Reproducibility of Linear Algebra Operations

Roman Iakymchuk[1]

joint work with

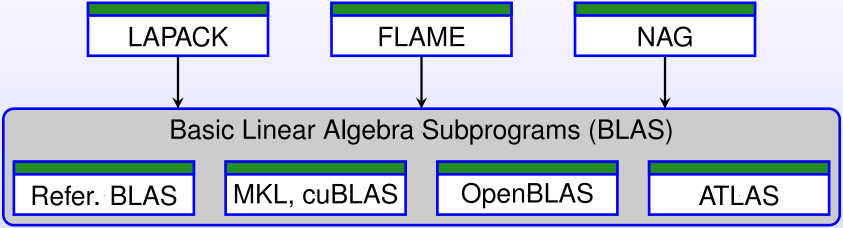
Sylvain Collange, David Defour, Erwin Laure,
Enrique S. Quitana Ortí and Stef Graillat

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

riakymch@kth.se

MS237: Algorithmic Revolution in Post Moore's Era:
Auto-Tuning and Accuracy Assurance
SIAM CSE17, Feb 27th - Mar 3rd, 2017, Atlanta, GA, USA

Linear Algebra Libraries



$\Downarrow$

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

Goals

- To ensure BLAS kernels yield precise and numerical reproducible results with comparable performance 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 matrix factorization

Outline

Outline

Computer Arithmetic

Problems

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

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

Computer Arithmetic

Problems

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

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

Computer Arithmetic

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

- 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)

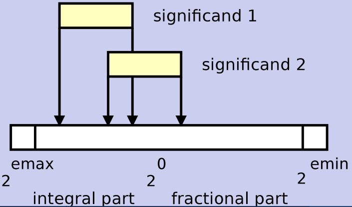
Algorithm 1 FPE of size 2 (Dekker and Knuth)

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

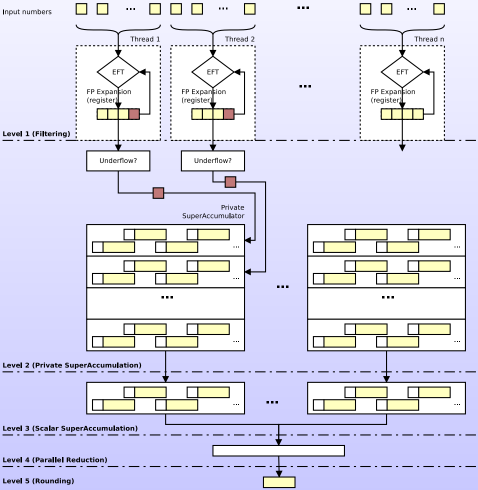
1: $r \leftarrow a + b$
2: $z \leftarrow r - a$
3: $s \leftarrow (a - (r - z)) + (b - z)$

- "Infinite" precision: reproducible independently from the inputs

→ Example: Kulisch accumulator (considered inefficient)



