Hierarchical approach for deriving a reproducible LU factorization

Roman Iakymchuk[1]

joint work with

Sylvain Collange, David Defour, Erwin Laure,
Enrique S. Quitana Ortí and Stef Graillat

[1]KTH Royal Institute of Technology, CSC, CST/PDC

riakymch@kth.se

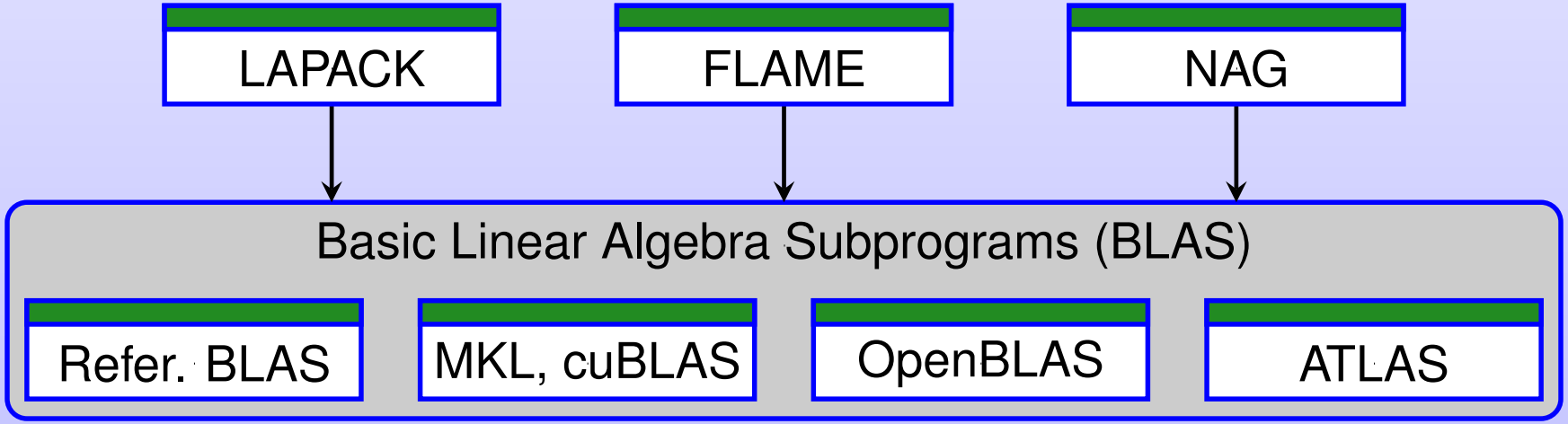
RAIM 2017, Oct 24-26, 2017
Lyon, France

Linear Algebra Libraries

BLAS-1 [1979]:	$y := y + \alpha x$ $\alpha := \alpha + x^T y$	$\alpha \in \mathbb{R}; x, y \in \mathbb{R}^n$	$2/3$
BLAS-2 [1988]:	$A := A + xy^T$ $y := A^{-1}x$	$A \in \mathbb{R}^{n \times n}; x, y \in \mathbb{R}^n$	2
BLAS-3 [1990]:	$C := C + AB$ $C := A^{-1}B$	$A, B, C \in \mathbb{R}^{n \times n}$	$n/2$

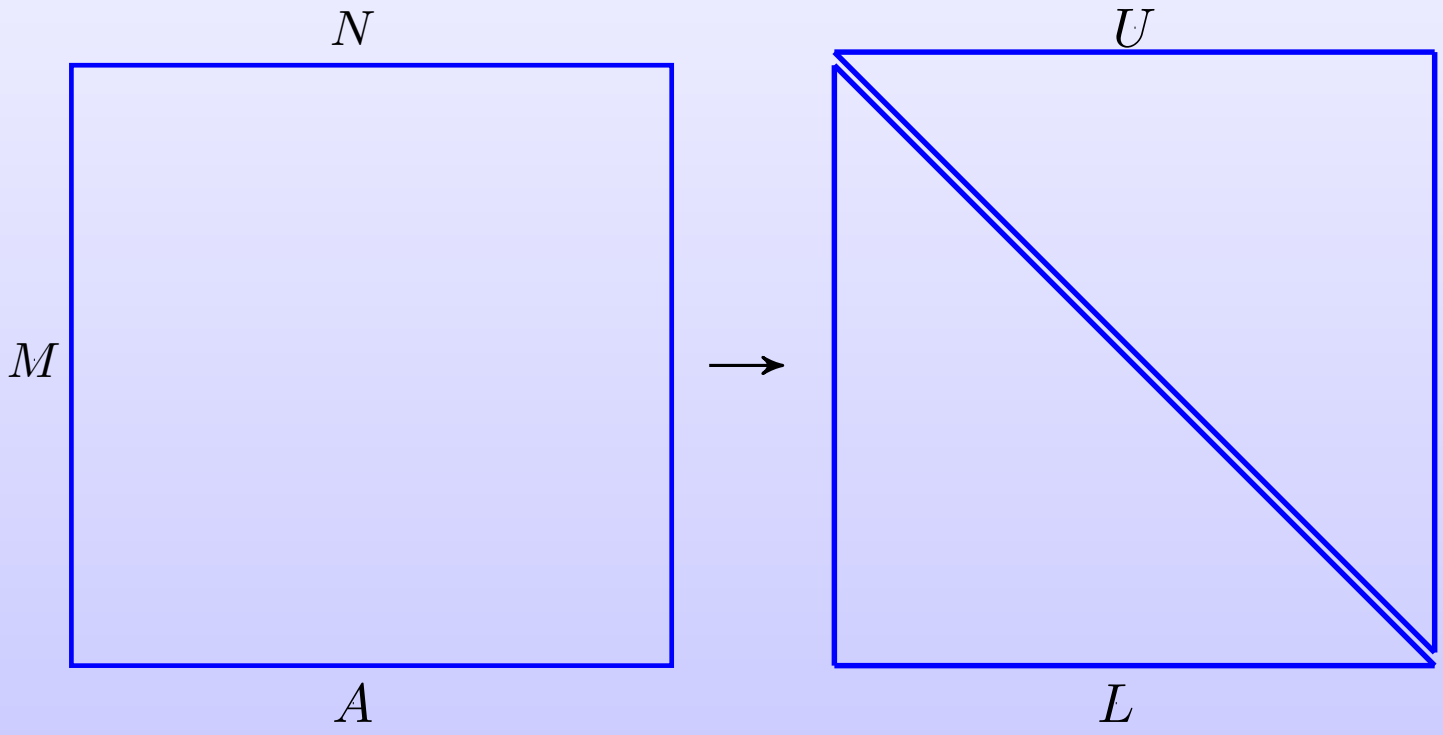
Linear Algebra Libraries

BLAS-1 [1979]:	$y := y + \alpha x$ $\alpha := \alpha + x^T y$	$\alpha \in \mathbb{R}; x, y \in \mathbb{R}^n$	$2/3$
BLAS-2 [1988]:	$A := A + xy^T$ $y := A^{-1}x$	$A \in \mathbb{R}^{n\times n}; x, y \in \mathbb{R}^n$	2
BLAS-3 [1990]:	$C := C + AB$ $C := A^{-1}B$	$A, B, C \in \mathbb{R}^{n\times n}$	$n/2$



