Reproducible Triangular Solvers for High-Performance Computing

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

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

stef.graillat@upmc.fr

Special Track on Wavelets and Validated Numerics

12th International Conference on
Information Technology: New Generations (ITNG 2015)
Las Vegas, Nevada, USA, April 13-15, 2015

Motivation and Goal

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

Motivation and Goal

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

- To compute BLAS operations with floating-point numbers efficiently and with the best possible accuracy on a wide range of architectures

ExBLAS – **Ex**act **BLAS**

- ExBLAS-1: `ExSCAL, ExDOT, ExAXPY, ...`
- ExBLAS-2: `ExGER, ExGEMV, ExTRSV, ExSYR, ...`
- ExBLAS-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

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

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 at ExaScale (1/2)

- **Reproducibility** – ability to obtain bit-wise identical results from run-to-run on the same input data on the same or different architectures
- **ExaScale** – ability to perform exaflops (10^{18} floating-point operations) per second

Reproducibility at ExaScale (1/2)

- **Reproducibility** – ability to obtain bit-wise identical results from run-to-run on the same input data on the same or different architectures
- **ExaScale** – ability to perform exaflops (10^{18} floating-point operations) per second

Challenges

- Increasing power of current computers
  $\rightarrow$ GPU accelerators, Intel Phi processors, etc.
- Enable to solve more complex problems
  $\rightarrow$ Quantum field theory, supernova simulation, etc.
- A high number of floating-point operations performed
  $\rightarrow$ Each of them leads to round-off error

Reproducibility at ExaScale (1/2)

- **Reproducibility** – ability to obtain bit-wise identical results from run-to-run on the same input data on the same or different architectures
- **ExaScale** – ability to perform exaflops $(10^{18}$ floating-point operations) per second

Challenges

- Increasing power of current computers
  $\rightarrow$ GPU accelerators, Intel Phi processors, etc.
- Enable to solve more complex problems
  $\rightarrow$ Quantum field theory, supernova simulation, etc.
- A high number of floating-point operations performed
  $\rightarrow$ Each of them leads to round-off error

$\Downarrow$

Difficult to obtain accurate and reproducible results

Reproducibility at ExaScale (2/2)

Needs for Reproducibility

- Debugging
  - Look inside the code step-by-step and might need to rerun multiple times on the same input data
- Understanding the reliability of output
- Contractual reasons (for security, ...)
- Prominent examples:
  - Nuclear energy
  - Weather and climate simulation

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

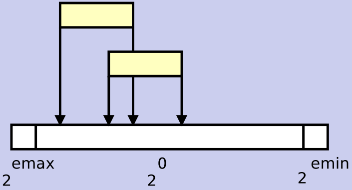
- Aims at benefiting from both FPEs and Kulisch long accumulators:
  - Fast and accurate computations with FPEs
  - “Infinite” precision of Kulisch long accumulators when needed

Algorithm 1 FPE of size n

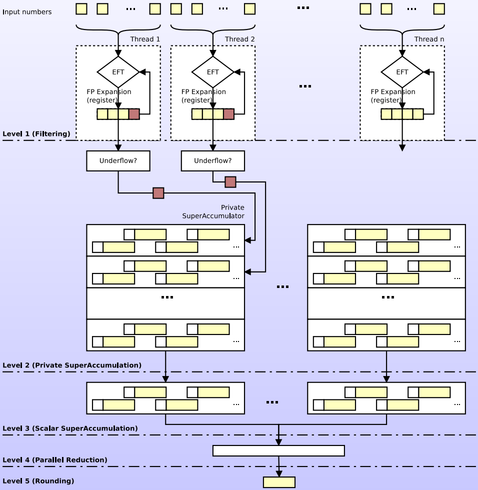
Function = ExpansionAccumulate(x)

```
for i = 0 → n − 1 do
    (a_i, x) ← TwoSum(a_i, x)
end for
if x ≠ 0 then
    Superaccumulate(x)
end if
```

