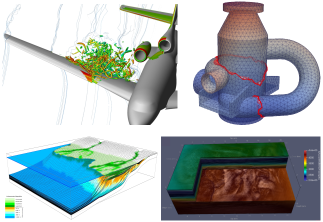
Approximate computing in numerical linear algebra: algorithms, analysis, and applications

Theo Mary
Sorbonne Université, CNRS, LIP6

Habilitation defense
7 October 2025

Exascale applications:

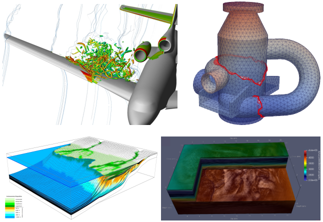
- Computationally demanding (speed, storage, and energy constraints)
- Numerically demanding (high accuracy target)



Exascale computers:

- Huge amounts of parallelism/concurrency
- Heterogeneity of the computing units: CPUs, GPUs, other accelerators
- Large gap between speed of computations and communications

Challenges of computing at exascale



Exascale applications:

- Computationally demanding (speed, storage, and energy constraints)
- Numerically demanding (high accuracy target)

Exascale methods

and software??

Exascale computers:

- Huge amounts of parallelism/concurrency
- Heterogeneity of the computing units: CPUs, GPUs, other accelerators
- Large gap between speed of computations and communications

Approximate computing

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

Given a target accuracy ε, how do we decide where (and how much) inexactness can be introduced?

Floating-point arithmetic

	Signif. bits	Exp. bits	Range ($f_{\max}/f_{\min}$)	Unit roundoff u
fp128	113	15	$2^{32766} \approx 10^{9863}$	$2^{-114} \approx 1 \times 10^{-34}$
fp64	52	11	$2^{2046} \approx 10^{616}$	$2^{-53} \approx 1 \times 10^{-16}$
fp32	23	8	$2^{254} \approx 10^{76}$	$2^{-24} \approx 6 \times 10^{-8}$
tfloat32	10	8	$2^{254} \approx 10^{76}$	$2^{-11} \approx 5 \times 10^{-4}$
fp16	10	5	$2^{30} \approx 10^{9}$	$2^{-11} \approx 5 \times 10^{-4}$
bfloat16	7	8	$2^{254} \approx 10^{76}$	$2^{-8} \approx 4 \times 10^{-3}$
fp8 (E4M3)	3	4	$2^{15} \approx 3 \times 10^{4}$	$2^{-4} \approx 6 \times 10^{-2}$
fp8 (E5M2)	2	5	$2^{30} \approx 10^{9}$	$2^{-3} \approx 1 \times 10^{-1}$
fp6 (E2M3)	3	2	$2^{3} \approx 8$	$2^{-4} \approx 6 \times 10^{-2}$
fp6 (E3M2)	2	3	$2^{7} \approx 128$	$2^{-3} \approx 0.125$
fp4 (E2M1)	1	2	$2^{3} \approx 8$	$2^{-2} \approx 0.25$

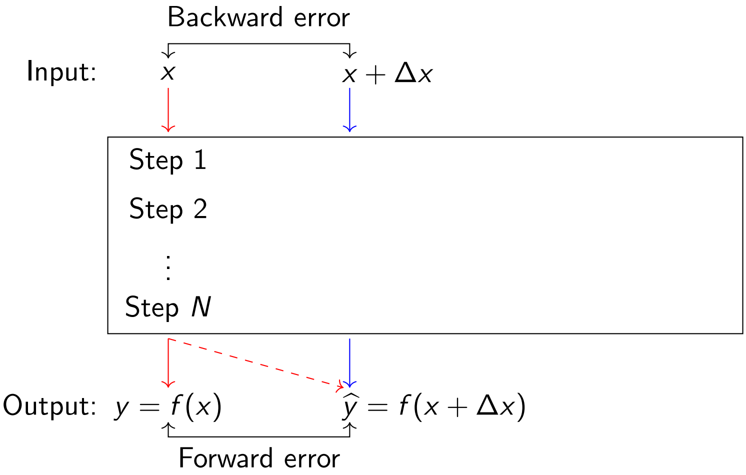
Lower precisions:

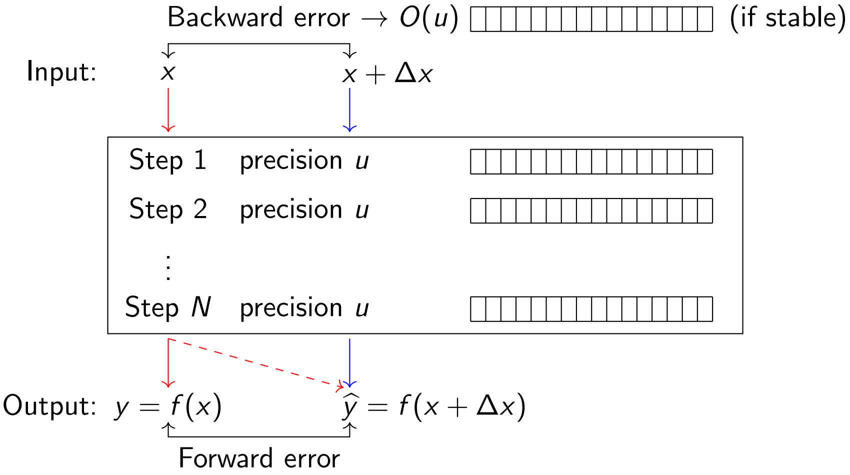
- ☺ Faster, consume less memory and energy
- ☹ Lower accuracy and narrower range

Standard model:

For any x such that $|x| \in [f_{\min}, f_{\max}]$,
$$\mathrm{fl}(x) = x(1+\delta), \quad |\delta| \le u$$

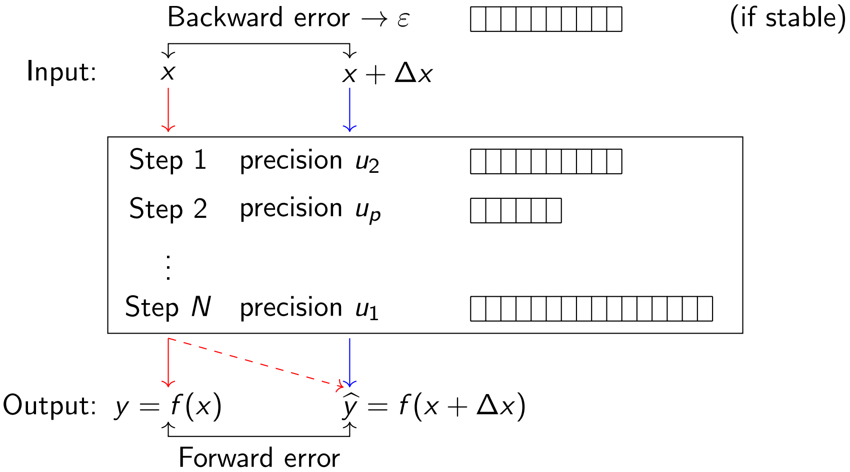
Backward and forward error, uniform and mixed precision





- **Uniform precision:** use one precision such that $u \le \varepsilon$

Backward and forward error, uniform and mixed precision



- **Uniform precision:** use one precision such that $u \leq \varepsilon$
- **Mixed precision:** use multiple precisions $u_1, \ldots, u_p$ such that their combined strategic use leads to an accuracy of ε

