SIAM Annual Meeting 2021

23 July 2021

Data-Aware Mixed Precision Algorithms

Theo Mary

Sorbonne Université, CNRS, LIP6

https://www-pequan.lip6.fr/~tmary/

Slides available at https://bit.ly/AN21data

Today's floating-point landscape

		Bits Signif. (t)	Exp.	Range	$u = 2^{-t}$
bfloat16	B	8	8	$10^{\pm 38}$	4×10^{-3}
fp16	H	11	5	$10^{\pm 5}$	5×10^{-4}
fp32	S	24	8	$10^{\pm 38}$	6×10^{-8}
fp64	D	53	11	$10^{\pm 308}$	1×10^{-16}
fp128	Q	113	15	$10^{\pm 4932}$	1×10^{-34}

- Low precision increasingly supported by hardware
- **Great benefits:** reduced storage, faster computations
- **Some risks too:** low precision, narrow range

Today's floating-point landscape

		Bits Signif. (t)	Exp.	Range	$u = 2^{-t}$
bfloat16	B	8	8	$10^{\pm 38}$	4×10^{-3}
fp16	H	11	5	$10^{\pm 5}$	5×10^{-4}
fp32	S	24	8	$10^{\pm 38}$	6×10^{-8}
fp64	D	53	11	$10^{\pm 308}$	1×10^{-16}
fp128	Q	113	15	$10^{\pm 4932}$	1×10^{-34}

- Low precision increasingly supported by hardware
- **Great benefits:** reduced storage, faster computations
- **Some risks too:** low precision, narrow range

$\Rightarrow$ **Mixed precision algorithms:** mix several precisions to

- Get the performance benefits of low precisions
- While preserving the accuracy and stability of the high precision

Today's floating-point landscape

		Bits Signif. (t)	Exp.	Range	$u = 2^{-t}$
bfloat16	B	8	8	$10^{\pm 38}$	4×10^{-3}
fp16	H	11	5	$10^{\pm 5}$	5×10^{-4}
fp32	S	24	8	$10^{\pm 38}$	6×10^{-8}
fp64	D	53	11	$10^{\pm 308}$	1×10^{-16}
fp128	Q	113	15	$10^{\pm 4932}$	1×10^{-34}

- Low precision increasingly supported by hardware
- **Great benefits:** reduced storage, faster computations
- **Some risks too:** low precision, narrow range

$\Rightarrow$ **Mixed precision algorithms:** mix several precisions to
  - Get the performance benefits of low precisions
  - While preserving the accuracy and stability of the high precision

- This talk: "new" class of mixed precision algorithms that exploit structure of the data

Role of the data in finite precision computations

- Traditional error analysis of numerical algorithms is mostly oblivious to the input data
- Example: let $s = x^T y = \sum_{i=1}^{n} x_i y_i$ be computed in precision u. The standard error bound is

$$|\widehat{s} - s| \le nu|x|^T|y| \quad \Rightarrow \quad \frac{|\widehat{s} - s|}{|s|} \le nu\kappa$$

- The bound only depends on the input x and y via the condition number $\kappa = \frac{|x|^T|y|}{|x^T y|}$. In particular, for nonnegative vectors, $\kappa = 1$ and so the bound is independent of the data.

Role of the data in finite precision computations

- Traditional error analysis of numerical algorithms is mostly oblivious to the input data
- Example: let $s = x^T y = \sum_{i=1}^{n} x_i y_i$ be computed in precision u. The standard error bound is

$$|\widehat{s} - s| \le nu|x|^T|y| \quad \Rightarrow \quad \frac{|\widehat{s} - s|}{|s|} \le nu\kappa$$

- The bound only depends on the input x and y via the condition number $\kappa = \frac{|x|^T|y|}{|x^T y|}$. In particular, for nonnegative vectors, $\kappa = 1$ and so the bound is independent of the data.
- Yet, the actual error does depend on the data, and strongly so! Examples with fp32, with $x_i = \mathrm{rand}(0,1)$ and $X = 10^8$ ($n = 10^6$).
  - $x_1 + x_2 + \ldots + x_{n-1} + x_n \quad \Rightarrow \quad$ error $\approx 2 \times 10^{-5}$
  - $X + x_2 + \ldots + x_{n-1} + x_n \quad \Rightarrow \quad$ error $\approx 4 \times 10^{-3}$
  - $x_1 + x_2 + \ldots + x_{n-1} + X \quad \Rightarrow \quad$ error $\approx 1 \times 10^{-7}$

