Advances in Numerical Linear Algebra

6-8 July 2022

Adaptive Precision Solvers and Preconditioners

Theo Mary

Sorbonne Université, CNRS, LIP6

Slides available at https://bit.ly/happyBirthdayNick

Solution of $Ax = b$, A large and sparse:

- **Direct methods**
  - Robust, black box solvers
  - High time and memory cost for factorization of A

- **Iterative methods**
  - Low time and memory per-iteration cost
  - Convergence is application dependent

Solution of $Ax = b$, A large and sparse:

- **Direct methods**
  - Robust, black box solvers
  - High time and memory cost for factorization of A

  $\Rightarrow$ Need fast factorization

- **Iterative methods**
  - Low time and memory per-iteration cost
  - Convergence is application dependent

  $\Rightarrow$ Need good preconditioner

Solution of $Ax = b$, A large and sparse:

- **Direct methods**
  - Robust, black box solvers
  - High time and memory cost for factorization of A

  $\Rightarrow$ Need fast factorization

- **Iterative methods**
  - Low time and memory per-iteration cost
  - Convergence is application dependent

  $\Rightarrow$ Need good preconditioner

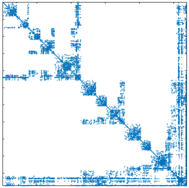
$\Rightarrow$ **Approximate factorizations...**

- as approximate fast direct methods, if
  - low accuracy is sufficient, or
  - matrix is structured (data sparsity)
- as high quality preconditioners otherwise

Dropping approximations (sparsification)

Dropping: replace with zero any value sufficiently small

$$|a_{ij}| \leq \epsilon \|A\| \quad \Rightarrow \quad a_{ij} \leftarrow 0$$

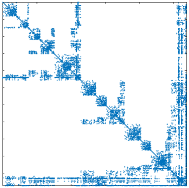


sparse A

Dropping approximations (sparsification)

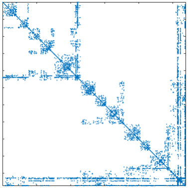
Dropping: replace with zero any value sufficiently small

$$|a_{ij}| \leq \epsilon \|A\| \quad \Rightarrow \quad a_{ij} \leftarrow 0$$



sparse A

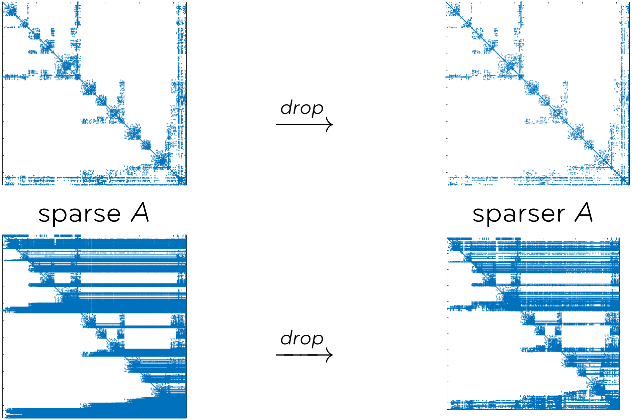
$\xrightarrow{\textit{drop}}$



sparser A

Dropping: replace with zero any value sufficiently small

$$\begin{cases} |\ell_{ij} u_{jj}| \le \epsilon \|A\| & \Rightarrow \quad \ell_{ij} \leftarrow 0 \\ |u_{ij}| \le \epsilon \|A\| & \Rightarrow \quad u_{ij} \leftarrow 0 \end{cases}$$

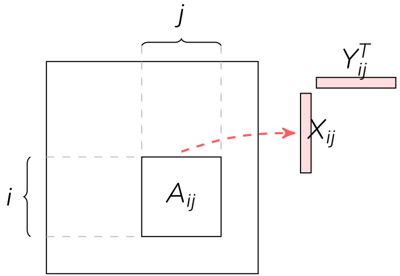


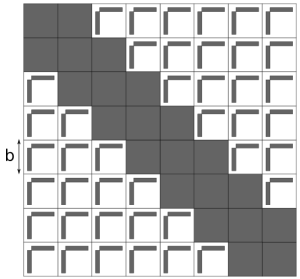
LU factors

incomplete *LU*

Low-rank approximations (data sparsification)

Low-rank compression: given $A = U\Sigma V^T$, if we truncate singular vectors associated with $\sigma_i \leq \epsilon$, we obtain $\widetilde{A}$ such that $\|\widetilde{A} - A\| \leq \epsilon$





Block Low Rank

Compress A_{ij} such that $\|\widetilde{A}_{ij} - A_{ij}\| \leq \epsilon\|A\|$:

- If $\|A_{ij}\| \leq \epsilon\|A\| \Rightarrow A_{ij} \leftarrow 0$ (drop block)
- otherwise replace A_{ij} with $\widetilde{A}_{ij} = X_{ij}Y_{ij}^T$

Evolution of the floating-point landscape

Common point: these methods only deal in absolutes: either we keep the data at full accuracy, or we discard it completely!

