Green computing via mixed precision

Theo Mary
Sorbonne Université, CNRS, LIP6

Energy Efficient Computing Workshop
Rutherford Appleton Laboratory
15-16 September 2025

Floating-point landscape

	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
- $\Rightarrow$ Mixed precision algorithms

Standard model of FPA:

For any x such that $|x| \in [f_{\min}, f_{\max}]$,

$$\mathrm{fl}(x) = x(1+\delta), \quad |\delta| \le u$$

Survey

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

1 Introduction 2
2 Floating-point arithmetics 6
3 Rounding error analysis model 14
4 Matrix multiplication 15
5 Nonlinear equations 18
6 Iterative refinement for $Ax = b$ 22
7 Direct methods for $Ax = b$ 25
8 Iterative methods for $Ax = b$ 35
9 Mixed precision orthogonalization and QR factorization 39
10 Least squares problems 42
11 Eigenvalue decomposition 43
12 Singular value decomposition 46
13 Multiword arithmetic 47
14 Adaptive precision algorithms 50
15 Miscellany 52

Mixed precision strategies

1. Iterative refinement
   Run baseline algorithm in low precision, refine result to high accuracy
2. Memory accessors
   Decouple the storage (low) precision and the compute (high) precision
3. Multiword arithmetic
   Emulate high precision with low precision
4. Adaptive precision
   Adapt the precision of each instruction to the problem/input at hand
5. Conclusion

1. Iterative refinement
2. Memory accessors
3. Multiword arithmetic
4. Adaptive precision
5. Conclusion

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

GMRES

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

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

Attainable accuracy of GMRES

Algorithm	Backward	Forward	Reference
MGS-GMRES	u_g	$\kappa(A)u_g$	Paige et al. (2006)

Attainable accuracy of GMRES

Algorithm	Backward	Forward	Reference
MGS-GMRES	u_g	$\kappa(A)u_g$	Paige et al. (2006)
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_f	u_g	u_p	max $\kappa(A)$
fp16	LU-IR		2×10^3
fp16	fp16	fp32	4×10^4
fp16	fp16	fp64	9×10^4
fp16	fp32	fp64	8×10^6
fp32	LU-IR		2×10^7
fp16	fp64	fp64	3×10^7
fp32	fp16	fp64	7×10^8
fp32	fp32	fp64	1×10^{10}
fp16	fp64	fp128	2×10^{11}
fp32	fp64	fp128	2×10^{15}

LU-IR vs GMRES-IR with fp32 MUMPS solver

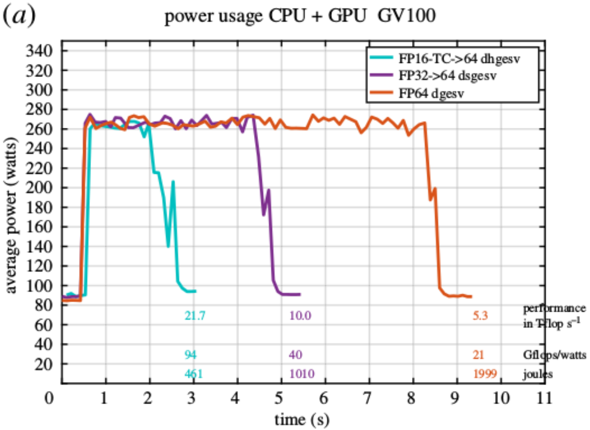
fp32 LU factorization + refinement to fp64 accuracy
($u_f \equiv$ fp32, $u = u_r = u_g = u_p \equiv$ fp64)

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]

Figure from [Haidar et al., 2020]



- Lowering precision decreases the time spent in the LU factorization, which is very energy-consuming, and increases the time spent in the refinement, which is much less energy-consuming ⇒ 4× speedup but 5× energy reduction

1 Iterative refinement

2 Memory accessors

3 Multiword arithmetic

4 Adaptive precision

5 Conclusion

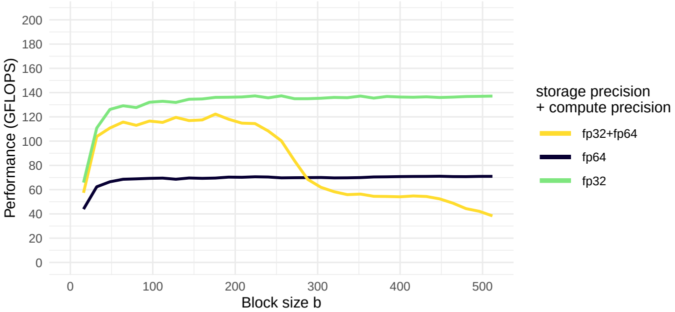
Memory accessors

- For memory-bound computations, performance and energy consumption are driven by volume of data transfers/communications
- Lowering storage precision reduces memory consumption and volume of data transfers ⇒ faster, more energy-efficient computation!
- Decouple storage and compute precisions: data is stored (compressed) in low precision and accessed (decompressed) back to high precision for computations [Anzt et al., 2019]
- Higher precision computations improves accuracy by reducing rounding error accumulation and may provide application-specific benefits (e.g., preconditioning)
- Some architectures (e.g., GPU tensor cores) provide this feature in hardware [Blanchard, Higham, Lopez, M., and Pranesh, SISC 2020] , [Lopez and M., IJHPCA 2023.]