Data-aware mixed precision computing

- Rounding errors are commensurate with the magnitude of the data involved in the computation

$$|\operatorname{fl}(a \text{ op } b) - a \text{ op } b| \leq u|a \text{ op } b|$$

Data-aware mixed precision computing

- Rounding errors are commensurate with the magnitude of the data involved in the computation

$$|\,\mathrm{fl}(a \text{ op } b) - a \text{ op } b| \le u|a \text{ op } b|$$

- Small elements produce small errors $\Rightarrow$ **opportunity for mixed precision !**

$$\begin{array}{lll} & 1.0\,1\,0\,1\,1\,0\,1 & \times 2^{0} \\ + & \underline{\qquad\qquad\quad 1.1\,0}(1\,0\,1\,1\,0) & \times 2^{-6} \\ = & 1.0\,1\,1\,0\,0\,0\,0 & \end{array}$$

- Different approaches exploit this fundamental observation at different levels of the computation.
  - **Matrix level:** multiword arithmetic
  - **Block level:** data sparse solvers such as BLR, block Jacobi
  - **Column/row level:** SVD, RRQR
  - **Element level:** SpMV, Krylov methods

Multiword arithmetic

- Double–double arithmetic most famous, but also recently:

$$\underbrace{x}_{\text{fp32}} \approx \underbrace{x_1}_{\text{fp16}} + \underbrace{x_2}_{\text{fp16}} \quad \text{or} \quad \underbrace{x_1}_{\text{bfloat16}} + \underbrace{x_2}_{\text{bfloat16}} + \underbrace{x_3}_{\text{bfloat16}}$$

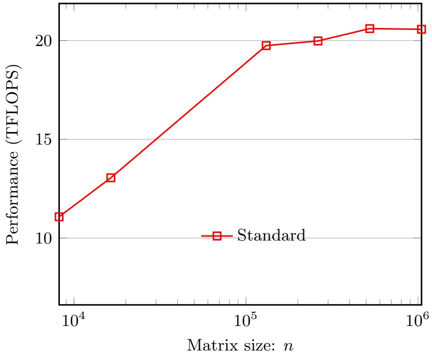
Markidis et al. (2018) Henry et al. (2019)

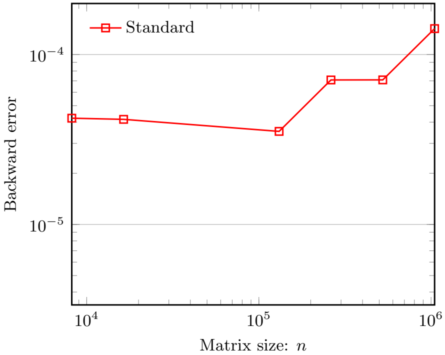
- Applying this elementwise to $A = A_1 + \ldots + A_p$ and $B = B_1 + \ldots + B_p$ to compute $C = AB$ as

$$C = \sum_{i=1}^{p} \sum_{j=1}^{p} A_i B_j$$

- The p^2 terms $A_i B_j$ are not all needed because $|A_i B_j| \le u_{16}^{i+j-2}$
- For the same reason, using a more accurate matrix mult. algorithm on $A_1 B_1$ is helpful

Double–fp16 arithmetic with tensor cores

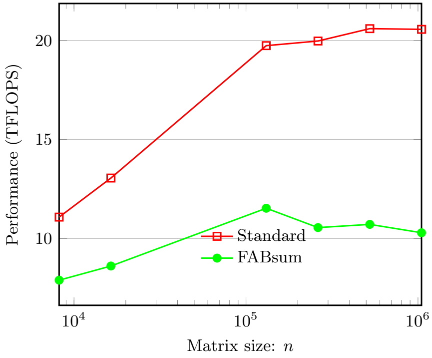


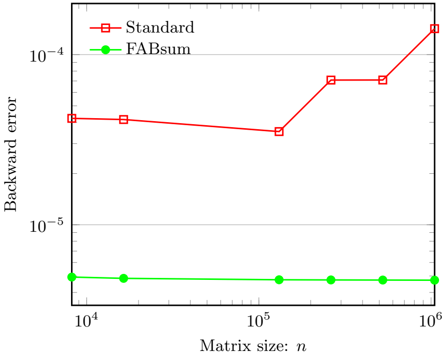


- Compute $C = AB$ as $A_1B_1 + A_2B_1 + A_1B_2$ on V100 GPUs
- Three variants:
  - Standard: use standard matrix mult. algorithm for each term