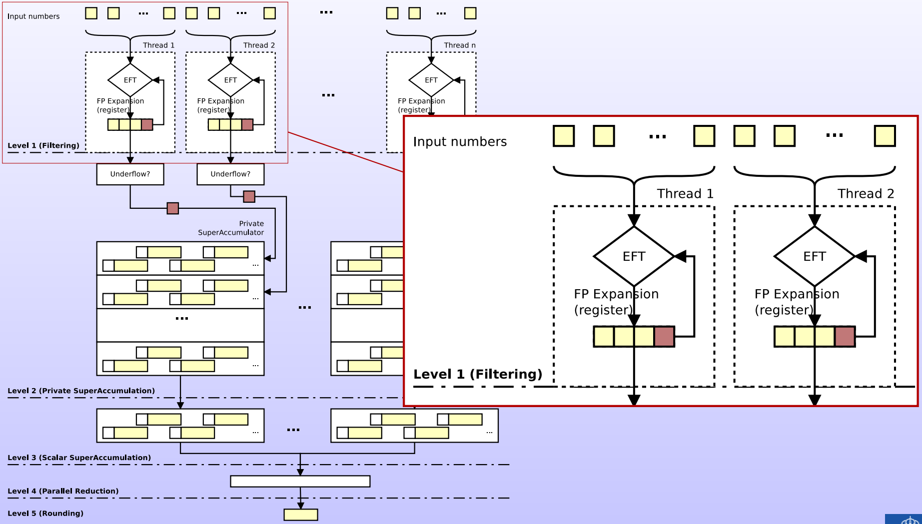
Exact Multi-Level Parallel Reduction



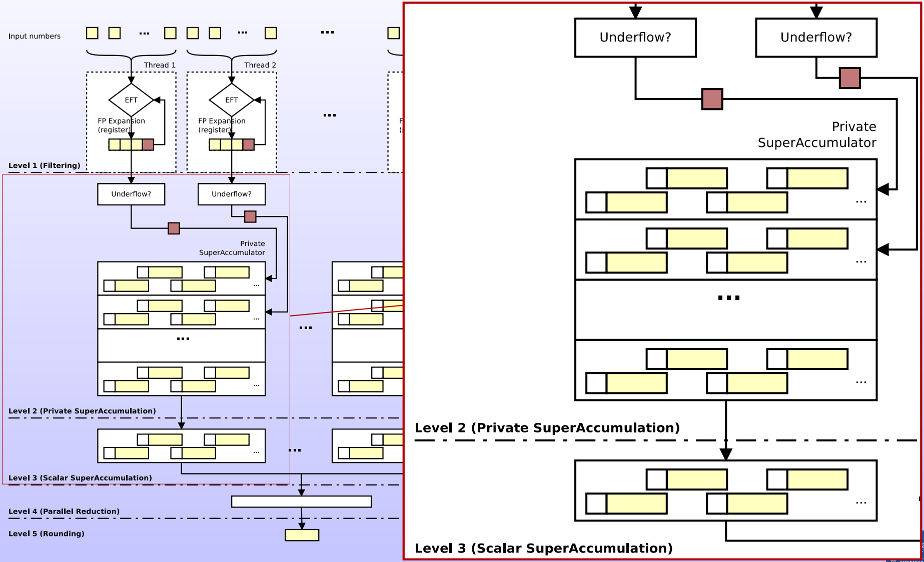
- Parallel algorithm with 5-levels
- Suitable for today's parallel architectures
- Based on FPE with EFT and Kulisch accumulator
- Guarantees "inf" precision

