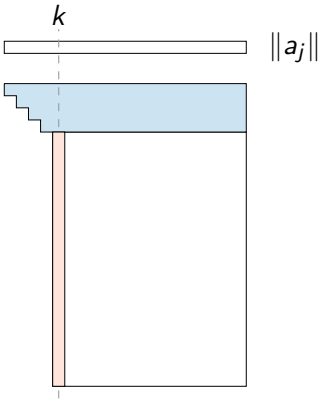
QR with column pivoting (QRCP)



At step $k = 1, \ldots, n$

1. select column j of largest norm
2. permute columns k and j

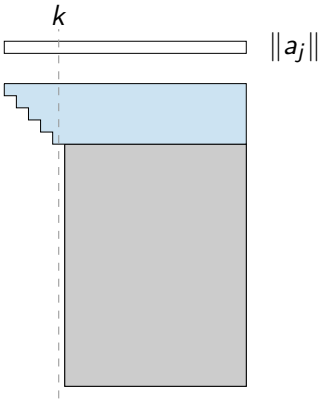
QR with column pivoting (QRCP)



At step $k = 1, \ldots, n$

1. select column j of largest norm
2. permute columns k and j
3. reduce column k via Householder transform

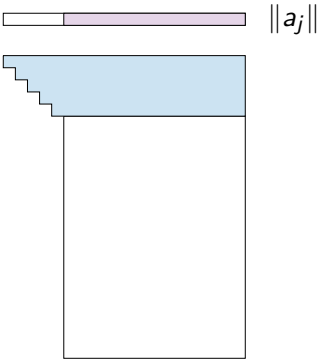
QR with column pivoting (QRCP)



At step $k = 1, \ldots, n$

1. select column j of largest norm
2. permute columns k and j
3. reduce column k via Householder transform
4. update trailing submatrix (at least row k)

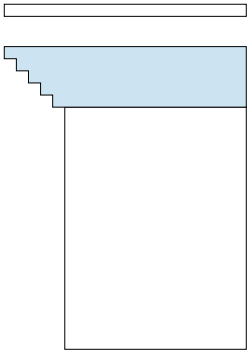
QR with column pivoting (QRCP)



At step $k = 1, \ldots, n$

1. select column j of largest norm
2. permute columns k and j
3. reduce column k via Householder transform
4. update trailing submatrix (at least row k)
5. update column norms

QR with column pivoting (QRCP)



At step $k = 1, \ldots, n$

1. select column j of largest norm
2. permute columns k and j
3. reduce column k via Householder transform
4. update trailing submatrix (at least row k)
5. update column norms

- ▲ Norm of the trailing submatrix is readily available
- ▼ Partial block version exists but still poor computational efficiency and parallelization

Randomized LRA

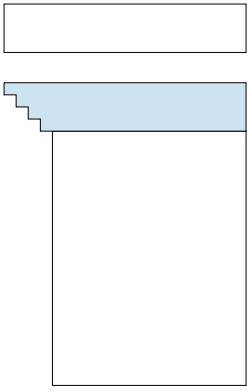
- Let $B = A\Omega$, where $\Omega \in \mathbb{R}^{n \times \ell}$ is a random Gaussian matrix ($\omega_{ij} \sim \mathcal{N}(0,1)$). Then $Q = \mathtt{qr}(B)$ provides a rank-ℓ approximation $A \approx Q(Q^T A)$ as good as the best rank-k approximation for $\ell = k + p$ and small p.
- A rank-k approximation can then be recovered with any deterministic LRA.
  Halko, Martinsson, Tropp (2011)

Input: $A \in \mathbb{R}^{m \times n}$, k, p
Output: $X \in \mathbb{R}^{m \times k}$, $Y \in \mathbb{R}^{n \times k}$ such that $A \approx XY^T$

$\Omega \leftarrow \mathtt{randn}(n, k+p)$
$B \leftarrow A\Omega$
$Q \leftarrow \mathtt{qr}(B)$
$C \leftarrow Q^T A$
$ZY^T \leftarrow \mathtt{LRA}(C, k)$
$X \leftarrow QZ$

- Matrix products are the computational bottleneck $\Rightarrow$ very efficient!

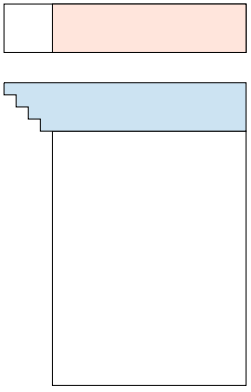
QR with randomized pivoting (QRRP)



Duersch and Gu (2017) Martinsson et al. (2017)

Compute a sample $S = \Omega A$ using a random matrix Ω. At step $k = 1 : b : n$

QR with randomized pivoting (QRRP)

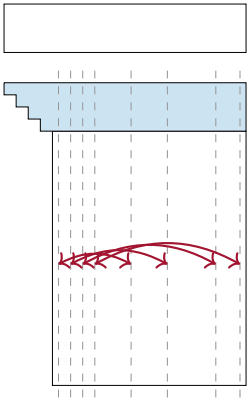


Duersch and Gu (2017) Martinsson et al. (2017)

Compute a sample $S = \Omega A$ using a random matrix Ω. At step $k = 1 : b : n$

1. compute QR of S with B-G pivoting to select the "best" b columns

QR with randomized pivoting (QRRP)

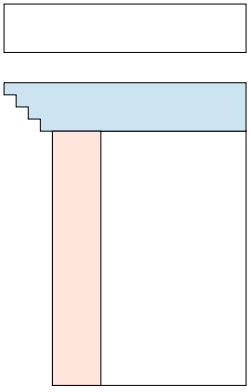


Duersch and Gu (2017) Martinsson et al. (2017)

Compute a sample $S = \Omega A$ using a random matrix Ω. At step $k = 1 : b : n$

1. compute QR of S with B-G pivoting to select the "best" b columns
2. permute the selected columns upfront

QR with randomized pivoting (QRRP)

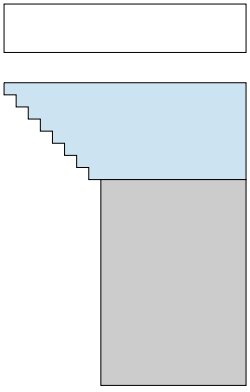


Duersch and Gu (2017) Martinsson et al. (2017)

Compute a sample $S = \Omega A$ using a random matrix Ω. At step $k = 1 : b : n$

1. compute QR of S with B-G pivoting to select the "best" b columns
2. permute the selected columns upfront
3. reduce b columns via Householder transform

QR with randomized pivoting (QRRP)

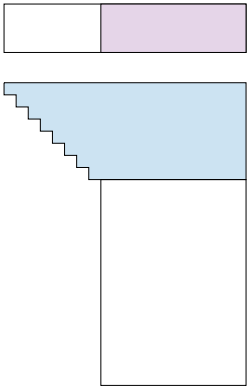


Duersch and Gu (2017) Martinsson et al. (2017)

