Numerical reproducibility and High-performance computing

Stef Graillat

LIP6/PEQUAN, Sorbonne Universités, UPMC Univ Paris 06, CNRS

Webinar on Reproducible Research: Numerical reproducibility
May 3rd, 2016, Grenoble, France

Outline of the talk

Outline of the talk

1. Introduction - motivations
2. Floating-point arithmetic
3. Numerical reproducibility and HPC

Motivations

- Reproducibility of experiments and analysis by others is one of the pillars of modern science
- However descriptions of experimental protocols, software, and analysis is often lacunar and rarely allows others to reproduce an experiment

By numerical reproducibility, we mean getting a bitwise identical floating-point result from multiple runs of the same code on the same inputs.

Motivations

2016 Petascale: we are able to perform 30 – 40 petaflops

2017 Petascale: we plan to perform 100 – 200 petaflops

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

$\Downarrow$

$\boxed{10^{18} \text{ rounding errors per second}}$

Improve the numerical quality and the numerical reproducibility of computations on high-performance computing (HPC) platforms, starting from multithreaded computations on multicore processors and targeting ultimately exascale computations.

Motivations

2016 Petascale: we are able to perform 30 – 40 petaflops

2017 Petascale: we plan to perform 100 – 200 petaflops

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

$\Downarrow$

$\boxed{10^{18} \text{ rounding errors per second}}$

Improve the numerical quality and the numerical reproducibility of computations on high-performance computing (HPC) platforms, starting from multithreaded computations on multicore processors and targeting ultimately exascale computations.

Motivations

2016 Petascale: we are able to perform 30 – 40 petaflops

2017 Petascale: we plan to perform 100 – 200 petaflops

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

$\Downarrow$

$\boxed{10^{18} \text{ rounding errors per second}}$

Improve the numerical quality and the numerical reproducibility of computations on high-performance computing (HPC) platforms, starting from multithreaded computations on multicore processors and targeting ultimately exascale computations.

Motivations

2016 Petascale: we are able to perform 30 – 40 petaflops

2017 Petascale: we plan to perform 100 – 200 petaflops

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

$$\Downarrow$$

$$\boxed{10^{18} \text{ rounding errors per second}}$$

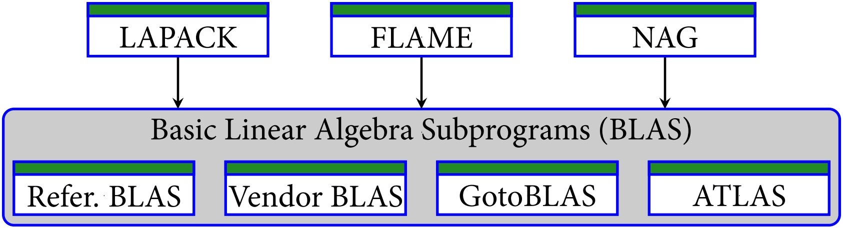
Improve the numerical quality and the numerical reproducibility of computations on high-performance computing (HPC) platforms, starting from multithreaded computations on multicore processors and targeting ultimately exascale computations.

Motivations

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

Motivations

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



Ultimate Goal

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

ExBLAS – Exact BLAS

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

Outline of the talk

Floating-point numbers

Normalized floating-point numbers $\mathbb{F} \subseteq \mathbb{R}$:

$$x = \pm \underbrace{x_0.x_1 \ldots x_{M-1}}_{mantissa} \times b^e, \quad 0 \le x_i \le b-1, \quad x_0 \neq 0$$

b : basis, M : precision, e : exponent such that $e_{\min} \le e \le e_{\max}$
epsilon machine $\epsilon = b^{1-M}$

Approximation of $\mathbb{R}$ by $\mathbb{F}$ with rounding $\mathrm{fl} : \mathbb{R} \to \mathbb{F}$.
Let $x \in \mathbb{R}$ then

$$\mathrm{fl}(x) = x(1+\delta), \quad |\delta| \le \mathbf{u}$$

Unit rounding $\mathbf{u} = \epsilon/2$ for rounding to the nearest

Standard model of floating-point arithmetic

Let $x, y \in \mathbb{F}$ and $\circ \in \{+, -, \cdot, /\}$.

The result $x \circ y$ is not in general a floating-point number

$$\mathrm{fl}(x \circ y) = (x \circ y)(1 + \delta), \quad |\delta| \leq \mathbf{u}$$

IEEE 754 standard (1985 and 2008)

Correctly rounded : arithmetic ops $(+, -, \times, /, \sqrt{\ })$ performed as if first calculated to infinite precision, then rounded.