Evolution of the floating-point landscape

Common point: these methods only deal in absolutes: either we keep the data at full accuracy, or we discard it completely!

		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}

Evolution of the floating-point landscape

Common point: these methods only deal in absolutes: either we keep the data at full accuracy, or we discard it completely!

		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}

We need **a new paradigm** that uses multiple, gradual levels of approximation

Mixed precision algorithms

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

Mixed precision algorithms

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

Adaptive precision algorithms: an emerging subclass

- Anzt, Dongarra, Flegar, Higham, and Quintana-Orti, *Adaptive precision in block-Jacobi preconditioning for iterative sparse linear system solvers* (2019).
- Doucet, Ltaief, Gratadour, and Keyes, *Mixed-precision tomographic reconstructor computations on hardware accelerator* (2019).
- Ahmad, Sundar, and Hall, *Data-driven mixed precision sparse matrix vector multiplication for GPUs* (2019).
- Ooi, Iwashita, Fukaya, Ida, and Yokota, *Effect of mixed precision computing on H-matrix vector multiplication in BEM analysis* (2020).
- Diffenderfer, Osei-Kuffuor, and Menon, *QDOT: Quantized dot product kernel for approximate high-performance computing* (2021).
- Abdulah, Cao, Pei, Bosilca, Dongarra, Genton, Keyes, Ltaief, and Sun, *Accelerating geostatistical modeling and prediction with mixed-precision computations* (2022).

Adaptive precision algorithms

- Given an algorithm and a prescribed accuracy ϵ, employ the minimal precision for each instruction

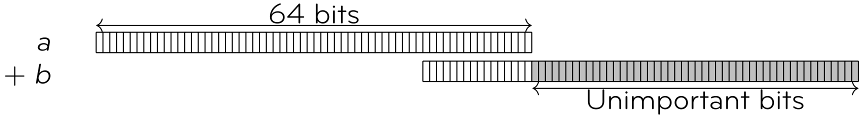
$\Rightarrow$ **First of all, why should the precisions vary?**

Adaptive precision algorithms

- Given an algorithm and a prescribed accuracy ϵ, employ the minimal precision for each instruction

⇒ **First of all, why should the precisions vary?**

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



⇒ **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

Adaptive precision SpMV

Graillat, Jézéquel, M., Molina (2022)

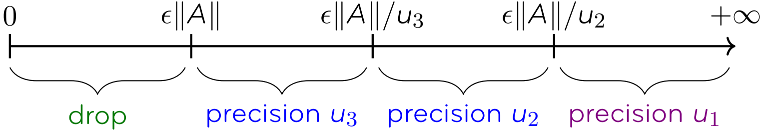
- **Goal:** compute the SpMV $y = Ax$ with accuracy ϵ using q precisions $u_1 \leq \epsilon < u_2 < \ldots < u_q$
- Split elements a_{ij} on each row i into q buckets $B_{i1}, \ldots, B_{iq}$, where bucket B_{ik} uses precision u_k

Adaptive precision SpMV

Graillat, Jézéquel, M., Molina (2022)

- **Goal:** compute the SpMV $y = Ax$ with accuracy ϵ using q precisions $u_1 \le \epsilon < u_2 < \ldots < u_q$
- Split elements a_{ij} on each row i into q buckets $B_{i1}, \ldots, B_{iq}$, where bucket B_{ik} uses precision u_k
- How should we build the buckets?

$$\begin{cases} |a_{ij}| \le \epsilon \|A\| & \Rightarrow \quad \text{drop} \\ |a_{ij}| \in [\epsilon \|A\|/u_{k+1}, \epsilon\|A\|/u_k) & \Rightarrow \quad \text{place in } B_{ik} \\ |a_{ij}| > \epsilon \|A\|/u_2 & \Rightarrow \quad \text{place in } B_{i1} \end{cases}$$

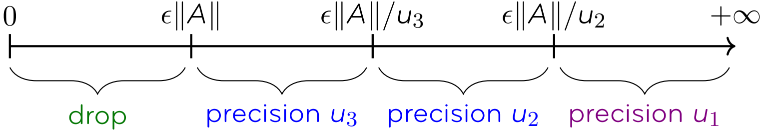


Adaptive precision SpMV

Graillat, Jézéquel, M., Molina (2022)

- **Goal:** compute the SpMV $y = Ax$ with accuracy ϵ using q precisions $u_1 \le \epsilon < u_2 < \ldots < u_q$
- Split elements a_{ij} on each row i into q buckets $B_{i1}, \ldots, B_{iq}$, where bucket B_{ik} uses precision u_k
- How should we build the buckets?