Compute a sample $S = \Omega A$ using a random matrix Ω. At step $k = 1 : b : n$

1. compute QR of S with B-G pivoting to select the "best" b columns
2. permute the selected columns upfront
3. reduce b columns via Householder transform
4. update trailing submatrix

QR with randomized pivoting (QRRP)

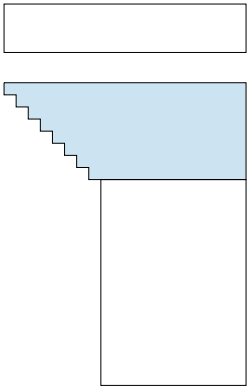


Duersch and Gu (2017) Martinsson et al. (2017)

Compute a sample $S = \Omega A$ using a random matrix Ω. At step $k = 1 : b : n$

1. compute QR of S with B-G pivoting to select the "best" b columns
2. permute the selected columns upfront
3. reduce b columns via Householder transform
4. update trailing submatrix
5. update S

QR with randomized pivoting (QRRP)



Duersch and Gu (2017) Martinsson et al. (2017)

Compute a sample $S = \Omega A$ using a random matrix Ω. At step $k = 1 : b : n$

1. compute QR of S with B-G pivoting to select the "best" b columns
2. permute the selected columns upfront
3. reduce b columns via Householder transform
4. update trailing submatrix
5. update S

- High efficiency and parallelization
- Norm of the trailing submatrix is indirectly available through the sample: $\|s_j\| = \sqrt{b}\|a_j\|$ works ok in practice

Mixed precision

Low-rank approximations

Mixed precision low-rank approximations

Mixed precision low-rank approximations

Three approaches to combine mixed precision and low-rank approximations:

- **Multiword arithmetic:** use multiword arithmetic to accelerate matrix products, by combining randomized methods with fast hardware such as NVIDIA tensor cores
- **Adaptive precision:** adapt the precision to the matrix at hand, taking advantage of the possibly rapid decay of singular values
- **Iterative refinement:** compute a low precision LRA and refine its accuracy iteratively, drawing inspiration from IR for $Ax = b$

Input: $A \in \mathbb{R}^{m\times n}$, k, p
Output: $X \in \mathbb{R}^{m\times k}$, $Y \in \mathbb{R}^{n\times k}$ such that $A \approx XY^T$

$\Omega \leftarrow \texttt{randn}(n, k+p)$
Compute the MW decomp. $A \approx A_1 + A_2$.
Compute the MW decomp. $\Omega \approx \Omega_1 + \Omega_2$.
$B \leftarrow A_1\Omega_1 + A_2\Omega_1 + A_1\Omega_2$
$Q \leftarrow \texttt{qr}(B)$
Compute the MW decomp. $Q \approx Q_1 + Q_2$.
$C \leftarrow A_1^T Q_1 + A_2^T Q_1 + A_1^T Q_2$
$ZY^T \leftarrow \texttt{LRA}(C, k)$
$X \leftarrow QZ$

- If the speed ratio between fp16/fp32 is s, then for asymptotically large matrices this algorithm should be $s/3$ faster compared with the uniform fp32 algorithm.

LRA with multiword arithmetic

- The expectation of the approximation error remains unchanged if $\Omega \sim \mathcal{N}(0, \sigma)$ as long as $\sigma \approx 1$. Ootomo and Yokota (2023)
- Generating Ω in a t-bit arithmetic yields $\sigma \approx 1 + 2^{-t} \Rightarrow$ can store Ω in low precision and reduce the cost of the $A\Omega$ product!

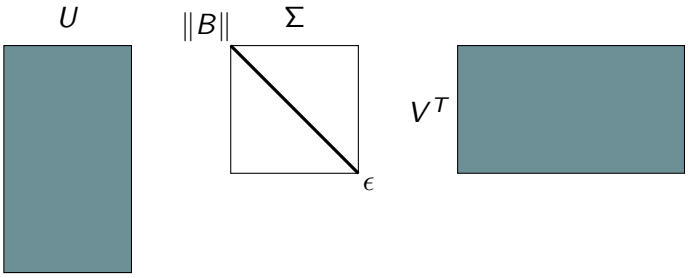
Input: $A \in \mathbb{R}^{m \times n}$, k, p
Output: $X \in \mathbb{R}^{m \times k}$, $Y \in \mathbb{R}^{n \times k}$ such that $A \approx XY^T$

$\Omega_1 \leftarrow \texttt{randn}(n, k+p)$ in fp16.
Compute the MW decomp. $A \approx A_1 + A_2$.
$B \leftarrow A_1\Omega_1 + A_2\Omega_1 + \cancel{A_1\Omega_2}$
$Q \leftarrow \texttt{qr}(B)$
Compute the MW decomp. $Q \approx Q_1 + Q_2$.
$C \leftarrow A_1^T Q_1 + A_2^T Q_1 + A_1^T Q_2$
$ZY^T \leftarrow \texttt{LRA}(C, k)$
$X \leftarrow QZ$

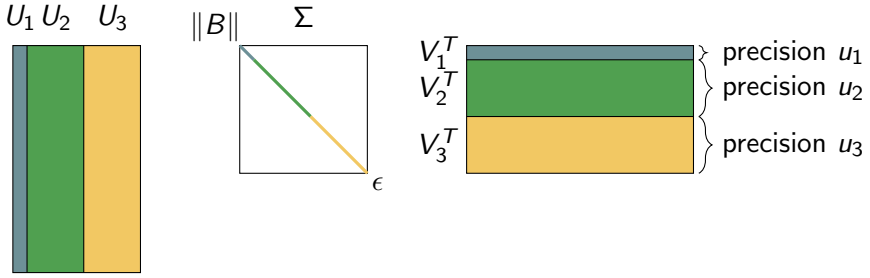
If the speed ratio between fp16/fp32 is s, then for asymptotically large matrices this algorithm should be $2s/5$ faster compared with the uniform fp32 algorithm.

- **Randomized LRA with multiword arithmetic**
  - ▲ Conceptually very simple and transparent for the user
  - ▲ Rigorous control of the accuracy via emulation
  - ▼ Restricted to randomized methods and to fast "tensor core" hardware

Adaptive precision low rank compression

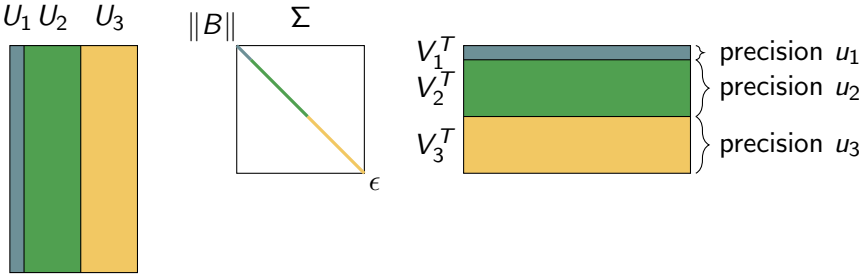


Adaptive precision low rank compression



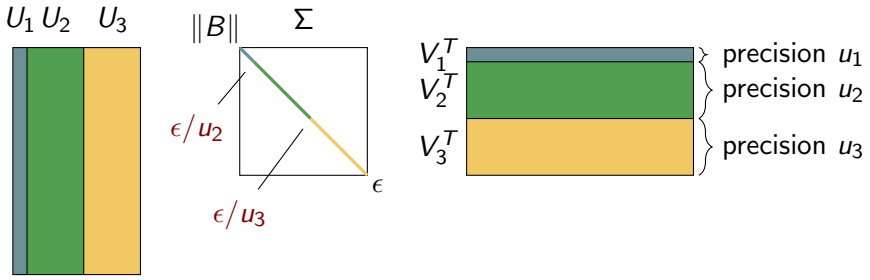
- **Adaptive precision compression:** partition U and V into q groups of decreasing precisions $u_1 \leq \epsilon < u_2 < \ldots < u_q$