BLAS-based block memory accessors

At what data granularity should we use memory accessors?

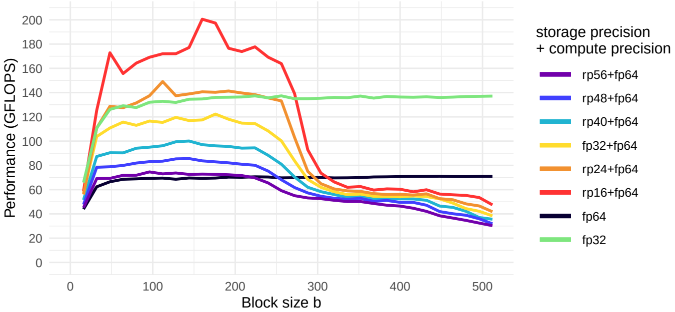
- Too small (e.g., variable-wise) $\Rightarrow$ need to rewrite all the code ☹
- Too large (e.g., matrix-wise) $\Rightarrow$ accessed data does not fit into fast memory, inefficient ☹
- Just right (e.g., block-wise) $\Rightarrow$ blocks fit into fast memory and computations can use BLAS ! ☺ [Amestoy, Jego, L'Excellent, M., and Pichon, preprint 2025]



BLAS-based block memory accessors

At what data granularity should we use memory accessors?

- Too small (e.g., variable-wise) ⇒ need to rewrite all the code ☹
- Too large (e.g., matrix-wise) ⇒ accessed data does not fit into fast memory, inefficient ☹
- Just right (e.g., block-wise) ⇒ blocks fit into fast memory and computations can use BLAS ! ☺ [Amestoy, Jego, L'Excellent, M., and Pichon, preprint 2025]



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

1 Iterative refinement

2 Memory accessors

3 Multiword arithmetic

4 Adaptive precision

5 Conclusion

Very low precisions on GPUs

Peak performance (TFLOPS)

	TFLOPS		TFLOPS/Watt	
	H200	B200	H200	B200
fp64	67	40	0.1	0.04
fp32	67	80	0.1	0.08
tfloat32	495	1,100	0.7	1.1
fp16/bfloat16	990	2,250	1.4	2.25
fp8/int8	2,000	4,500	2.9	4.5
fp4	–	9,000	–	9

fp64/fp8 FLOPS ratio:

- H200 (2022): $30\times$
- B200 (2025): $112\times$

Power consumption:

- H200: 700 W
- B200: 1000 W

- Very low precisions are very fast and energy-efficient, but hardly usable “as is”

$\Rightarrow$ use them to emulate higher precisions

Multiword arithmetic on mixed precision MMA units

- Step 1: compute the multiword decompositions

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

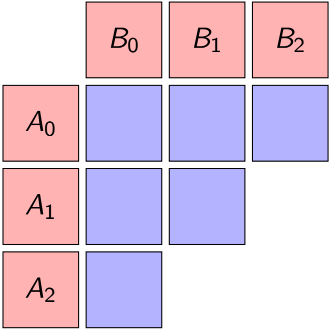
  with A_i and B_j stored in precision u_{low}

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

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

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

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



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

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

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

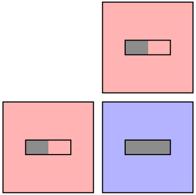
$$A \approx \sum_{i=0}^{s_A-1} \alpha^i A_i \quad \text{and} \quad B \approx \sum_{j=0}^{s_B-1} \beta^j B_j$$

  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

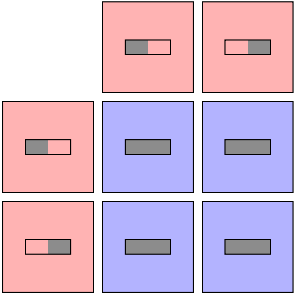
$$A \approx \sum_{i=0}^{s_A-1} \alpha^i A_i \quad \text{and} \quad B \approx \sum_{j=0}^{s_B-1} \beta^j B_j$$

  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

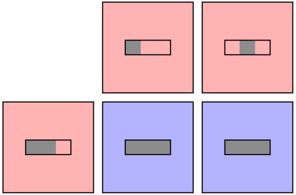
$$A \approx \sum_{i=0}^{s_A - 1} \alpha^i A_i \quad \text{and} \quad B \approx \sum_{j=0}^{s_B - 1} \beta^j B_j$$

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 $p^{1/s_A + 1/s_B} \le 2^{t-\ell}$.

