ExBLAS: Reproducible and Accurate BLAS Library

Roman Iakymchuk[1,2], Sylvain Collange[3], David Defour[4], and Stef Graillat[1]

[1] Sorbonne Universités, UPMC Univ Paris VI, UMR 7606, LIP6
[2] Sorbonne Universités, UPMC Univ Paris VI, ICS
[3] INRIA – Centre de recherche Rennes – Bretagne Atlantique
[4] DALI–LIRMM, Université de Perpignan

`roman.iakymchuk@lip6.fr`

NRE2015 at SC15, November 20, 2015
Austin, TX, USA

Motivation

2015 Petascale: we able to perform 33.86 petaflops

Motivation

2015 Petascale: we able to perform 33.86 petaflops

2017 Petascale: we plan to perform $100 - 200$ petaflops

Motivation

2015 Petascale: we able to perform 33.86 petaflops

2017 Petascale: we plan to perform $100 - 200$ petaflops

2020 Exascale: we aim to perform exaflops (10^{18} flops)

Motivation

2015 Petascale: we able to perform 33.86 petaflops

2017 Petascale: we plan to perform $100 - 200$ petaflops

2020 Exascale: we aim to perform exaflops (10^{18} flops)

$$\Downarrow$$

$\boxed{10^{18}}$ round-off errors per second

Goal

- 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`, ...

Outline

Problems

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

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

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

Performance-optimized floating-point libraries are prone to non-reproducibility for various reasons:

- Changing Data Layouts:
  - Data partitioning
  - Data alignment

Sources of Non-Reproducibility

Performance-optimized floating-point libraries are prone to non-reproducibility for various reasons:

- 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)
  - Cache line size
  - Number of processors
  - Network topology

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 (slow, no accuracy guarantees)

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 (slow, no accuracy guarantees)

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

  $\rightarrow$ 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

  $\rightarrow$ 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 (slow, no accuracy guarantees)

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

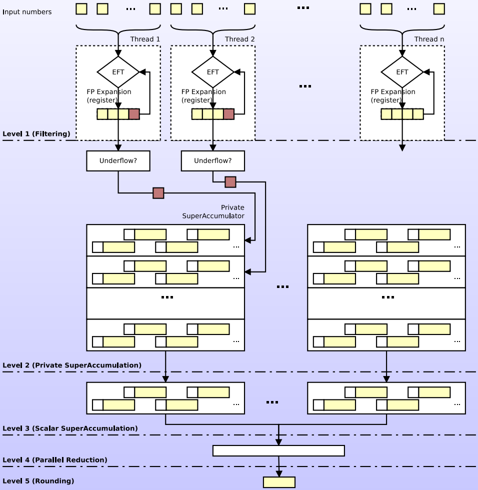
  → Example: double-double or quad-double (Briggs, Bailey, Hida, Li) (work well on a set of relatively close numbers)

  - “Infinite” precision: reproducible independently from the inputs

  → Example: Kulisch accumulator (considered inefficient)

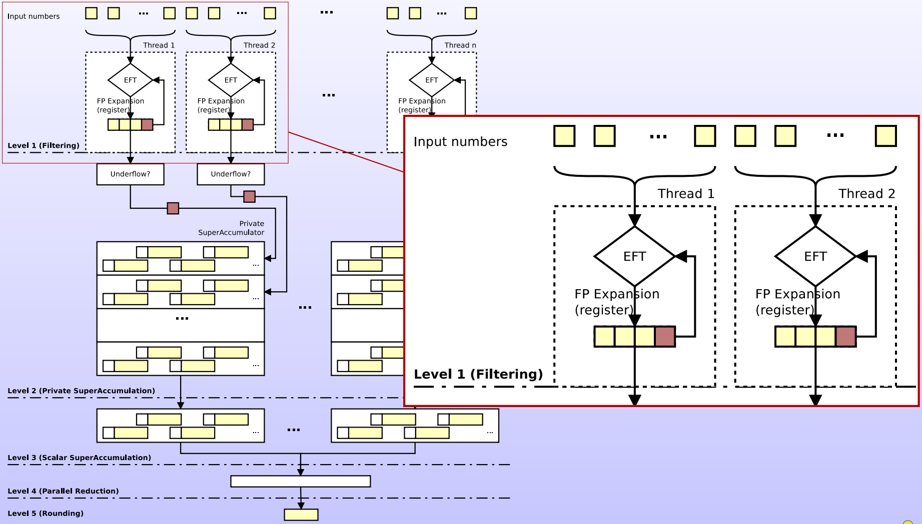
- **Libraries**
  - ReproBLAS: Reproducible BLAS (Demmel and Nguyen)