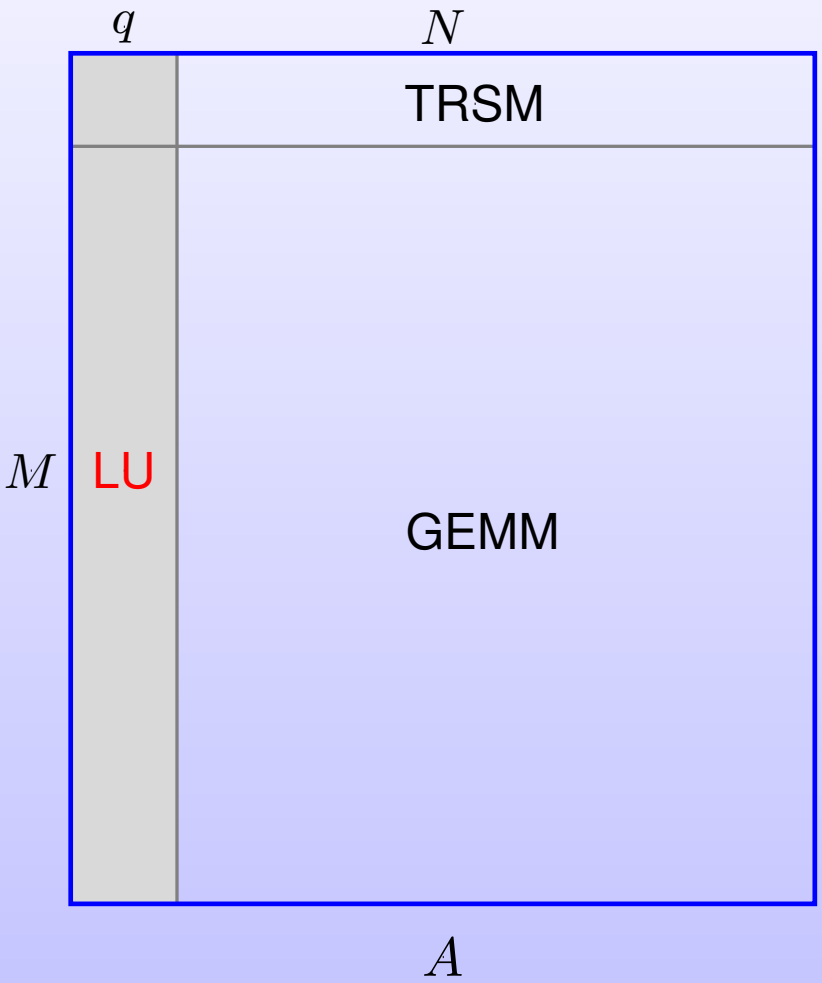
LU Factorization

$$\boxed{Ax = b} \Rightarrow \boxed{A = LU}$$



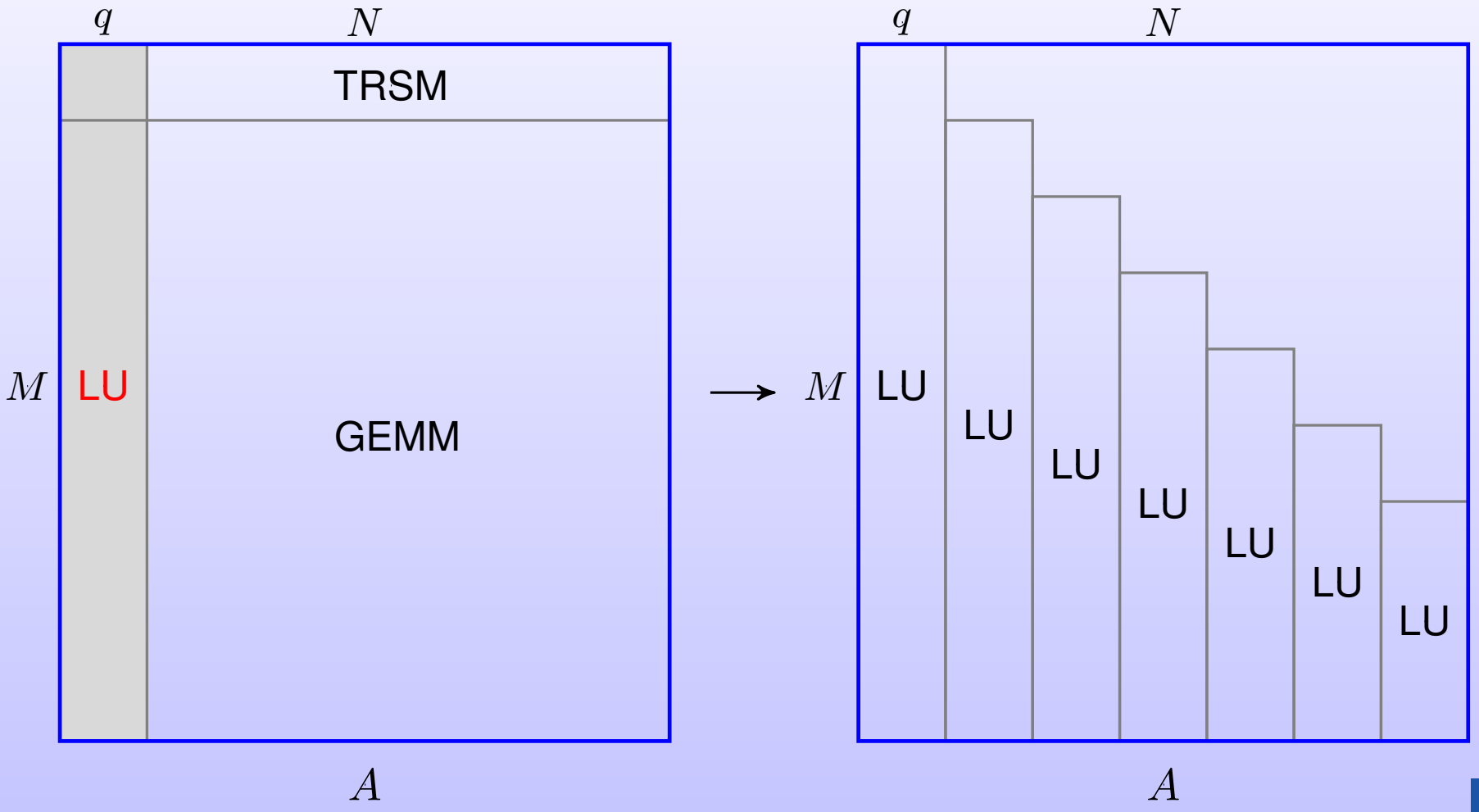
LU Factorization

$$A = LU$$



LU Factorization

$$A = LU$$



Motivation

Unblocked LU Factorization

Variant 1

$$a_{01} := L_{00}^{-1} a_{01} \quad \texttt{TRSV}$$
$$a_{10}^T := a_{10}^T U_{00}^{-1} \quad \texttt{TRSV}$$
$$\alpha_{11} := \alpha_{11} - a_{10}^T a_{01} \quad \texttt{DOT}$$

Variant 2

$$a_{10}^T := a_{10}^T U_{00}^{-1} \quad \texttt{TRSV}$$
$$\alpha_{11} := \alpha_{11} - a_{10}^T a_{01} \quad \texttt{DOT}$$
$$a_{12}^T := a_{12}^T - a_{10}^T A_{02} \quad \texttt{GEMV}$$

Variant 3

$$a_{01} := L_{00}^{-1} a_{01} \quad \texttt{TRSV}$$
$$\alpha_{11} := \alpha_{11} - a_{10}^T a_{01} \quad \texttt{DOT}$$
$$a_{21} := \frac{a_{21} - A_{20} a_{01}}{\alpha_{11}} \quad \texttt{GEMV}$$

Variant 4

$$\alpha_{11} := \alpha_{11} - a_{10}^T a_{01} \quad \texttt{DOT}$$
$$a_{21} := \frac{a_{21} - A_{20} a_{01}}{\alpha_{11}} \quad \texttt{GEMV}$$
$$a_{12}^T := a_{12}^T - a_{10}^T A_{02} \quad \texttt{GEMV}$$

Variant 5

$$a_{21} := a_{21} / \alpha_{11} \quad \texttt{SCAL}$$
$$A_{22} := A_{22} - a_{21} a_{12}^T \quad \texttt{GER}$$

Outline

Outline

Accuracy and Reproducibility

Problems

- Floating-point arithmetic suffers from rounding errors
- Floating-point operations ($+$,$\times$) are commutative but non-associative

$$(-1+1)+2^{-53} \neq -1+(1+2^{-53}) \quad \text{in double precision}$$

Accuracy and Reproducibility

Problems

- Floating-point arithmetic suffers from rounding errors
- Floating-point operations $(+, \times)$ are commutative but non-associative

$$2^{-53} \neq 0 \quad \text{in double precision}$$

Accuracy and Reproducibility

Problems

- Floating-point arithmetic suffers from rounding errors
- Floating-point operations ($+$,$\times$) are commutative but non-associative

$$(-1+1)+2^{-53} \neq -1+(1+2^{-53}) \quad \text{in double precision}$$

- Consequence: results of floating-point computations depend on the order of computation
- Results computed by performance-optimized parallel floating-point libraries may be often inconsistent: each run returns a different result

- **Reproducibility** – ability to obtain bit-wise identical results from run-to-run on the same input data on the same or different architectures

Sources of Non-Reproducibility

- Changing Data Layouts:
  - Data partitioning
  - Data alignment
- Changing Hardware Resources
  - Number of threads
  - Fused Multiply-Add support
  - Intermediate precision (64 bits, 80 bits, 128 bits, etc)
  - Data path (SSE, AVX, GPU warp, etc)
  - Number of processors
  - Network topology

Outline

Exact Multi-Level Parallel Reduction

Background

- Fixed FP expansions (FPE) with Error-Free Transformations

→ Example: double-double or quad-double (Briggs, Bailey, Hida, Li) (work well on a set of relatively close numbers)

Algorithm 1 FPE of size 2 (Dekker and Knuth)

Function[r, s] = TwoSum(a, b)

1: $r \leftarrow a + b$
2: $z \leftarrow r - a$
3: $s \leftarrow (a - (r - z)) + (b - z)$

Exact Multi-Level Parallel Reduction

Background

- Fixed FP expansions (FPE) with Error-Free Transformations

→ Example: double-double or quad-double (Briggs, Bailey, Hida, Li) (work well on a set of relatively close numbers)

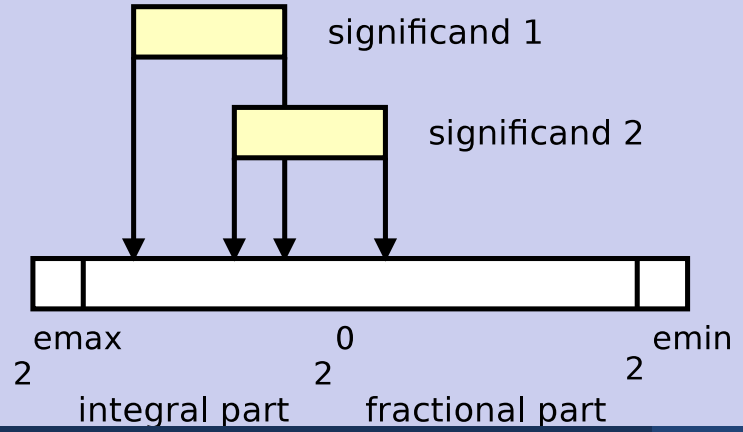
Algorithm 1 FPE of size 2 (Dekker and Knuth)

Function[r, s] = TwoSum(a, b)

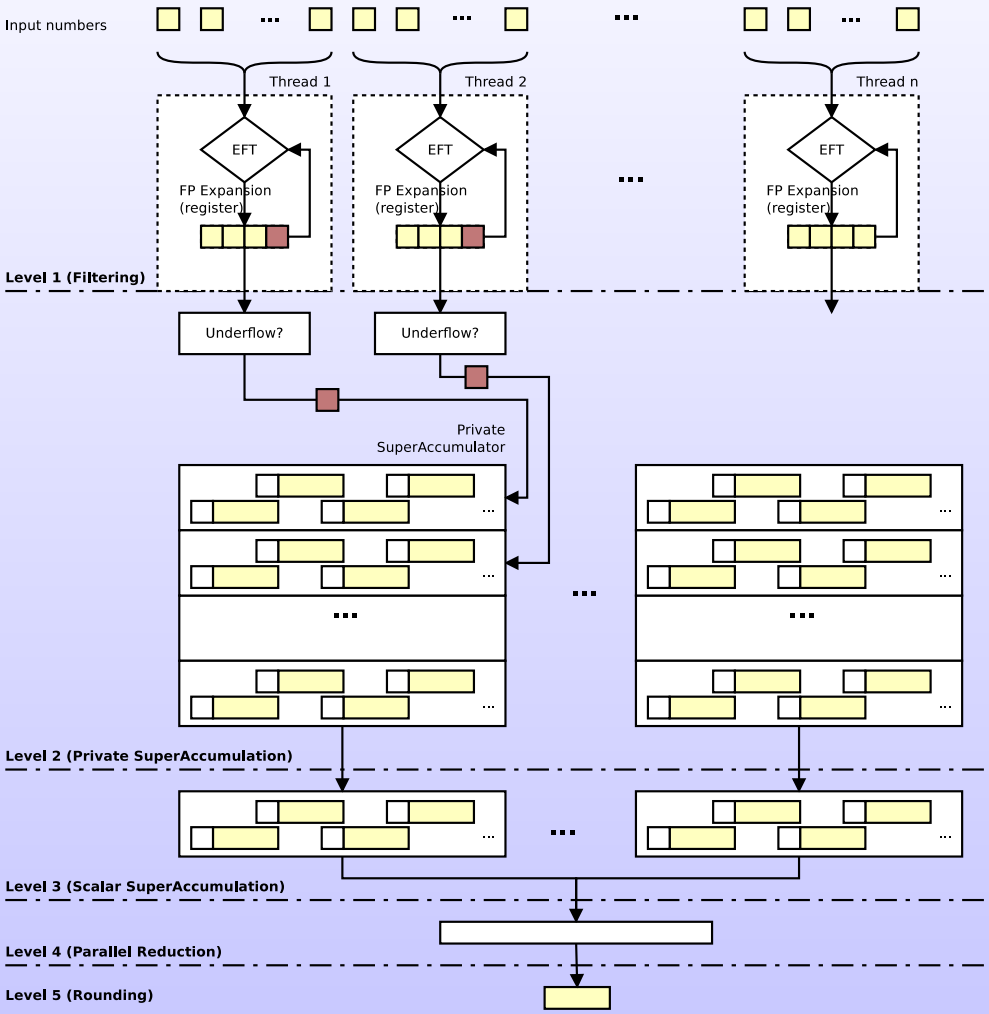
1: $r \leftarrow a + b$
2: $z \leftarrow r - a$
3: $s \leftarrow (a - (r - z)) + (b - z)$