Part I

Matrix multiplication

Inner products

Consider the computation

$$s = x^T y = \sum_{k=1}^{n} x_k y_k$$

The backward error is bounded by γ_n:

$$\widehat{s} = \sum_{k=1}^{n} x_k y_k (1+\theta_k), \qquad 1+\theta_k = \prod_{i=1}^{n_k \le n} (1+\delta_i), \qquad |\theta_k| \le \gamma_n = \frac{nu}{1-nu} = nu + O(u^2)$$

The forward error is bounded by:

$$\frac{|\widehat{s} - s|}{|s|} \le \gamma_n \kappa \approx n\kappa u, \qquad \kappa = \frac{|x|^T |y|}{|x^T y|} = \frac{\sum_{k=1}^{n} |x_k y_k|}{|\sum_{k=1}^{n} x_k y_k|}$$

This error can be large when

- The dimension n is large **(accumulation)**
- The condition number κ is large **(cancellation)**
- The unit roundoff u is large **(low precision)**

Part I

Matrix multiplication

Dealing with rounding error accumulation

$n\kappa u$

Probabilistic rounding error analysis

Worst-case nu bound only attained when all δ_i are in the same direction ($+u$ or $-u$)

Wilkinson's conjecture (1961)

$$n \to \sqrt{n}$$

Probabilistic rounding error analysis

Worst-case nu bound only attained when all δ_i are in the same direction ($+u$ or $-u$)

Wilkinson's conjecture (1961)

$$n \to \sqrt{n}$$

Probabilistic model of rounding errors (Model M)

In the computation of interest, the rounding errors δ_i are mean independent random variables of mean zero: $\mathbb{E}(\delta_i \mid \delta_1, \ldots, \delta_{i-1}) = \mathbb{E}(\delta_i) = 0$.

Probabilistic backward error bound (Higham and M., SISC 2019, SISC 2020)

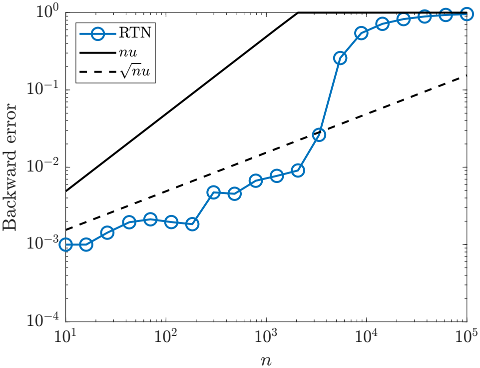
Let δ_i, $i = 1 : n$, satisfy Model M. Then, for any $\lambda > 0$, the relation

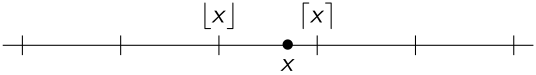
$$\prod_{i=1}^{n}(1+\delta_i) = 1 + \theta_n, \quad |\theta_n| \le \widetilde{\gamma}_n(\lambda) := \exp\left(\lambda\sqrt{n}u + \frac{nu^2}{1-u}\right) - 1$$

$$\le \lambda\sqrt{n}u + O(u^2)$$

holds with probability at least $P(\lambda) = 1 - 2\exp(-\lambda^2/2)$.

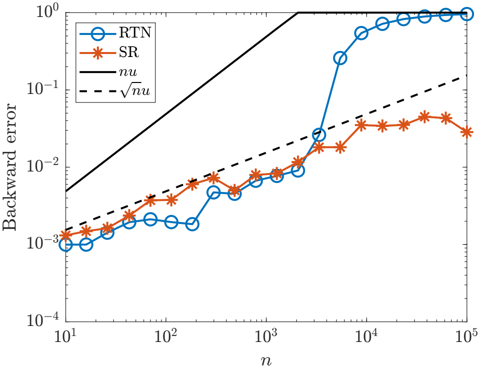
Stochastic rounding

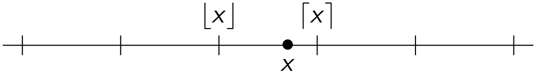




$$\mathrm{SR}(x) = \begin{cases} \lceil x \rceil & \text{with probability } p = \frac{x - \lfloor x \rfloor}{\lceil x \rceil - \lfloor x \rfloor} \\ \lfloor x \rfloor & \text{with probability } 1 - p \end{cases}$$

Stochastic rounding





$$\mathrm{SR}(x) = \begin{cases} \lceil x \rceil & \text{with probability } p = \frac{x - \lfloor x \rfloor}{\lceil x \rceil - \lfloor x \rfloor} \\ \lfloor x \rfloor & \text{with probability } 1 - p \end{cases}$$

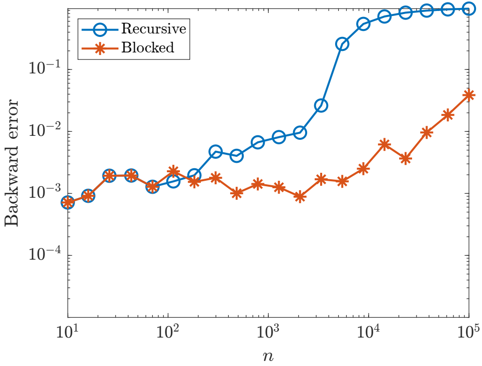
Backward error bound with SR (Connolly, Higham and M., SISC 2021)

SR enforces Model M. Therefore, the $\lambda\sqrt{nu}$ bound holds unconditionally with SR.

Blocked summation

Blocked summation

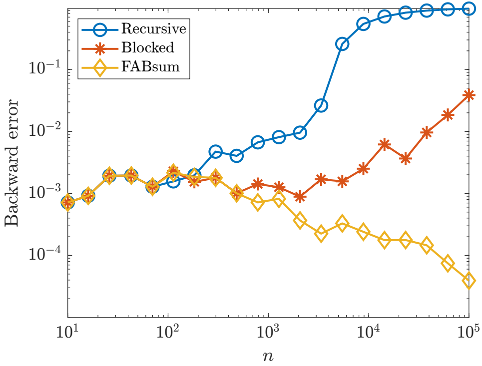
1: **for** $i = 1 : n/b$ **do**
2: $\quad s_i = \sum_{k=(i-1)b+1}^{ib} x_k y_k$
3: **end for**
4: $s = \sum_{i=1}^{n/b} s_i$



- Smaller bounds can be obtained by reducing the number of additions any given summand is involved in
- Blocked summation $nu \to (b + n/b - 1)u$

Fast and Accurate Blocked summation (FABsum)

1: **for** $i = 1 : n/b$ **do**
2: $\quad s_i = \sum_{k=(i-1)b+1}^{ib} x_k y_k$ in precision u
3: **end for**
4: $s = \sum_{i=1}^{n/b} s_i$ in precision u^2



- Smaller bounds can be obtained by reducing the number of additions any given summand is involved in
- Blocked summation $nu \to (b + n/b - 1)u$
- FABsum [Blanchard, Higham, M., SISC 2020] : $bu + O(u^2) \to$ independent of n to first order

