Hierarchical Approach for Deriving a Reproducible LU factorization

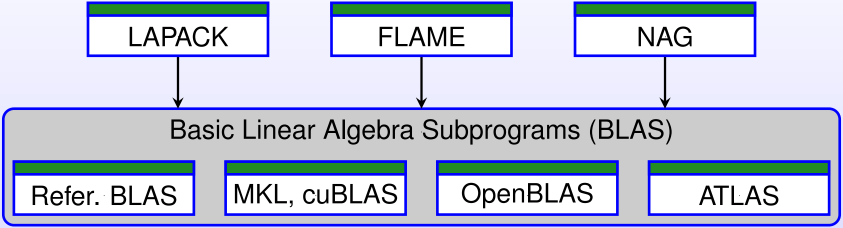
Roman Iakymchuk[1], David Defour[2], Stef Graillat[3], and Enrique S. Quitana Ortí[4]

[1]KTH Royal Institute of Technology, CSC, CST/PDC
[2]Université de Perpignan, DALI–LIRMM
[3]Sorbonne Universités, UPMC Univ Paris VI, UMR 7606, LIP6
[4]Universidad Jaime I, 12.071-Castellón, Spain

David.Defour@univ-perp.fr

NRE16 at SC16, Nov 18th, 2016
Salt Lake City, Utah, USA

Linear Algebra Libraries



$\Downarrow$

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]:	$C := C + AB$ $C := A^{-1}B$	$A, B, C \in \mathbb{R}^{n \times n}$	$n/2$

Goals

- To compute BLAS operations with floating-point numbers fast and precise, ensuring their numerical reproducibility, on a wide range of architectures

ExBLAS – **Ex**act **BLAS**

- **Ex**BLAS-1: `ExSUM, ExSCAL, ExDOT, ExAXPY, ...`
- **Ex**BLAS-2: `ExGER, ExGEMV, ExTRSV, ExSYR, ...`
- **Ex**BLAS-3: `ExGEMM, ExTRSM, ExSYR2K, ...`

Goals

- To compute BLAS operations with floating-point numbers fast and precise, ensuring their numerical reproducibility, on a wide range of architectures

ExBLAS – **Ex**act **BLAS**

- **Ex**BLAS-1: `ExSUM, ExSCAL, ExDOT, ExAXPY, ...`
- **Ex**BLAS-2: `ExGER, ExGEMV, ExTRSV, ExSYR, ...`
- **Ex**BLAS-3: `ExGEMM, ExTRSM, ExSYR2K, ...`

- Use the ExBLAS kernels to construct exact higher-level operations such as the LU factorization

Outline

Outline

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

Existing Solutions

- **Fix the Order of Computations**
  - Sequential mode: intolerably costly at large-scale systems
  - Fixed reduction trees: substantial communication overhead

$\rightarrow$ Example: Intel **C**onditional **N**umerical **R**eproducibility
($\sim 2x$ for datum, no accuracy guarantees)

Existing Solutions

- **Fix the Order of Computations**
  - Sequential mode: intolerably costly at large-scale systems
  - Fixed reduction trees: substantial communication overhead

  → Example: Intel **C**onditional **N**umerical **R**eproducibility
  ($\sim 2x$ for datum, no accuracy guarantees)

- **Eliminate/Reduce the Rounding Errors**
  - Fixed-point arithmetic: limited range of values
  - Fixed FP expansions with Error-Free Transformations (EFT)

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

  - "Infinite" precision: reproducible independently from the inputs

  → Example: Kulisch accumulator (considered inefficient)

Existing Solutions

- **Fix the Order of Computations**
  - Sequential mode: intolerably costly at large-scale systems
  - Fixed reduction trees: substantial communication overhead

  → Example: Intel **C**onditional **N**umerical **R**eproducibility ($\sim 2x$ for datum, no accuracy guarantees)

- **Eliminate/Reduce the Rounding Errors**
  - Fixed-point arithmetic: limited range of values
  - Fixed FP expansions with Error-Free Transformations (EFT)