→ **bit-wise reproducibility**

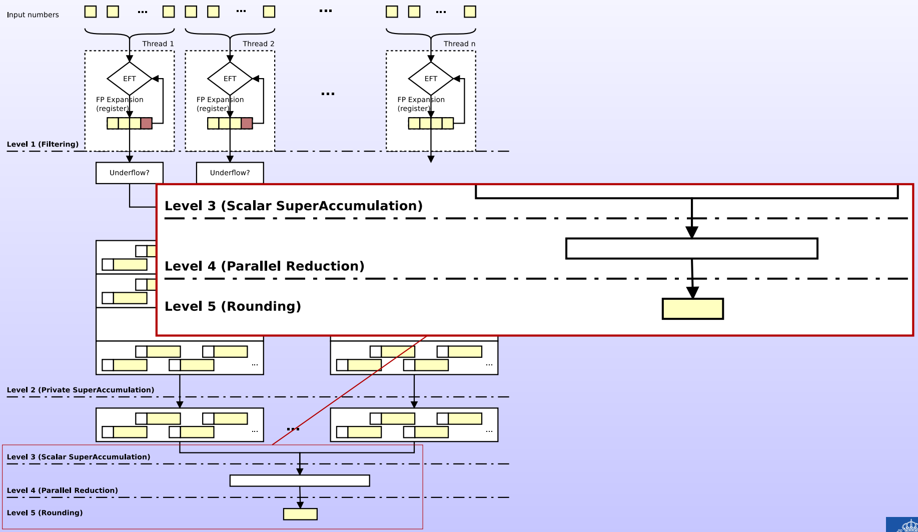
Level 1: Filtering



Level 2 and 3: Scalar Superaccumulator



Level 4 and 5: Reduction and Rounding



Outline

ExBLAS Highlights

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

ExSCAL

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

ExBLAS Highlights

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

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

[a] Routines in blue are already in ExBLAS

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

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

ExSCAL

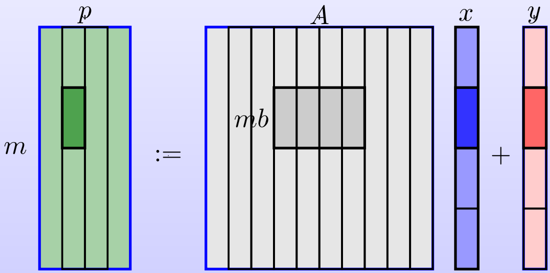
- $x := \alpha * x \rightarrow$ correctly rounded and reproducible
- Within LU: $x := 1/\alpha * x \rightarrow$ not correctly rounded
- ExInvSCAL: $x := x/\alpha \rightarrow$ correctly rounded and reproducible

ExGER

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

Matrix-Vector Product

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

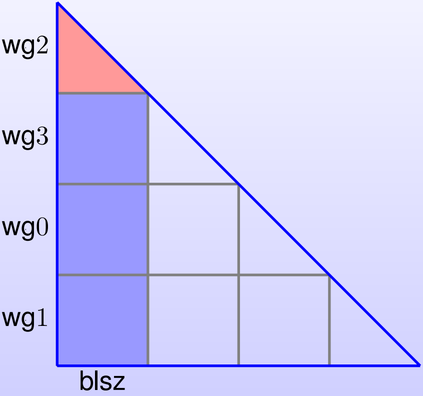


- Based on ExDOT
- `TwoProd`(a, b)

  1: $r \leftarrow a * b$

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

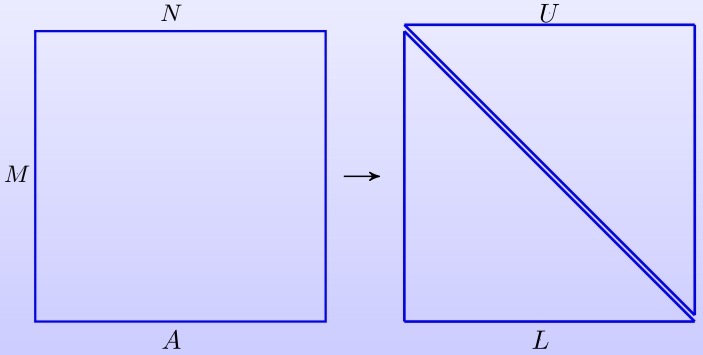
for $i = 0 : blsz : n$ **do**
  for $k = i : i + blsz$ **do**
    for $j = 1 : k - 1$ **do**
      $[r, e] \leftarrow TwoProd(l_{kj}, -x_j)$
      $ExpansionAccumulate(r)$
      $ExpansionAccumulate(e)$
    end for
    $ExpansionAccumulate(b_k)$
    $\widehat{s} \leftarrow RNDN(acc(k))$
    $x_k \leftarrow \widehat{s}/l_{kk}$
  end for
  for $k = i + blsz : n$ **do**
    for $j = i : i + blsz$ **do**
      $[r, e] \leftarrow TwoProd(l_{kj}, -x_j)$
      $ExpansionAccumulate(r)$
      $ExpansionAccumulate(e)$
    end for
  end for
end for