- "Infinite" precision: reproducible independently from the inputs

→ Example: Kulisch accumulator (considered inefficient)



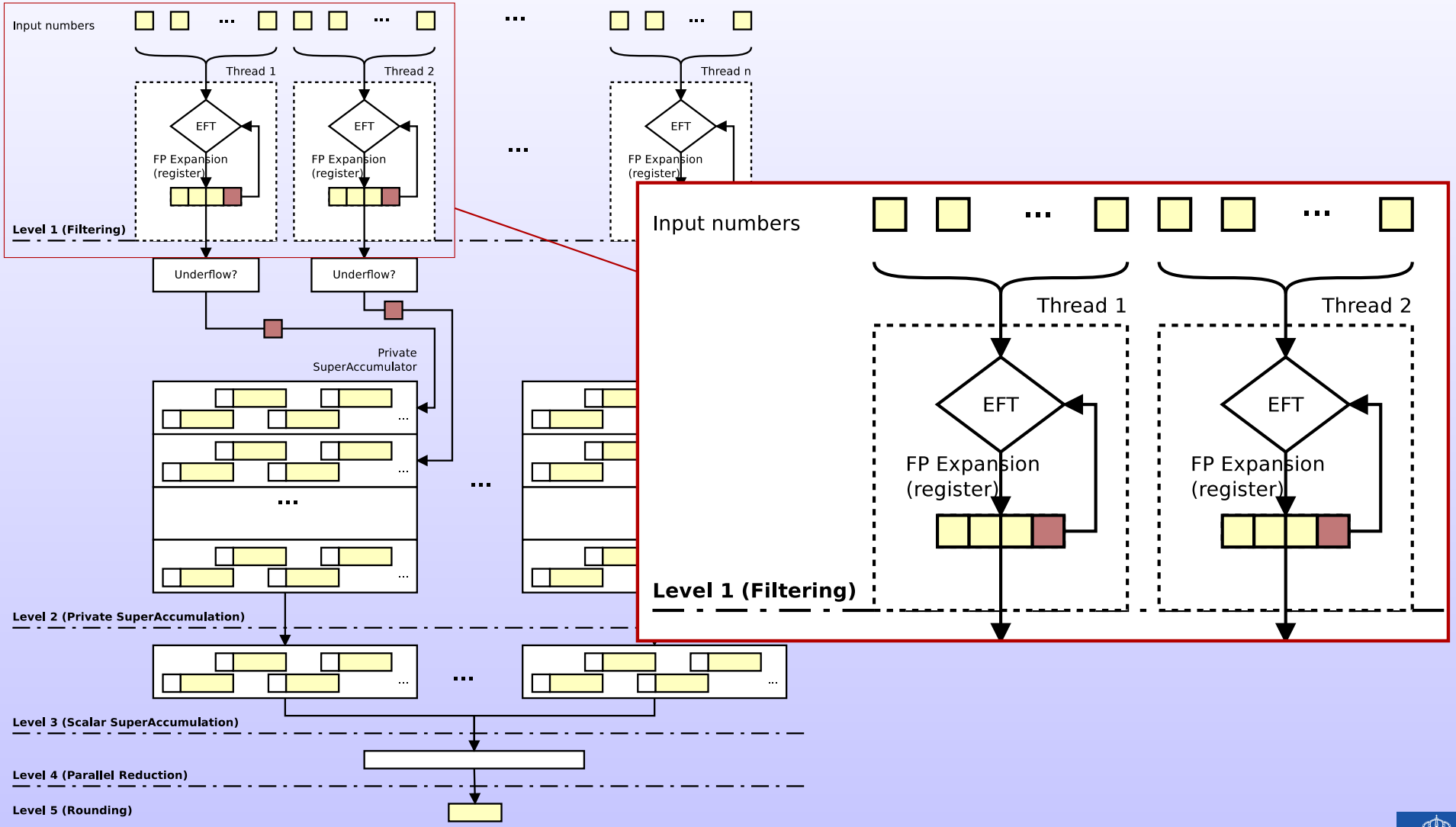
Exact Multi-Level Parallel Reduction



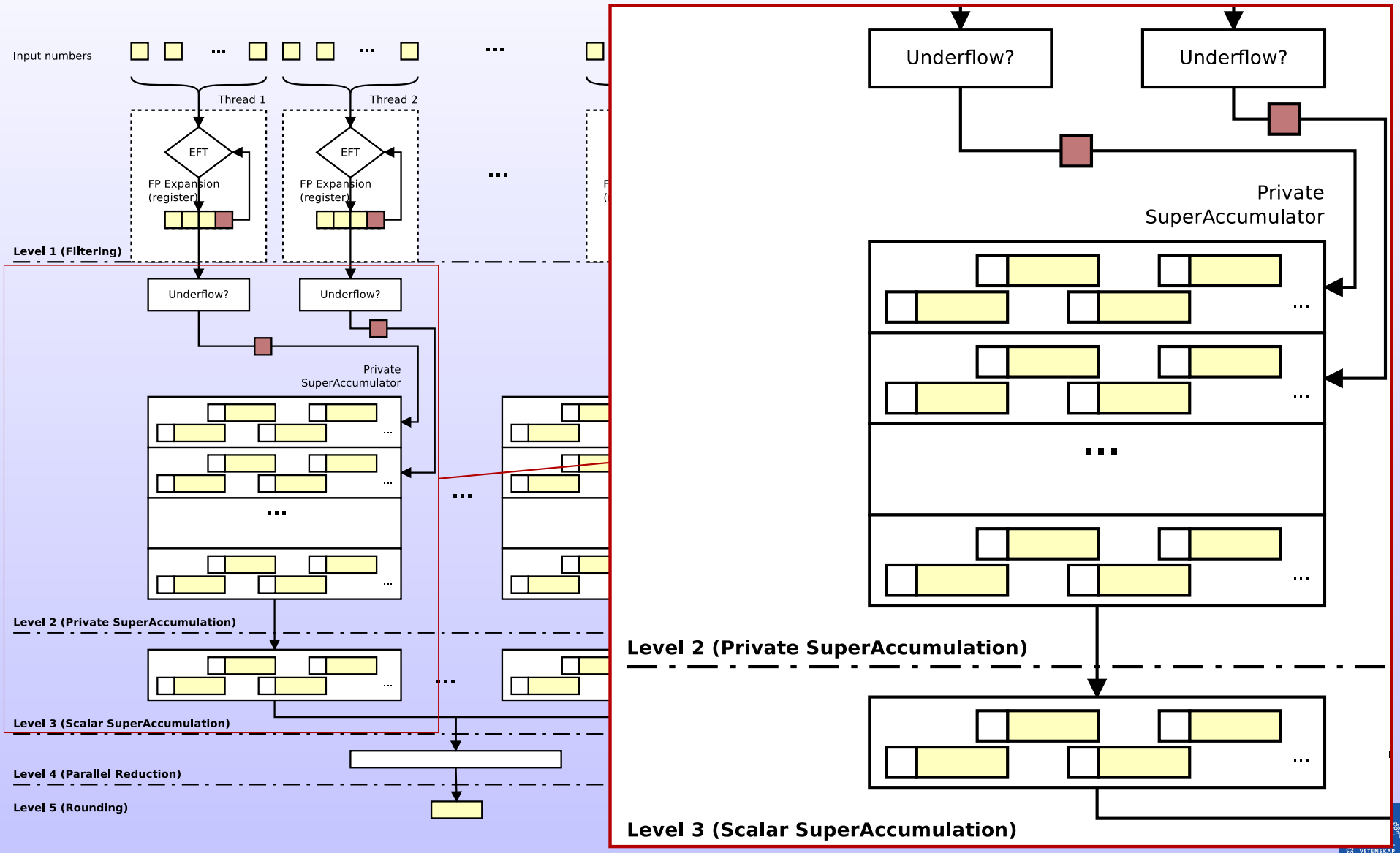
- Parallel algorithm with 5-levels
- Suitable for today's parallel architectures
- Based on FPE with EFT and Kulisch accumulator
- Guarantees "inf" precision