  → Example: double-double or quad-double (Briggs, Bailey, Hida, Li) (work well on a set of relatively close numbers)
  - “Infinite” precision: reproducible independently from the inputs

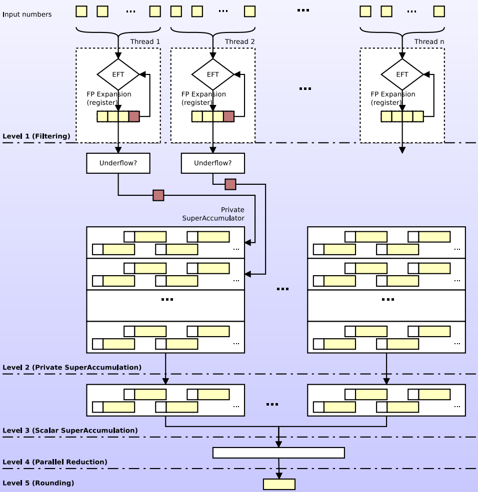
  → Example: Kulisch accumulator (considered inefficient)

- **Libraries**
  - ReproBLAS: Reproducible BLAS (Demmel, Nguyen, Ahrens)

  → For BLAS-1, GEMV, and GEMM on CPUs
  - RARE-BLAS: Repr. Accur. Rounded and Eff. BLAS (Chohra, Langlois, Parello) → For BLAS-1 and GEMV on CPUs

Outline

Our Multi-Level Reproducible Summation



- 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**

Outline

ExBLAS Highlights

ExBLAS Status

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

ExBLAS Highlights

ExBLAS Status

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

ExSCAL

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

ExBLAS Highlights

ExBLAS Status

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

ExSCAL

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

ExBLAS Highlights

ExBLAS Status

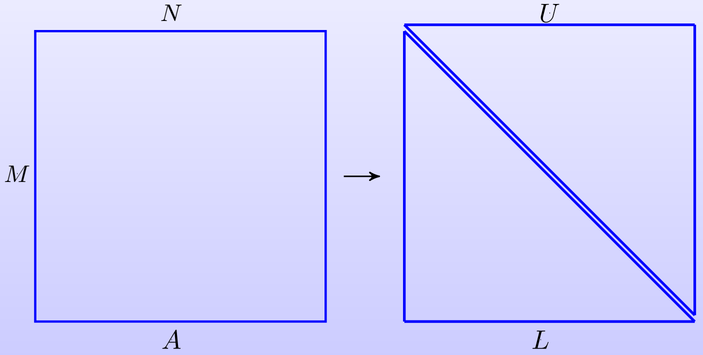
- ExBLAS-1: `ExSUM`, `ExSCAL`, `ExDOT`, `ExAXPY`, ...
- ExBLAS-2: `ExGER`, `ExGEMV`, `ExTRSV`, `ExSYR`, ...
- ExBLAS-3: `ExGEMM`, `ExTRSM`, `ExSYR2K`, ...