Partitioning of a lower triangular matrix L

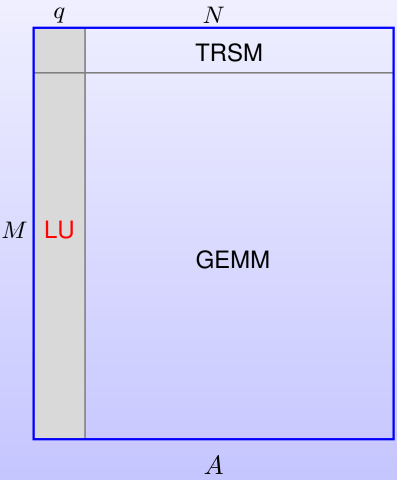
LU Factorization

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



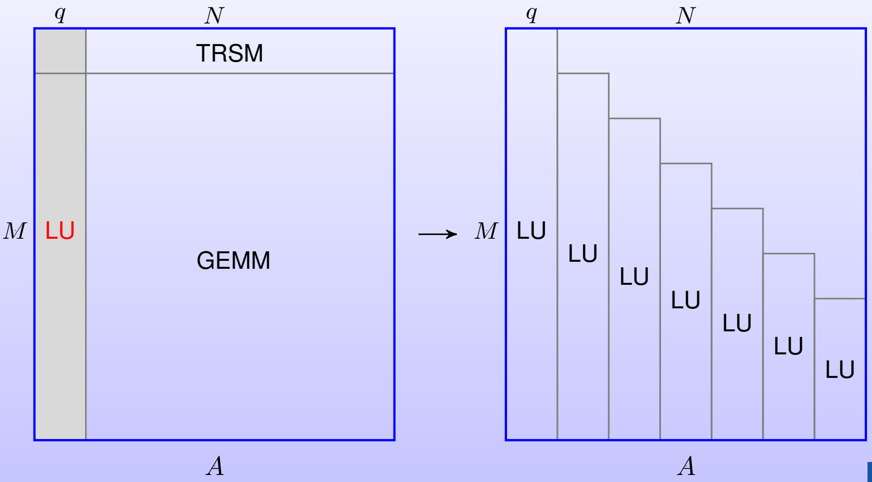
LU Factorization

$$\boxed{A = LU}$$



LU Factorization

$$A = LU$$



An unblocked LU Factorization

LU Factorization

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

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

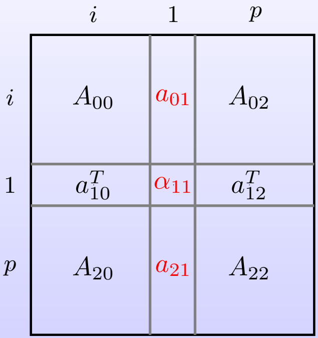
$$\alpha_{11} := \alpha_{11} - a_{10}^T a_{01} \quad \textbf{(dot)}$$