Adaptive precision low rank compression



- **Adaptive precision compression:** partition U and V into q groups of decreasing precisions $u_1 \leq \epsilon < u_2 < \ldots < u_q$
- Why does it work? $B = \mathbf{B_1} + \mathbf{B_2} + \mathbf{B_3}$ with $|B_i| \leq O(\|\Sigma_i\|)$

Adaptive precision low rank compression

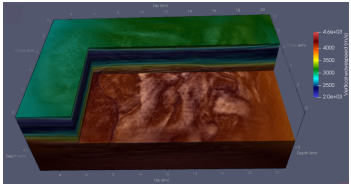


- **Adaptive precision compression:** partition U and V into q groups of decreasing precisions $u_1 \leq \epsilon < u_2 < \ldots < u_q$
- Why does it work? $B = \mathbf{B_1} + \mathbf{B_2} + \mathbf{B_3}$ with $|B_i| \leq O(\|\Sigma_i\|)$
- With p precisions and a partitioning such that $\|\Sigma_k\| \leq \epsilon \|B\|/u_k$, $\|B - \widehat{U}_\epsilon \Sigma_\epsilon \widehat{V}_\epsilon\| \lesssim (2p-1)\epsilon\|B\|$

  Amestoy, Boiteau, Buttari, Gerest, Jézéquel, L'Excellent, M. (2022)

Performance illustration on Full-Waveform Inversion

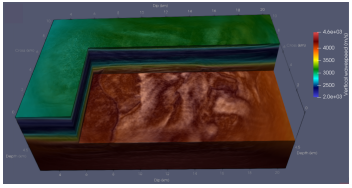
- Adastra MUMPS4FWI project led by WIND team
- Application: Gorgon Model, reservoir 23km x 11km x 6.5km, grid size 15m, Helmholtz equation, 25-Hz
- Complex matrix, 531 Million dofs, storage(A)=220 GBytes;
- FR cost: flops for one LU factorization= 2.6×10^{18};
  Estimated storage for LU factors= 73 TBytes



(25-Hz Gorgon FWI velocity model)

Performance illustration on Full-Waveform Inversion

- Adastra MUMPS4FWI project led by WIND team
- Application: Gorgon Model, reservoir 23km x 11km x 6.5km, grid size 15m, Helmholtz equation, 25-Hz
- Complex matrix, 531 Million dofs, storage(A)=220 GBytes;
- FR cost: flops for one LU factorization= 2.6×10^{18};
  Estimated storage for LU factors= 73 TBytes



(25-Hz Gorgon FWI velocity model)

FR (Full-Rank); BLR with $\varepsilon = 10^{-5}$; 48 000 cores (500 MPI $\times$ 96 threads/MPI)

FR: fp32; Mixed precision BLR: 3 precisions (32bits, 24bits, 16bits) for storage

LU size (TBytes)			Flops		Time BLR + Mixed (sec)			Scaled Resid.
FR	BLR	+mixed	FR	BLR+mixed	Analysis	Facto	Solve	BLR+mixed
73	34	26	2.6×10^{18}	0.5×10^{18}	446	5500	27	7×10^{-4}

in practice: hundreds to thousands of Solve steps (sparse right hand sides (sources))

- Can we compute an LRA directly in adaptive precision form?
- At step i of a the Householder QR factorization, for each column b we compute $\widehat{b}^i = H^i(\widehat{b}^{i-1} + \Delta b^{i-1})$ where $\|\Delta b^{i-1}\| \leq mu\|\widehat{b}^{i-1}\| = mu\|b\|$
- In the specific case of the QR factorization, the H transforms have a peculiar structure:

$$H^i \widehat{b}^{i-1} = \begin{bmatrix} I^{i-1} & \\ & \bar{H}^i \end{bmatrix} \begin{bmatrix} \widehat{b}^{i-1}_{1:i-1} \\ \widehat{b}^{i-1}_{i:m} \end{bmatrix}$$

and, therefore,

$$\|\Delta b^{i-1}\| \leq (m-i)u\|\widehat{b}^{i-1}_{i:m}\|$$

- Can we compute an LRA directly in adaptive precision form?
- At step i of a the Householder QR factorization, for each column b we compute $\widehat{b}^i = H^i(\widehat{b}^{i-1} + \Delta b^{i-1})$ where $\|\Delta b^{i-1}\| \leq mu\|\widehat{b}^{i-1}\| = mu\|b\|$
- In the specific case of the QR factorization, the H transforms have a peculiar structure:

$$H^i \widehat{b}^{i-1} = \begin{bmatrix} I^{i-1} & \\ & \bar{H}^i \end{bmatrix} \begin{bmatrix} \widehat{b}^{i-1}_{1:i-1} \\ \widehat{b}^{i-1}_{i:m} \end{bmatrix}$$

  and, therefore,

$$\|\Delta b^{i-1}\| \leq (m-i)u\|\widehat{b}^{i-1}_{i:m}\|$$

- **Introducing mixed precision:** because all the H^i and $\bar{H}^i$ are unitary transformations, $\|\widehat{b}^{i-1}_{i:m}\|$ will be monotonically decreasing for $i = 1, \ldots, k$ $\longrightarrow$ starting at some i, u can be increased without increasing the error

Theorem

Assume that a truncated QR factorization is computed such that $k \leq n$ Householder transformations are computed and applied to a matrix $A \in \mathbb{R}^{m \times n}$ using p different precisions of increasing unit roundoff u^i. Let k^i be the number of transformations that are computed using precision i. The computed $\widehat{R}_i$ and $\widehat{Q}_i$ satisfy

$$\|A - \sum_{i=1}^{p} \widehat{Q}_i \widehat{R}_i\| \leq \|A^{p+1}\| + \sum_{i=1}^{p} c_{mk^i} u^i \|A^i\|.$$

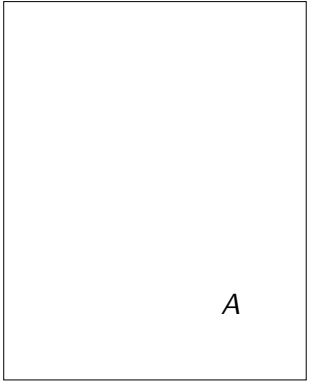
where A^i is the trailing submatrix after $\sum_{j=1}^{i-1} k_j$ transformations.