High precision accumulation: matrix multiply–accumulate

- Some hardware (e.g., GPU tensor cores) provide **matrix multiply–accumulate (MMA)** operations $C \leftarrow C + A \times B$ where A and B are stored in precision u_{low} (e.g., fp16) but accumulated into C in precision u_{high} (e.g., fp32)
- Can be leveraged to reduce the error bound from nu_{low} to $2u_{\text{low}} + nu_{\text{high}}$ [Blanchard, Higham, Lopez, M., Pranesh, SISC 2020]

	fp16	fp16/fp32 tensor cores	
		fp32 storage	fp16 storage
Accuracy	1×10^{-3}	1×10^{-5}	
Speed (TFLOPS)	—	50	

- In LU factorization, accumulating the updates $A_{ij} \leftarrow A_{ij} - L_{ik}U_{kj}$ in fp32 significantly boosts the accuracy [Haidar et al., 2018]
  . . . but performance limited by data transfers

High precision accumulation: matrix multiply–accumulate

- Some hardware (e.g., GPU tensor cores) provide **matrix multiply–accumulate (MMA)** operations $C \leftarrow C + A \times B$ where A and B are stored in precision u_{low} (e.g., fp16) but accumulated into C in precision u_{high} (e.g., fp32)
- Can be leveraged to reduce the error bound from nu_{low} to $2u_{\text{low}} + nu_{\text{high}}$ [Blanchard, Higham, Lopez, M., Pranesh, SISC 2020]

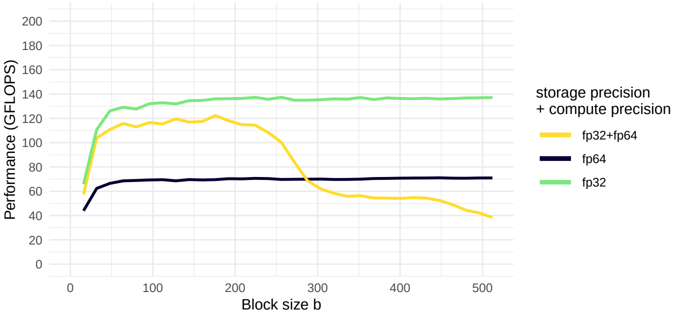
	fp16	fp16/fp32 tensor cores	
		fp32 storage	fp16 storage
Accuracy	1×10^{-3}	1×10^{-5}	3×10^{-5}
Speed (TFLOPS)	—	50	140

- In LU factorization, accumulating the updates $A_{ij} \leftarrow A_{ij} - L_{ik}U_{kj}$ in fp32 significantly boosts the accuracy [Haidar et al., 2018]
  ... but performance limited by data transfers $\Rightarrow$ store matrix in fp16 and accumulate in fp32 buffers [Lopez and M., IJHPCA 2023]

$$B_{ij} = \sum_k L_{ik}U_{kj}, \quad A_{ij} \leftarrow A_{ij} - B_{ij}$$

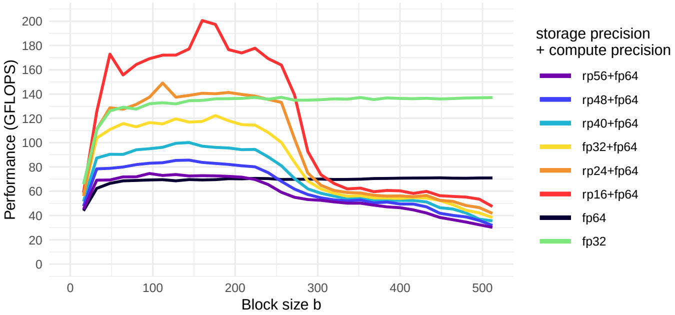
High precision accumulation: memory accessors

- Implementing high precision accumulation in software: decouple storage (low) and compute (high) precisions [Anzt et al., 2019]
- Memory accessor to efficiently load and convert data to high precision
- Blockwise accessor achieves efficient compromise between BLAS usage and data transfer reduction [Amestoy, Jego, L'Excellent, M., Pichon, preprint 2025]



High precision accumulation: memory accessors

- Implementing high precision accumulation in software: decouple storage (low) and compute (high) precisions [Anzt et al., 2019]
- Memory accessor to efficiently load and convert data to high precision
- Blockwise accessor achieves efficient compromise between BLAS usage and data transfer reduction [Amestoy, Jego, L'Excellent, M., Pichon, preprint 2025]



- Storage precision needs not be hardware supported, can use custom formats

Part I

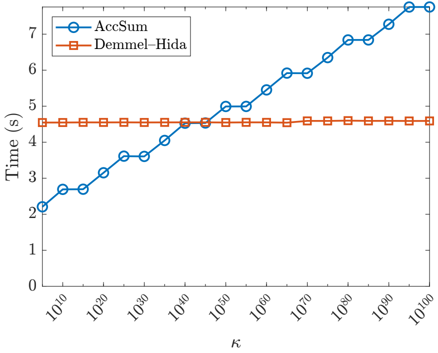
Matrix multiplication

Dealing with cancellation

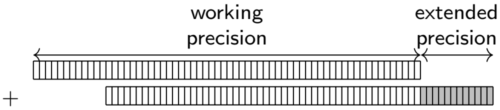
$n\kappa u$

Condense & Distill

- Distillation methods employ compensated operations to reinject errors back into the sum until it becomes well-conditioned (ex: AccSum [Rump et al., 2008])
  - ☺ Only uses standard operations
  - ☺ Entirely in the working precision
  - ☹ Cost strongly depends on κ

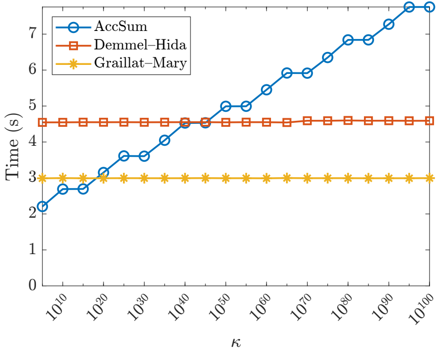


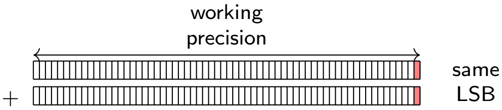
- Condensation methods add numbers of similar exponent together in extended precision (ex: [Demmel and Hida, 2004])
  - ☹ Requires access to the exponent
  - ☹ Requires extended precision
  - ☺ Cost independent of κ



- Distillation methods employ compensated operations to reinject errors back into the sum until it becomes well-conditioned (ex: AccSum [Rump et al., 2008])
  - ☺ Only uses standard operations
  - ☺ Entirely in the working precision
  - ☹ Cost strongly depends on κ

- Condensation methods add numbers of similar exponent together in extended precision (ex: [Demmel and Hida, 2004])
  - ☹ Requires access to the exponent and LSB
  - ☹ ~~Requires extended precision~~
  - ☺ Cost independent of κ





Can do without extended precision by enforcing same exponent and same LSB [Graillat and M., SISC 2025]

Part I

Matrix multiplication

Emulating high precision with low precision

$n\kappa u$

Multiword arithmetic on mixed precision MMA units

- Step 1: compute the multiword decompositions

$$A \approx \sum_{i=0}^{s-1} u_{\mathrm{low}}^{i} A_i \quad \text{and} \quad B \approx \sum_{j=0}^{s-1} u_{\mathrm{low}}^{j} B_j$$

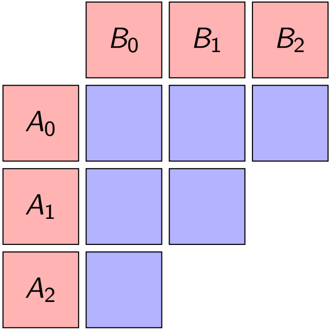
with A_i and B_j stored in precision u_{low}

- Step 2: compute the $s(s+1)/2$ leading products

$$C = \sum_{i+j<s} u_{\mathrm{low}}^{i+j} A_i B_j$$

with a mixed precision MMA with accumulation precision u_{high}

Example: **fp32 emulation** with bfloat16 tensor cores ($50\times$ speed ratio on Blackwell)
$u_{\mathrm{low}} \equiv$ fp16, $u_{\mathrm{high}} \equiv$ fp32, $s = 3$



Multiword MMA error bound (Fasi, Higham, Lopez, M., Mikaitis, SISC 2023)

The computed $\widehat{C}$ satisfies $|\widehat{C} - AB| \le \big((p+1)u_{\mathrm{low}}^{s} + c(n,s)u_{\mathrm{high}}\big)|A||B|$.