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

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

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

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

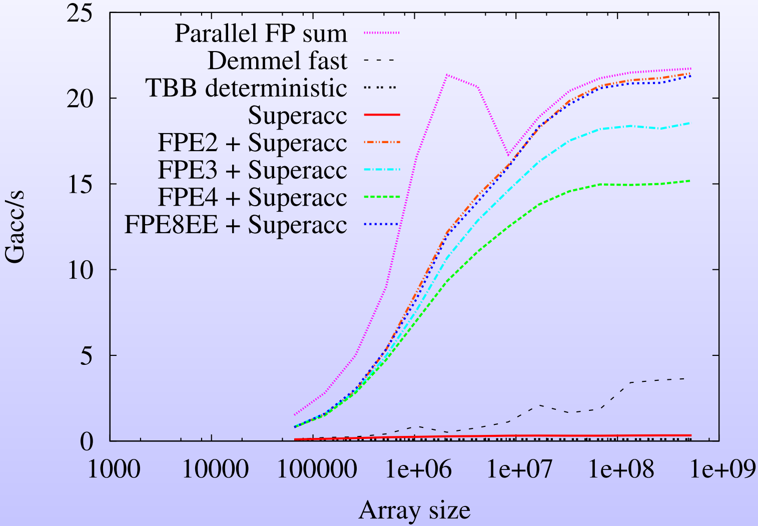


3×3 partitioning of A

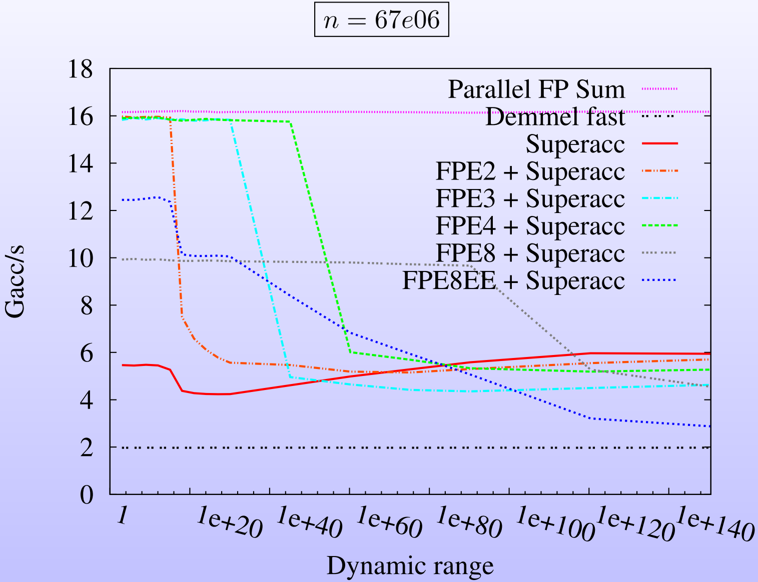
Outline

Parallel Reduction

Performance Scaling on Intel Xeon Phi



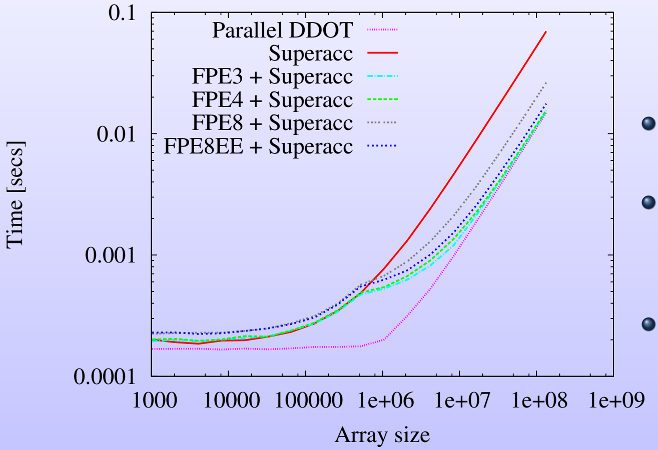
Data-Dependent Performance on NVIDIA Tesla K20c



Dot Product

Performance Scaling on NVIDIA Tesla K20c

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

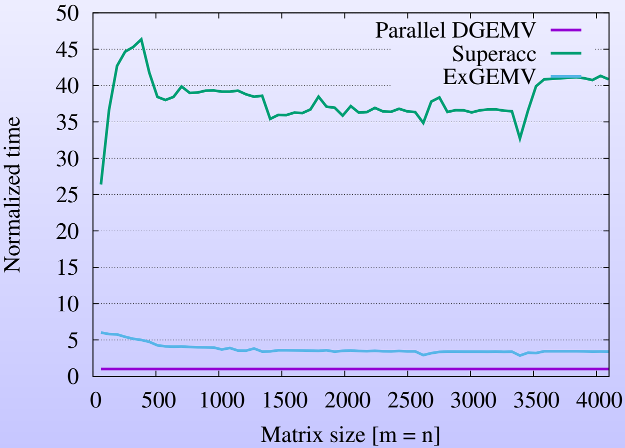


- Based on `TwoProd` and ExSUM
- `TwoProd`(a, b)
  1: $r \leftarrow a * b$
  2: $s \leftarrow fma(a, b, -r)$
- $fma(a, b, c) = a * b + c$