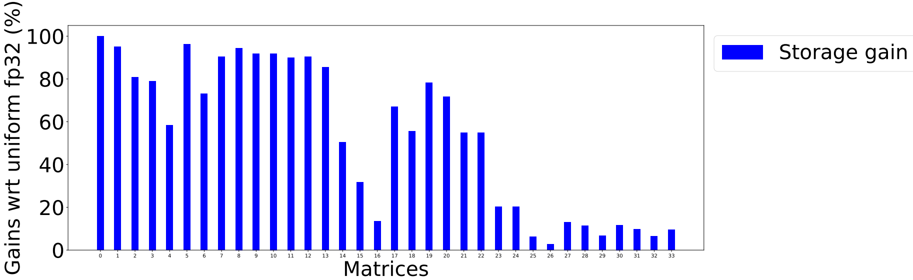
$$\begin{cases} |a_{ij}| \le \epsilon \|A\| & \Rightarrow \text{drop} \\ |a_{ij}| \in [\epsilon\|A\|/u_{k+1}, \epsilon\|A\|/u_k) & \Rightarrow \text{place in } B_{ik} \\ |a_{ij}| > \epsilon\|A\|/u_2 & \Rightarrow \text{place in } B_{i1} \end{cases}$$



- **Theorem**: the computed $\widehat{y}$ satisfies $\|\widehat{y} - y\| \le c\epsilon\|A\|\|x\|$

Adaptive precision SpMV

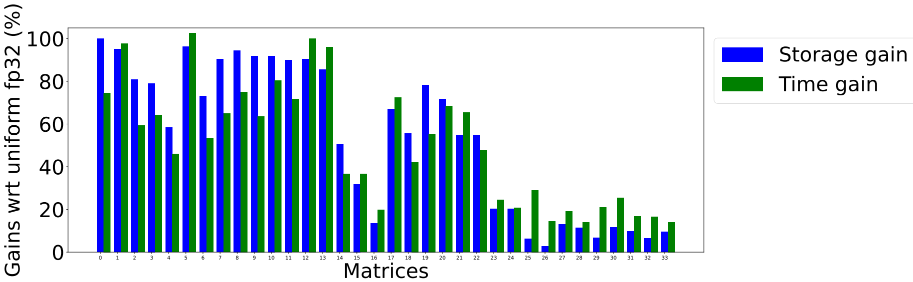
- 34 matrices from SuiteSparse of order $47k$–$11M$
- Timings on 24-core computer



Up to $36\times$ storage reduction

Adaptive precision SpMV

- 34 matrices from SuiteSparse of order $47k$–$11M$
- Timings on 24-core computer



Up to $36\times$ storage reduction $\Rightarrow$ up to $7\times$ time reduction

GMRES-based iterative refinement

GMRES

$r = b - Ax_0$
$\beta = \|r\|_2$
$q_1 = r/\beta$
for $k = 1, 2, \ldots$ **do**
  $y = Aq_k$
  for $j = 1 : k$ **do**
    $h_{jk} = q_j^T y$
    $y = y - h_{jk} q_j$
  end for
  $h_{k+1,k} = \|y\|_2$
  $q_{k+1} = y / h_{k+1,k}$
  Solve $\min_{c_k} \|Hc_k - \beta e_1\|_2$.
  $x_k = x_0 + Q_k c_k$
end for

GMRES-IR

for $i = 1, 2, \ldots$ **do**
  $r_i = b - Ax_{i-1}$
  Solve $Ad_i = r_i$ by GMRES
  $x_i = x_{i-1} + d_i$
end for

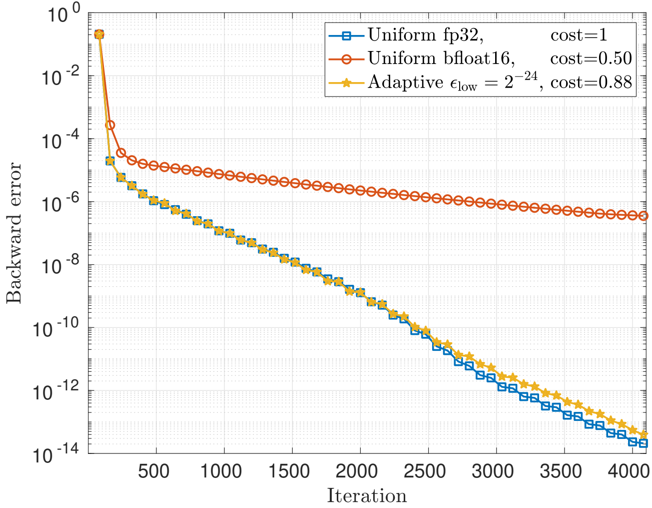
GMRES

$r = b - Ax_0$
$\beta = \|r\|_2$
$q_1 = r/\beta$
for $k = 1, 2, \ldots$ **do**
  $y = Aq_k \rightarrow \epsilon_{\text{low}}$
  for $j = 1: k$ **do**
    $h_{jk} = q_j^T y$
    $y = y - h_{jk} q_j$
  end for
  $h_{k+1,k} = \|y\|_2$
  $q_{k+1} = y / h_{k+1,k}$
  Solve $\min_{c_k} \|Hc_k - \beta e_1\|_2$.
  $x_k = x_0 + Q_k c_k$
end for