Multiword arithmetic for large integer matrix products

Goal: compute $C = AB$, $A \in \mathbb{Z}^{m\times n}$, $B \in \mathbb{Z}^{n\times q}$, coefficients bounded by p

- modular matrix product $C = AB \bmod p$ in computer algebra
- approximating floating-point product as scaled integer product [Ozaki et al.]

- Step 1: compute the decompositions

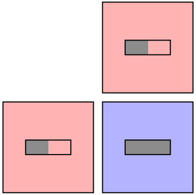
$$A \approx \sum_{i=0}^{s_A-1} \alpha^i A_i \quad \text{and} \quad B \approx \sum_{j=0}^{s_B-1} \beta^j B_j$$

  with coefficients A_i and B_j bounded by $\alpha = \lceil p^{1/s_A} \rceil$ and $\beta = \lceil p^{1/s_B} \rceil$

- Step 2: compute the $s_A s_B$ products

$$C = \sum_{i,j} \alpha^i \beta^j A_i B_j$$

  by blocks of size $b = 2^\ell$



Bound on p (Berthomieu, Graillat, Lesnoff, M., preprint 2025)

The product is computed exactly in t-bit arithmetic if $p^{1/s_A + 1/s_B} \le 2^{t-\ell}$.

Multiword arithmetic for large integer matrix products

Goal: compute $C = AB$, $A \in \mathbb{Z}^{m\times n}$, $B \in \mathbb{Z}^{n\times q}$, coefficients bounded by p

- modular matrix product $C = AB \bmod p$ in computer algebra
- approximating floating-point product as scaled integer product [Ozaki et al.]

- Step 1: compute the decompositions

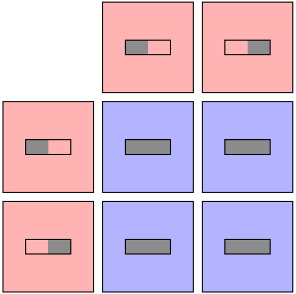
$$A \approx \sum_{i=0}^{s_A-1} \alpha^i A_i \quad \text{and} \quad B \approx \sum_{j=0}^{s_B-1} \beta^j B_j$$

  with coefficients A_i and B_j bounded by $\alpha = \lceil p^{1/s_A} \rceil$ and $\beta = \lceil p^{1/s_B} \rceil$

- Step 2: compute the $s_A s_B$ products

$$C = \sum_{i,j} \alpha^i \beta^j A_i B_j$$

  by blocks of size $b = 2^\ell$



Bound on p (Berthomieu, Graillat, Lesnoff, M., preprint 2025)

The product is computed exactly in t-bit arithmetic if $p^{1/s_A + 1/s_B} \le 2^{t-\ell}$.

Multiword arithmetic for large integer matrix products

Goal: compute $C = AB$, $A \in \mathbb{Z}^{m \times n}$, $B \in \mathbb{Z}^{n \times q}$, coefficients bounded by p

- modular matrix product $C = AB \bmod p$ in computer algebra
- approximating floating-point product as scaled integer product [Ozaki et al.]

- Step 1: compute the decompositions

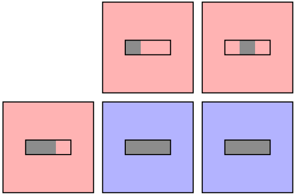
$$A \approx \sum_{i=0}^{s_A - 1} \alpha^i A_i \quad \text{and} \quad B \approx \sum_{j=0}^{s_B - 1} \beta^j B_j$$

  with coefficients A_i and B_j bounded by $\alpha = \lceil p^{1/s_A} \rceil$ and $\beta = \lceil p^{1/s_B} \rceil$

- Step 2: compute the $s_A s_B$ products

$$C = \sum_{i,j} \alpha^i \beta^j A_i B_j$$

  by blocks of size $b = 2^\ell$



Bound on p (Berthomieu, Graillat, Lesnoff, M., preprint 2025)

The product is computed exactly in t-bit arithmetic if $p^{1/s_A + 1/s_B} \leq 2^{t-\ell}$.

Multiword arithmetic for large integer matrix products

Goal: compute $C = AB$, $A \in \mathbb{Z}^{m\times n}$, $B \in \mathbb{Z}^{n\times q}$, coefficients bounded by p

- modular matrix product $C = AB \bmod p$ in computer algebra
- approximating floating-point product as scaled integer product [Ozaki et al.]

- Step 1: compute the decompositions

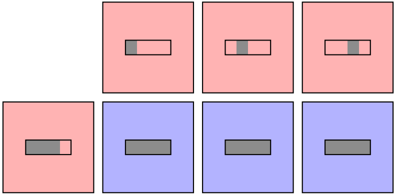
$$A \approx \sum_{i=0}^{s_A-1} \alpha^i A_i \quad \text{and} \quad B \approx \sum_{j=0}^{s_B-1} \beta^j B_j$$

with coefficients A_i and B_j bounded by $\alpha = \lceil p^{1/s_A} \rceil$ and $\beta = \lceil p^{1/s_B} \rceil$

- Step 2: compute the $s_A s_B$ products

$$C = \sum_{i,j} \alpha^i \beta^j A_i B_j$$

by blocks of size $b = 2^\ell$



Bound on p (Berthomieu, Graillat, Lesnoff, M., preprint 2025)

The product is computed exactly in t-bit arithmetic if $p^{1/s_A + 1/s_B} \leq 2^{t-\ell}$.