Matrix-Vector Product

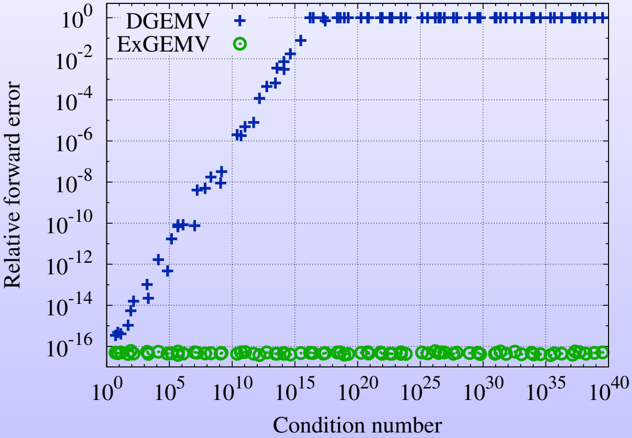
Performance Scaling on NVIDIA Tesla K80

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



- Blocked ExGEMV
- Based on ExDOT

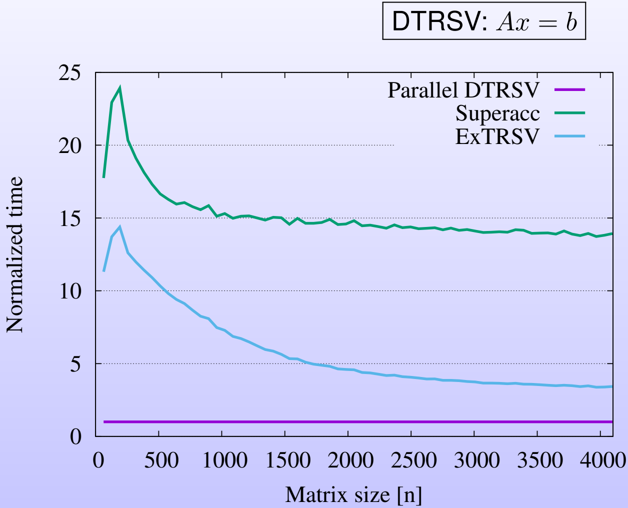
GEMV: $y := Ax$



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

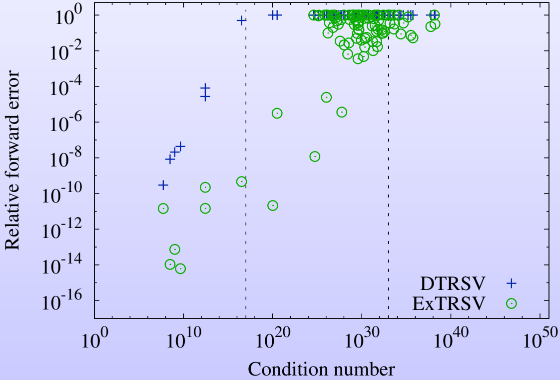
Triangular Solver

Performance Scaling on NVIDIA Tesla K420



- Blocked ExTRSV
- Internal ExGEMV
- Based on ExDOT

TRSV: $Ax = b$



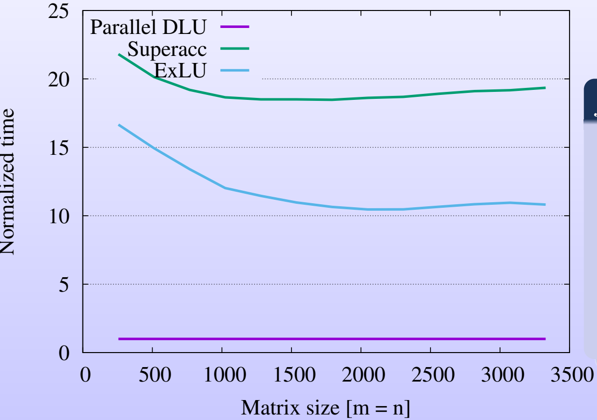
1: $x_1 \leftarrow fl(b_1/l_{11})$
2: **for** $i = 2 \to n$ **do**
3: $\quad s \leftarrow b_i$
4: $\quad$ **for** $j = 1 \to i-1$ **do**
5: $\quad\quad s \leftarrow s - l_{ij}x_j$
6: $\quad$ **end for**
7: $\quad x_i \leftarrow fl(RNDN(s)/l_{ii})$
8: **end for**

