



**HAL**  
open science

# Adaptive Precision Sparse Matrix-Vector Product and its application to Krylov Solvers

Roméo Molina, Stef Graillat, Fabienne Jézéquel, Théo Mary

► **To cite this version:**

Roméo Molina, Stef Graillat, Fabienne Jézéquel, Théo Mary. Adaptive Precision Sparse Matrix-Vector Product and its application to Krylov Solvers. 13èmes Rencontres Arithmétique de l'Informatique Mathématique (RAIM 2022), Nov 2022, Nantes, France. hal-03931125

**HAL Id: hal-03931125**

**<https://hal.science/hal-03931125v1>**

Submitted on 9 Jan 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

**RAIM 2022**

3rd November 2022

**Adaptive Precision Sparse Matrix–Vector  
Product  
and its Application to Krylov Solvers**

**Roméo Molina**

LIP6, Sorbonne Université

Service Online, Département Informatique, IJCLab

Joint work with

**Stef Graillat, Fabienne Jézéquel, and Theo Mary**

# Today's floating-point landscape

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

- Low precision increasingly supported by hardware
- **Great benefits:**
  - Reduced **storage**, data movement, and communications
  - Reduced **energy** consumption (5× with fp16, 9× with bfloat16)
  - Increased **speed** on emerging hardware (16× on A100 from fp32 to fp16/bfloat16)
- **Some limitations too:**
  - Low accuracy (large  $u$ )
  - Narrow range

Mix several precisions in the same code with the goal of

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

**Terminology varies:** Mixed precision, Multiprecision, Adaptive precision, Variable precision, Transprecision, Dynamic precision, . . .

Mix several precisions in the same code with the goal of

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

**Terminology varies:** Mixed precision, Multiprecision, Adaptive precision, Variable precision, Transprecision, Dynamic precision, . . .

**How** to select the right precision for the right variable/operation

- **Precision tuning:** autotuning based on the source code, my thesis area: CADNA / PROMISE...
  - ▲ Does not need any understanding of what the code does
  - ▼ Does not have any understanding of what the code does
- **This work:** another point of view, **exploit as much as possible the knowledge we have about the code**

# Adaptive precision algorithms

- Given an algorithm and a prescribed accuracy  $\varepsilon$ , adaptively select the minimal precision for each computation

⇒ **Why does it make sense to make the precision vary?**

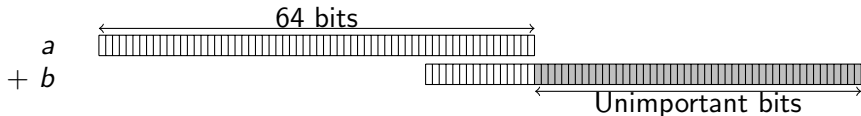
# Adaptive precision algorithms

- Given an algorithm and a prescribed accuracy  $\varepsilon$ , adaptively select the minimal precision for each computation

⇒ **Why does it make sense to make the precision vary?**

- Because not all computations are equally “important”!

Example:



and small elements produce small errors :

$$|\text{fl}(a \text{ op } b) - a \text{ op } b| \leq u |a \text{ op } b|, \quad \text{op} \in \{+, -, *, \div\}$$

⇒ **Opportunity for mixed precision:** adapt the precisions to the data at hand by storing and computing “less important” (usually smaller) data in lower precision

# Adaptive precision at the variable level?

- Pushing adaptive precision to the extreme: can we benefit from storing **each variable** in a (possibly) different precision?
  - Example:  $Ax = b$  with adaptive precision for each  $A_{ij}$ 
    - **Is it worth it?**  
Need to have elements of **widely different magnitudes**
    - **Is it practical?**  
Probably not for compute-bound applications, but could it work for **memory-bound** ones?
- ⇒ **Natural candidate: sparse matrices**