Part II

Linear systems

Linear system solution methods

Goal: solve $Ax = b$, where A is large and often 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

Linear system solution methods

Goal: solve $Ax = b$, where A is large and often 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

$\Rightarrow$ **Approximate factorizations...**
  - as fast direct methods
  - as high quality preconditioners

Part II

Linear systems

Iterative refinement

Iterative refinement

Iterative refinement

```
Compute an initial approximation x

repeat
    r = b − Ax in precision u_r
    Solve Ac = r in "precision" ε
    x = x + c in precision u
until convergence
```

Convergence of IR

- Attainable accuracy $u + u_r\kappa(A)$ (independent of ε!)
- Convergence rate $\propto \kappa(A)\varepsilon$

- Can use direct, iterative, or any kind of approximate solvers

LU-based iterative refinement

LU-IR [Langou et al., 2006]

Compute $A = LU$ in precision u_f
Solve $LUx = b$ in precision u_f
repeat
    $r = b - Ax$ in precision u_r
    Solve $LUc = r$ in precision u_f
    $x = x + c$ in precision u
until convergence

Convergence of LU-IR

- Attainable accuracy $u + u_r\kappa(A)$ (independent of u_f!)
- Convergence rate $\propto \kappa(A)u_f$

- Can use direct, iterative, or any kind of approximate solvers
- LU-IR: use low precision LU factorization and solve $LUc \approx r$

GMRES-based iterative refinement

GMRES-IR [Carson and Higham, 2017]

Compute $A = LU$ in precision u_f
Solve $LUx = b$ in precision u_f
repeat
  $r = b - Ax$ in precision u_r
  Solve $Ac = r$ via LU-preconditioned GMRES
  $x = x + c$ in precision u
until convergence

Convergence of GMRES-IR

- Attainable accuracy $u + u_r \kappa(A)$
- Convergence rate of GMRES-IR
  $\propto$ attainable accuracy of GMRES

- What precisions for GMRES?
- What preconditioning style (left, right, flexible)?

GMRES

$r_0 = b - Ax_0$
$s_0 = M^{-1} r_0$
$\beta = \|s_0\|,\ v_1 = s_0/\beta,\ k = 1$
repeat
  $z_k = Av_k$
  $w_k = M^{-1} z_k$
  for $i = 1, \dots, k$ **do**
    $h_{i,k} = v_i^T w_k$
    $w_k = w_k - h_{i,k} v_i$
  end for
  $h_{k+1,k} = \|w_k\|,\ v_{k+1} = w_k / h_{k+1,k}$
  $V_k = [v_1, \dots, v_k]$
  $H_k = \{h_{i,j}\}_{1<i<j+1; 1<j<k}$
  $y_k = \mathrm{argmin}_y \|\beta e_1 - H_k y\|$
  $k = k + 1$
until $\|\beta e_1 - H_k y_k\| \leq \varepsilon_{\mathrm{in}}$
$x_k = x_0 + V_k y_k$

Attainable accuracy of GMRES

Algorithm	Backward	Forward	Reference
Householder GMRES	u_g	$\kappa(A)u_g$	Drkošová et al. (1995)
MGS-GMRES	u_g	$\kappa(A)u_g$	Paige et al. (2006)
Flexible GMRES	$\kappa(Z_k)u_g$	$\kappa(A)\kappa(Z_k)u_g$	Arioli and Duff (2009)

Attainable accuracy of GMRES

Algorithm	Backward	Forward	Reference
Householder GMRES	u_g	$\kappa(A)u_g$	Drkošová et al. (1995)
MGS-GMRES	u_g	$\kappa(A)u_g$	Paige et al. (2006)
Flexible GMRES	$\kappa(Z_k)u_g$	$\kappa(A)\kappa(Z_k)u_g$	Arioli and Duff (2009)
Left-precond. MGS-GMRES products with $M^{-1}A$ in prec. u_p	$u_g + \kappa(A)u_p$	$\kappa(M^{-1}A)(u_g + \kappa(A)u_p)$	Amestoy, Buttari, Higham L'Excellent, M., Vieublé (2024)

- Preconditioned GMRES is *not* stable

$\Rightarrow$ Motivates mixed precision GMRES with $u_p \ll u_g$

- Attainable accuracy depends on u_f only through $\kappa(M^{-1}A)$

u_g	u_p	max $\kappa(A)$
LU-IR		2×10^3
fp16	fp32	4×10^4
fp16	fp64	9×10^4
fp32	fp64	8×10^6
fp64	fp64	3×10^7
fp64	fp128	2×10^{11}

$u_f \equiv$ fp16, $u \equiv$ fp64

LU-IR vs GMRES-IR with fp32 MUMPS solver

fp32 LU factorization + refinement to fp64 accuracy
($u_f \equiv$ fp32, $u = u_r = u_g = u_p \equiv$ fp64)

Matrix	$\kappa(A)$	time gain		memory gain	
		LU-IR	GMRES-IR	LU-IR	GMRES-IR
ElectroPhys10M	10^1	**1.7**×	1.6×	**2.0**×	1.6×
Bump_2911	10^6	**1.6**×	1.4×	**2.0**×	1.7×
DrivAer6M	10^6	**1.4**×	1.2×	**2.0**×	1.5×
Queen_4147	10^6	**1.7**×	1.5×	**2.0**×	1.6×
tminlet3M	10^7	**2.2**×	1.9×	**2.0**×	1.4×
perf009ar	10^8	0.8×	**0.9**×	**1.9**×	1.5×
elasticity-3d	10^9	—	**1.3**×	—	**1.5**×
lfm_aug5M	10^{12}	**2.1**×	2.0×	**2.0**×	1.7×
Long_Coup_dt0	10^{12}	**1.4**×	**1.4**×	**2.0**×	1.6×
CarBody25M	10^{13}	—	**0.6**×	—	**1.4**×
thmgaz	10^{14}	**1.5**×	1.2×	**2.0**×	1.4×

- Up to 2× **time and memory reduction**, even for ill-conditioned problems
- GMRES-IR usually more expensive than LU-IR, but more robust

[Amestoy, Buttari, Higham, L'Excellent, M., Vieublé, TOMS 2022]

Attainable accuracy of GMRES, continued

Algorithm	Backward	Forward	Reference
Householder GMRES	u_g	$\kappa(A)u_g$	Drkošová et al. (1995)
MGS-GMRES	u_g	$\kappa(A)u_g$	Paige et al. (2006)
Flexible GMRES	$\kappa(Z_k)u_g$	$\kappa(A)\kappa(Z_k)u_g$	Arioli and Duff (2009)
Left-precond. MGS-GMRES products with $M^{-1}A$ in prec. u_p	$u_g + \kappa(A)u_p$	$\kappa(M^{-1}A)(u_g + \kappa(A)u_p)$	Amestoy, Buttari, Higham L'Excellent, M., Vieublé (2024)

Attainable accuracy of GMRES, continued

Algorithm	Backward	Forward	Reference
Householder GMRES	u_g	$\kappa(A)u_g$	Drkošová et al. (1995)
MGS-GMRES	u_g	$\kappa(A)u_g$	Paige et al. (2006)
Flexible GMRES	$\kappa(Z_k)u_g$	$\kappa(A)\kappa(Z_k)u_g$	Arioli and Duff (2009)
Left-precond. MGS-GMRES products with $M^{-1}A$ in prec. u_p	$u_g + \kappa(A)u_p$	$\kappa(M^{-1}A)(u_g + \kappa(A)u_p)$	Amestoy, Buttari, Higham L'Excellent, M., Vieublé (2024)
Modular GMRES	modular	modular	Buttari, Higham, M., Vieublé (2025)

- Modular GMRES framework to:
  - unify all existing results
  - derive new results more easily