Fasi, Higham, Lopez, M., Mikaitis, Pranesh (2021)

Double–fp16 arithmetic with tensor cores

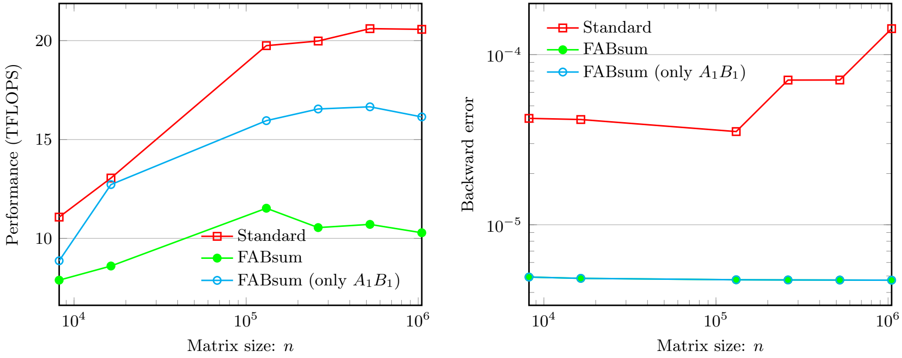




- Compute $C = AB$ as $A_1B_1 + A_2B_1 + A_1B_2$ on V100 GPUs
- Three variants:
  - Standard: use standard matrix mult. algorithm for each term
  - FABsum: use FABsum (more accurate) algorithm for each term

Fasi, Higham, Lopez, M., Mikaitis, Pranesh (2021)

Double–fp16 arithmetic with tensor cores



- Compute $C = AB$ as $A_1B_1 + A_2B_1 + A_1B_2$ on V100 GPUs
- Three variants:
  - Standard: use standard matrix mult. algorithm for each term
  - FABsum: use FABsum (more accurate) algorithm for each term
  - FABsum (only A_1B_1): use FABsum on A_1B_1 and standard alg. on A_1B_2 and A_2B_1

Fasi, Higham, Lopez, M., Mikaitis, Pranesh (2021)

BLR matrices

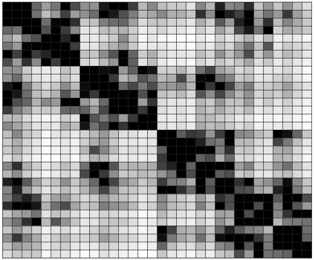
Target: solve $Ax = b$.

Can we exploit the data structure of matrix A?

BLR matrices

Target: solve $Ax = b$.

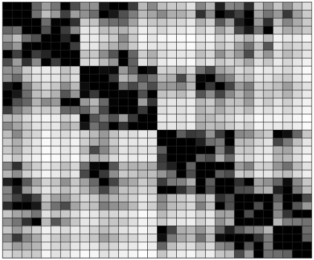
Can we exploit the data structure of matrix A?



- Block low rank (BLR) matrices use a flat 2D block partitioning
  Amestoy et al. (2015, 2017, 2019)
- Diagonal blocks are full rank
- Off-diagonal blocks A_{ij} are approximated by low-rank blocks T_{ij} satisfying $\|A_{ij} - T_{ij}\| \leq \varepsilon \|A\|$

Target: solve $Ax = b$.

Can we exploit the data structure of matrix A?