  → For BLAS-1 on CPUs only



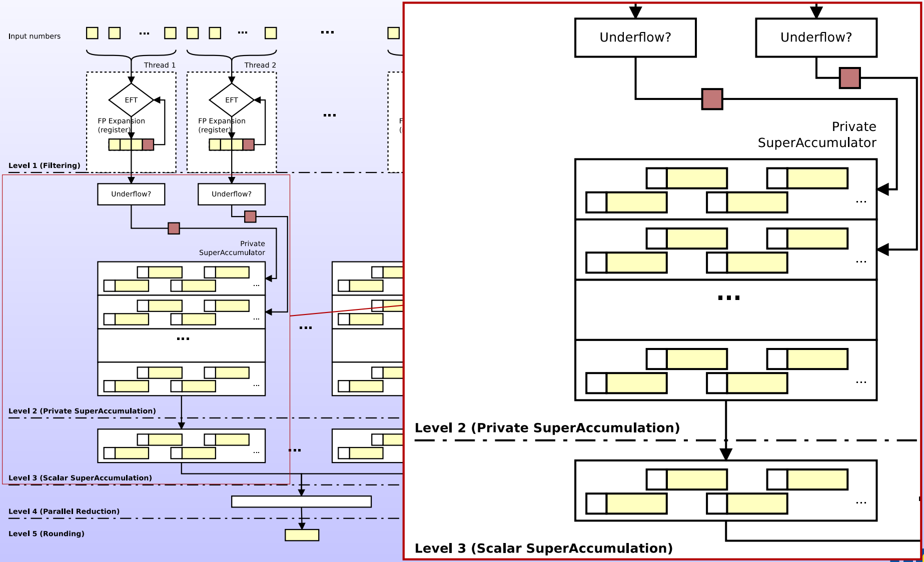
- Parallel algorithm with 5-levels
- Suitable for today's parallel architectures
- Based on FPE with EFT and Kulisch accumulator
- Guarantees "inf" precision

→ **bit-wise reproductibility**



Input numbers
Thread 1
Thread 2
Thread n
EFT
FP Expansion (register)
Level 1 (Filtering)
Underflow?
Private SuperAccumulator
Level 2 (Private SuperAccumulation)
Level 3 (Scalar SuperAccumulation)
Level 4 (Parallel Reduction)
Level 5 (Rounding)

Level 2 and 3: Scalar Superaccumulator



Level 4 and 5: Reduction and Rounding

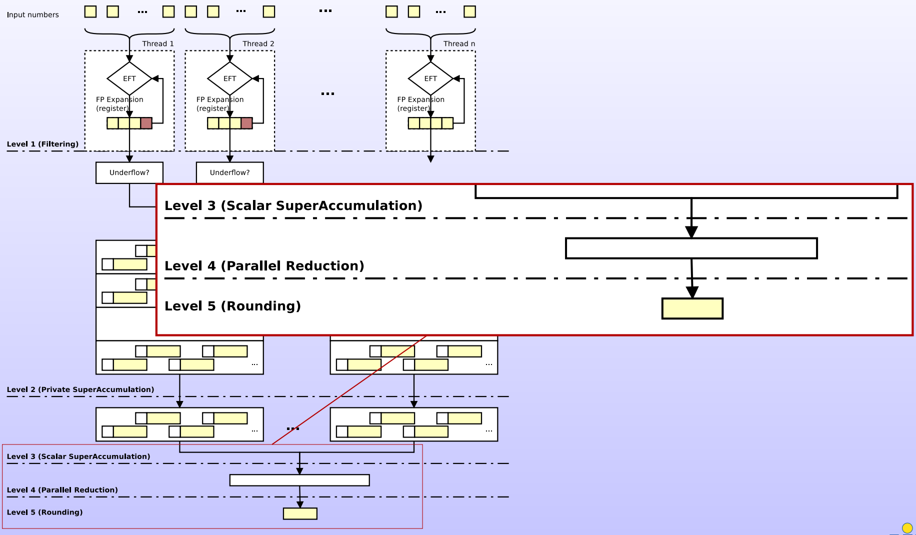
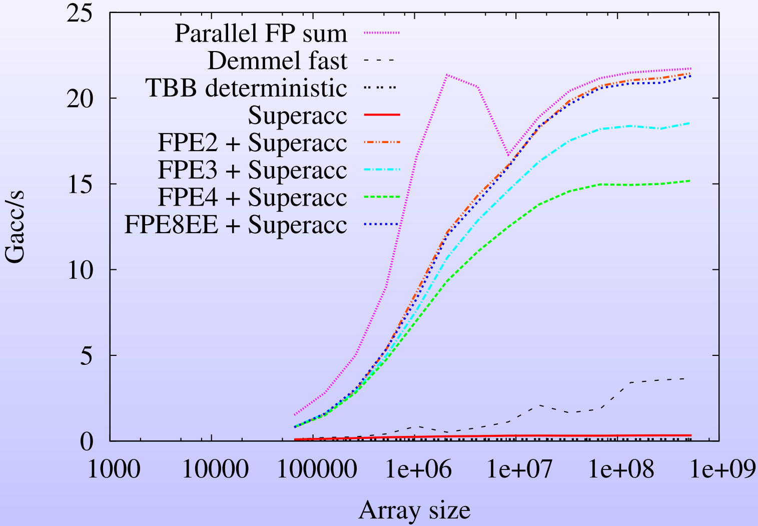