Kulisch long accumulator



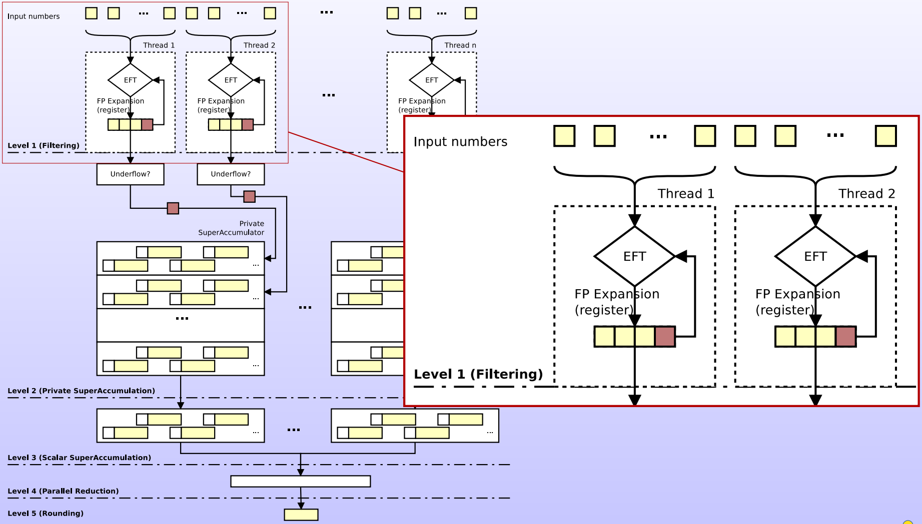
Multi-Level Reproducible Summation



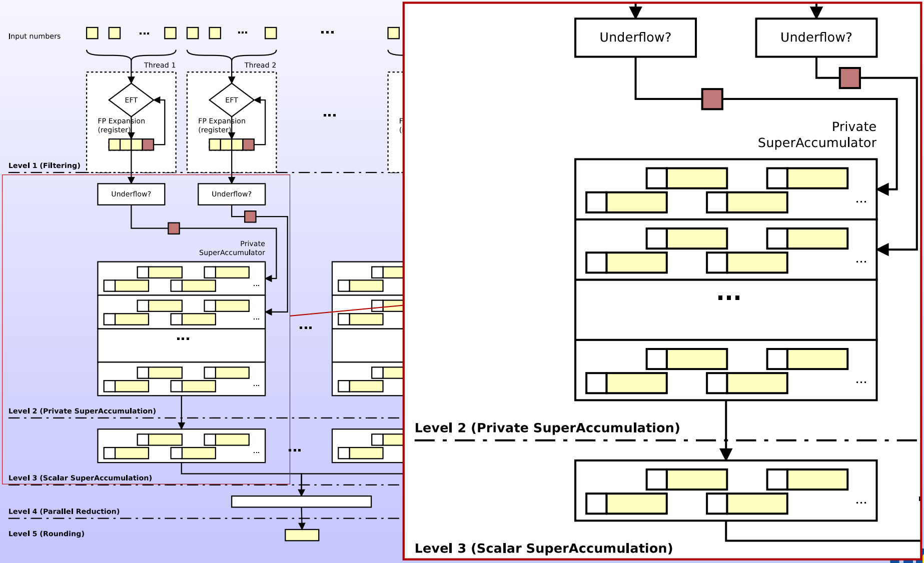
- Parallel algorithm with 5-levels
- Suitable for today's parallel architectures
- Based on FPE with EFT and Kulisch accumulator
- Guarantees "inf" precision