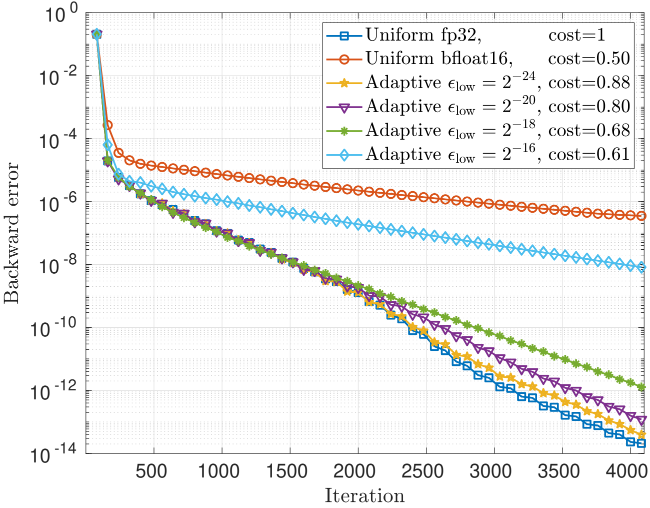
GMRES-IR

for $i = 1, 2, \ldots$ **do**
  $r_i = b - Ax_{i-1} \rightarrow \epsilon_{\text{high}}$
  Solve $Ad_i = r_i$ by GMRES
  $x_i = x_{i-1} + d_i$
end for

ML_Laplace ($\epsilon_{\text{high}} = 2^{-53}$, restart $= 80$, Jacobi preconditioner)
3 precisions (fp64, fp32, bfloat16) + dropping



ML_Laplace ($\epsilon_{\text{high}} = 2^{-53}$, restart $= 80$, Jacobi preconditioner)
3 precisions (fp64, fp32, bfloat16) + dropping



Adaptive precision ILU

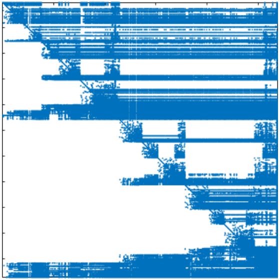
$$\begin{cases} |\ell_{ij} u_{jj}| \leq \epsilon \|A\| & \Rightarrow \quad \text{drop } \ell_{ij} \\ |u_{ij}| \leq \epsilon \|A\| & \Rightarrow \quad \text{drop } u_{ij} \end{cases}$$

$$\begin{cases} |\ell_{ij} u_{jj}| \leq \epsilon \|A\| & \Rightarrow \quad \text{drop } \ell_{ij} \\ |\ell_{ij} u_{jj}| \in [\epsilon \|A\|/u_{k+1}, \epsilon \|A\|/u_k) & \Rightarrow \quad \text{convert } \ell_{ij} \text{ to precision } u_k \\ |\ell_{ij} u_{jj}| > \epsilon \|A\|/u_2 & \Rightarrow \quad \text{keep } \ell_{ij} \text{ in precision } u_1 \end{cases}$$

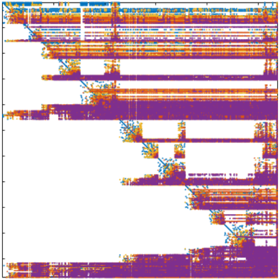
$$\begin{cases} |u_{ij}| \leq \epsilon \|A\| & \Rightarrow \quad \text{drop } u_{ij} \\ |u_{ij}| \in [\epsilon \|A\|/u_{k+1}, \epsilon \|A\|/u_k) & \Rightarrow \quad \text{convert } u_{ij} \text{ to precision } u_k \\ |u_{ij}| > \epsilon \|A\|/u_2 & \Rightarrow \quad \text{keep } u_{ij} \text{ in precision } u_1 \end{cases}$$

$$\rightarrow \|A - LU\| \leq O(\epsilon) \|A\|$$

Adaptive precision ILU

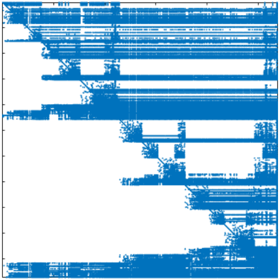


Incomplete LU
$\epsilon = 4 \times 10^{-7}$
storage$(L + U) = 81k$
$\kappa(U^{-1}L^{-1}A) = 60$

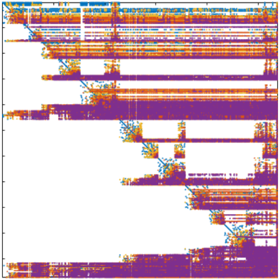


Adaptive LU
$\epsilon = 4 \times 10^{-7}$
storage$(L + U) = 43k$
$\kappa(U^{-1}L^{-1}A) = 60$

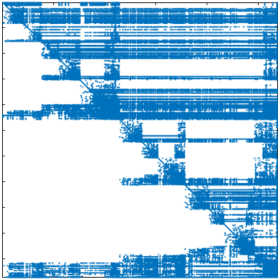
Adaptive precision ILU



Incomplete LU
$\epsilon = 4 \times 10^{-7}$
$\text{storage}(L+U) = 81k$
$\kappa(U^{-1}L^{-1}A) = 60$