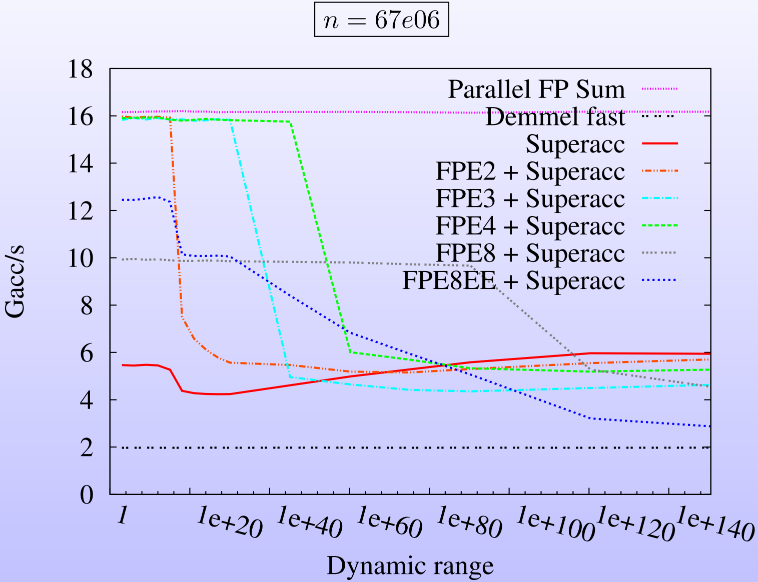
Table : Hardware platforms employed in the experimental evaluation

Intel Core i7-4770 (Haswell)	4 cores with HT
Mesu cluster (Intel Sandy Bridge)	$64 \times 2 \times 8$ cores
Intel Xeon Phi 3110P	60 cores $\times$ 4-way MT
NVIDIA Tesla K20c	13 SMs $\times$ 192 CUDA cores
NVIDIA Quadro K5000	8 SMs $\times$ 192 CUDA cores
AMD Radeon HD 7970	32 CUs $\times$ 64 units

Performance Scaling on Intel Xeon Phi



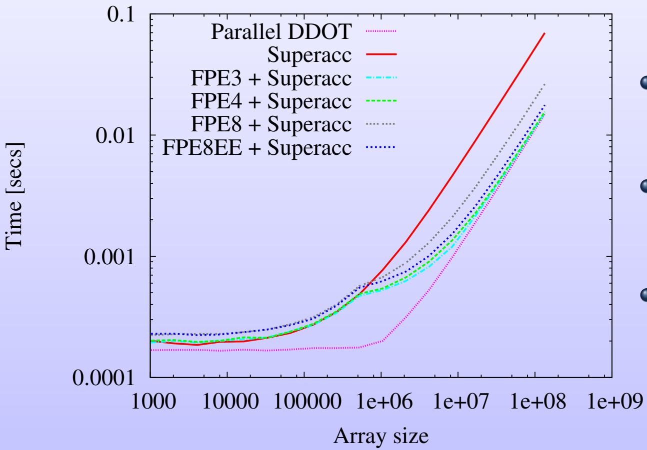
Data-Dependent Performance on NVIDIA Tesla K20c