Attainable accuracy of GMRES, continued

Algorithm	Backward	Forward	Reference
Householder GMRES	u_g	$\kappa(A)u_g$	Drkošová et al. (1995)
MGS-GMRES	u_g	$\kappa(A)u_g$	Paige et al. (2006)
Flexible GMRES	$\kappa(Z_k)u_g$	$\kappa(A)\kappa(Z_k)u_g$	Arioli and Duff (2009)
Left-precond. MGS-GMRES products with $M^{-1}A$ in prec. u_p	$u_g + \kappa(A)u_p$	$\kappa(M^{-1}A)(u_g + \kappa(A)u_p)$	Amestoy, Buttari, Higham L'Excellent, M., Vieublé (2024)
Modular GMRES	modular	modular	Buttari, Higham, M., Vieublé (2025)
$\Rightarrow$ CGS2-GMRES	u_g	$\kappa(A)u_g$	"
$\Rightarrow$ s-step GMRES	$\kappa(Z_k)u_g$	$\kappa(A)\kappa(Z_k)u_g$	Carson and Ma (2025)
$\Rightarrow$ sketched GMRES	$\kappa(Z_k)u_g$	$\kappa(A)\kappa(Z_k)u_g$	Burke, Carson, Ma (2025)
$\Rightarrow$ mixed precision GMRES		see next slide	Buttari, Liu, M., Vieublé (2025)

- Modular GMRES framework to:
  - unify all existing results
  - derive new results more easily

Mixed precision preconditioned GMRES

Attainable forward error (Buttari, Liu, M., Vieublé, preprint 2025)

- Left:
  $\kappa(M^{-1}A)\boldsymbol{u_g} + \kappa(M^{-1}A)\boldsymbol{u_m} + \kappa(A)\boldsymbol{u_a}$
- Right:
  $\kappa(AM^{-1})\kappa(M)\boldsymbol{u_g} + \kappa(M)\boldsymbol{u_m} + \kappa(A)\boldsymbol{u_a}$
- Flexible:
  $\kappa(AM^{-1})\kappa(M)\boldsymbol{u_g} + \kappa(A)\boldsymbol{u_a}$

Key observations

- For a given preconditioning style, the term in front of each precision is different
- For a given precision, the term in front of it depends on the preconditioning style

Mixed precision preconditioned GMRES

$r_0 = b - Ax_0$	$\boldsymbol{u_a}$
$s_0 = M^{-1}r_0$	$\boldsymbol{u_m}$
$\beta = \|s_0\|,\ v_1 = s_0/\beta,\ k = 1$	$\boldsymbol{u_g}$
repeat	
$\quad z_k = Av_k$	$\boldsymbol{u_a}$
$\quad w_k = M^{-1}z_k$	$\boldsymbol{u_m}$
$\quad$ **for** $i = 1, \dots, k$ **do**	
$\qquad h_{i,k} = v_i^T w_k$	$\boldsymbol{u_g}$
$\qquad w_k = w_k - h_{i,k}v_i$	$\boldsymbol{u_g}$
$\quad$ **end for**	
$\quad h_{k+1,k} = \|w_k\|,\ v_{k+1} = w_k/h_{k+1,k}$	$\boldsymbol{u_g}$
$\quad V_k = [v_1, \dots, v_k]$	
$\quad H_k = \{h_{i,j}\}_{1<i<j+1;1<j<k}$	
$\quad y_k = \mathrm{argmin}_y \|\beta e_1 - H_k y\|$	$\boldsymbol{u_g}$
$\quad k = k + 1$	
until $\|\beta e_1 - H_k y_k\| \leq \varepsilon_{\mathrm{in}}$	
$x_k = x_0 + V_k y_k$	$\boldsymbol{u_g}$

Mixed precision preconditioned GMRES

Attainable forward error (Buttari, Liu, M., Vieublé, preprint 2025)

- Left:
  $\kappa(M^{-1}A)\boldsymbol{u_g} + \kappa(M^{-1}A)\boldsymbol{u_m} + \kappa(A)\boldsymbol{u_a}$
- Right:
  $\kappa(AM^{-1})\kappa(M)\boldsymbol{u_g} + \kappa(M)\boldsymbol{u_m} + \kappa(A)\boldsymbol{u_a}$
- Flexible:
  $\kappa(AM^{-1})\kappa(M)\boldsymbol{u_g} + \kappa(A)\boldsymbol{u_a}$

Key observations

- For a given preconditioning style, the term in front of each precision is different
- For a given precision, the term in front of it depends on the preconditioning style

	Left	Right	Flexible
$u_a = u_m \ll u_g$	exists	new	new
$u_a = u_g \ll u_m$	new	new	exists
$u_a \ll u_g = u_m$	new	new	new
$u_a \ll u_g \ll u_m$	new	new	new
$u_a \ll u_m \ll u_g$	new	new	new
$u_g \ll u_a = u_m$	new	new	new
$u_g \ll u_a \ll u_m$	new	new	new
$u_m \ll u_a = u_g$	new	new	new
$u_m \ll u_a \ll u_g$	new	new	new
$u_g = u_m \ll u_a$	new	new	new
$u_g \ll u_m \ll u_a$	new	new	new
$u_m \ll u_g \ll u_a$	new	new	new

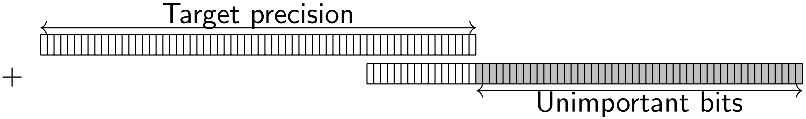
Provably meaningful
Not provably meaningful

Part II

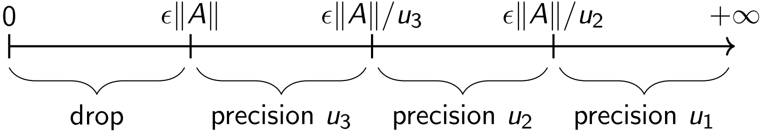
Linear systems

Adaptive precision SpMV

Adaptive precision SpMV: theory and algorithm



- Goal: compute $y = Ax$, where A is a sparse matrix, with a target accuracy ε
- Given p available precisions $u_1 < \varepsilon < u_2 < \ldots < u_p$, define partition



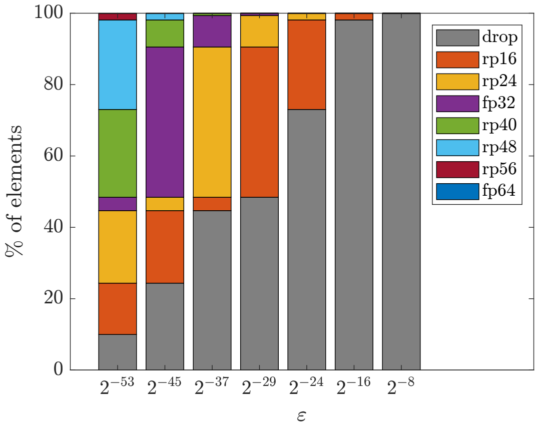
$\Rightarrow$ the precision of each element is chosen inversely proportional to its magnitude

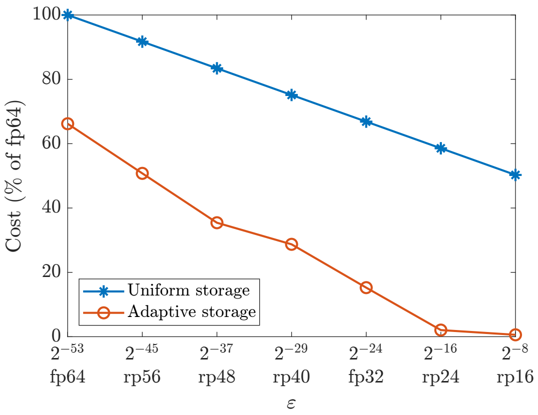
Error bound for adaptive precision SpMV (Graillat, Jézéquel, M., Molina, SISC 2022)