→ **bit-wise reproducibility**

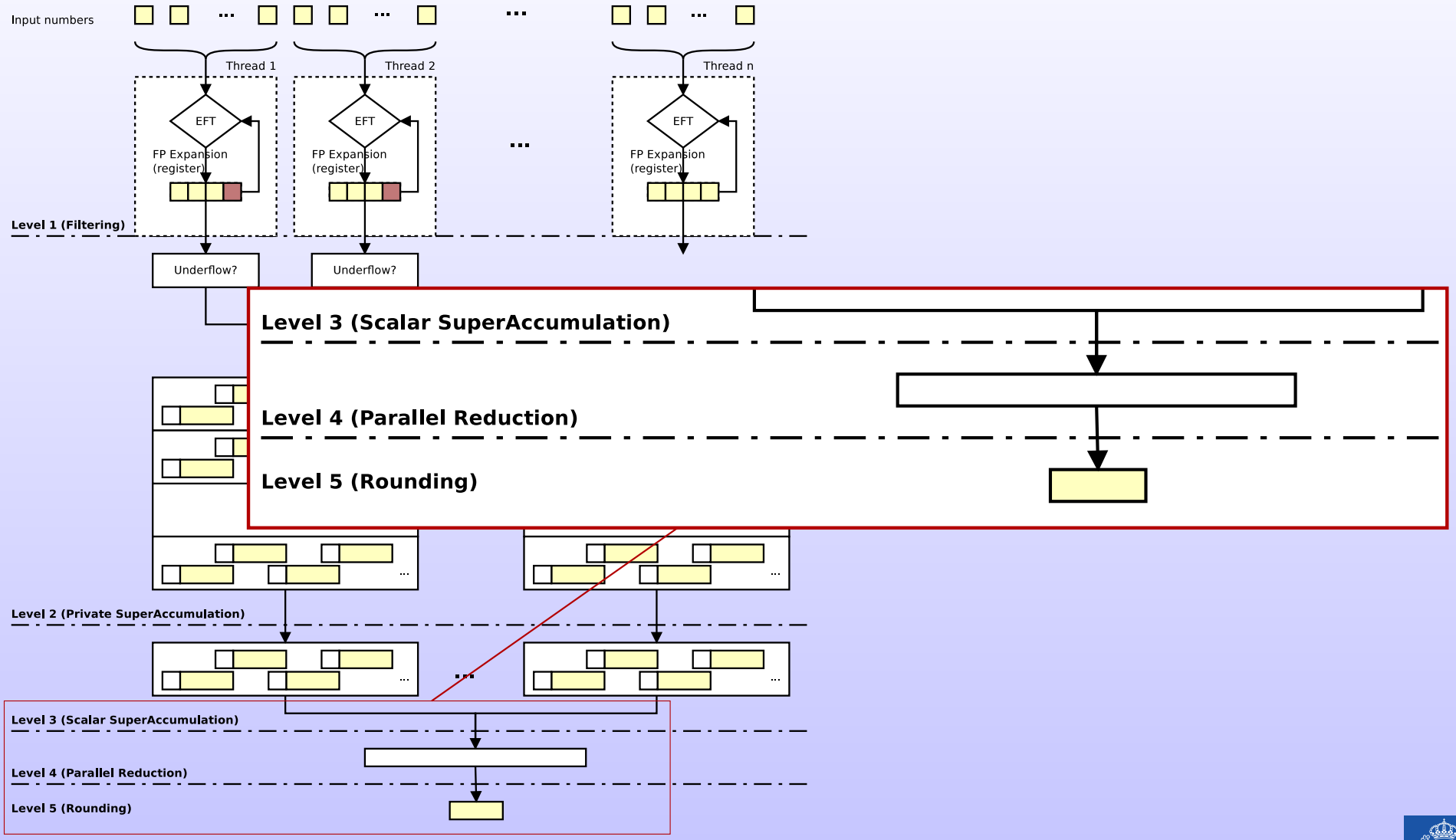
Level 1: Filtering



Level 2 and 3: Scalar Superaccumulator



Level 4 and 5: Reduction and Rounding



Outline

ExBLAS Highlights (1/2)

ExBLAS Status

- ExBLAS-1: `ExSUM`[a], `ExSCAL`, `ExDOT`, `ExAXPY`, ...
- ExBLAS-2: `ExGER`, `ExGEMV`, `ExTRSV`, `ExSYR`, ...
- ExBLAS-3: `ExGEMM`, `ExTRSM`, `ExSYR2K`, ...

[a] Routines in blue are already in ExBLAS

BLAS-1 routines

- Some are virtually built upon `ExSUM`

→ For instance, `ExDOT` = `TwoProd` + `ExSUM`

→ `TwoProd`(a,b):

1: $res \leftarrow a \cdot b$,
2: $err \leftarrow \mathrm{FMA}(a, b, -res)$

- The others are embarrassingly parallel

→ `ExAXPY` = $\mathrm{FMA}(\alpha, x[i], y[i])$

ExBLAS Highlights (2/2)

ExSCAL

- $x := \alpha * x \rightarrow$ correctly rounded and reproducible

ExBLAS Highlights (2/2)

ExSCAL

- $x := \alpha * x \rightarrow$ correctly rounded and reproducible
- Within LU: $x := 1/\alpha * x \rightarrow$ not correctly rounded

ExBLAS Highlights (2/2)

ExSCAL

- $x := \alpha * x \rightarrow$ correctly rounded and reproducible
- Within LU: $x := 1/\alpha * x \rightarrow$ not correctly rounded
- ExInvSCAL: $x := x/\alpha \rightarrow$ correctly rounded and reproducible

ExGER

- General case: $A := \alpha * x * y^T + A$

ExBLAS Highlights (2/2)

ExSCAL

- $x := \alpha * x \rightarrow$ correctly rounded and reproducible
- Within LU: $x := 1/\alpha * x \rightarrow$ not correctly rounded
- ExInvSCAL: $x := x/\alpha \rightarrow$ correctly rounded and reproducible

ExGER

- General case: $A := \alpha * x * y^T + A$
- Within LU: $A := x * y^T + A$. Using FMA $\rightarrow$ correctly rounded and reproducible

An unblocked LU Factorization: Variant 5

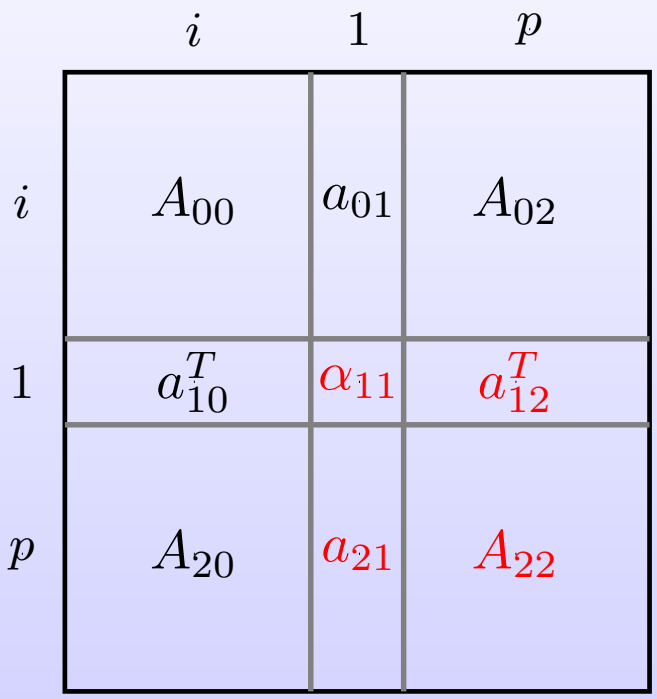
LU Factorization

$$\pi_1 := PivIndex\left(\frac{\alpha_{11}}{a_{21}}\right) \qquad (\textbf{max})$$

$$\left(\frac{\alpha_{11}}{a_{21}}\right) := P(\pi_1)\left(\frac{\alpha_{11}}{a_{21}}\right) \qquad (\textbf{swap})$$

$$a_{21} := a_{21}/\alpha_{11} \qquad (\textbf{scal})$$

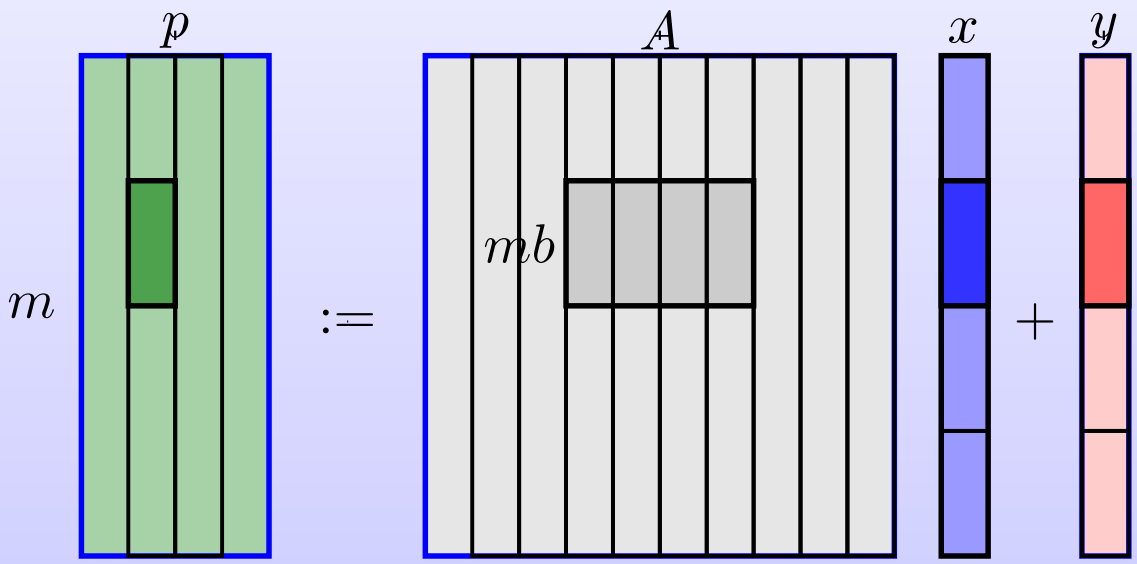
$$A_{22} := A_{22} - a_{21}a_{12}^T \qquad (\textbf{ger})$$



3×3 partitioning of A

Matrix-Vector Product

DGEMV: $y := \alpha Ax + \beta y$



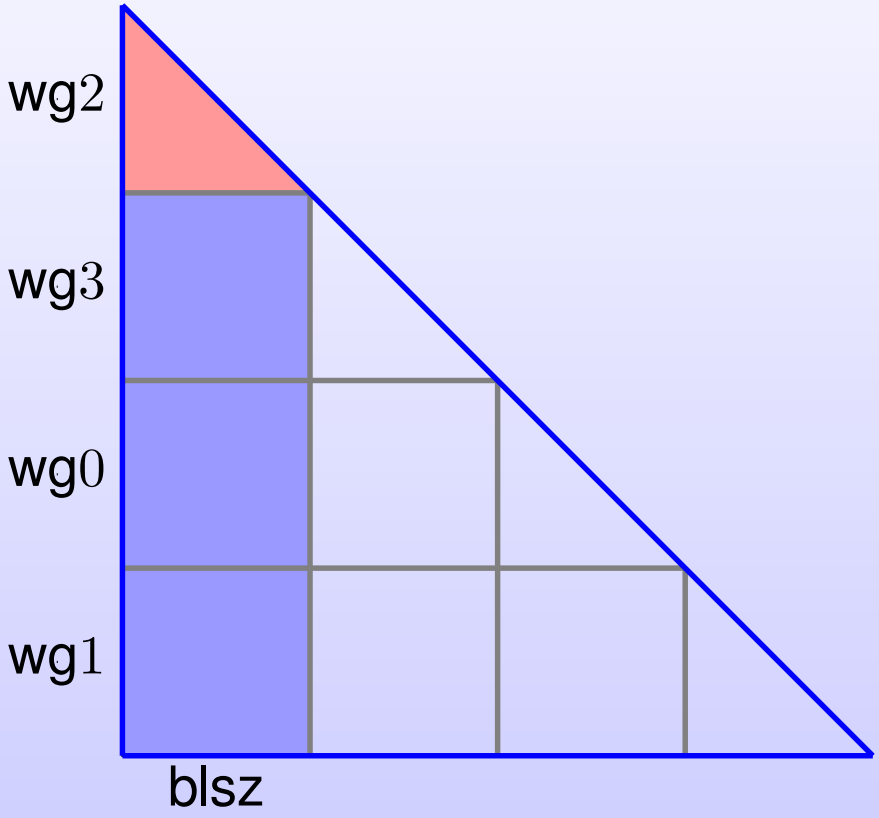
- Based on ExDOT
- `TwoProd`(a, b)