- Block low rank (BLR) matrices use a flat 2D block partitioning
  Amestoy et al. (2015, 2017, 2019)
- Diagonal blocks are full rank
- Off-diagonal blocks A_{ij} are approximated by low-rank blocks T_{ij} satisfying $\|A_{ij} - T_{ij}\| \le \varepsilon \|A\|$

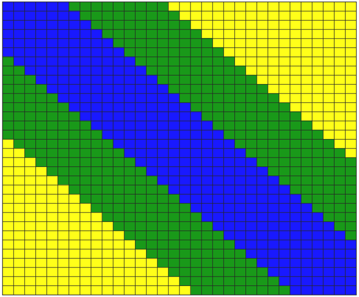
Behavior of low-rank methods in uniform finite precision now well understood Higham and M. (2021)

- Since typically $u \ll \varepsilon$, effect of rounding errors usually insignificant $\Rightarrow$ ε controls the backward error of BLR LU
- Using low precision ($u \ge \varepsilon$) **uniformly** not desirable (loss of low-rankness)

Data-driven mixed precision BLR matrices

Idea: store blocks far away from the diagonal in lower precisions

Abdulah et al. (2019) Doucet et al. (2019) Abdulah et al. (2021)

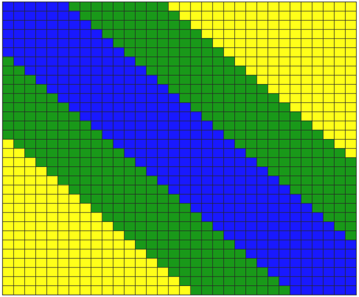


- **double**
- **single**
- **half**

Data-driven mixed precision BLR matrices

Idea: store blocks far away from the diagonal in lower precisions

Abdulah et al. (2019) Doucet et al. (2019) Abdulah et al. (2021)



- **double**
- **single**
- **half**

Data-aware analysis:

- Converting A_{ij} to precision $\mathbf{u_{low}}$ introduces an error $\mathbf{u_{low}}\|A_{ij}\|$

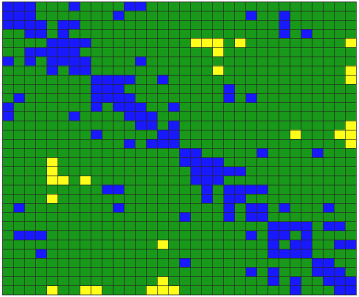
$\Rightarrow$ If $\|A_{ij}\| \le \varepsilon\|A\|/\mathbf{u_{low}}$, block can be safely stored in precision $\mathbf{u_{low}}$

Data-driven mixed precision BLR matrices

Idea: store blocks far away from the diagonal in lower precisions

Abdulah et al. (2019)   Doucet et al. (2019)   Abdulah et al. (2021)

(Poisson, $\varepsilon = 10^{-10}$)



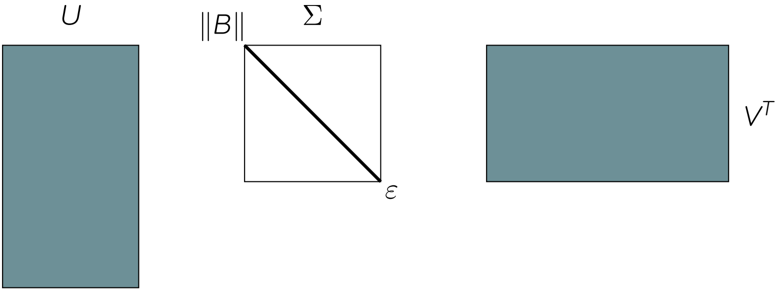
- **double**
- **single**
- **half**

Data-aware analysis:

- Converting A_{ij} to precision $\mathbf{u_{low}}$ introduces an error $\mathbf{u_{low}}\|A_{ij}\|$

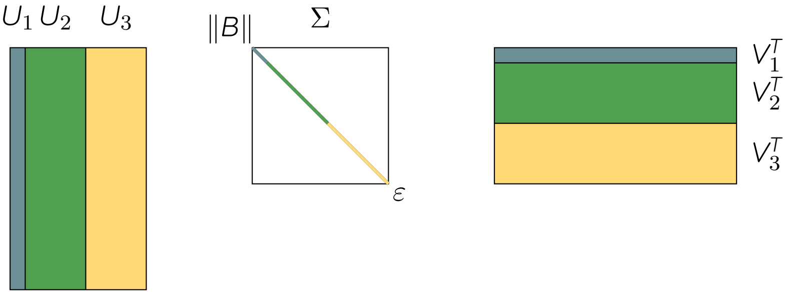
$\Rightarrow$ If $\|A_{ij}\| \le \varepsilon\|A\|/\mathbf{u_{low}}$, block can be safely stored in precision $\mathbf{u_{low}}$