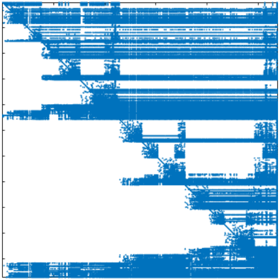


Adaptive LU
$\epsilon = 4 \times 10^{-7}$
$\text{storage}(L+U) = 43k$
$\kappa(U^{-1}L^{-1}A) = 60$

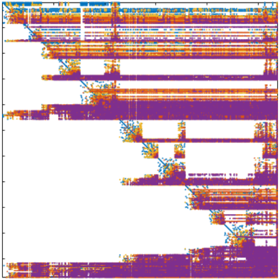


Incomplete LU
$\epsilon = 6 \times 10^{-5}$
$\text{storage}(L+U) = 43k$
$\kappa(U^{-1}L^{-1}A) = 2 \times 10^{7}$

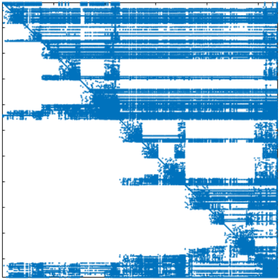
Adaptive precision ILU



Incomplete LU
$\epsilon = 4 \times 10^{-7}$
storage$(L+U) = 81k$
$\kappa(U^{-1}L^{-1}A) = 60$



Adaptive LU
$\epsilon = 4 \times 10^{-7}$
storage$(L+U) = 43k$
$\kappa(U^{-1}L^{-1}A) = 60$

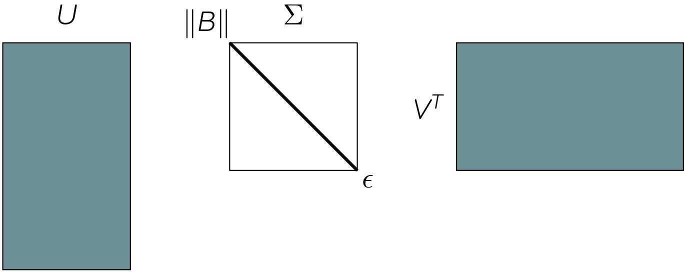


Incomplete LU
$\epsilon = 6 \times 10^{-5}$
storage$(L+U) = 43k$
$\kappa(U^{-1}L^{-1}A) = 2 \times 10^{7}$

Unlike SpMV, practical implementation seems challenging...
(future work)

Adaptive precision low rank compression

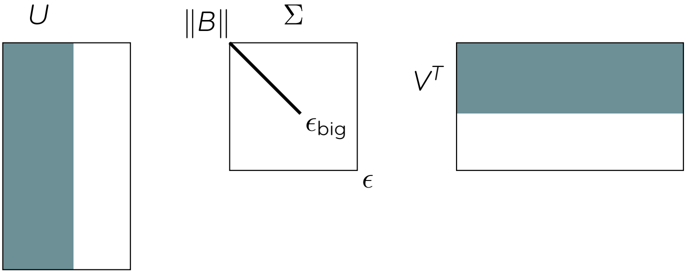
Amestoy, Boiteau, Buttari, Gerest, Jézéquel, L'Excellent, M. (2021)



How to increase low-rank compression?

Adaptive precision low rank compression

Amestoy, Boiteau, Buttari, Gerest, Jézéquel, L'Excellent, M. (2021)

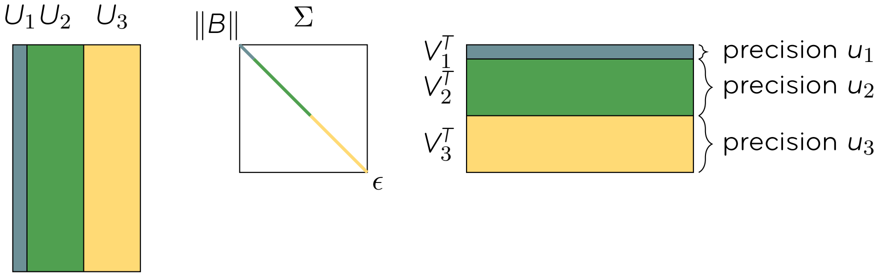


How to increase low-rank compression?

- Standard approach: increase ϵ to discard more vectors

Adaptive precision low rank compression

Amestoy, Boiteau, Buttari, Gerest, Jézéquel, L'Excellent, M. (2021)

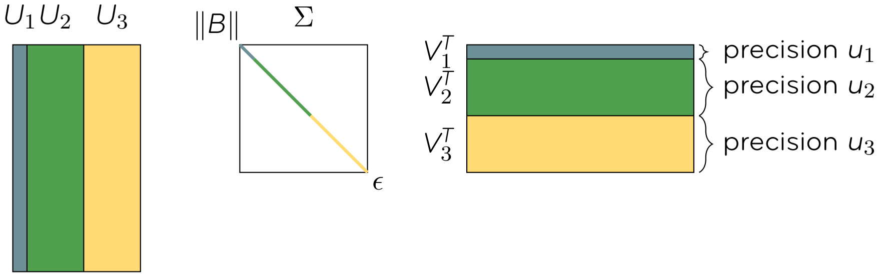


How to increase low-rank compression?