Buttari, M., Pacteau (2024)

Truncated Householder QR in mixed precision

Using this result into an algorithm:

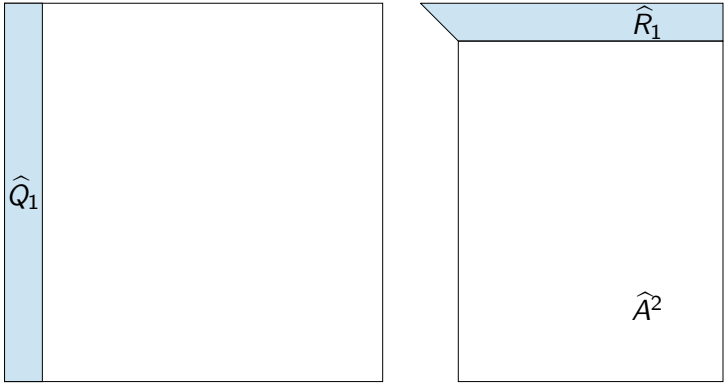
1. start the factorization with $u_1 \leq \varepsilon$



Truncated Householder QR in mixed precision

Using this result into an algorithm:

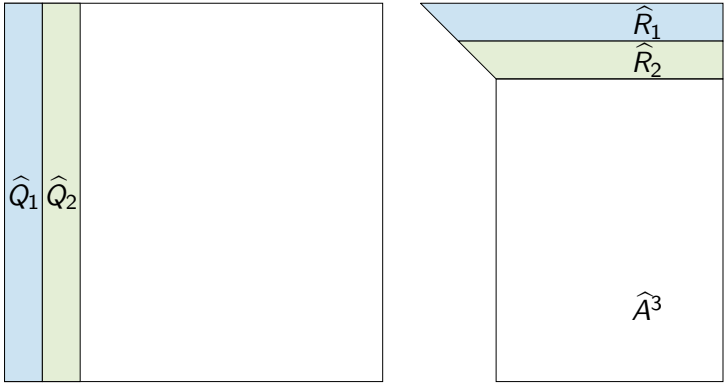
1. start the factorization with $u_1 \leq \varepsilon$
2. if after k_1 transformations $\|A^2\| \leq \varepsilon/u_2\|A\|$, switch to prec. u_2



Truncated Householder QR in mixed precision

Using this result into an algorithm:

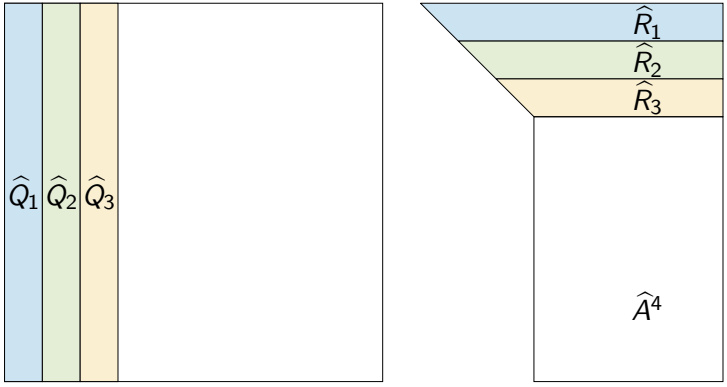
1. start the factorization with $u_1 \leq \varepsilon$
2. if after k_1 transformations $\|A^2\| \leq \varepsilon/u_2\|A\|$, switch to prec. u_2
3. same for precisions $2, \ldots, p$



Truncated Householder QR in mixed precision

Using this result into an algorithm:

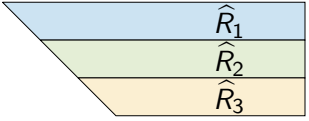
1. start the factorization with $u_1 \leq \varepsilon$
2. if after k_1 transformations $\|A^2\| \leq \varepsilon/u_2\|A\|$, switch to prec. u_2
3. same for precisions $2, \ldots, p$
4. if after $k_1 + \cdots + k_p$ transformations $\|A^{p+1}\| \leq \varepsilon\|A\|$, stop



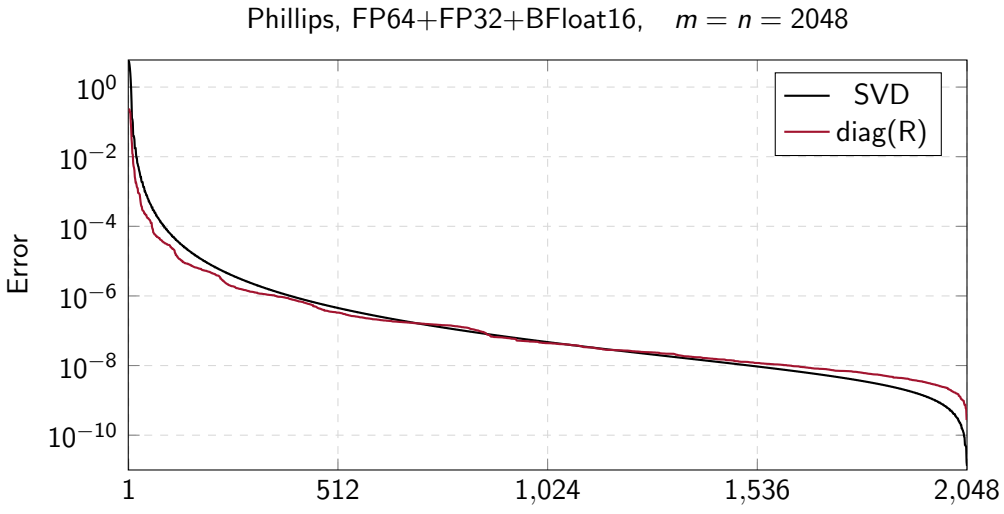
Truncated Householder QR in mixed precision

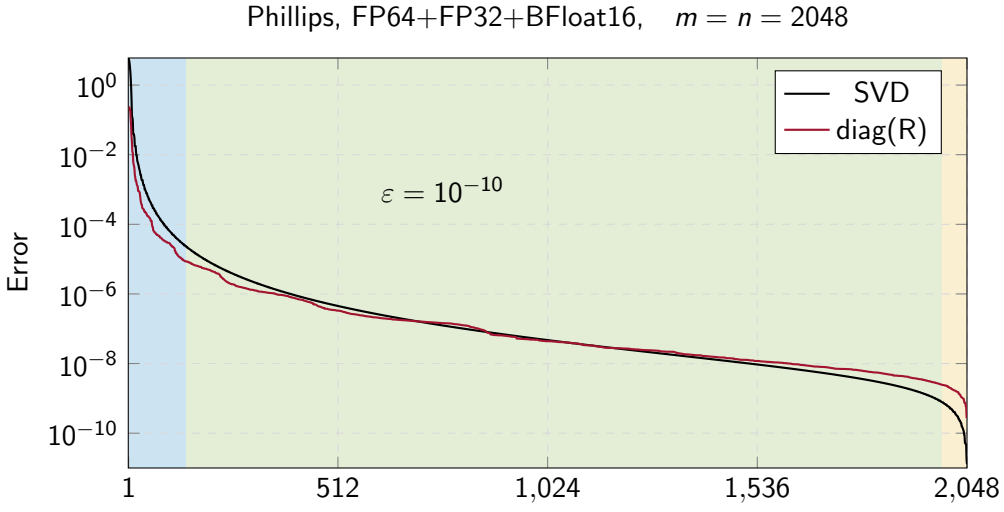
Using this result into an algorithm:

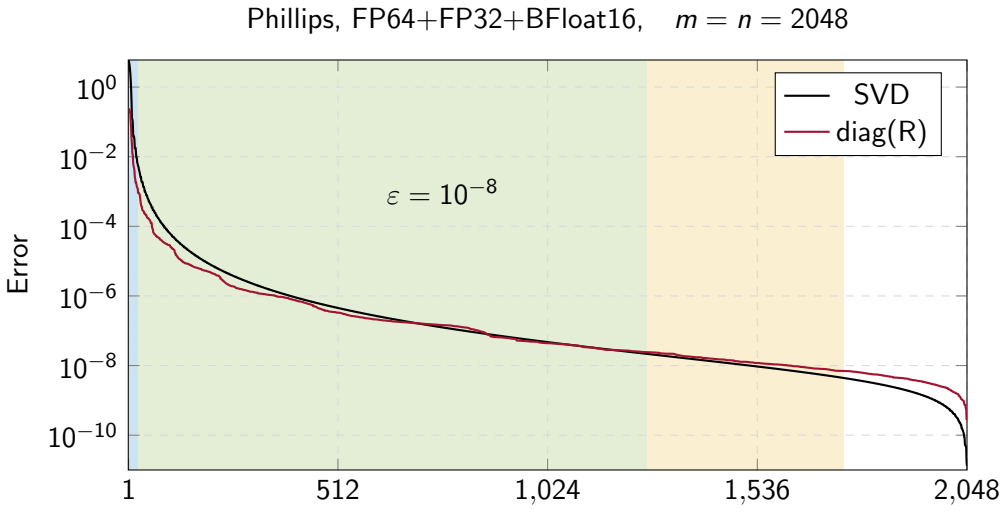
1. start the factorization with $u_1 \leq \varepsilon$
2. if after k_1 transformations $\|A^2\| \leq \varepsilon/u_2\|A\|$, switch to prec. u_2
3. same for precisions $2, \ldots, p$
4. if after $k_1 + \cdots + k_p$ transformations $\|A^{p+1}\| \leq \varepsilon\|A\|$, stop

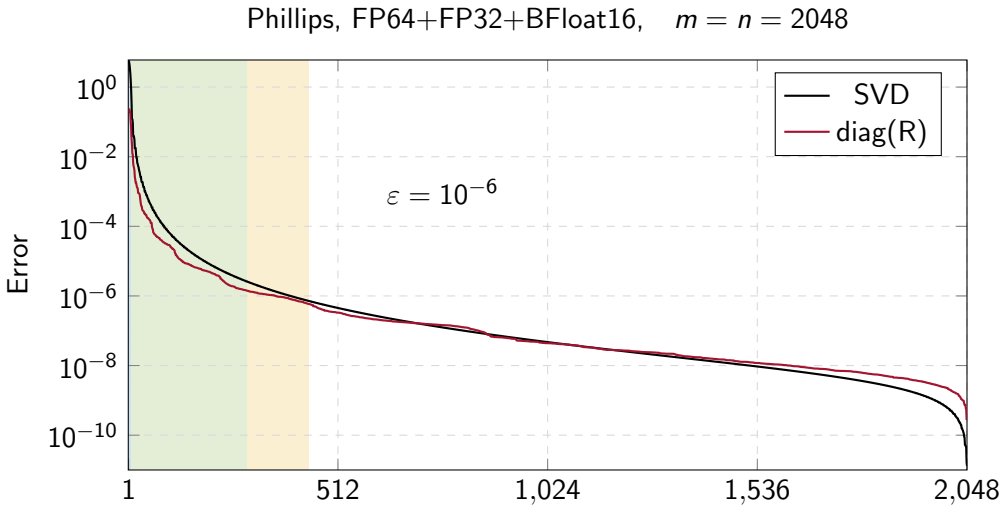


$$\|A - \widehat{Q}_1\widehat{R}_1 - \widehat{Q}_2\widehat{R}_2 - \widehat{Q}_3\widehat{R}_3\| \leq \beta\varepsilon\|A\|$$

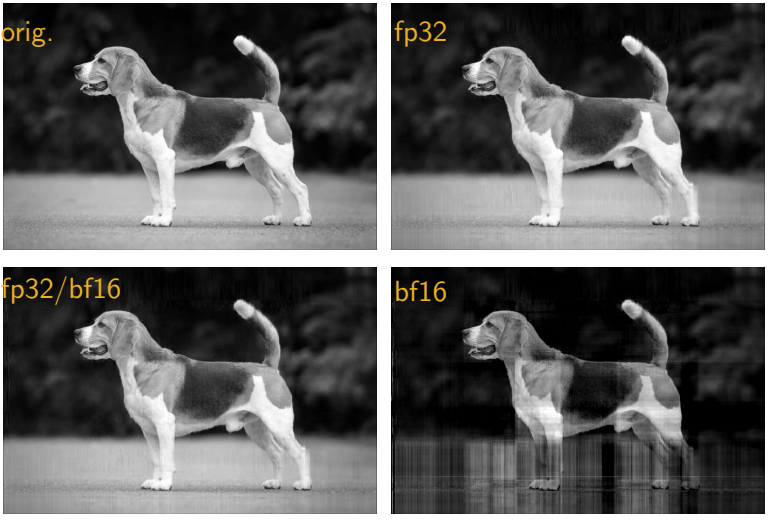

Phillips, FP64+FP32+BFloat16, $m = n = 2048$
Error
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
1
512
1,024
1,536
2,048
SVD
diag(R)


Phillips, FP64+FP32+BFloat16, $m = n = 2048$
Error
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
$\varepsilon = 10^{-10}$
SVD
diag(R)
1
512
1,024
1,536
2,048


Phillips, FP64+FP32+BFloat16, $m = n = 2048$
Error
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
$\varepsilon = 10^{-8}$
SVD
diag(R)
1
512
1,024
1,536
2,048