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

Adaptive precision SpMV: implementation and results

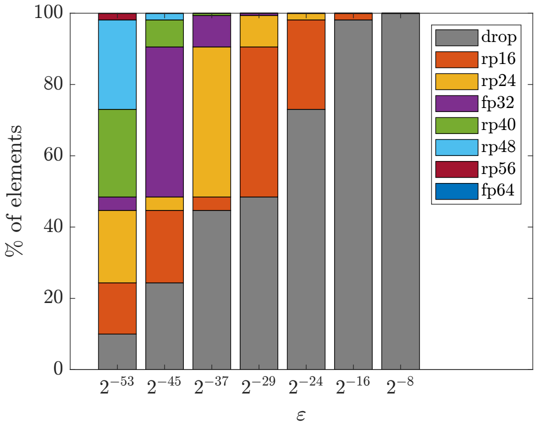
- The more precisions we have, the more we can reduce storage ⇒ exploit custom precisions with memory accessor [Graillat, Jézéquel, M., Molina, Mukunoki, Europar 2024]
- Gains are entirely matrix-dependent. Here, storage reduced by at least 30% and potentially much more for larger ε.

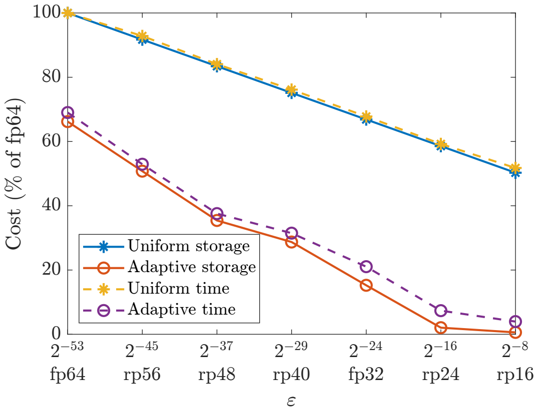




Adaptive precision SpMV: implementation and results

- The more precisions we have, the more we can reduce storage ⇒ exploit custom precisions with memory accessor [Graillat, Jézéquel, M., Molina, Mukunoki, Europar 2024]
- Gains are entirely matrix-dependent. Here, storage reduced by at least 30% and potentially much more for larger ε. Time cost matches storage!



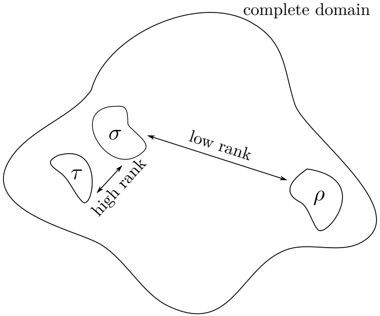


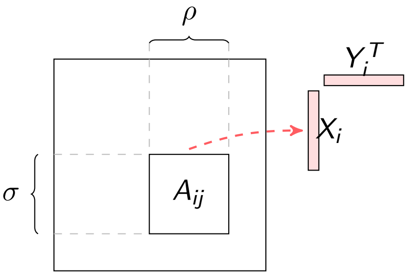
Part II

Linear systems

Block low-rank LU

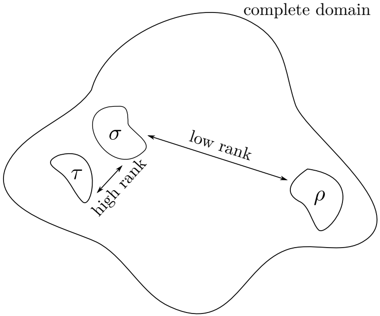
Block low-rank (BLR) matrices

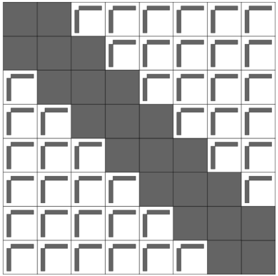




$$\|A_{ij} - X_{ij} Y_{ij}^T\| \leq \varepsilon \|A\|$$

Block low-rank (BLR) matrices





Block low-rank matrix
[Amestoy et al., 2015]

BLR LU factorization

BLR LU factorization (CFU variant)

for $k = 1\colon p$ **do**
  COMPRESS:
  for $i = k+1\colon p$ **do**
    $\widetilde{A}_{ik} = X_{ik} Y_{ik}^T$ such that $\|\widetilde{A}_{ik} - A_{ik}\| \le \varepsilon \|A\|$
    $\widetilde{A}_{ki} = X_{ki} Y_{ki}^T$ such that $\|\widetilde{A}_{ki} - A_{ki}\| \le \varepsilon \|A\|$
  end for
  FACTOR:
  $A_{kk} = L_{kk} U_{kk}$
  for $i = k+1\colon p$ **do**
    $\widetilde{L}_{ik} = \widetilde{A}_{ik} U_{kk}^{-1} = X_{ik} Y_{ik}^T \ (Y_{ik} \leftarrow U_{kk}^{-T} Y_{ik})$
    $\widetilde{U}_{ki} = L_{kk}^{-1} \widetilde{A}_{ki} = X_{ki} Y_{ki}^T \ (X_{ki} \leftarrow L_{kk}^{-1} X_{ki})$
  end for
  UPDATE:
  for $i = k+1\colon p$ **do**
    for $j = k+1\colon p$ **do**
      $A_{ij} \leftarrow A_{ij} - \widetilde{L}_{ik} \widetilde{U}_{kj} = A_{ij} - X_{ik}(Y_{ik}^T X_{ki}) Y_{ki}^T$
    end for
  end for
end for

Backward stability (Higham and M., IMAJNA 2021)

The computed BLR LU factors satisfy

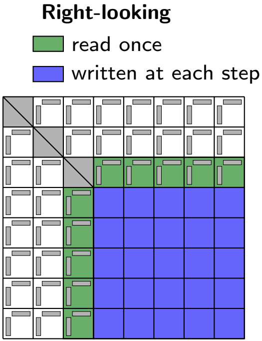
$$\widetilde{L}\widetilde{U} = A + \Delta A, \quad \|\Delta A\| \le (c_n u + \varepsilon)\|A\|.$$

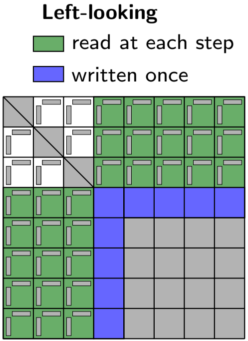
Asymptotic complexity (Amestoy, Buttari, L'Excellent, M., SISC 2017)

Work	Space
Dense matrices*	
$O(n^3) \to O(n^2)$	$O(n^2) \to O(n^{3/2})$
Sparse matrices*,†	
$O(n^2) \to O(n^{4/3})$	$O(n^{4/3}) \to O(n \log n)$

* assuming constant ranks, † regular 3D problem

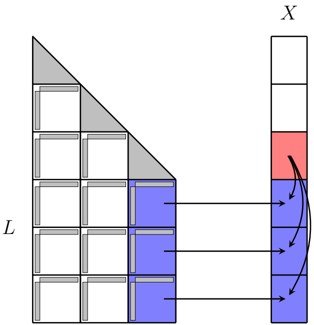
Communication-avoiding BLR factorization and solve