- Standard approach: increase ϵ to discard more vectors
- **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

Amestoy, Boiteau, Buttari, Gerest, Jézéquel, L'Excellent, M. (2021)

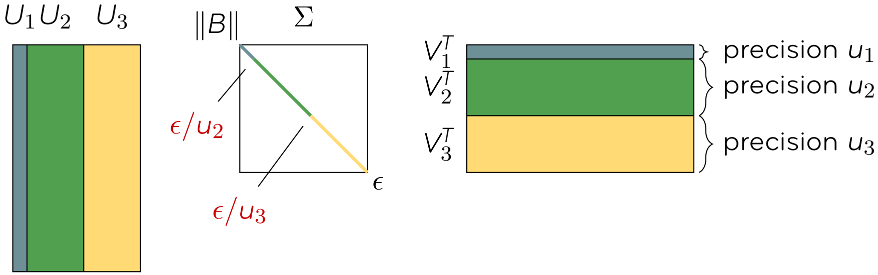


How to increase low-rank compression?

- Standard approach: increase ϵ to discard more vectors
- **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

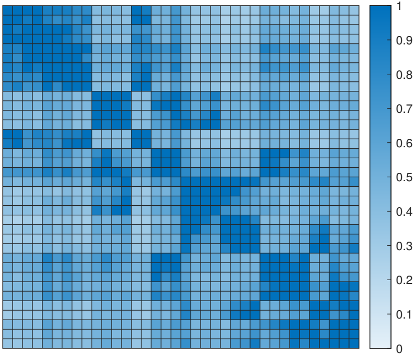
Amestoy, Boiteau, Buttari, Gerest, Jézéquel, L'Excellent, M. (2021)



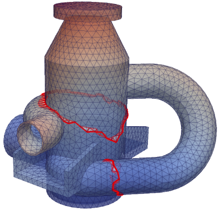
How to increase low-rank compression?

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

Adaptive precision BLR compression



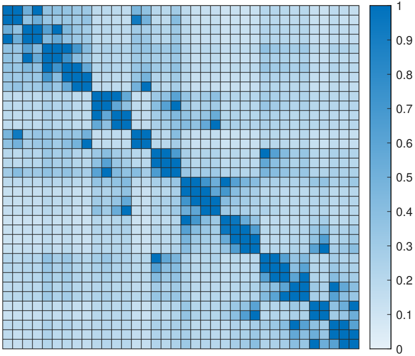
Normalized storage cost of each block



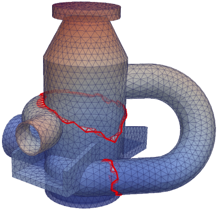
Matrix perf009d
(RIS pump from EDF)

100% entries in fp64

Adaptive precision BLR compression



Normalized storage cost of each block



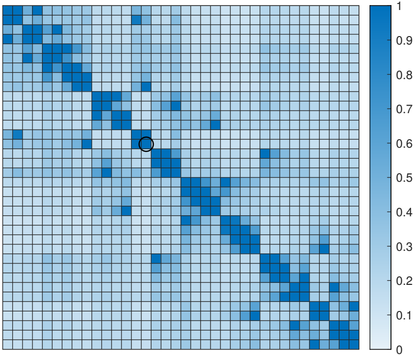
Matrix perf009d
(RIS pump from EDF)

100% entries in fp64

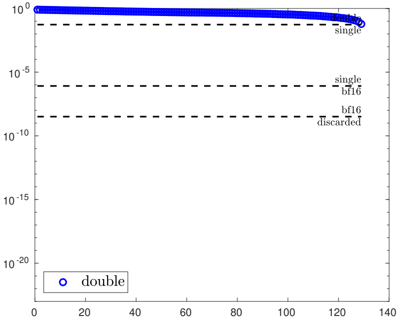
$$\rightarrow \begin{cases} 13\% \text{ in fp64} \\ 53\% \text{ in fp32} \\ 33\% \text{ in bfloat16} \end{cases}$$

$\Rightarrow 2\times$ storage reduction

Adaptive precision BLR compression



Normalized storage cost of each block



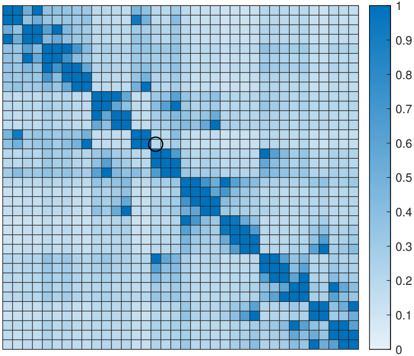
block (15,15)

100% entries in fp64

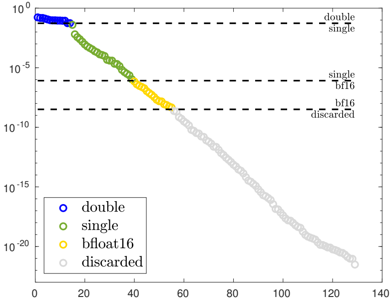
$$\rightarrow \begin{cases} 13\% \text{ in fp64} \\ 53\% \text{ in fp32} \\ 33\% \text{ in bfloat16} \end{cases}$$