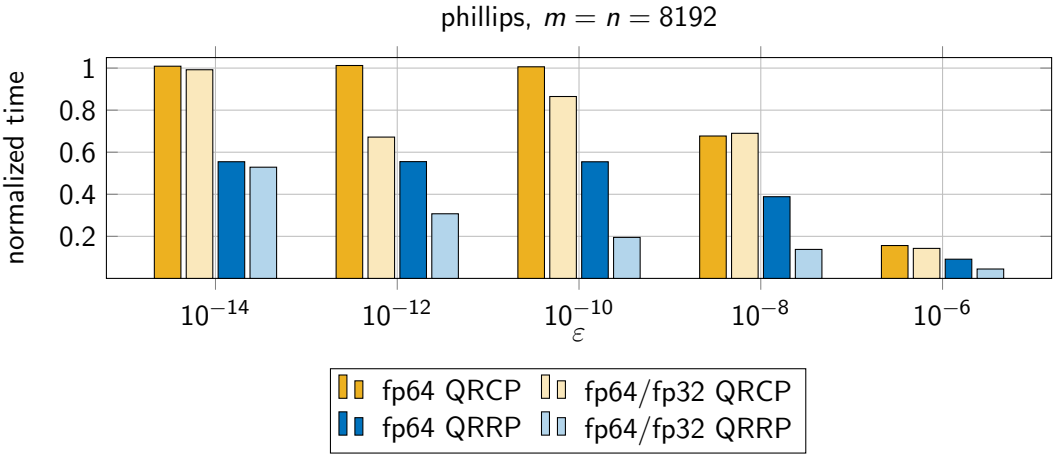

Phillips, FP64+FP32+BFloat16, $m = n = 2048$
Error
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
$\varepsilon = 10^{-6}$
SVD
diag(R)
1
512
1,024
1,536
2,048

Experiments: Julia, image compression



With $\varepsilon = 0.04$ the rank is 191 but only 13 steps are done in fp32 and the rest in bf16 (original size is 1057×1600)

Experiments: Fortran, performance



Discussion

- **Randomized LRA with multiword arithmetic**
  - ▲ Conceptually very simple and transparent for the user
  - ▲ Rigorous control of the accuracy via emulation
  - ▼ Restricted to randomized methods and to fast "tensor core" hardware
- **Adaptive precision LRA**
  - ▲ Can be applied to a wide range of LRA methods
  - ▲ Rigorous control of the accuracy via adaptive criterion
  - ▼ Performance gains conditioned on rapid decay of singular values

Iterative refinement for LRA

Can we refine a low precision LRA into a higher precision one?

1. Apply method to input in low precision
2. Compute residual error in high precision
3. Apply method to residual error in low precision
4. Combine result of (1) and (3) to obtain refined result in high precision

Iterative refinement for LRA

Can we refine a low precision LRA into a higher precision one?

Input: a matrix A
Output: its low-rank factors $X_1 Y_1^T$

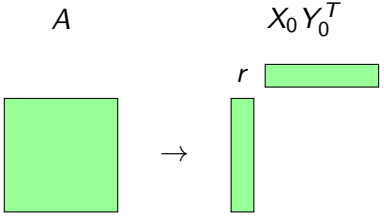
1: Compute LRA $X_0 Y_0^T \approx A$ in precision u_{low}.
2: Compute $E = A - X_0 Y_0^T$ in precision u_{high}.
3: Compute LRA $X_E Y_E^T \approx E$ in precision u_{low}.
4: $X_1 = [X_0 \ X_E]$ and $Y_1 = [Y_0 \ Y_E]$.

Iterative refinement for LRA

Can we refine a low precision LRA into a higher precision one?

Input: a matrix A
Output: its low-rank factors $X_1 Y_1^T$

1: Compute LRA $X_0 Y_0^T \approx A$ in precision u_{low}.
2: Compute $E = A - X_0 Y_0^T$ in precision u_{high}.
3: Compute LRA $X_E Y_E^T \approx E$ in precision u_{low}.
4: $X_1 = [X_0 \ X_E]$ and $Y_1 = [Y_0 \ Y_E]$.

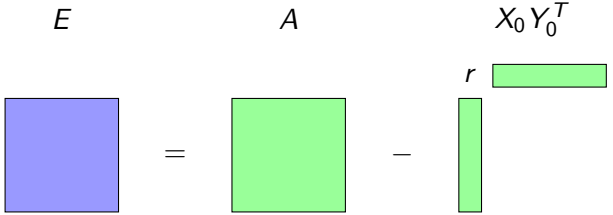


Iterative refinement for LRA

Can we refine a low precision LRA into a higher precision one?

Input: a matrix A
Output: its low-rank factors $X_1 Y_1^T$

1: Compute LRA $X_0 Y_0^T \approx A$ in precision u_{low}.
2: Compute $E = A - X_0 Y_0^T$ in precision u_{high}.
3: Compute LRA $X_E Y_E^T \approx E$ in precision u_{low}.
4: $X_1 = [X_0 \ X_E]$ and $Y_1 = [Y_0 \ Y_E]$.

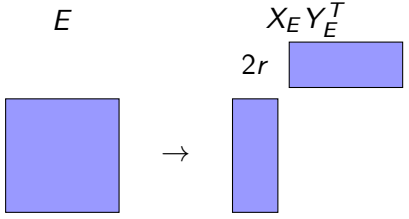


Iterative refinement for LRA

Can we refine a low precision LRA into a higher precision one?

Input: a matrix A
Output: its low-rank factors $X_1 Y_1^T$

1: Compute LRA $X_0 Y_0^T \approx A$ in precision u_{low}.
2: Compute $E = A - X_0 Y_0^T$ in precision u_{high}.
3: Compute LRA $X_E Y_E^T \approx E$ in precision u_{low}.
4: $X_1 = [X_0 \ X_E]$ and $Y_1 = [Y_0 \ Y_E]$.

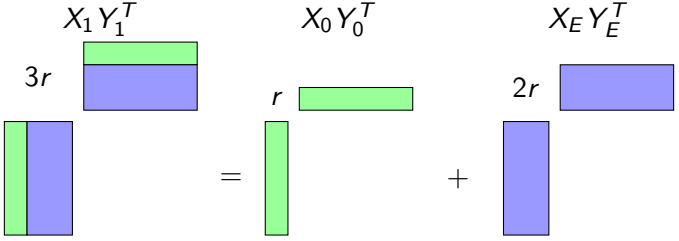


Iterative refinement for LRA

Can we refine a low precision LRA into a higher precision one?

Input: a matrix A
Output: its low-rank factors $X_1 Y_1^T$

1: Compute LRA $X_0 Y_0^T \approx A$ in precision u_{low}.
2: Compute $E = A - X_0 Y_0^T$ in precision u_{high}.
3: Compute LRA $X_E Y_E^T \approx E$ in precision u_{low}.
4: $X_1 = [X_0 \ X_E]$ and $Y_1 = [Y_0 \ Y_E]$.



Iterative refinement for LRA

Can we refine a low precision LRA into a higher precision one?

Input: a matrix A
Output: its low-rank factors $X_1 Y_1^T$

1: Compute LRA $X_0 Y_0^T \approx A$ in precision u_{low}.
2: Compute $E = A - X_0 Y_0^T$ in precision u_{high}.
3: Compute LRA $X_E Y_E^T \approx E$ in precision u_{low}.
4: $X_1 = [X_0 \ X_E]$ and $Y_1 = [Y_0 \ Y_E]$.