Dot Product

Performance Scaling on NVIDIA Tesla K20c

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

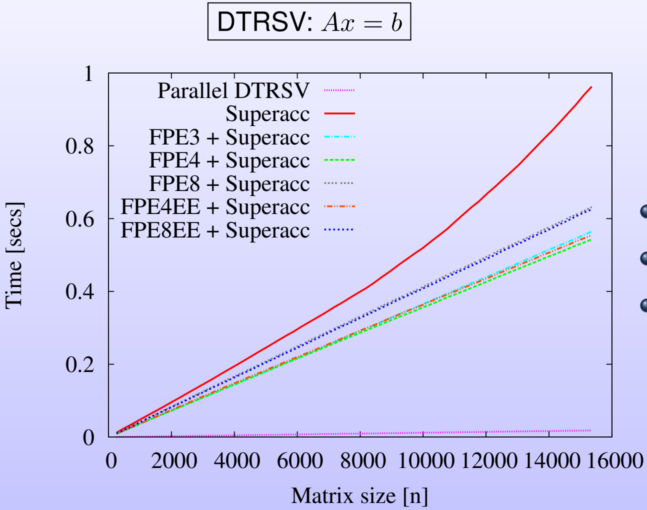


- Based on TwoProduct and Reproducible Summation
- TwoProduct(a, b)

  1: $r \leftarrow a * b$

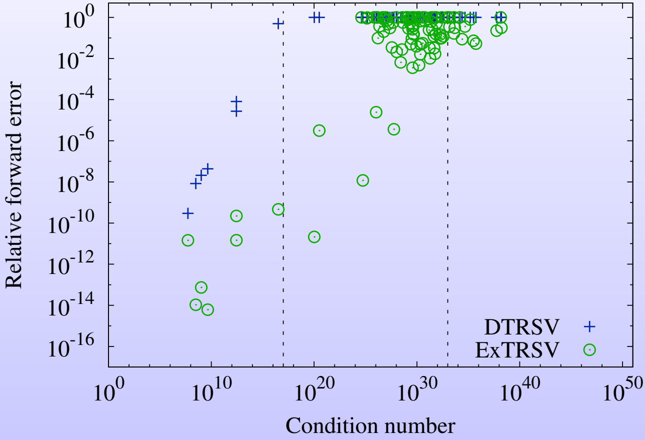
  2: $s \leftarrow fma(a, b, -r)$
- $fma(a, b, c) = a * b + c$