$\Rightarrow 2\times$ storage reduction

Adaptive precision BLR compression



Normalized storage cost of each block



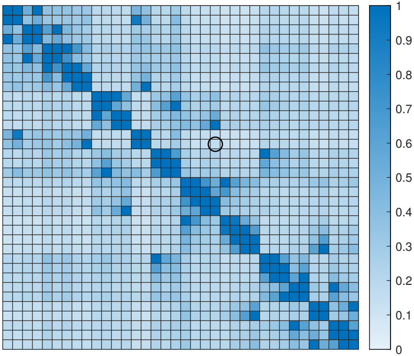
block (15,16)

100% entries in fp64

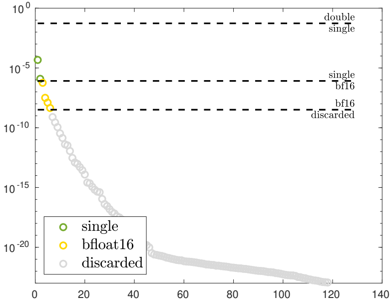
$$\rightarrow \begin{cases} 13\% \text{ in fp64} \\ 53\% \text{ in fp32} \\ 33\% \text{ in bfloat16} \end{cases}$$

$\Rightarrow 2\times$ storage reduction

Adaptive precision BLR compression



Normalized storage cost of each block



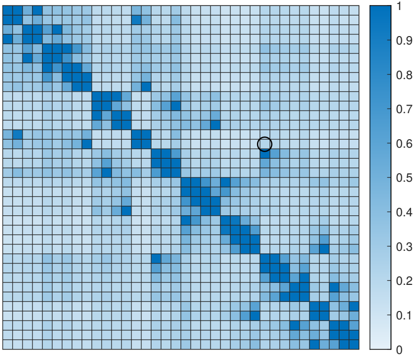
block (15,22)

100% entries in fp64

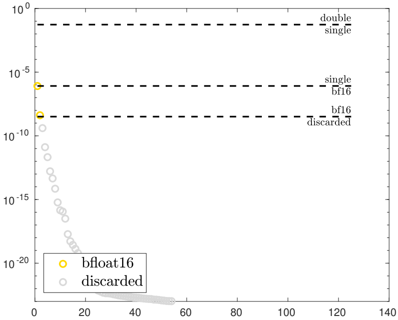
$$\rightarrow \begin{cases} 13\% \text{ in fp64} \\ 53\% \text{ in fp32} \\ 33\% \text{ in bfloat16} \end{cases}$$

$\Rightarrow 2\times$ storage reduction

Adaptive precision BLR compression



Normalized storage cost of each block



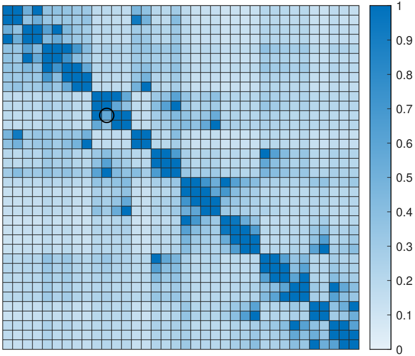
block (15,27)

100% entries in fp64

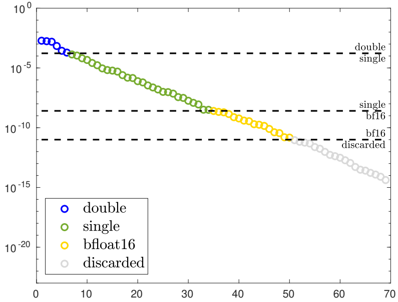
$$\rightarrow \begin{cases} 13\% \text{ in fp64} \\ 53\% \text{ in fp32} \\ 33\% \text{ in bfloat16} \end{cases}$$

$\Rightarrow 2\times$ storage reduction

Adaptive precision BLR compression



Normalized storage cost of each block



block (12,11)

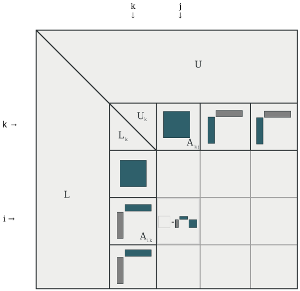
100% entries in fp64

$\rightarrow \begin{cases} 13\% \text{ in fp64} \\ 53\% \text{ in fp32} \\ 33\% \text{ in bfloat16} \end{cases}$

$\Rightarrow 2\times$ storage reduction

Adaptive precision BLR LU factorization

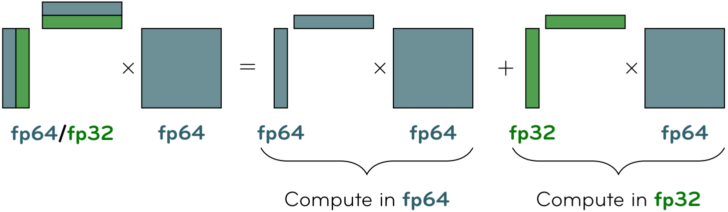
- Step k:
  - Compute $L_{kk}U_{kk} = A_{kk}$
  - Update
    $A_{ij} \leftarrow A_{ij} - (A_{ik}U_{kk}^{-1}) \times (L_{kk}^{-1}A_{kj})$