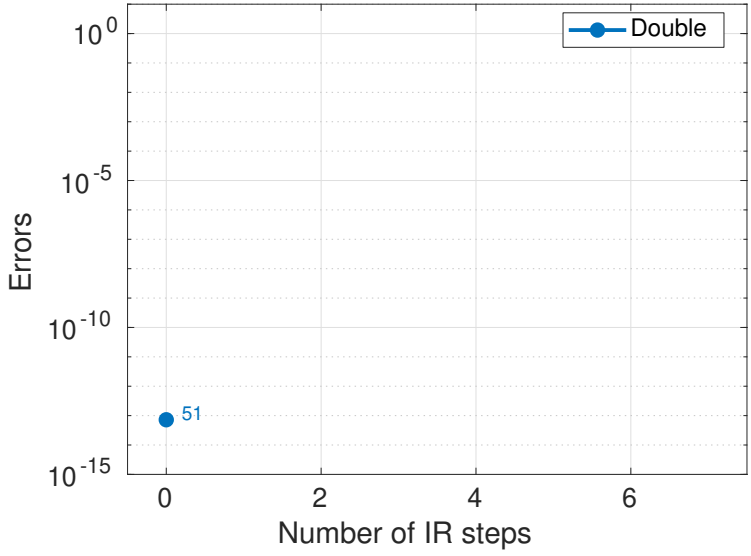
- Can recompress $X_1 Y_1^T$ from rank $3r$ to rank r
- Achieves u_{low}^2 accuracy with most of the work done in precision u_{low}
- Can repeat process: after i iterations, the computed $X_i Y_i^T$ satisfies

$$\|A - X_i Y_i^T\| \leq (\phi^{i+1} + \xi + O(u_{\text{low}} u_{\text{high}}))\|A\|$$

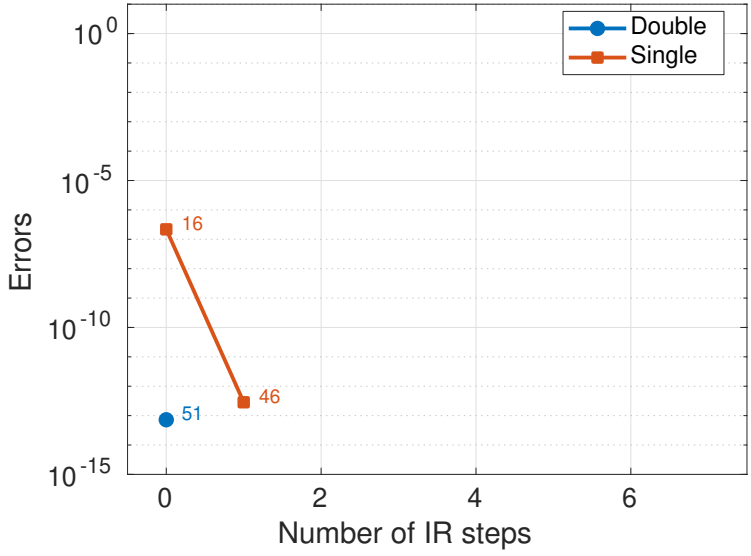
  - $\phi = O(u_{\text{low}})$ is the convergence speed
  - $\xi = O(u_{\text{high}})$ is the attainable accuracy

Baboulin, Kaya, M., Robeyns (2023)

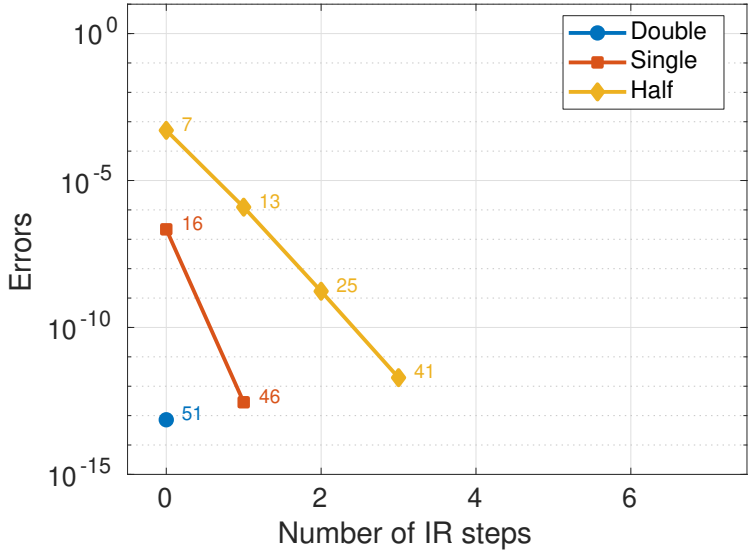
LRA-IR experiments (MATLAB)



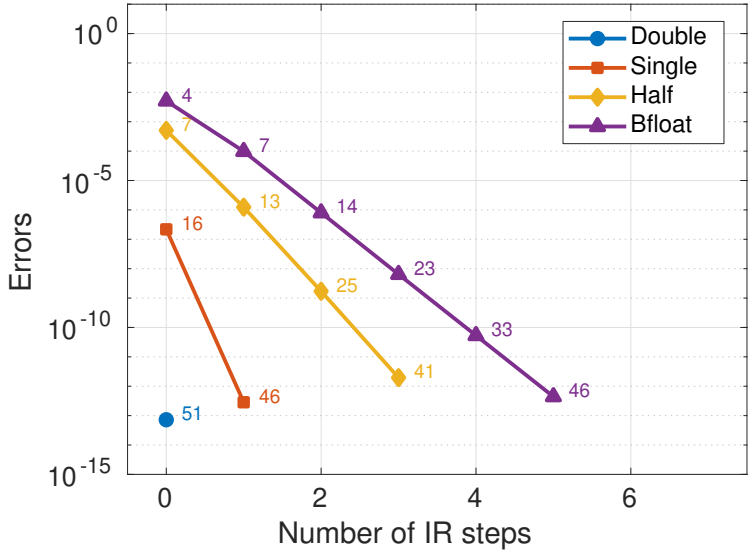
LRA-IR experiments (MATLAB)



LRA-IR experiments (MATLAB)



LRA-IR experiments (MATLAB)



Cost analysis of LRA-IR

IR faster than standard high precision LRA in two scenarios:

- If the ranks r_i at the early iterations are much smaller than the final rank: $r_i \ll r \Rightarrow$ requires rapid decay of singular values
- If the low precision is much faster than high precision $\Rightarrow$ requires specialized hardware (e.g., NVIDIA tensor cores)

LRA-IR therefore bridges the gap between adaptive precision LRA and multiword LRA!

- Large singular values are computed with low precision but high accuracy, à la multiword arithmetic
- Small singular values are computed with low precision and low accuracy

Randomized SVD with tensor cores

Input: a matrix $A_{32} \in \mathbb{R}^{m \times n}$, the target rank k
Output: $X_{16} \in \mathbb{R}^{m \times k}$ and $Y_{16} \in \mathbb{R}^{n \times k}$ such that $A_{32} \approx X_{16} Y_{16}^T$.