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

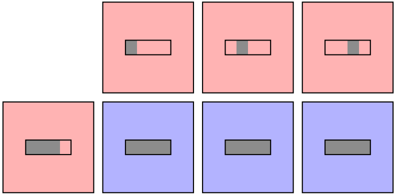
$$A \approx \sum_{i=0}^{s_A-1} \alpha^i A_i \quad \text{and} \quad B \approx \sum_{j=0}^{s_B-1} \beta^j B_j$$

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 $p^{1/s_A+1/s_B} \leq 2^{t-\ell}$.

1 Iterative refinement

2 Memory accessors

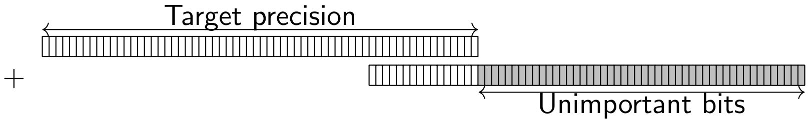
3 Multiword arithmetic

4 Adaptive precision

5 Conclusion

Adaptive precision: main principle

- Not all variables/operations need the same precision!
  Example:



$\Rightarrow$ Here, b can be stored and computed in low precision

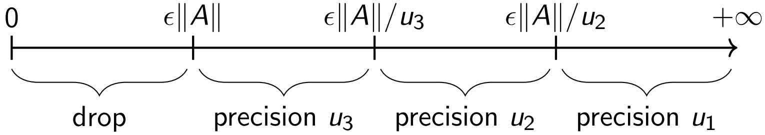
- Adaptive precision algorithms exploit this observation by dynamically selecting the minimal precision for each variable/operation, depending on the data and on the prescribed accuracy ε

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

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

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

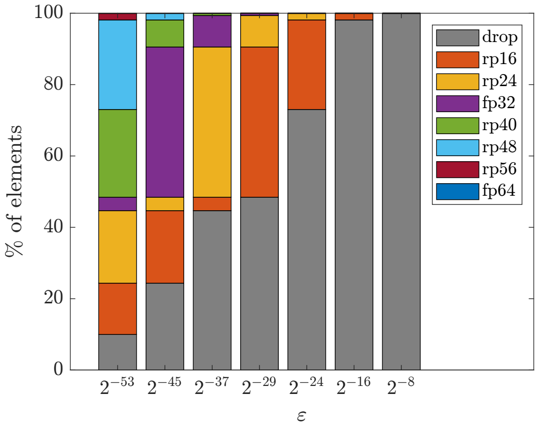


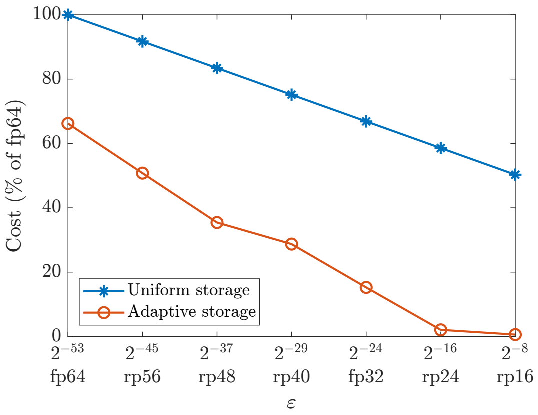
Error bound for adaptive precision SpMV (Graillat, Jézéquel, M., Molina, SISC 2022)

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

Adaptive precision SpMV: implementation and results

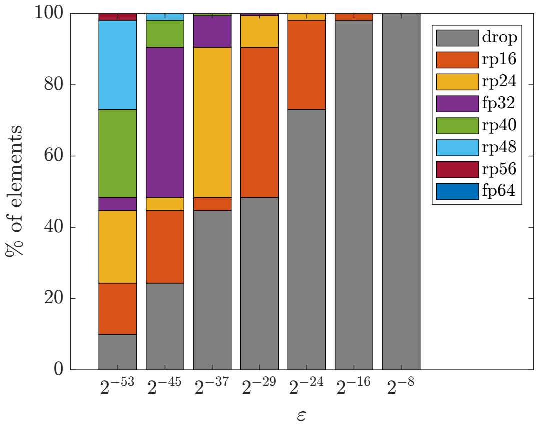
- The more precisions we have, the more we can reduce storage $\Rightarrow$ exploit custom precisions with memory accessor [Graillat, Jézéquel, M., Molina, Mukunoki, Europar 2024]
- Gains are entirely matrix-dependent. Here, storage reduced by at least 30% and potentially much more for larger ε.