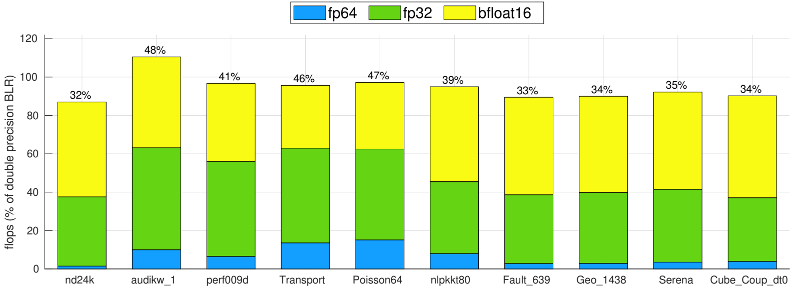
- Stability of LU factorization:
  $\widehat{L}\widehat{U} = A + \Delta A$
  - **Standard LU** (Wilkinson) :
    $\|\Delta A\| \lesssim 3n^3\rho_n u_1\|A\|$
  - **BLR LU** (Higham & M.) :
    $\|\Delta A\| \lesssim (c_1\epsilon + c_2\rho_n u_1)\|A\|$
  - **Adaptive prec. BLR LU** (this work) :
    $\|\Delta A\| \lesssim (c_1'\epsilon + c_2'\rho_n u_1)\|A\|$

Error analysis determines which precision is needed for each flop

Example of kernel: LR $\times$ matrix multiplication:



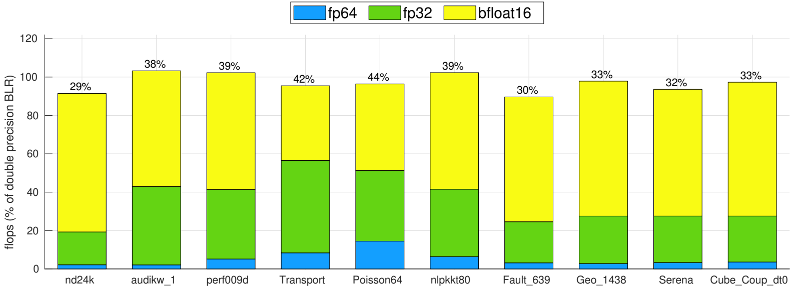
Adaptive precision BLR LU factorization



$$\varepsilon = 10^{-12}$$

Top of the bars: cost w.r.t. fp64 BLR, assuming
1 flop(fp64) = 2 flops(fp32) = 4 flops(bfloat16)

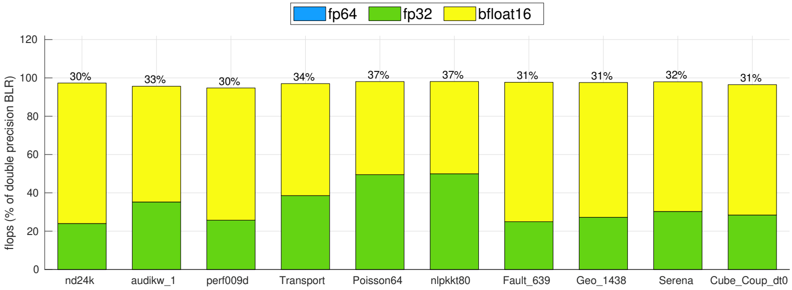
Adaptive precision BLR LU factorization



$\varepsilon = 10^{-9}$

Top of the bars: cost w.r.t. fp64 BLR, assuming
1 flop(fp64) = 2 flops(fp32) = 4 flops(bfloat16)

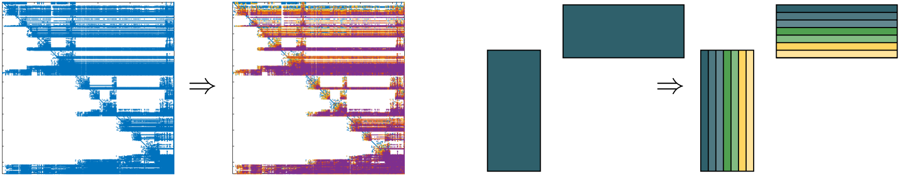
Adaptive precision BLR LU factorization



$$\varepsilon = 10^{-6}$$

Top of the bars: cost w.r.t. fp64 BLR, assuming
1 flop(fp64) = 2 flops(fp32) = 4 flops(bfloat16)

Take-home picture



We now live in a multiprecision world,
we need to rethink our algorithms accordingly

Slides at https://bit.ly/happyBirthdayNick
Check out our papers:
 Adaptive SpMV: https://bit.ly/adapt2022-SpMV
 Adaptive BLR: https://bit.ly/adapt2022-BLR

Thank you!