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

LU Factorization

Performance Scaling on NVIDIA Tesla K420

$A = LU$



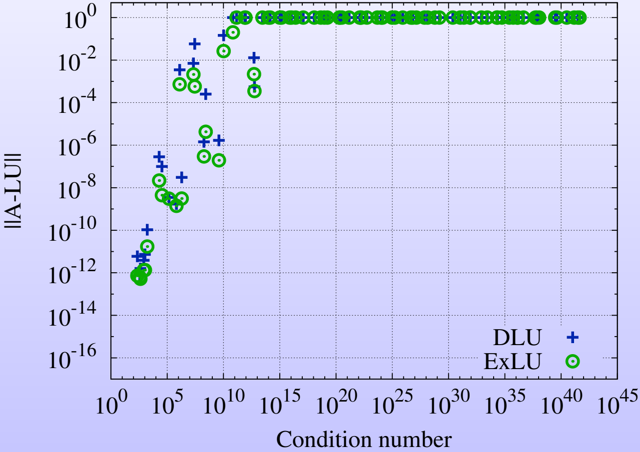
jik variant of LU

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

Accuracy

$A = LU$



- Slightly better accuracy than DLU
- But, always reproducible

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
  - Warps/threads 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
  - Warps/threads 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
  - Warps/threads scheduling
  - Partitioning/blocking
  - Reduction trees
- Deliver comparable performance to the classic implementations of the memory-bound operations
- Reproducible underlying kernels $\rightarrow$ 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
  - Warps/threads scheduling
  - Partitioning/blocking
  - Reduction trees
- Deliver comparable performance to the classic implementations of the memory-bound operations
- Reproducible underlying kernels $\rightarrow$ reproducible LU

Future directions

- Lightweight approach for compute-intensive operations
- Performance portability
- Applicability in real-world codes

Thank you for your attention!

https://exblas.lip6.fr

ExBLAS

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

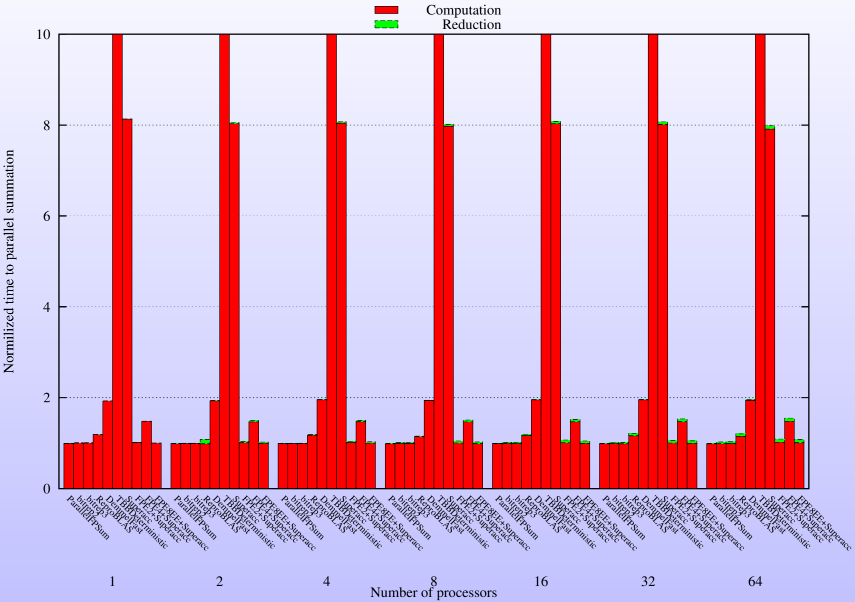
[a] Routines in blue are already in ExBLAS

Higher Level Operations

- Unblocked LU factorization variants, including GER+SCAL
- Towards blocked LU factorization

Parallel Summation with MPI

Performance Scaling on Mesu cluster; $n = 16e06$



Efforts on Reproducibility

Standardalization

- IEEE 754-2018: `TwoSum` and `TwoProd`
- Jim Demmel et. al.: "A Proposal for a Next-Generation BLAS"

Journals

- TOMS: Replicated Computational Results

Conferences, Workshops, and Minisymposiums

- ARITH 2016-17: Arithmetic challenges in HPC and exascale computing (accuracy, reproducibility, ...)
- SC 2015-16: Workshop on Numerical Reproducibility at Exascale
- IPDPS 2017 and Euro-par 2014-16: Workshop on Reproducibility in Parallel Computing
- SIAM PP 2016: Numerical Reproducibility for High-Performance Computing
- SIAM CSE 2017: Algorithmic evolution in Post Moore's Era: Auto-Tuning and Accuracy Assurance