Data-driven mixed precision low rank compression



- Low-rank compress based on, e.g., SVD: $\Rightarrow \|B - U\Sigma V^T\| \leq \varepsilon$, everything stored in **double precision**

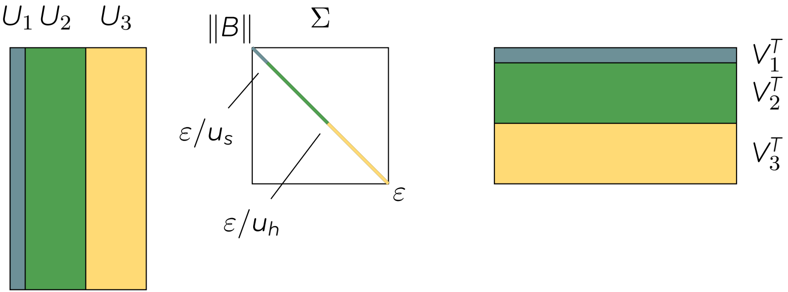
Data-driven mixed precision low rank compression



- Low-rank compress based on, e.g., SVD: $\Rightarrow \|B - U\Sigma V^T\| \leq \varepsilon$, everything stored in **double precision**
- Mixed precision compression: partition the SVD into several groups of different precision

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

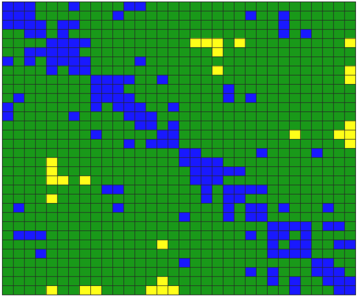
Data-driven mixed precision low rank compression



- Low-rank compress based on, e.g., SVD: $\Rightarrow \|B - U\Sigma V^T\| \leq \varepsilon$, everything stored in **double precision**
- Mixed precision compression: partition the SVD into several groups of different precision
  Amestoy, Boiteau, Buttari, Gerest, Jézéquel, L'Excellent, M. (2021)
- Why does it work ? **Data-aware analysis:**
- Converting U_i and V_i to precision u_i introduces error proportional $u_i\|\Sigma_i\| \Rightarrow$ Need to partition Σ such that $\|\Sigma_i\| \leq \varepsilon/u_i$

Back to mixed precision BLR matrices

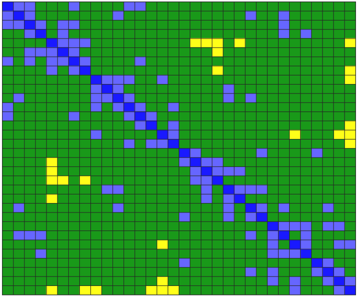
(Poisson, $\varepsilon = 10^{-10}$)



- **double**
- **single**
- **half**

Back to mixed precision BLR matrices

(Poisson, $\varepsilon = 10^{-10}$)



- **double** $\Rightarrow \begin{cases} \textbf{double} \\ \textbf{double/single/half} \end{cases}$
- **single** $\Rightarrow$ **single/half**
- **half**

Up to $3.3\times$ BLR LU flops reduction with almost no error increase

Amestoy et al. (2021)

Mixed precision at the element level ?

Given a matrix A, can we benefit from storing **each of its elements** in different precisions?

- **Is it worth it ?**
  Need to have elements of widely different magnitudes, and yet not structured in any obvious way (by blocks or columns, etc.)
- **Is it practical ?**
  Probably not for compute-bound applications, but could it work for memory-bound ones?

Mixed precision at the element level ?

Given a matrix A, can we benefit from storing **each of its elements** in different precisions?