Performance Scaling on NVIDIA Tesla K20c



- Blocked ExTRSV
- Based on ExDOT
- Internal ExGEMV

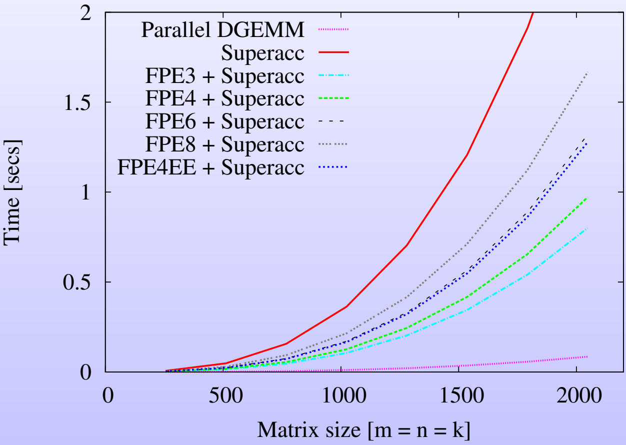
DTRSV: $Ax = b$



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

- Round superaccs for each element of the solution
- Saturated by division

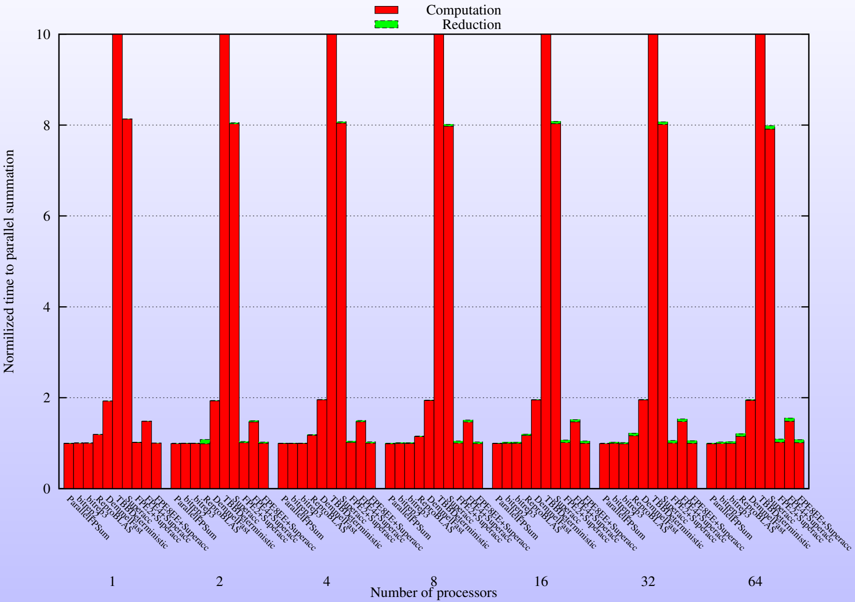
DGEMM: $C := \alpha AB + \beta C$



- Extensive usage of memory $\rightarrow$ lower performance

Parallel Summation with MPI

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



Conclusions and Future Work

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

Conclusions and Future Work

- Compute the results with no errors due to rounding
- Provide bit-wise reproducible results independently from
  - Data permutation, data assignment
  - Thread scheduling
- Deliver comparable performance to the classic implementations
- Perfect scaling with the increase of the problem size or the number of cores

Conclusions and Future Work

- Compute the results with no errors due to rounding
- Provide bit-wise reproducible results independently from
  - Data permutation, data assignment
  - Thread scheduling
- Deliver comparable performance to the classic implementations
- Perfect scaling with the increase of the problem size or the number of cores
- The ExTRSV and ExGEMM performance needs to be enhanced

Conclusions and Future Work

- Compute the results with no errors due to rounding
- Provide bit-wise reproducible results independently from
  - Data permutation, data assignment
  - Thread scheduling
- Deliver comparable performance to the classic implementations
- Perfect scaling with the increase of the problem size or the number of cores
- The ExTRSV and ExGEMM performance needs to be enhanced

ExBLAS – **Ex**act **BLAS**

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

Acknowledgement

This work undertaken (partially) in the framework of CALSIMLAB is supported by the public grant ANR-11-LABX-0037-01 and used the HPC resources of ICS funded by Region Île-de-France and the project Equip@Meso (ANR-10-EQPX-29-01) both overseen by the French National Research Agency (ANR) as part of the "Investissements d'Avenir" program (ANR-11-IDEX-0004-02)

This work was partially supported by the AllScale project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671603

This work was partially supported by the INTERTWinE project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671602

Thank you for your attention!

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

ExBLAS -- Exact BLAS

Main / HomePage

MENU

ACTIONS

- View
- Edit
- History
- Print

SEARCH

Find

About ExBLAS

ExBLAS stands for Exact (fast, accurate, and reproducible) Basic Linear Algebra Subprograms.

The increasing power of current computers enables one to solve more and more complex problems. This, therefore, requires to perform a high number of floating-point operations, each one leading to a round-off error. Because of round-off error propagation, some problems must be solved with a longer floating-point format.

As Exascale computing is likely to be reached within a decade, getting accurate results in floating-point arithmetic on such computers will be a challenge. However, another challenge will be the reproducibility of the results -- meaning getting a bitwise identical floating-point result from multiple runs of the same code -- due to non-associativity of floating-point operations and dynamic scheduling on parallel computers.

ExBLAS aims at providing new algorithms and implementations for fundamental linear algebra operations -- like those included in the BLAS library -- that deliver reproducible and accurate results with small or without losses to their performance on modern parallel architectures such as Intel Xeon Phi many-core processors and GPU accelerators. We construct our approach in such a way that it is independent from data partitioning, order of computations, thread scheduling, or reduction tree schemes.