# Sparse matrix–vector product (SpMV)

$$y = Ax, A \in \mathbb{R}^{m \times n}$$

```
for  $i = 1 : m$  do
   $y_i = 0$ 
  for  $j \in \text{nnz}_i(A)$  do
     $y_i = y_i + a_{ij}x_j$ 
  end for
end for
```

- Standard error analysis for  $y = Ax$  performed in a uniform precision  $\varepsilon$  gives,

$$|\hat{y}_i - y_i| \leq n_i \varepsilon \sum_{j \in \text{nnz}_i(A)} |a_{ij}x_j|$$

- Idea:** store elements of  $A$  in a precision inversely proportional to their magnitude (**smaller elements in lower precision**)

```
for  $i = 1 : m$  do
   $y_i = 0$ 
  for  $k = 1 : p$  do
     $y_i^{(k)} = 0$ 
    for  $j \in \text{nnz}_i(A)$  do
      if  $a_{ij}x_j \in B_{ik}$  then
         $y_i^{(k)} = y_i^{(k)} + a_{ij}x_j$  at precision  $u_k$ 
      end if
    end for
     $y_i = y_i + y_i^{(k)}$ 
  end for
end for
```

- Split row  $i$  of  $A$  into  $p$  buckets  $B_{ik}$  and sum elements of  $B_{ik}$  in precision  $u_k$
- Error analysis:  $|\hat{y}_i^{(k)} - y_i^{(k)}| \leq n_i^{(k)} u_k \sum_{a_{ij}x_j \in B_{ik}} |a_{ij}x_j|$

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

⇒ Build the buckets such that  $u_k \sum_{a_{ij}x_j \in B_{ik}} |a_{ij}x_j| \approx \varepsilon \sum_j |a_{ij}x_j|$

- By setting  $B_{ik}$  to the interval  $(\varepsilon\beta_i/u_{k+1}, \varepsilon\beta_i/u_k]$ , we obtain  $|\hat{y}_i^{(k)} - y_i^{(k)}| \leq n_i^{(k)} \varepsilon\beta_i$  and so  $|\hat{y}_i - y_i| \leq n_i \varepsilon\beta_i$

- Two possible choices for  $\beta_i$ :

- $\beta_i = \sum_j |a_{ij}x_j| \Rightarrow$  guarantees  $O(\varepsilon)$  **componentwise** error:

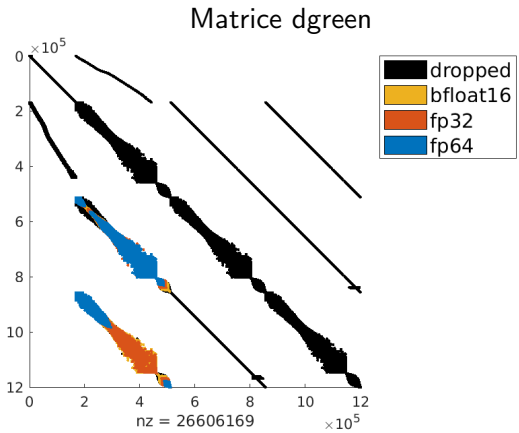
$$|\hat{y}_i - y_i| \leq n\varepsilon \sum_j |a_{ij}x_j| \quad \forall i \in \{1, \dots, n\}$$

- $\beta_i = \|A\| \|x\| \Rightarrow$  guarantees  $O(\varepsilon)$  **normwise** error:

$$\|\hat{y}_i - y_i\| \leq n\varepsilon \|A\| \|x\|$$

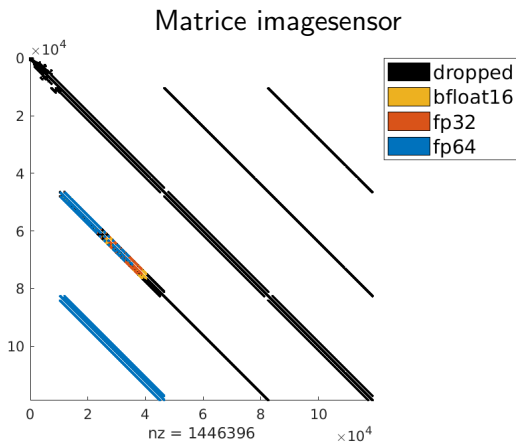
- Smallest elements are simply **dropped**