  1: $r \leftarrow a * b$

  2: $s \leftarrow fma(a, b, -r)$
- $fma(a, b, c) = a * b + c$

Triangular Solver

```
for i = 0 : blsz : n do
    for k = i : i + blsz do
        for j = 1 : k − 1 do
            [r, e] ← TwoProd(l_kj, −x_j)
            ExpansionAccumulate(r)
            ExpansionAccumulate(e)
        end for
        ExpansionAccumulate(b_k)
        ŝ ← RNDN(acc(k))
        x_k ← ŝ / l_kk
    end for
    for k = i + blsz : n do
        for j = i : i + blsz do
            [r, e] ← TwoProd(l_kj, −x_j)
            ExpansionAccumulate(r)
            ExpansionAccumulate(e)
        end for
    end for
end for
```



Partitioning of a lower triangular matrix L

Unblocked left-looking algorithm of LU

jik or jki variant

$$\left(\begin{array}{c} a_{01} \\ \hline \alpha_{11} \\ \hline a_{21} \end{array}\right) := P(p_0) \left(\begin{array}{c} a_{01} \\ \hline \alpha_{11} \\ \hline a_{21} \end{array}\right) \quad (\textbf{swap})$$

$$a_{01} := L_{00}^{-1} a_{01} \quad (\textbf{trsv})$$

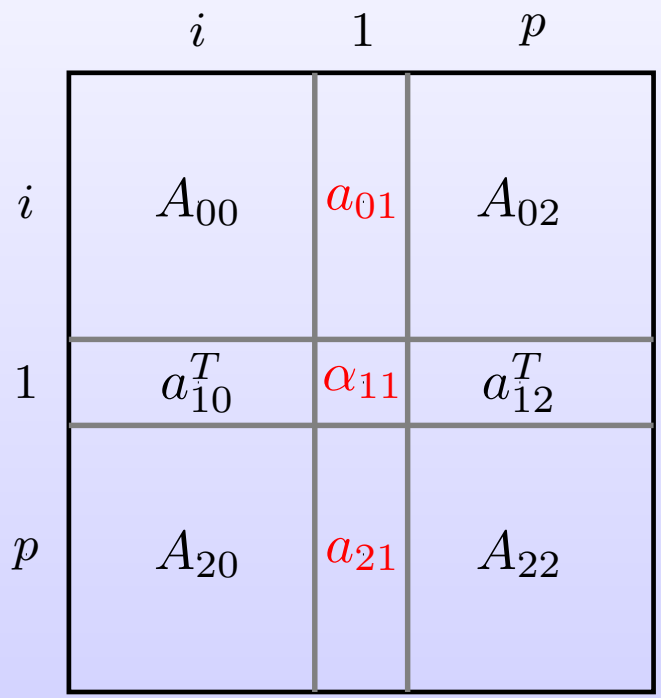
$$\alpha_{11} := \alpha_{11} - a_{10}^T a_{01} \quad (\textbf{dot})$$

$$a_{21} := a_{21} - A_{20} a_{01} \quad (\textbf{gemv})$$

$$\pi_1 := PivIndex \left(\begin{array}{c} \alpha_{11} \\ \hline a_{21} \end{array}\right) \quad (\textbf{max})$$

$$\left(\begin{array}{c} \alpha_{11} \\ \hline a_{21} \end{array}\right) := P(\pi_1) \left(\begin{array}{c} \alpha_{11} \\ \hline a_{21} \end{array}\right) \quad (\textbf{swap})$$