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

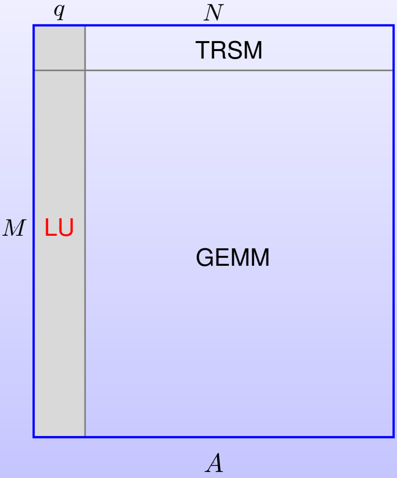
LU Factorization

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



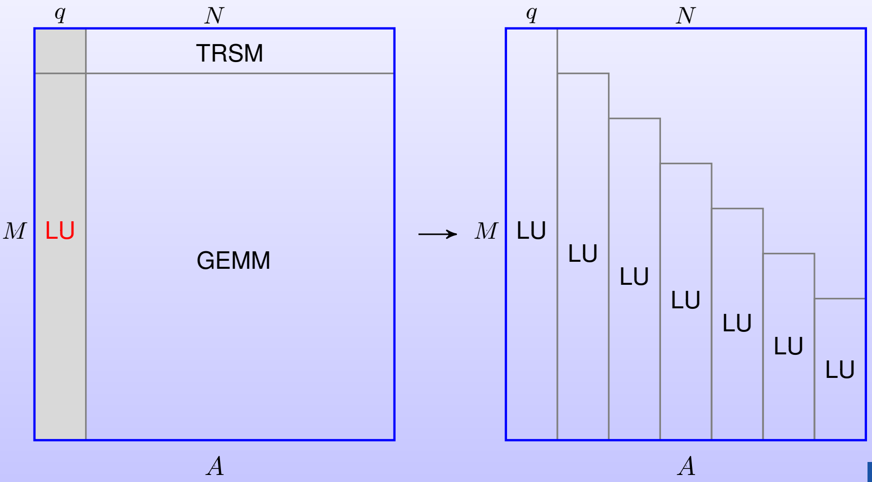
LU Factorization

$$\boxed{A = LU}$$



LU Factorization

$$\boxed{A = LU}$$



An unblocked LU Factorization

LU Factorization

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

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

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

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

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

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

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

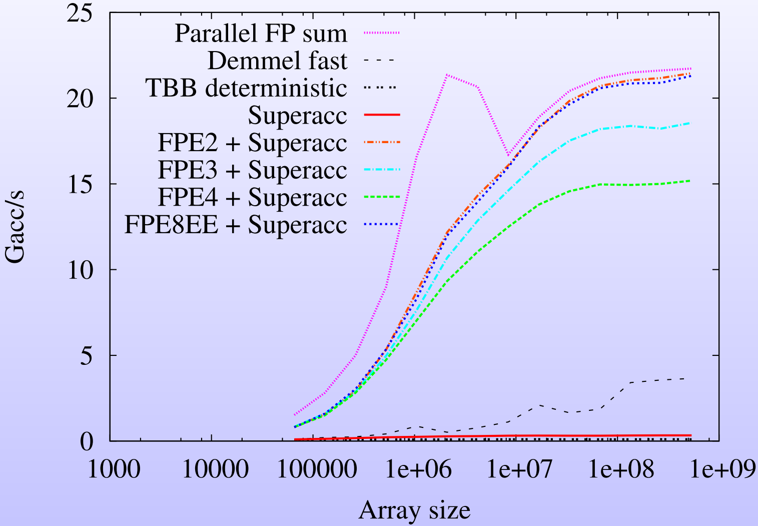
	i	1	p
i	A_{00}	a_{01}	A_{02}
1	a_{10}^T	α_{11}	a_{12}^T
p	A_{20}	a_{21}	A_{22}