→ bit-wise reproductibility

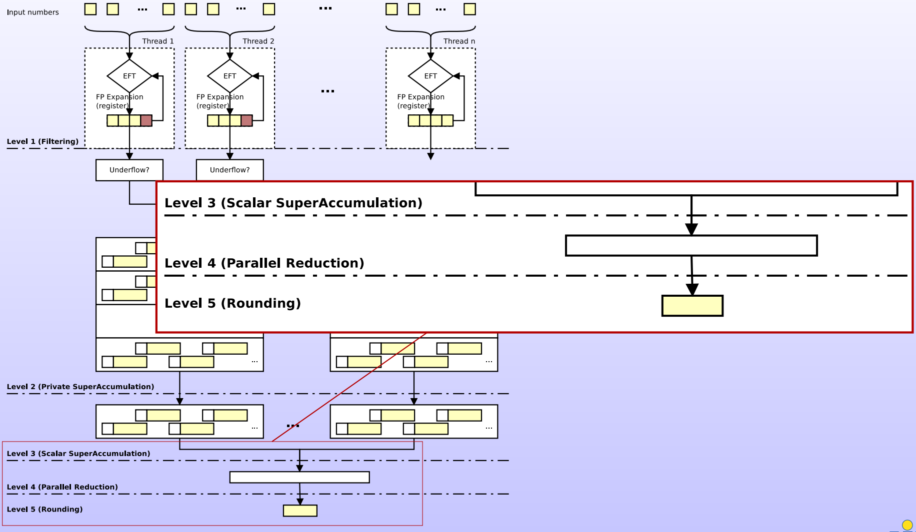
Level 1: Filtering



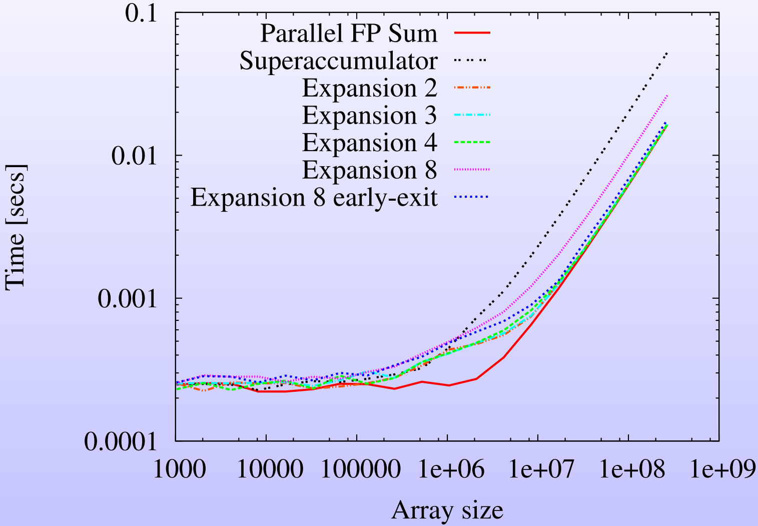
Level 2 and 3: Scalar Superaccumulator



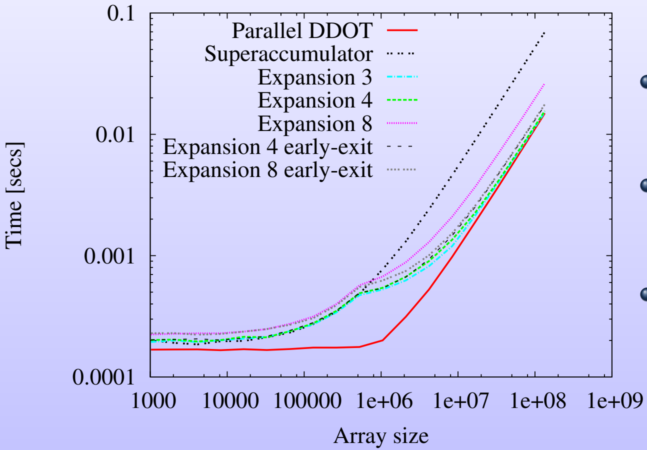
Level 4 and 5: Reduction and Rounding



Performance Scaling on NVIDIA Tesla K20c



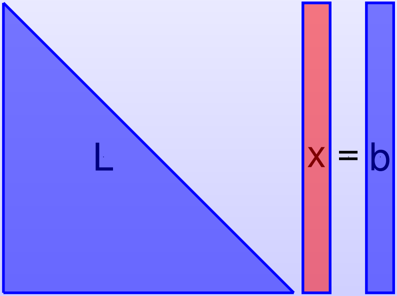
DDOT: $\alpha := x^T y = \sum_i^N x_i y_i$



- Based on TwoProduct and Reproducible Summation
- TwoProduct(a, b)
  1: $r \leftarrow a * b$
  2: $s \leftarrow fma(a, b, -r)$
- $fma(a, b, c) = a * b + c$

Triangular Solver

TRSV (Triangular solver): $Lx = b$



Algorithm 2 Forward substitution

$x_1 \leftarrow b_1 / l_{11}$
for $i = 2 \rightarrow n$ **do**
    $s \leftarrow b_i$
    for $j = 1 \rightarrow i - 1$ **do**
        $s \leftarrow s - l_{ij} x_j$
    end for
    $x_i \leftarrow s / l_{ii}$
end for