$$a_{21} := a_{21} / \alpha_{11} \quad (\textbf{scal})$$

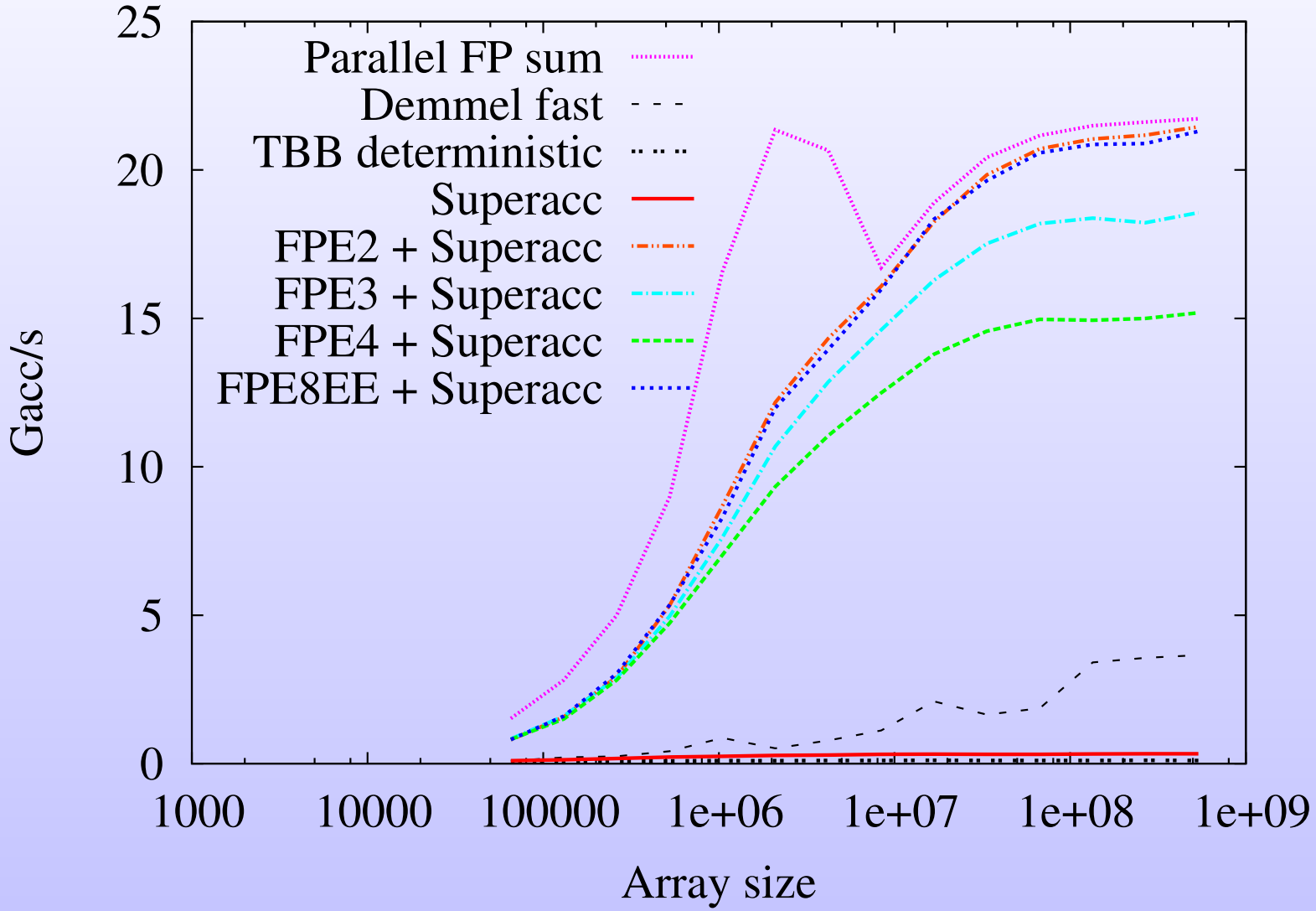


3×3 partitioning of A

Outline

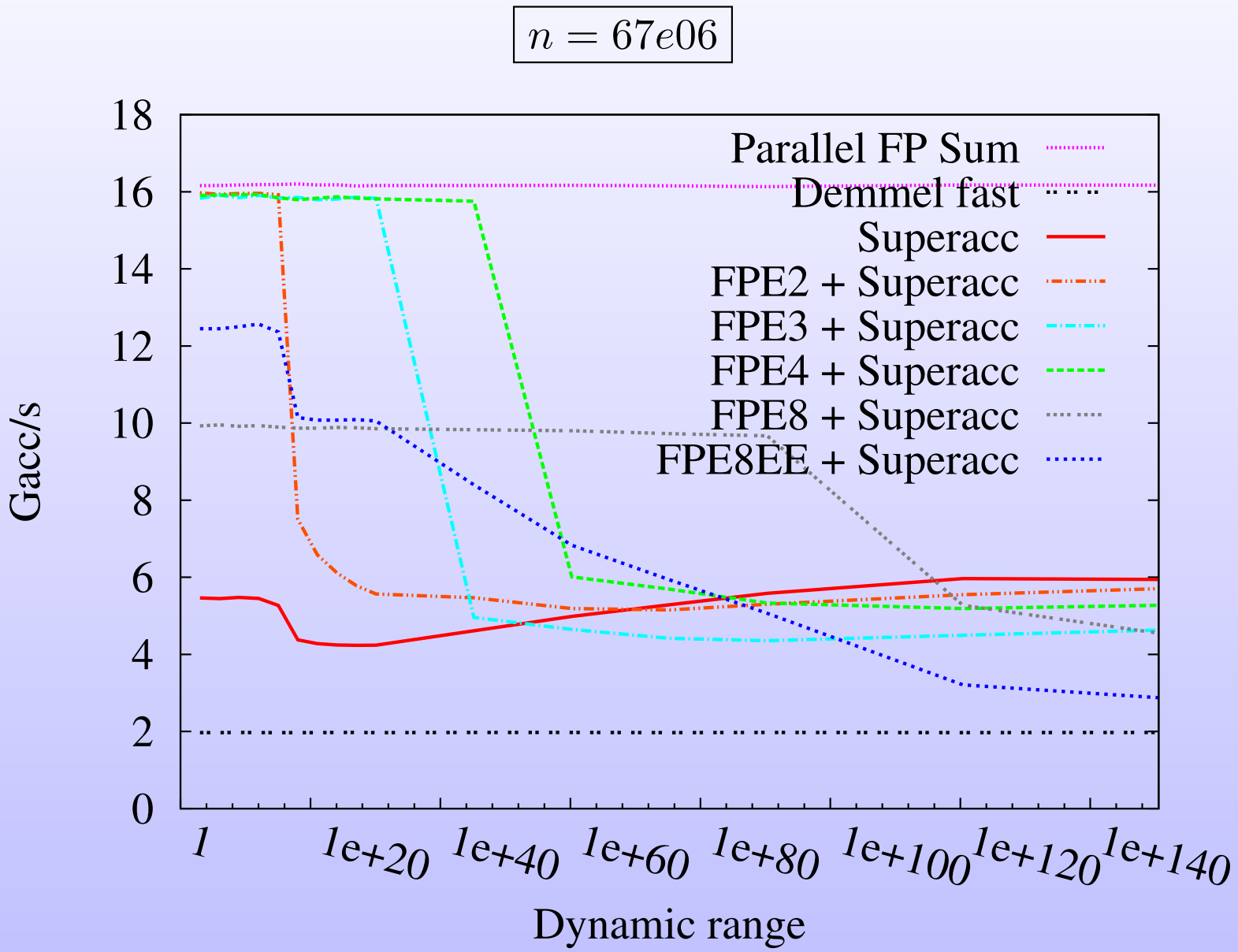
Parallel Reduction

Performance Scaling on Intel Xeon Phi



Parallel Reduction

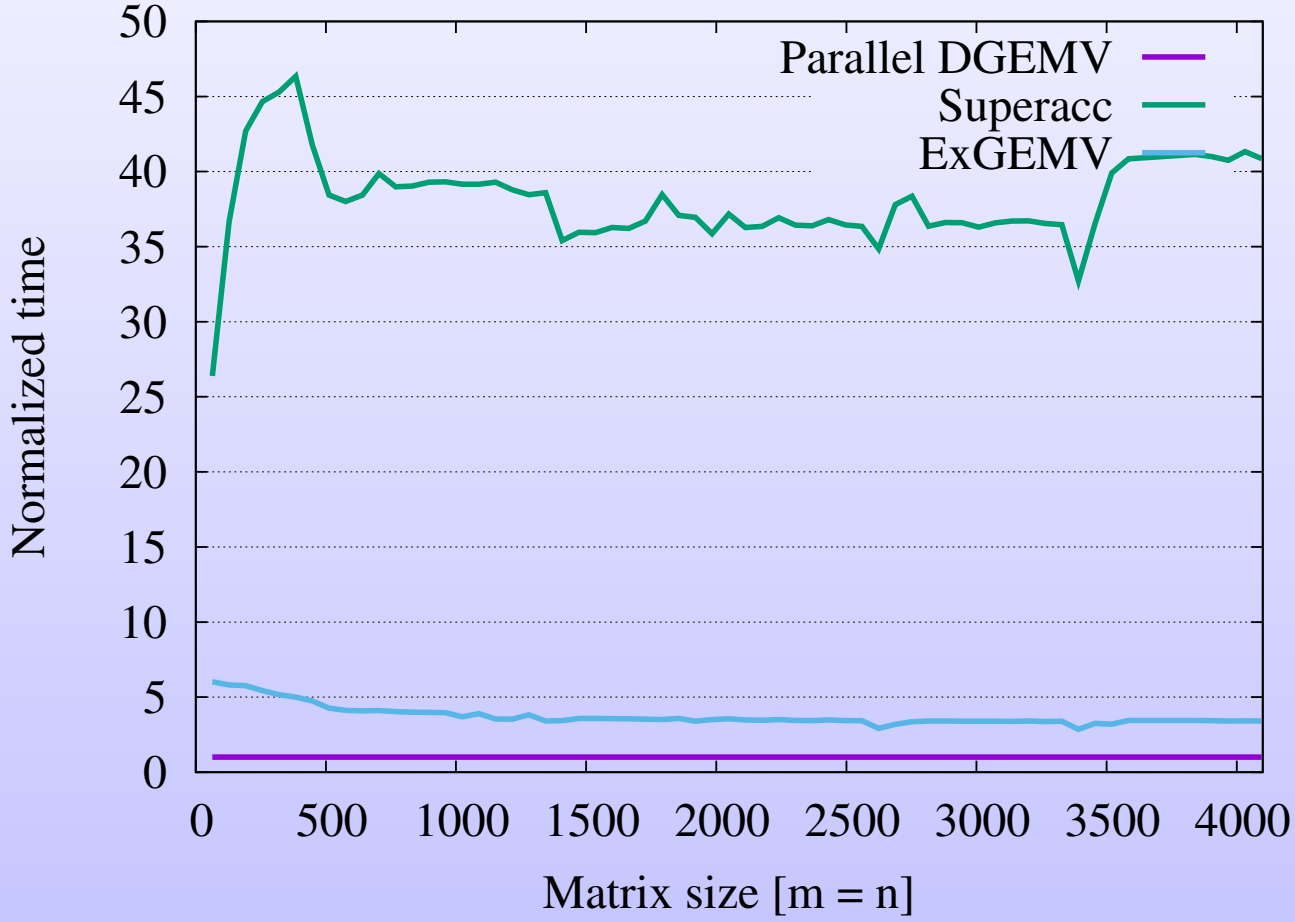
Data-Dependent Performance on NVIDIA Tesla K20c