3×3 partitioning of A

Outline

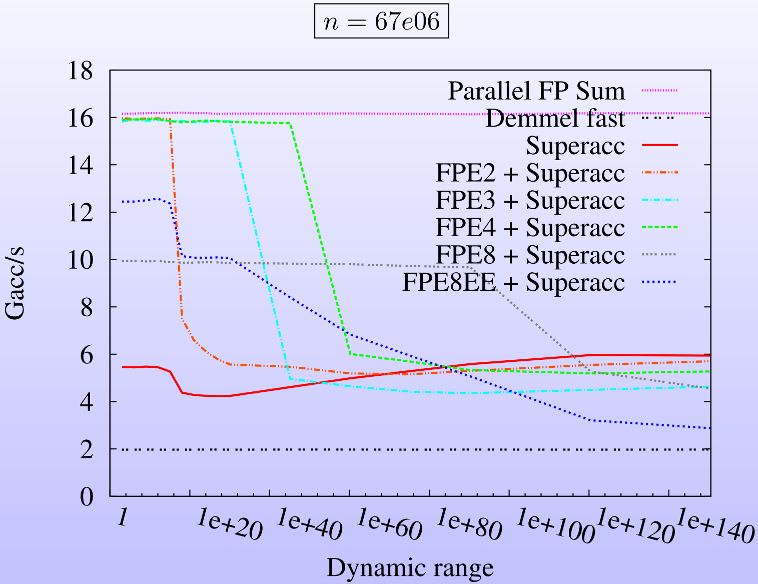
Parallel Reduction

Performance Scaling on Intel Xeon Phi



Parallel Reduction

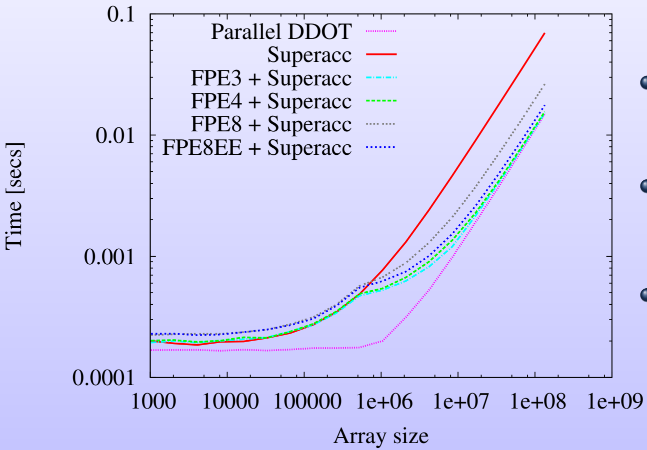
Data-Dependent Performance on NVIDIA Tesla K20c



Dot Product

Performance Scaling on NVIDIA Tesla K20c

DDOT: $\alpha := x^T y = \sum_i^N x_i y_i$

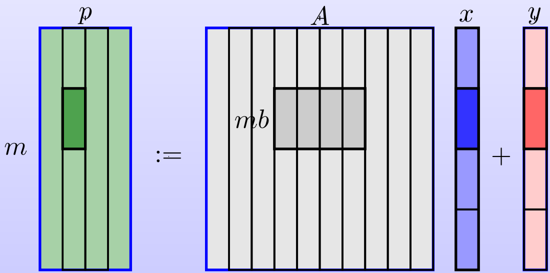


- Based on TwoProduct and Reproducible Summation
- TwoProduct(a, b)