$\Omega_{16} = \texttt{randn}(n, k)$
$A_{16} = \texttt{fp16}(A_{32})$
$B_{32} = \texttt{tcgemm}_{16|32}(A_{16}, \Omega_{16})$
$Q_{32} = \texttt{qr}(B_{32})$
$X_{16} = \texttt{fp16}(Q_{32})$
$Y_{16} = \texttt{tcgemm}_{16|32}(A_{16}^T, X_{16})$

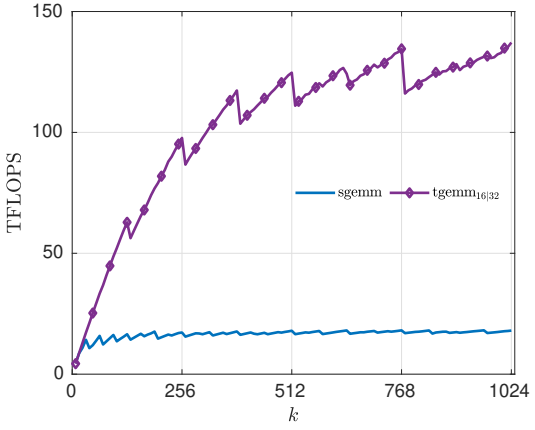
Randomized SVD with tensor cores + IR

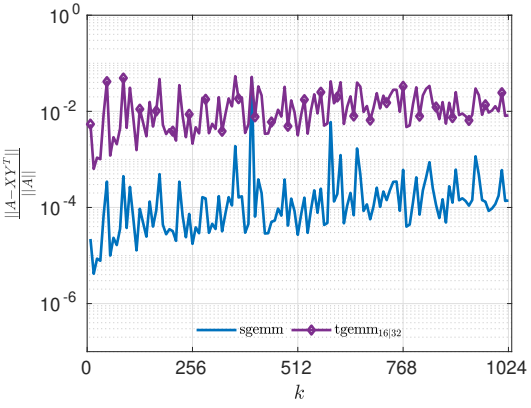
Input: a matrix $A_{32} \in \mathbb{R}^{m \times n}$, the target rank k
Output: $X_{16} \in \mathbb{R}^{m \times k}$ and $Y_{16} \in \mathbb{R}^{n \times k}$ such that $A_{32} \approx X_{16} Y_{16}^T$.

$[X_{16}, Y_{16}] = \texttt{RandLRA}(A_{32}, k)$
$E_{32} = A_{32} - \texttt{tcgemm}_{16|32}(X_{16}, Y_{16}^T)$
$[X'_{16}, Y'_{16}] = \texttt{RandLRA}(E_{32}, 2k)$
$X_{16} = [X_{16}, X'_{16}]$
$Y_{16} = [Y_{16}, Y'_{16}]$

- Since input of `tcgemm` is already in fp16, can use tensor cores to compute E_{32} with fp32 accuracy!

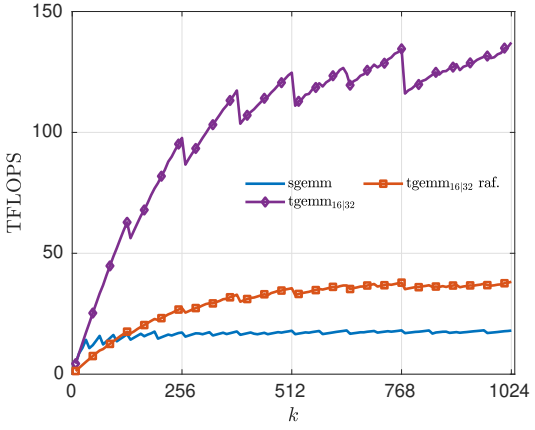
LRA-IR experiments (tensor cores)

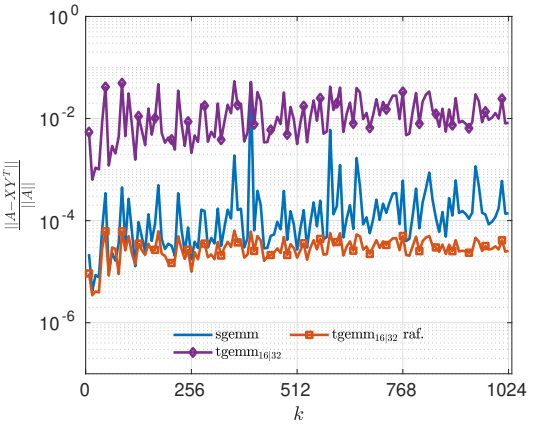




Baboulin, Donfack, Kaya, M., Robeyns (2024)

LRA-IR experiments (tensor cores)





Baboulin, Donfack, Kaya, M., Robeyns (2024)

Discussion

- **Randomized LRA with multiword arithmetic**
  - ▲ Conceptually very simple and transparent for the user
  - ▲ Rigorous control of the accuracy via emulation
  - ▼ Restricted to randomized methods and to fast "tensor core" hardware
- **Adaptive precision LRA**
  - ▲ Can be applied to a wide range of LRA methods
  - ▲ Rigorous control of the accuracy via adaptive criterion
  - ▼ Performance gains conditioned on rapid decay of singular values
- **Iterative refinement for LRA**
  - ▲ Unifies both previous methods: can take advantage of *both* fast hardware and rapid decay of singular values
  - ▲ Rigorous control of the accuracy via refinement
  - ▼ Requires more flops than either of the two previous methods

Discussion

- **Randomized LRA with multiword arithmetic**
  - ▲ Conceptually very simple and transparent for the user
  - ▲ Rigorous control of the accuracy via emulation
  - ▼ Restricted to randomized methods and to fast "tensor core" hardware
- **Adaptive precision LRA**
  - ▲ Can be applied to a wide range of LRA methods
  - ▲ Rigorous control of the accuracy via adaptive criterion
  - ▼ Performance gains conditioned on rapid decay of singular values
- **Iterative refinement for LRA**
  - ▲ Unifies both previous methods: can take advantage of *both* fast hardware and rapid decay of singular values
  - ▲ Rigorous control of the accuracy via refinement
  - ▼ Requires more flops than either of the two previous methods

Take-away: mixed precision, low-rank approximations, and randomization synergize well together!

References

- Multiword matrix multiplication
  - Fasi, Higham, Lopez, M., Mikaitis (2023)
  - Blanchard, Higham, Lopez, M., Pranesh (2020)
- Iterative refinement for $Ax = b$
  - Amestoy, Buttari, Higham, L'Excellent, M., Vieublé (2024)
  - Amestoy, Buttari, Higham, L'Excellent, M., Vieublé (2023)
- Adaptive precision SpMV
  - Graillat, Jézéquel, M., Molina (2024)
  - Graillat, Jézéquel, M., Molina, Mukunoki (2024)
- Adaptive precision LRA
  - Amestoy, Boiteau, Buttari, Gerest, Jézéquel, L'Excellent, M. (2022)
  - Buttari, M., Pacteau (2024)
- Iterative refinement for LRA
  - Baboulin, Kaya, M., Robeyns (2023)
  - Baboulin, Donfack, Kaya, M., Robeyns (2024)

Thanks!
Questions?