Matrix-Vector Product

Performance Scaling on NVIDIA Tesla K80

$$\texttt{GEMV}: y := \alpha A x + \beta y$$

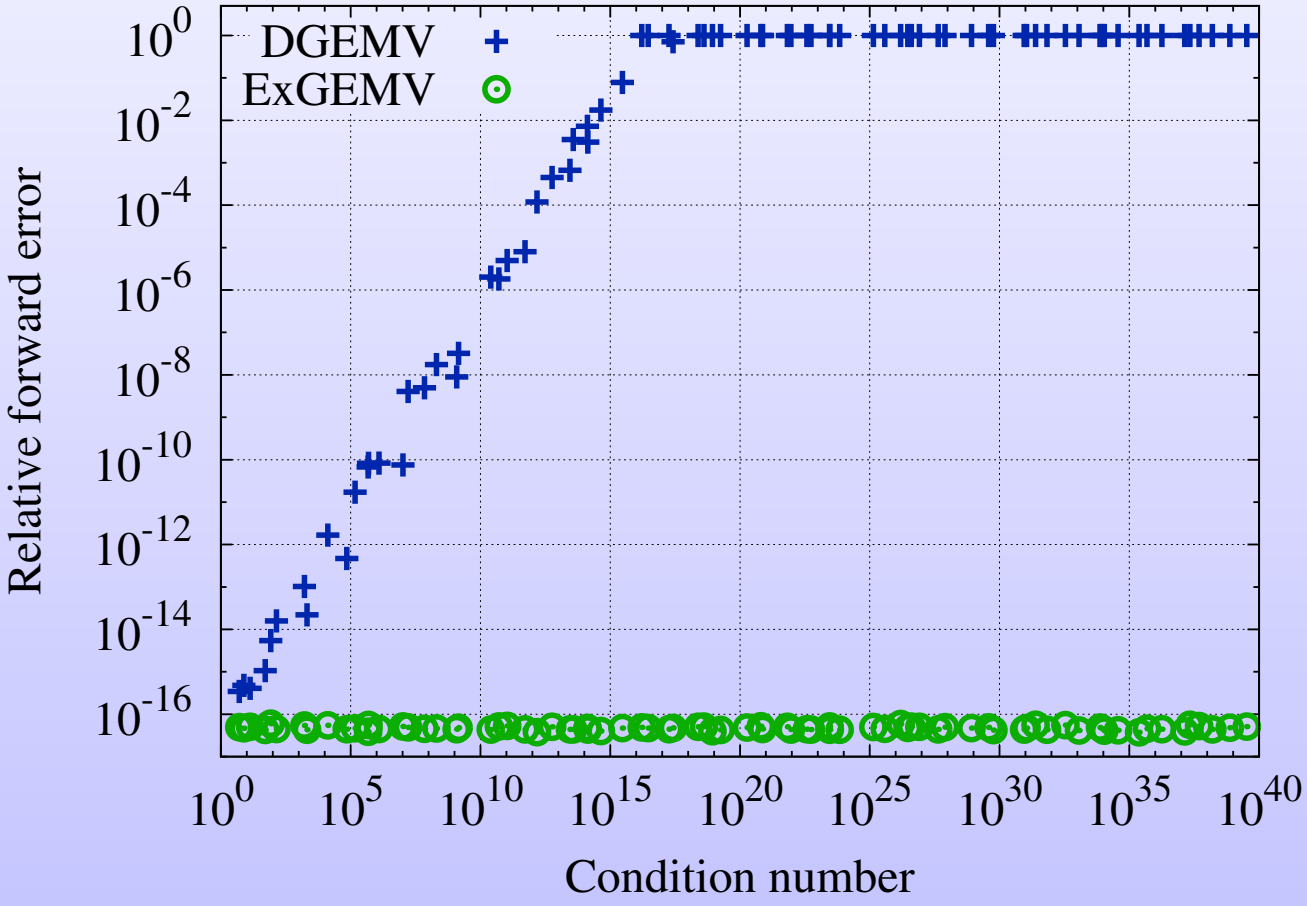


- Blocked `ExGEMV`
- Based on `ExDOT`

Matrix-Vector Product

Accuracy

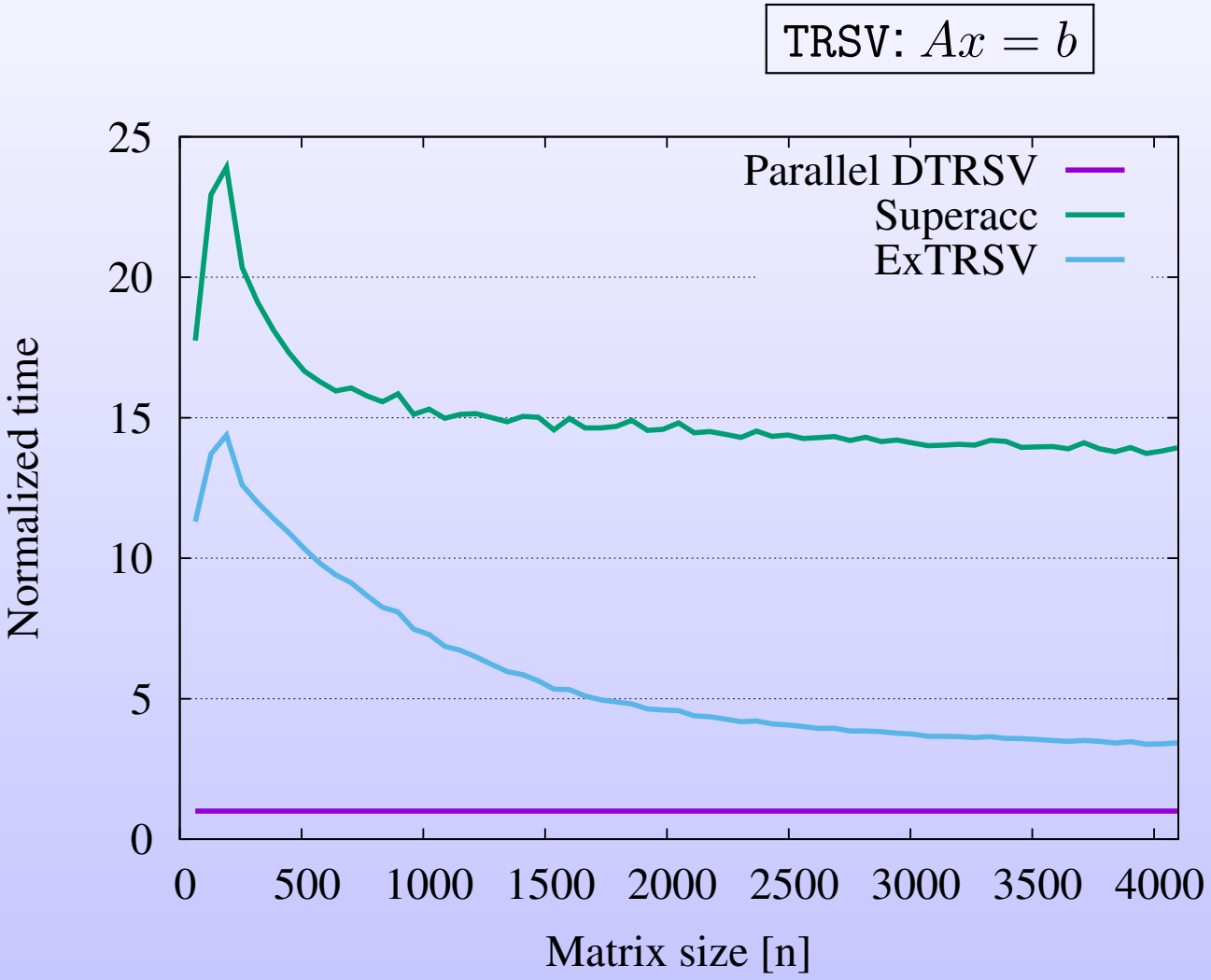
$\boxed{\texttt{GEMV}\colon y := Ax}$



- Preserve every bit of information
- Correctly-rounded
- $\mathrm{cond}(A, x) = \frac{\||A|\cdot|x|\|}{\|A\cdot x\|}$

Triangular Solver

Performance Scaling on NVIDIA Tesla K420

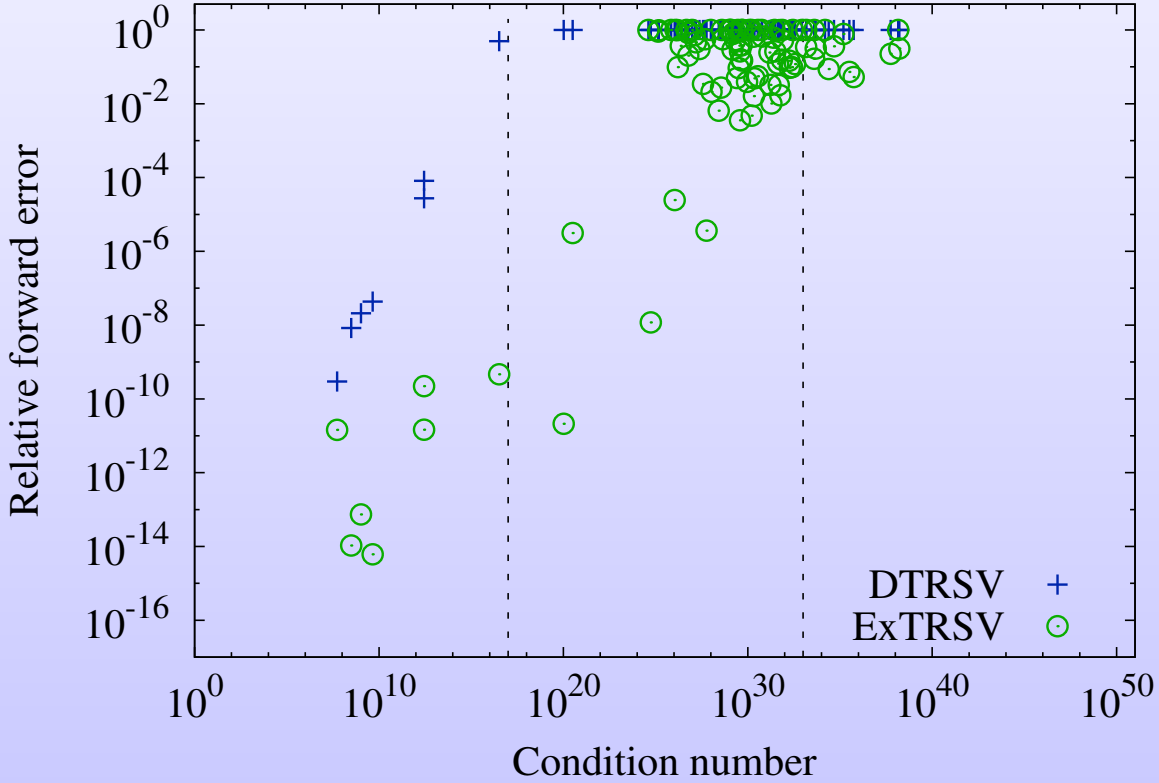


- Blocked `ExTRSV`
- Internal `ExGEMV`
- Based on `ExDOT`

Triangular Solver

Accuracy

$\boxed{\texttt{TRSV}\colon Ax = b}$



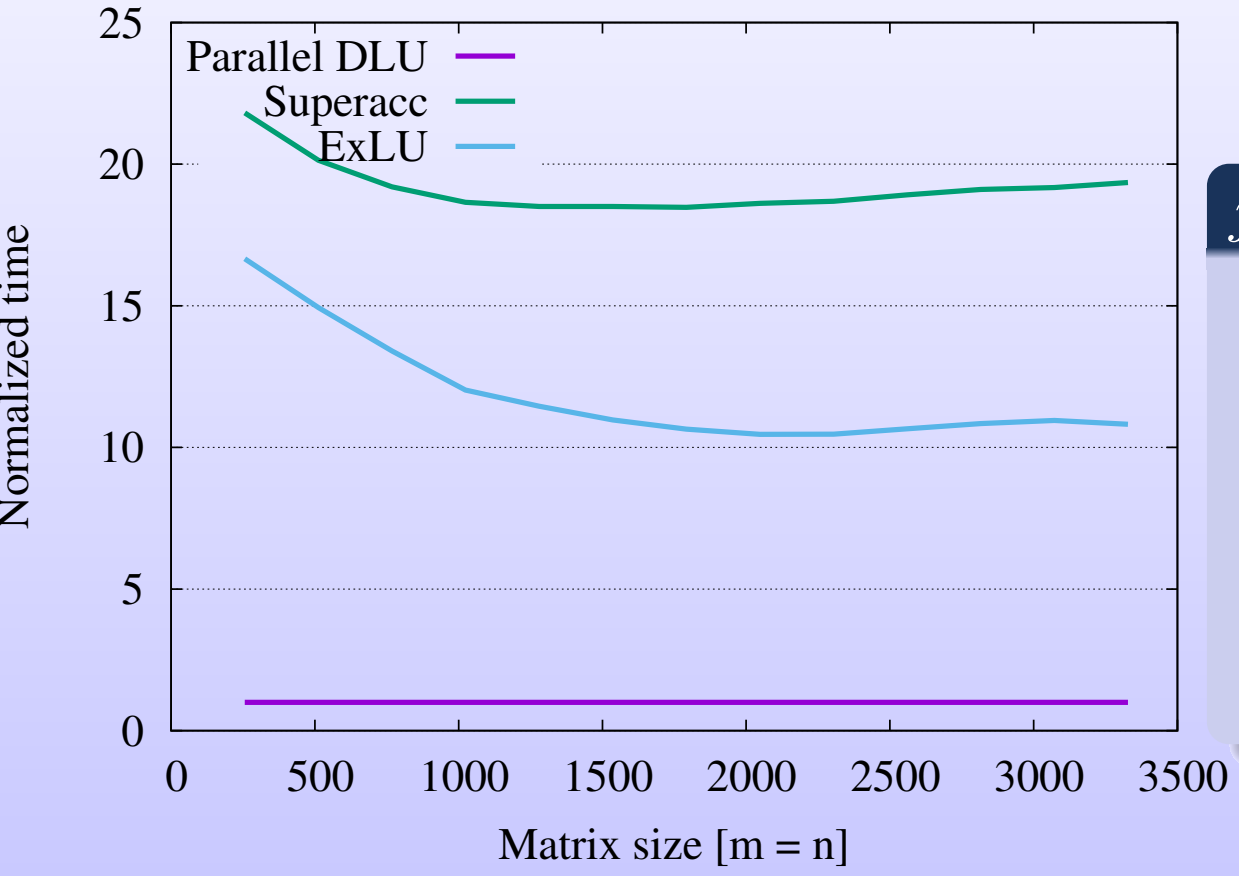
1: $x_1 \leftarrow fl(b_1/l_{11})$
2: **for** $i = 2 \rightarrow n$ **do**
3: $\quad s \leftarrow b_i$
4: $\quad$ **for** $j = 1 \rightarrow i-1$ **do**
5: $\quad\quad s \leftarrow s - l_{ij}x_j$
6: $\quad$ **end for**
7: $\quad x_i \leftarrow fl(RNDN(s)/l_{ii})$
8: **end for**

$$\mathrm{cond}(A, x) = \frac{\||A^{-1}||A||x|\|_\infty}{\|x\|_\infty}$$

LU Factorization (1/2)