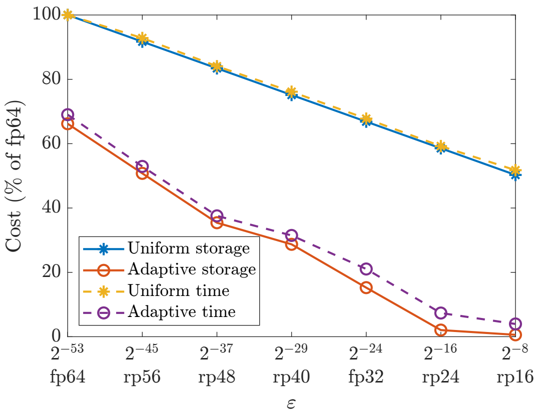




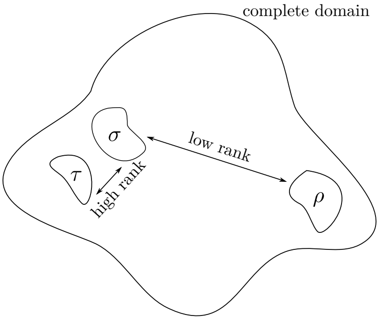
Adaptive precision SpMV: implementation and results

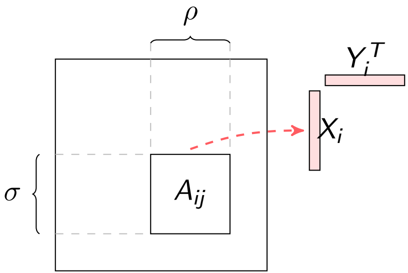
- The more precisions we have, the more we can reduce storage ⇒ exploit custom precisions with memory accessor [Graillat, Jézéquel, M., Molina, Mukunoki, Europar 2024]
- Gains are entirely matrix-dependent. Here, storage reduced by at least 30% and potentially much more for larger ε. Time cost matches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





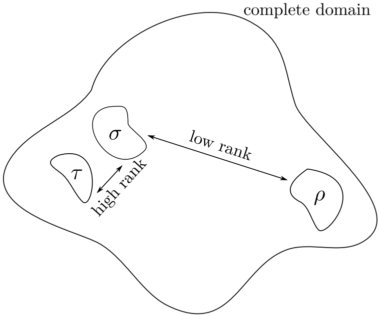
Block low-rank (BLR) matrices

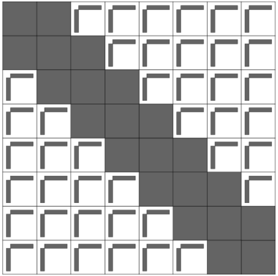




$$\|A_{ij} - X_{ij} Y_{ij}^T\| \leq \varepsilon \|A\|$$

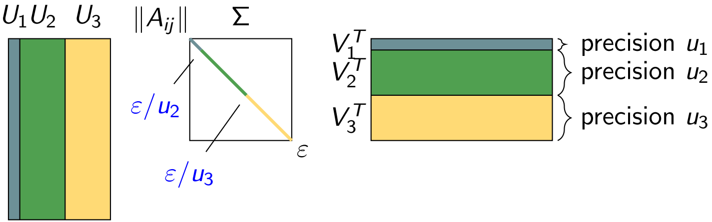
Block low-rank (BLR) matrices





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

Adaptive precision BLR



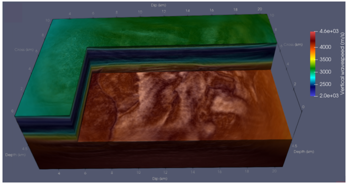
Error bound (Amestoy, Boiteau, Buttari, Gerest, Jézéquel, L'Excellent, M., IMAJNA 2022)

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

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

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}

48 000 cores needed due to memory requirements $\Rightarrow$ important not to waste them (and the associated energy consumption)! Efficiency computation:

- 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)$/5500/10^{12}$)

1 Iterative refinement

2 Memory accessors

3 Multiword arithmetic

4 Adaptive precision

5 Conclusion

Conclusion

- Iterative refinement reduces the cost of the most compute-intensive, energy-consuming step (LU factorization)
- Memory accessors reduce the cost of data transfers, the most energy-consuming operation in memory-bound computations
- Multiword arithmetic enables the use of energy-efficient specialized GPUs with very low precisions
- Adaptive precision optimizes the use of precisions for the given problem
- Mixed precision can reduce the number of ressources needed to solve a problem

Conclusion

- Iterative refinement reduces the cost of the most compute-intensive, energy-consuming step (LU factorization)
- Memory accessors reduce the cost of data transfers, the most energy-consuming operation in memory-bound computations
- Multiword arithmetic enables the use of energy-efficient specialized GPUs with very low precisions
- Adaptive precision optimizes the use of precisions for the given problem
- Mixed precision can reduce the number of ressources needed to solve a problem

Thanks! Questions?