# Visualise mixed-precision gains



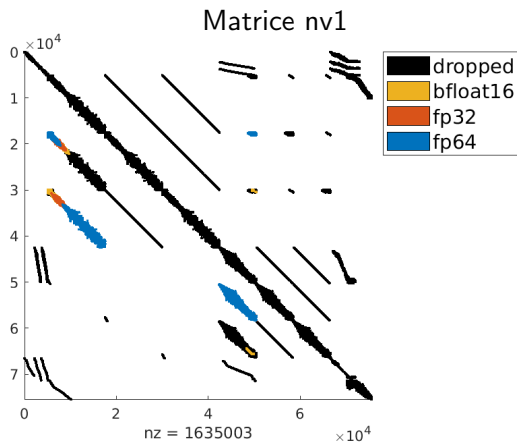
For some matrices, many elements can be dropped that leads to major gains.

# Visualise mixed-precision gains



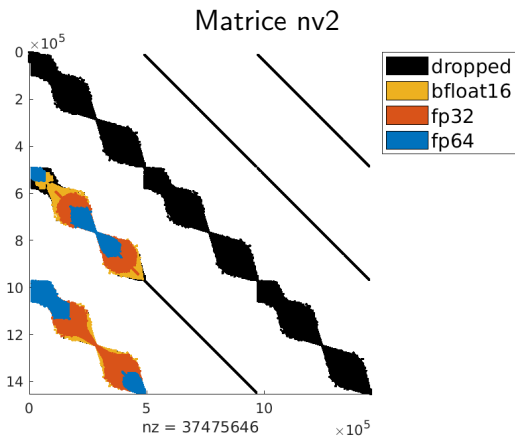
For some matrices, many elements can be dropped that leads to major gains.

# Visualise mixed-precision gains



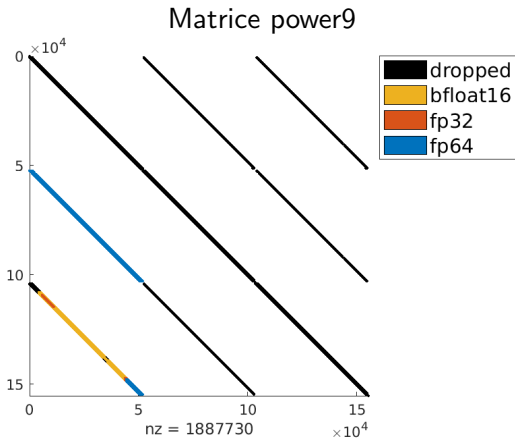
For some matrices, many elements can be dropped that leads to major gains.

# Visualise mixed-precision gains



For some matrices, many elements can be dropped that leads to major gains.

# Visualise mixed-precision gains



For some matrices, many elements can be dropped that leads to major gains.



- 33 matrices coming from SuiteSparse collection and industrial partners with at most 166M non-zeros

- 33 matrices coming from SuiteSparse collection and industrial partners with at most 166M non-zeros
- 3 different **accuracy targets**:
  - $\epsilon = 2^{-24}$
  - $\epsilon = 2^{-37}$
  - $\epsilon = 2^{-53}$

Various sets of precision formats:

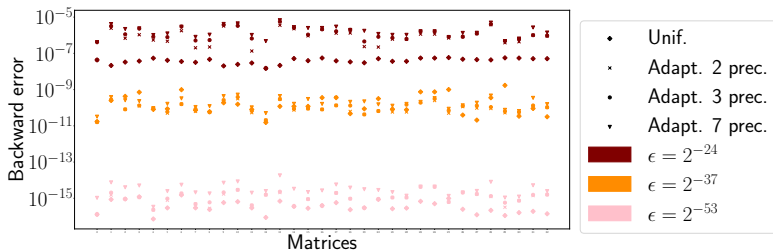
- 2 **precisions**: fp32, fp64
- 3 **precisions**: bfloat16, fp32, fp64
- 7 **precisions**: bfloat16, "fp24", fp32, fp64, "fp40", "fp48", "fp56"

---

	Bits	
	Mantissa	Exponent
bfloat16	8	8
"fp24"	8	8
fp32	24	8
"fp40"	29	11
"fp48"	37	11
"fp56"	45	11
fp64	53	11

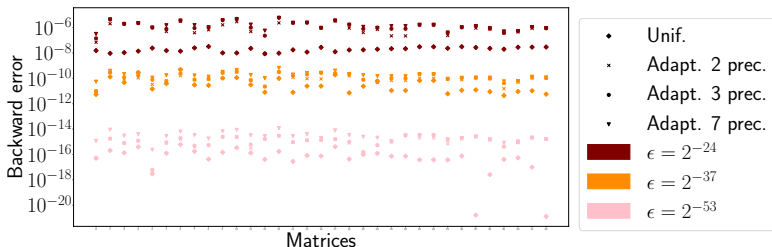
---

## Maintaining componentwise accuracy



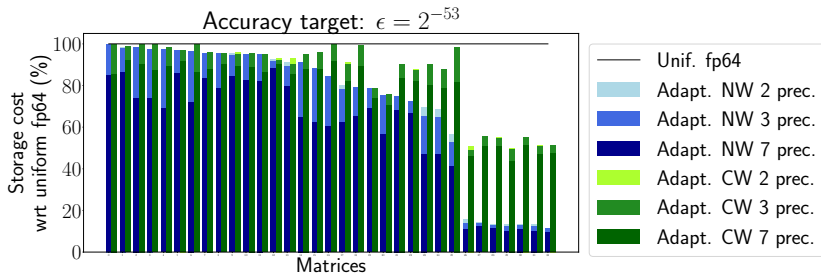
Adaptive methods preserve an accuracy close to the accuracy of uniform methods.

## Maintaining normwise accuracy



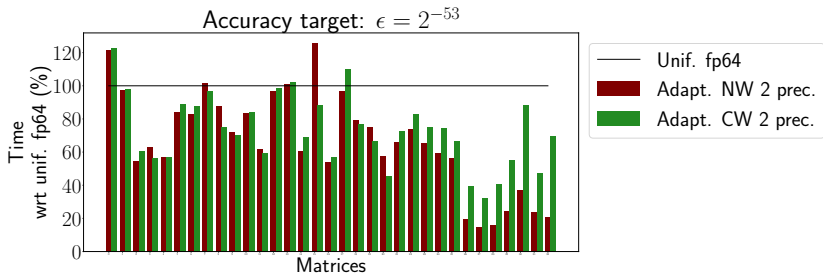
Adaptive methods preserve an accuracy close to the accuracy of uniform methods.