- **Is it worth it ?**
  Need to have elements of widely different magnitudes, and yet not structured in any obvious way (by blocks or columns, etc.)
- **Is it practical ?**
  Probably not for compute-bound applications, but could it work for memory-bound ones?
- Natural candidate: SpMV

Data-aware analysis of SpMV

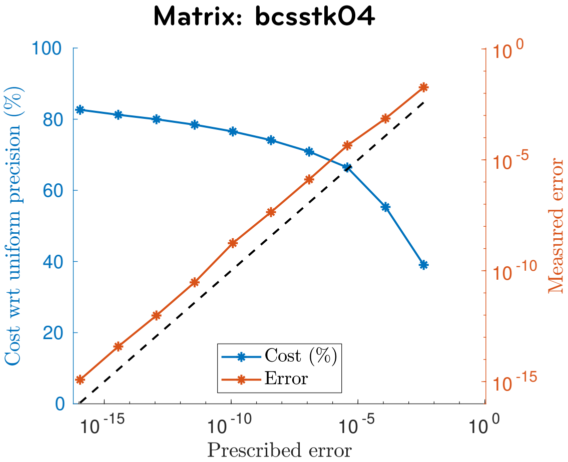
- Split row i of A into p buckets B_{ik} and sum elements of B_{ik} in precision u_k

$$y_i = \sum_{i=1}^{p} y_i^{(k)}, \quad y_i^{(k)} = \sum_{a_{ij}x_j \in B_{ik}} a_{ij}x_j$$

$$|\widehat{y}_i^{(k)} - y_i^{(k)}| \le n_i^{(k)} u_k \sum_{a_{ij}x_j \in B_{ik}} |a_{ij}x_j|$$

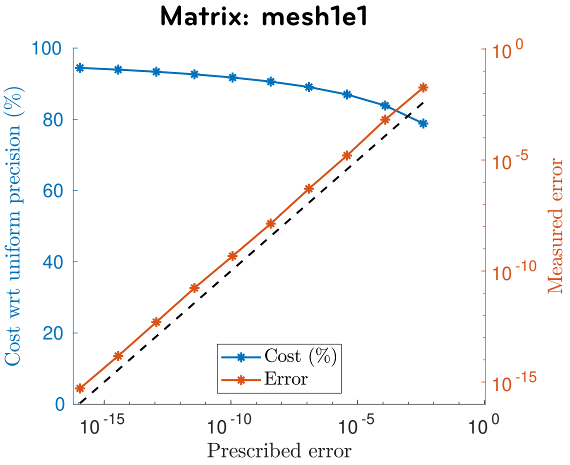
- Can guarantee a backward error of order ε by ensuring that the quantities $\sum_{a_{ij}x_j \in B_{ik}} |a_{ij}x_j|$ do not exceed ε

$\Rightarrow$ Explicit rule for building the buckets B_{ik}: put small elements in low precision bucket first, move to higher precision buckets when "full"