Performance Scaling on NVIDIA Tesla K420

$$A = LU$$

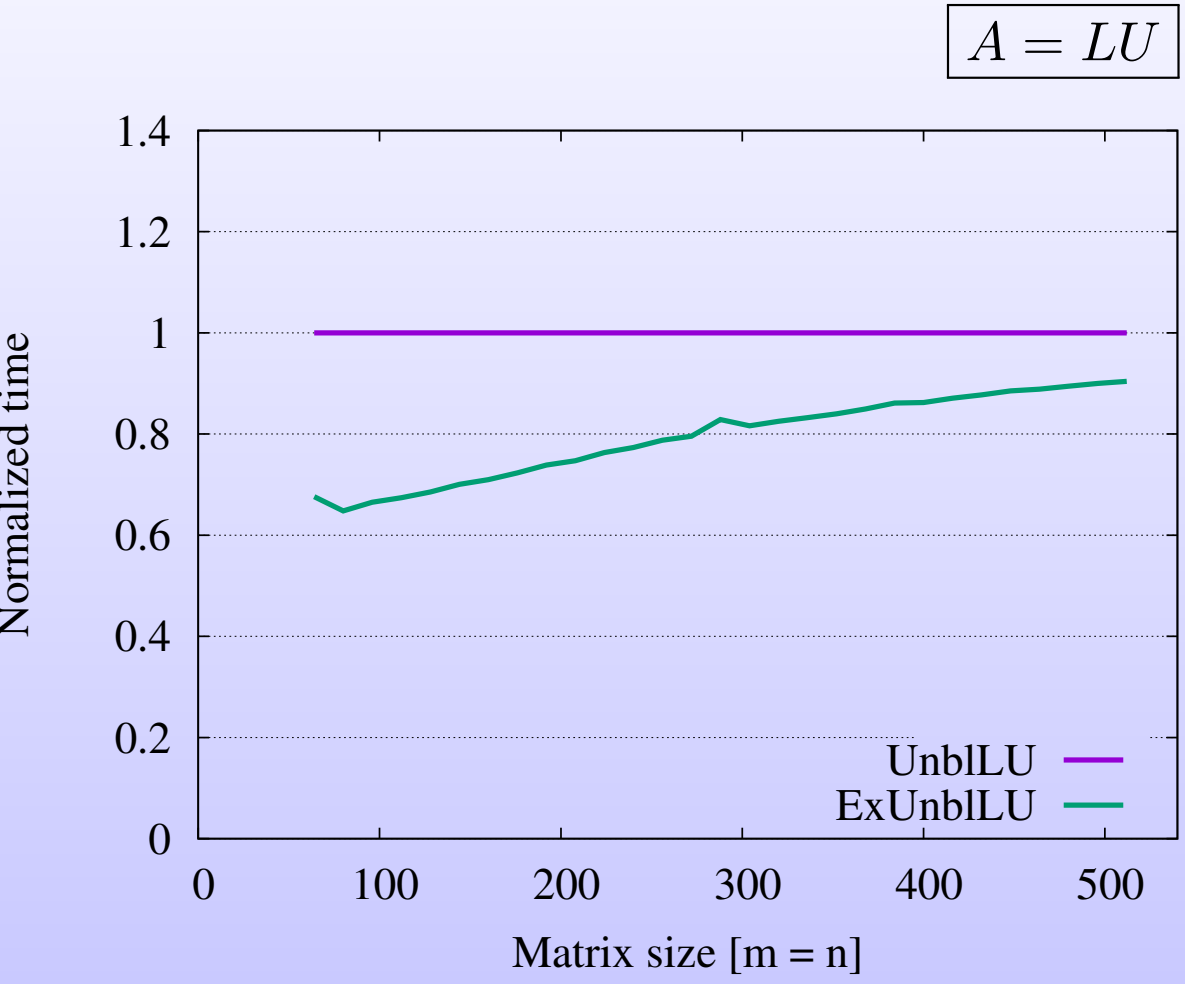


jik variant of LU

`swap()`	
$a_{01} \leftarrow L_{00}^{-1} a_{01}$	`trsv`
$\alpha_{11} \leftarrow \alpha_{11} - a_{10}^T a_{01}$	`dot`
$a_{21} \leftarrow a_{21} - A_{20} a_{01}$	`gemv`
`max()`	
`swap()`	
$a_{21} \leftarrow a_{21} / \alpha_{11}$	`scal`

LU Factorization (2/2)

Performance Scaling on NVIDIA Tesla K420



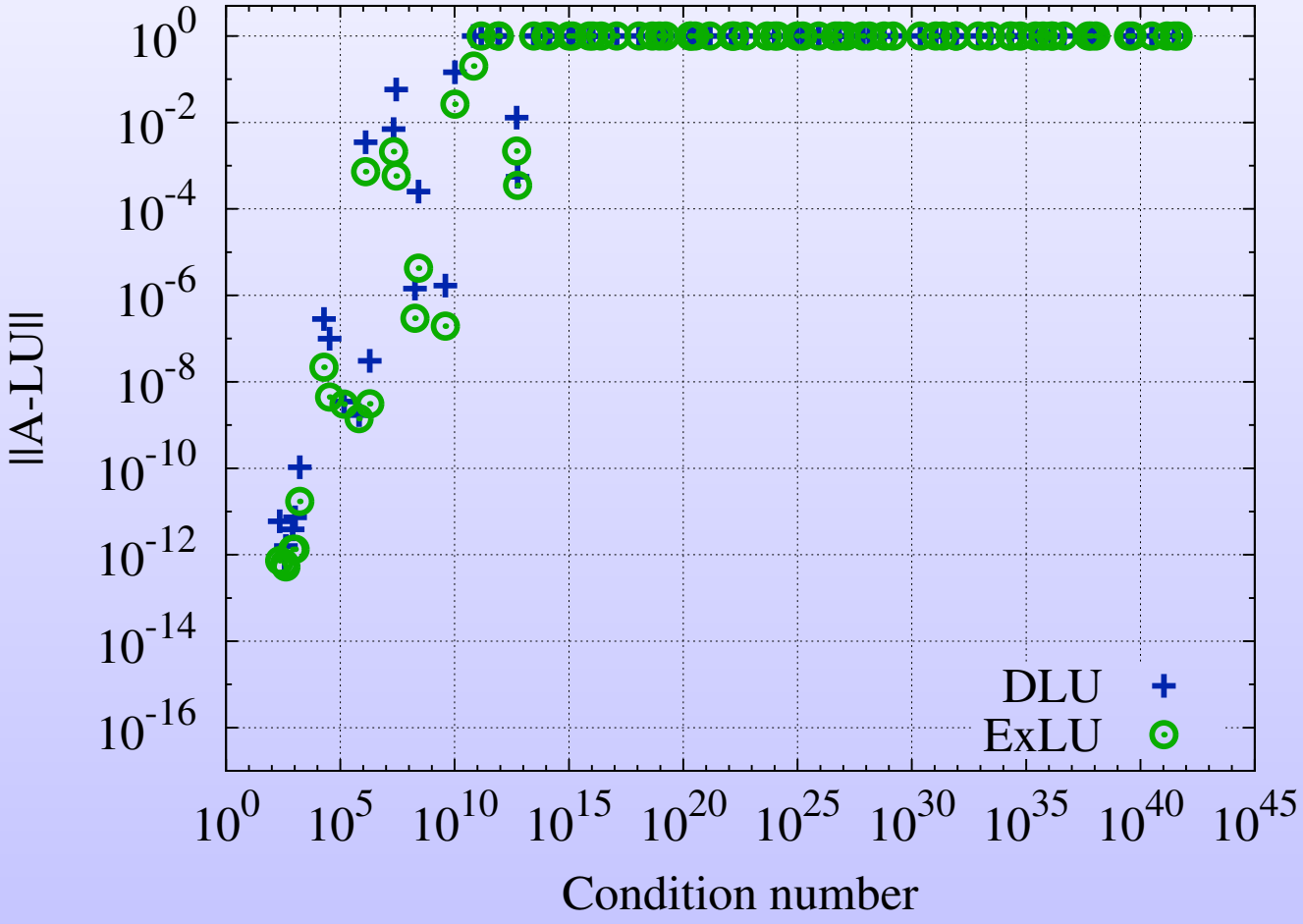
variant 5 of LU

`max()`	
`swap()`	
$a_{21} := a_{21}/\alpha_{11}$	`scal`
$A_{22} := A_{22} - a_{21}a_{12}^T$	`ger`

LU Factorization

Accuracy

$$A = LU$$



- Slightly better accuracy than DLU
- But, always reproducible

Outline

Conclusions and Future Work

Conclusions

- Leveraged a long accumulator and EFTs to design **reproducible and correctly-rounded** `ExSUM` **and** `ExDOT`
- Provided an extension to `InvSCAL` to perform division directly
- Developed a reproducible and accurate `ExGEMV`
- Derived two unblocked variants, e.g. jik, of the LU factorisation with partial pivoting
- Provide **bit-wise reproducible** results independently from
  - Data permutation, data assignment, partitioning/blocking
  - Thread scheduling
  - Reduction trees

Conclusions and Future Work

Conclusions

- Leveraged a long accumulator and EFTs to design **reproducible and correctly-rounded** `ExSUM` **and** `ExDOT`
- Provided an extension to `InvSCAL` to perform division directly
- Developed a reproducible and accurate `ExGEMV`
- Derived two unblocked variants, e.g. jik, of the LU factorisation with partial pivoting
- Provide **bit-wise reproducible** results independently from
  - Data permutation, data assignment, partitioning/blocking
  - Thread scheduling
  - Reduction trees

Future directions

- Reproducible solution of linear systems
- Application of our implementations in real-world codes

Thank you for your attention!

URL: `https://exblas.lip6.fr`

ExBLAS Status

- ExBLAS-1: `ExSUM`[a], `ExSCAL`, `ExDOT`, `ExAXPY`, ...
- ExBLAS-2: `ExGER`, `ExGEMV`, `ExTRSV`, `ExSYR`, ...
- ExBLAS-3: `ExGEMM`, `ExTRSM`, `ExSYR2K`, ...

[a] Routines in blue are already in ExBLAS