Triangular Solver

Matrix Partitioning

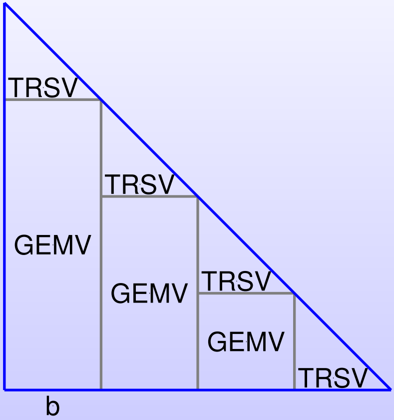


Figure : Partitioning of L in GotoBLAS

Triangular Solver

Matrix Partitioning

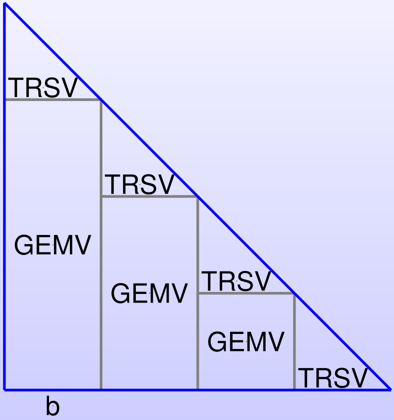
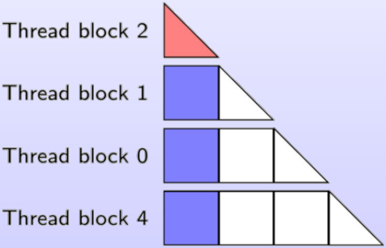
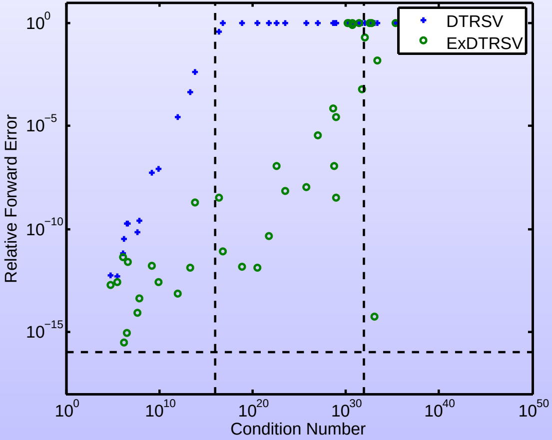


Figure : Partitioning of L in GotoBLAS

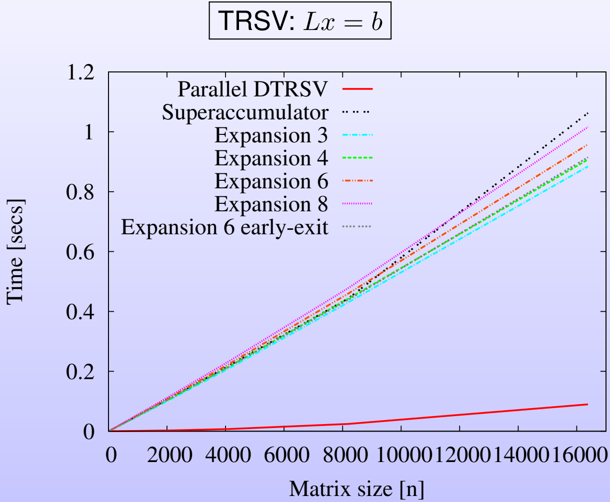


Source: A fast triangular solve on GPUs by Hogg

$$\frac{\|x-\widehat{x}\|}{\|x\|} \leq nu\,\mathrm{cond}(T,x) + \mathrm{O}(u^2)$$



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**



- Use $n \times b$ threads and accumulators
- Higher usage of memory and switches to accumulators $\rightarrow$ lower performance
- But, it is reproducible

Reproducible Triangular Solvers

- Provides bit-wise identical reproducibility, regardless of
  - Data permutation, data assignment
  - Thread scheduling, etc.

Reproducible Triangular Solvers

- Provides bit-wise identical reproducibility, regardless of
  - Data permutation, data assignment
  - Thread scheduling, etc.
- Is for the moment too slow

Conclusions

Reproducible Triangular Solvers

- Provides bit-wise identical reproducibility, regardless of
  - Data permutation, data assignment
  - Thread scheduling, etc.
- Is for the moment too slow
- The DTRSV performance needs to be enhanced

Future Work

- ExTRSV on Intel Phi and Intel CPUs
- ExTRSV using superaccumulators of different sizes
- ExTRSV with iterative refinement and FPEs

ExBLAS – **Exact BLAS**

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

Acknowledgement

Thank you for your attention!

→ This work undertaken (partially) in the framework of CALSIMLAB is supported by the public grant ANR-11-LABX-0037-01 overseen by the French National Research Agency (ANR) as part of the “Investissements d’Avenir” program (reference: ANR-11-IDEX-0004-02)

→ This work was granted access to the HPC resources of The Institute for scientific Computing and Simulation financed by Region Île-de-France and the project Equip@Meso (reference ANR-10-EQPX-29-01) overseen by the French National Research Agency (ANR) as part of the “Investissements d’Avenir” program

On the Web

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.