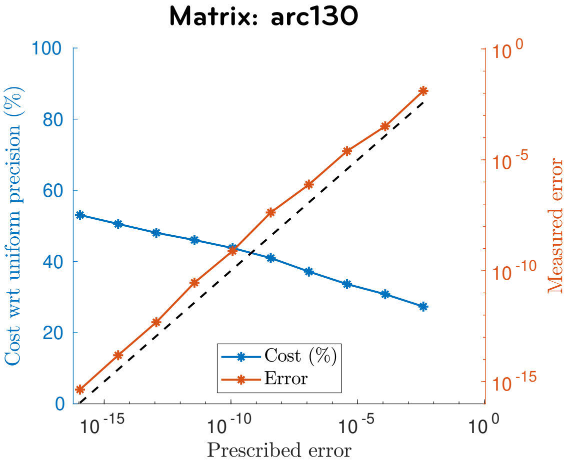
Results for $x = (1, \ldots, 1)^T$

Mixed precision SpMV: experiments



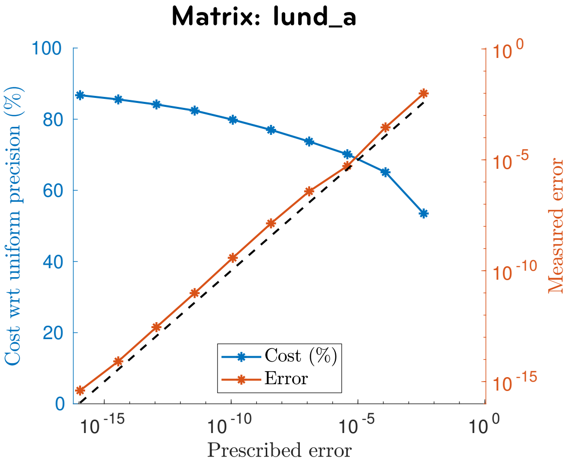
Results for $x = (1, \ldots, 1)^T$

Mixed precision SpMV: experiments



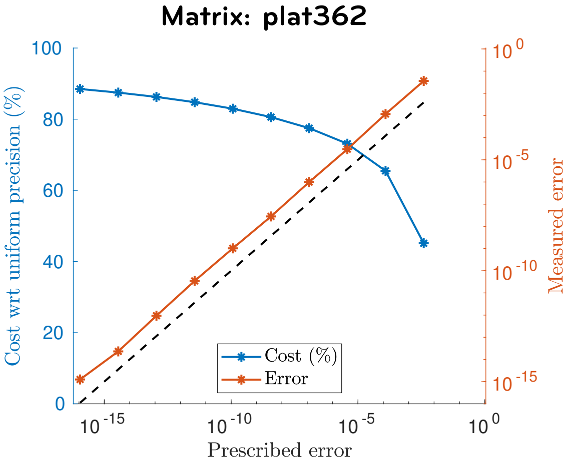
Results for $x = (1, \ldots, 1)^T$

Mixed precision SpMV: experiments



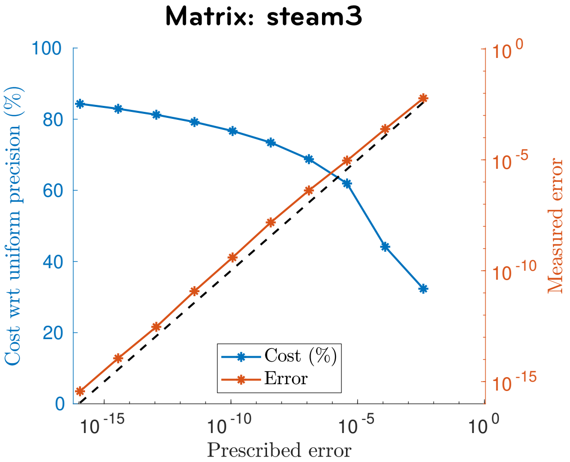
Results for $x = (1, \ldots, 1)^T$

Mixed precision SpMV: experiments



Results for $x = (1, \ldots, 1)^T$

Mixed precision SpMV: experiments



Results for $x = (1, \ldots, 1)^T$

Mixed precision SpMV: role of vector x

- Critical issue: accuracy of SpMV depends on x, but not practical to change precision of A based on x
- Can still use it and cross fingers ...
- More promising avenue: use it in a setting where x is guaranteed to be "nice"

Mixed precision SpMV: role of vector x

- Critical issue: accuracy of SpMV depends on x, but not practical to change precision of A based on x
- Can still use it and cross fingers ...
- More promising avenue: use it in a setting where x is guaranteed to be "nice"

$\Rightarrow$ **Krylov solvers**! In GMRES, x is orthonormal.

Mixed precision SpMV: role of vector x

- Critical issue: accuracy of SpMV depends on x, but not practical to change precision of A based on x
- Can still use it and cross fingers ...
- More promising avenue: use it in a setting where x is guaranteed to be "nice"

$\Rightarrow$ **Krylov solvers**! In GMRES, x is orthonormal.

Matrix	Iterations		SpMV mixed cost
	Uniform	Mixed	(% of unif.)
arc_130	5	5	16%
bcsstk04	79	79	59%
steam3	45	45	26%
lund_a	121	121	71%
mesh1e1	14	14	89%

Results with unpreconditioned unrestarted GMRES (tol $= 10^{-6}$)

Graillat, Jézéquel, M., Molina (2021)

Conclusions

- Rounding errors are commensurate with the data involved in the computation $\Rightarrow$ **rounding errors not all equally important**
- Creates opportunities for mixed precision: **adapt the precision to the magnitude of the data**!

Conclusions

- Rounding errors are commensurate with the data involved in the computation ⇒ **rounding errors not all equally important**
- Creates opportunities for mixed precision: **adapt the precision to the magnitude of the data**!

- Several (seemingly unconnected) mixed precision algorithms rely on this idea!
- **Multiword** arithmetic does it at the **matrix level**
- Mixed precision **BLR** solvers do it at the **block level**
- Mixed precision **SVD** does it at the **column level**
- Can even go to the **element level** in some applications: **SpMV** seems a promising candidate, especially within **GMRES**

Thank you! Questions?

Slides available at https://bit.ly/AN21data