## Theoretical storage gains targetting $\epsilon = 2^{-53}$ accuracy (fp64)



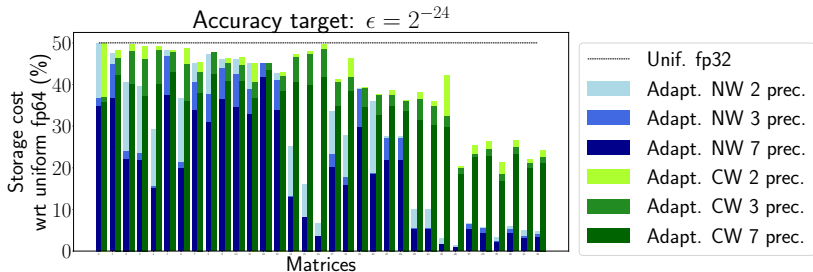
Up to **88%** of storage reduction

## Actual time gains targetting $\epsilon = 2^{-53}$ accuracy (fp64)



Up to **85%** of time reduction

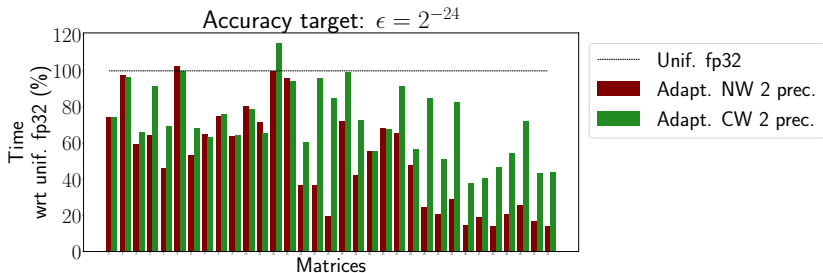
## Theoretical storage gains targetting $\epsilon = 2^{-24}$ accuracy (fp32)



Up to **97%** of storage reduction

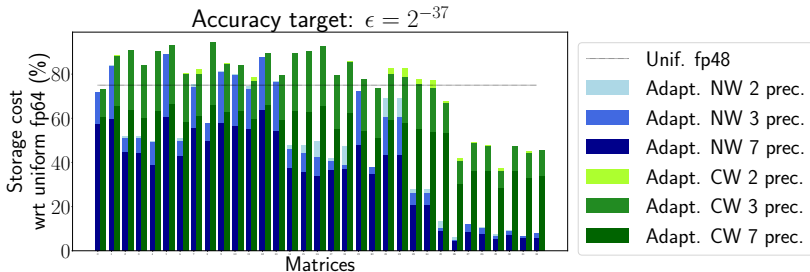


## Actual time gains targetting $\epsilon = 2^{-24}$ accuracy (fp32)



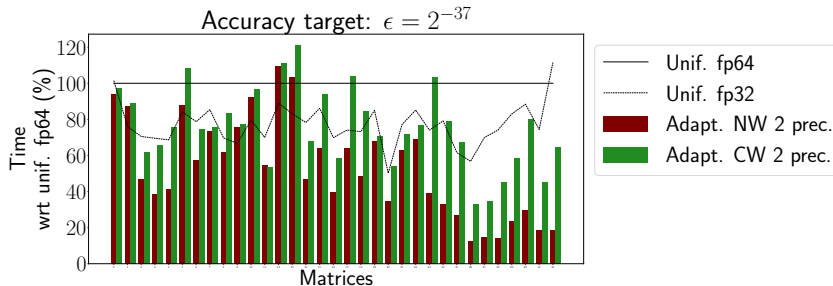
Up to **88%** of time reduction

## Theoretical storage gains targetting $\epsilon = 2^{-37}$ accuracy (unavailable in hardware!)



We are able to target any kind of accuracy with only natively supported precisions.

**Actual time gains targetting  $\epsilon = 2^{-37}$  accuracy (unavailable in hardware!)**



We are able to target any kind of accuracy with only natively supported precisions.

## Performance of GMRES rely on SpMV

```
r = b - Ax0
β = ||r||2
q1 = r/β
for k = 1, 2, ... do
  y = Aqk
  for j = 1: k do
    hjk = qjT y
    y = y - hjkqj
  end for
  hk+1,k = ||y||2
  qk+1 = y/hk+1,k
  Solve the least squares problem minck ||Hck - βe1||2
  xk = x0 + Qkck
end for
```

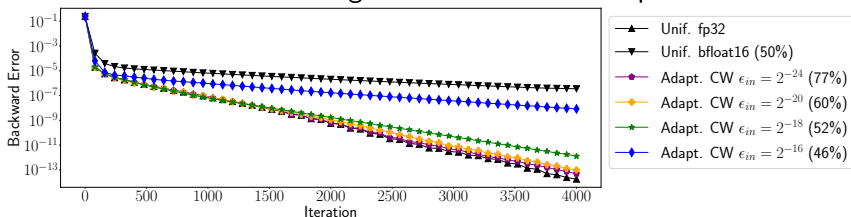
How does the adaptive method affect the convergence?