  1: $r \leftarrow a * b$

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

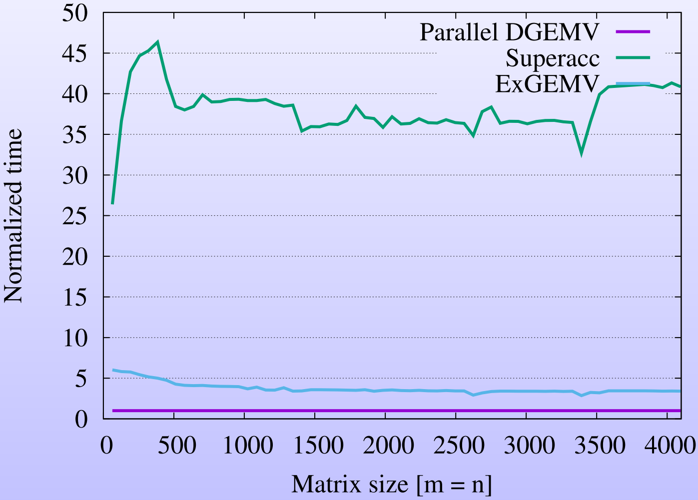
DGEMV: $y := \alpha A x + \beta y$



Matrix-Vector Product

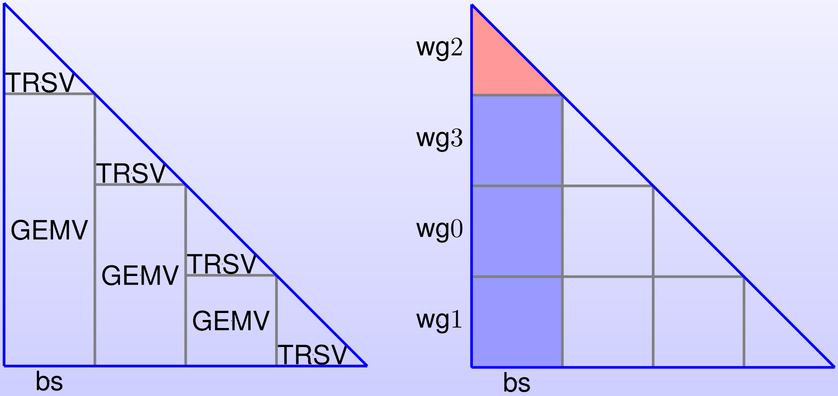
Performance Scaling on NVIDIA Tesla K80

DGEMV: $y := \alpha A x + \beta y$



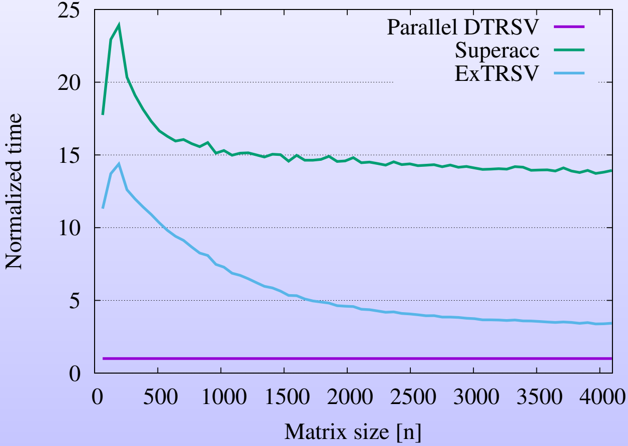
Triangular Solver

Matrix Partitioning



Partitioning of a lower triangular matrix L

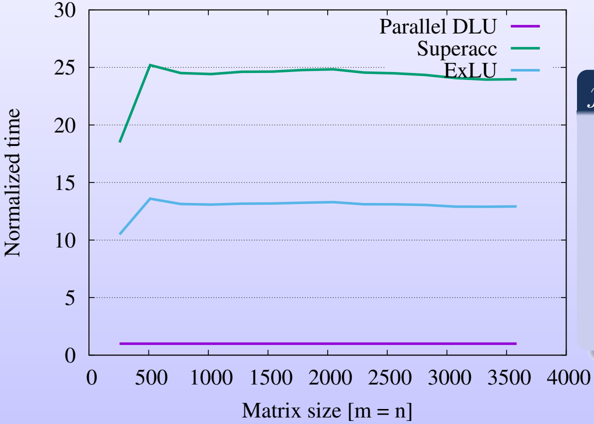
DTRSV: $Ax = b$



- Blocked ExTRSV
- Based on ExDOT
- Internal ExGEMV

Performance Scaling on NVIDIA Tesla K80

$A = LU$



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

Outline

Conclusions and Future Work

Conclusions

- Compute the results with no errors due to rounding
- Provide bit-wise reproducible results independently from
  - Data permutation, data assignment
  - Thread scheduling
  - Partitioning/blocking
  - Reduction trees

Conclusions and Future Work

Conclusions

- Compute the results with no errors due to rounding
- Provide bit-wise reproducible results independently from
  - Data permutation, data assignment
  - Thread scheduling
  - Partitioning/blocking
  - Reduction trees
- Deliver comparable performance to the classic implementations of the memory-bound operations

Conclusions and Future Work

Conclusions

- Compute the results with no errors due to rounding
- Provide bit-wise reproducible results independently from
  - Data permutation, data assignment
  - Thread scheduling
  - Partitioning/blocking
  - Reduction trees
- Deliver comparable performance to the classic implementations of the memory-bound operations
- Reproducible underlying kernels → reproducible LU

Conclusions and Future Work

Conclusions

- Compute the results with no errors due to rounding
- Provide bit-wise reproducible results independently from
  - Data permutation, data assignment
  - Thread scheduling
  - Partitioning/blocking
  - Reduction trees
- Deliver comparable performance to the classic implementations of the memory-bound operations
- Reproducible underlying kernels → reproducible LU

Future directions

- Enhance compute-intensive operations and the LU factorization
- Cover the other variants of the unblocked LU factorization
- 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