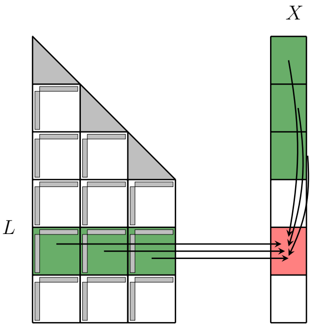




- Left-looking factorization avoids accessing uncompressed blocks
  [Amestoy, Buttari, L'Excellent, M., TOMS 2019]

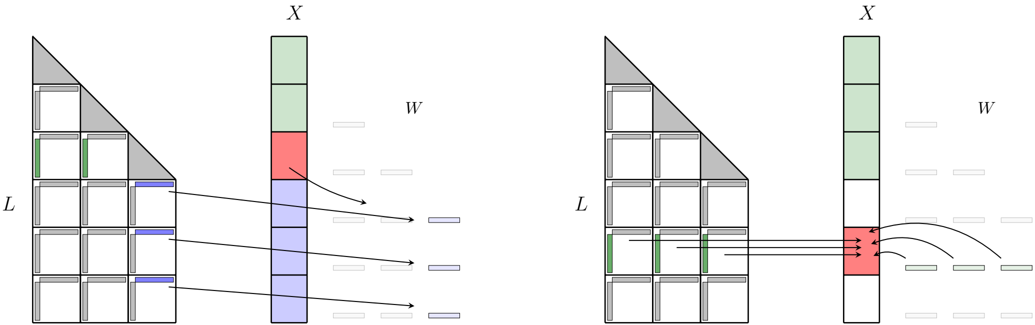
Communication-avoiding BLR factorization and solve





- Left-looking factorization avoids accessing uncompressed blocks
  [Amestoy, Buttari, L'Excellent, M., TOMS 2019]
- Solve with many RHS suffers from uncompressed RHS accesses (both in right and left looking)

Communication-avoiding BLR factorization and solve



- Left-looking factorization avoids accessing uncompressed blocks
  [Amestoy, Buttari, L'Excellent, M., TOMS 2019]
- Solve with many RHS suffers from uncompressed RHS accesses (both in right and left looking) $\Rightarrow$ hybrid algorithm only requires a single pass on RHS
  [Amestoy, Boiteau, Buttari, Gerest, Jézéquel, L'Excellent, M., SIMAX 2024]

Part II

Linear systems

Mixed precision block low-rank LU

BLR + iterative refinement

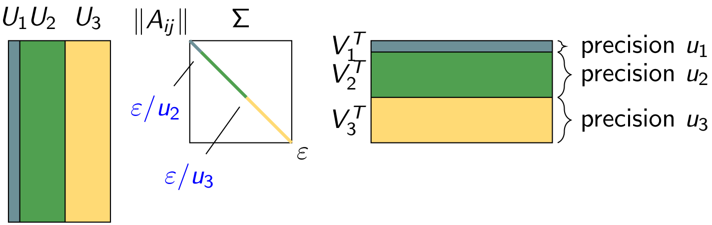
- Error analysis: replace $\mathbf{u_f}$ by $\mathbf{u_f} + \varepsilon$ in the convergence rate
  [Amestoy, Buttari, Higham, L'Excellent, M., Vieublé, TOMS 2022]
- Illustrative result on tminlet3M matrix ($n = 3 \times 10^6$, $\kappa(A) = 10^7$):

fp32+BLR(ε) LU factorization + refinement to fp64 accuracy

	time gain		memory gain	
	LU-IR	GMRES-IR	LU-IR	GMRES-IR
$\varepsilon = 10^{-8}$	2.2×	2.0×	2.1×	1.5×
$\varepsilon = 10^{-6}$	**4.2**×	3.7×	2.9×	2.6×
$\varepsilon = 10^{-4}$	—	3.3×	—	**3.4**×

- GMRES-IR allows to push BLR further!

Adaptive precision BLR



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

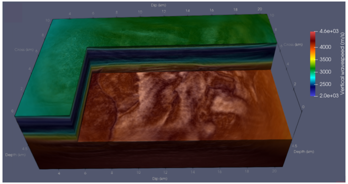
Given p precisions $u_1 < \varepsilon < u_2 < \ldots < u_p$, partition each block $A_{ij} = \sum_{k=1}^{p} U_k \Sigma_k V_k^T$ such that $\|\Sigma_k\| \le \varepsilon \|A\| / u_k$ and let $\widehat{U}_k = \mathrm{fl}_k(U_k)$ and $\widehat{V}_k = \mathrm{fl}_k(V_k)$. Then

$$\|A - \sum_{k=1}^{p} \widehat{U}_k \Sigma_k \widehat{V}_k^T\| \le (2p-1)\varepsilon\|A\|.$$

- Can also be used in BLR LU factorization while preserving stability

Performance impact illustration

- Adastra MUMPS4FWI project led by WIND team [Operto et al., The Leading Edge 2023]
- 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; Adaptive precision BLR: 3 precisions (32bits, 24bits, 16bits) for storage

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

Efficiency on 48 000 cores?

- Theoretical peak: 3686 TFLOPS (48000 $\times$ 2.4GHz $\times$ 2 (fp32) $\times$ 16 flop/cycle)
- Speed w.r.t. BLR flops: 364 TFLOPS (10% of the peak) ($0.5 \times 10^{18} \times 4$ (complex)$/5500/10^{12}$)

Part III

Conclusion

Other important topics

- Krylov solver variants
  - Augmented/deflated GMRES [Jang, Jolivet, M.]
  - Relaxed/inexact GMRES [Agullo, Giraud, Jolivet, M., Tabouret]
  - BiCGStab [Anciaux-Sedrakian, Dorfsman, Guignon, Jézéquel, M.]
- Domain decomposition
  - Mixed precision [Caruso, Jolivet, M., Nataf, Tournier]
- Hierarchical/multilevel BLR matrix formats
  - MBLR [Amestoy, Buttari, L'Excellent, M., SISC 2019]
  - BLR^2 [Ashcraft, Buttari, M., SIMAX 2021]
  - Butterfly [Gribonval, M., Riccietti]
- Low-rank approximation algorithms
  - Mixed precision randomized LRA [Buttari, M., Pacteau, SISC 2025]
  - Randomized interpolative decomposition [Buttari, Hoogveld, M.]
  - Iterative refinement/emulation for LRA [Baboulin, Kaya, M., Robeyns, SISC 2025]
- Tensors
  - Error analysis [Baboulin, Kaya, M., Robeyns]
- Neural networks
  - Error analysis [Beuzeville, Buttari, Gratton, M.]
  - Mixed precision [El Arar, Filip, M., Riccietti]

Some perspectives

Many open problems and promising research directions!

- Emulation
  - Towards increasingly specialized hardware and increasingly lower precisions
    $\Rightarrow$ need hardware-driven algorithms
- Adaptive precision
  - How do we industrialize approximate computing?
    $\Rightarrow$ need robust, almost black-box algorithms
- AI & approximate computing
  - Computing for AI and AI for computing

Final take-aways

- Need rigorous error analyses to understand and control accuracy
  $\Rightarrow$ need numerical analysts!
- Challenging to efficiently translate approximations into performance
  $\Rightarrow$ need HPC researchers!
- Performance and accuracy can only be meaningfully assessed on real-world data
  $\Rightarrow$ need application scientists!