- Adaptive SpMV achieve larger speedups for lower accuracy targets
- GMRES-based iterative refinement particularly attractive

```
1: for  $i = 1, 2, \dots$  do  
2:    $r_i = b - Ax_{i-1}$   
3:   Solve  $Ad_i = r_i$  by GMRES  
4:    $x_i = x_{i-1} + d_i$   
5: end for
```

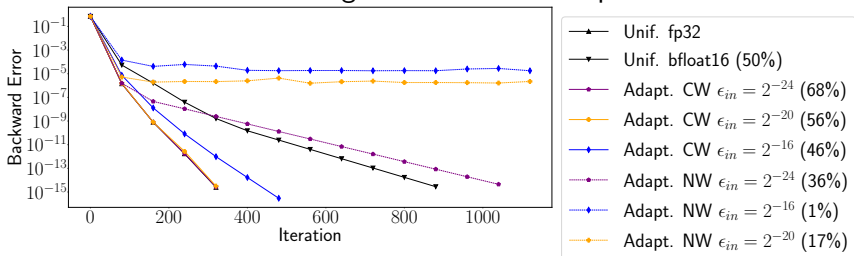
- SpMV line 2 in CW adaptive, target  $\epsilon = 2^{-53}$
- GMRES line 3 experimented with multiple variants
- Use of simple Jacobi preconditioner
- Fixed outside threshold  $\epsilon_{out} = 2^{-53}$
- Varying inside threshold  $\epsilon_{in} = 2^{-24}$
- Fixed inside restart to 80 iterations

## GMRES convergence for matrix ML\_Laplace



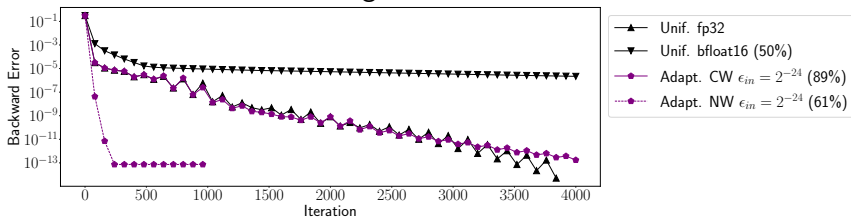
- With reasonable accuracy targets, adaptive SpMV, does not affect the convergence scheme
- Choose the best compromise between the iteration cost and the number of iterations

## GMRES convergence for matrix CoupCons3D



- For a given  $\epsilon_{in}$  value, NW variants achieve a lower cost but a slower convergence than CW ones
- $\epsilon_{in} = 2^{-24}$  leads to the best NW variant, which converges in 1040 iterations with an SpMV cost of 36% of the fp32 uniform one
- $\epsilon_{in} = 2^{-20}$  leads to the best CW variant, which converges in 320 iterations with a corresponding SpMV cost of 56%

## GMRES convergence for matrix Geo\_1438



- Surprising behavior
- Consistently reproduced and occurs for several other matrices in our set
- Aggressive dropping of small coefficients might lead to a “nicer” matrix for which GMRES can converge quickly.



- **Adaptive precision SpMV algorithm**
  - Builds computing buckets according to the elements magnitude
  - Targets any accuracy
  - Matrix-dependent gains
- **Application to Krylov solvers**
  - Reasonable accuracy targets preserves convergence scheme

More info



**Thank you! Any questions?**