Type	Size	Mantissa	Exponent	Unit rounding	Interval
binary32	32 bits	23+1 bits	8 bits	$\mathbf{u} = 2^{1-24} \approx 1,92 \times 10^{-7}$	$\approx 10^{\pm 38}$
binary64	64 bits	52+1 bits	11 bits	$\mathbf{u} = 2^{1-53} \approx 2,22 \times 10^{-16}$	$\approx 10^{\pm 308}$

Error-free transformation (EFT) for addition

$$x = a \oplus b \quad \Rightarrow \quad a + b = x + y \quad \text{with } y \in \mathbb{F},$$

Algorithm of Dekker (1971) and Knuth (1974)

Algorithm 1 (EFT of the sum of 2 floating-point numbers)

function $[x, y] = \texttt{TwoSum}(a, b)$

$x = a \oplus b$

$z = x \ominus a$

$y = (a \ominus (x \ominus z)) \oplus (b \ominus z)$

EFT for multiplication

$$x = a \otimes b \quad \Rightarrow \quad a \times b = x + y \quad \text{with } y \in \mathbb{F},$$

Given $a, b, c \in \mathbb{F}$,

- $\mathtt{FMA}(a, b, c)$ is the nearest floating-point number $a \cdot b + c \in \mathbb{F}$

Algorithm 2 (EFT of the product of 2 floating-point numbers)

function $[x, y] = \mathtt{TwoProduct}(a, b)$
  $x = a \otimes b$
  $y = \mathtt{FMA}(a, b, -x)$

The FMA is available for example on PowerPC, Itanium, Cell, Xeon Phi, Haswell processors.

Floating-point expansions (FPE)

Representation using floating-point numbers: non-evaluated sum of floating-point numbers

$$\sum_{i=0}^{n} f_i$$

where the f_i are floating-point numbers, if possible with exponents sufficiently wide apart so that the mantissas do not overlap.

double-double library

A double-double number is a non-evaluated pair (a_h, a_l) of IEEE 754 floating-point numbers satisfying $a = a_h + a_l$ et $|a_l| \leq \mathbf{u}|a_h|$.

Algorithm 3 (Addition of a double b and a double-double (a_h, a_l))

function $[c_h, c_l] = \texttt{add\_dd\_d}(a_h, a_l, b)$

  $[t_h, t_l] = \texttt{TwoSum}(a_h, b)$

  $[c_h, c_l] = \texttt{TwoSum}(t_h, (t_l \oplus a_l))$

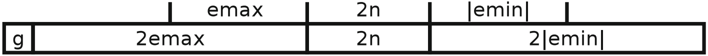
double-double library

Algorithm 4 (Product of a double-double (a_h, a_l) by a double b)

function $[c_h, c_l] = \texttt{prod\_dd\_d}(a_h, a_l, b)$
  $[s_h, s_l] = \texttt{TwoProduct}(a_h, b)$
  $[t_h, t_l] = \texttt{TwoSum}(s_h, (a_l \otimes b))$
  $[c_h, c_l] = \texttt{TwoSum}(t_h, (t_l \oplus s_l))$

Kulisch accumulator

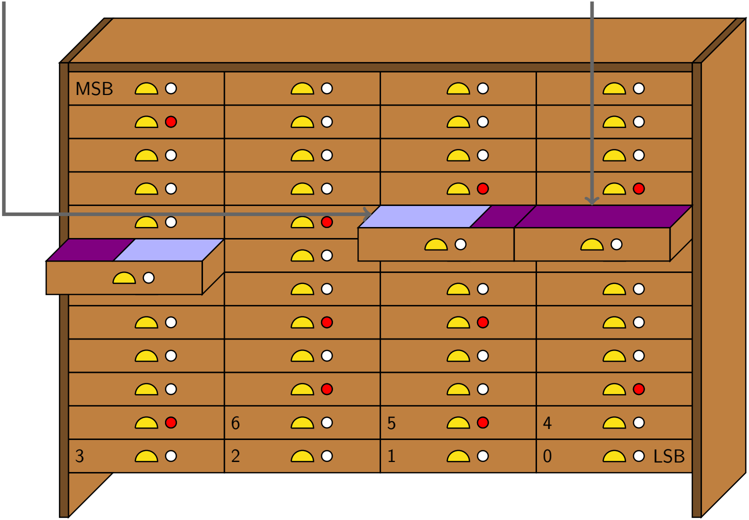
Computing without error due to the limited range of floating-point numbers



In double precision, $n = 53$ bits, $emin = -1022$, $emax = 1023$ and $k = 92$ bits

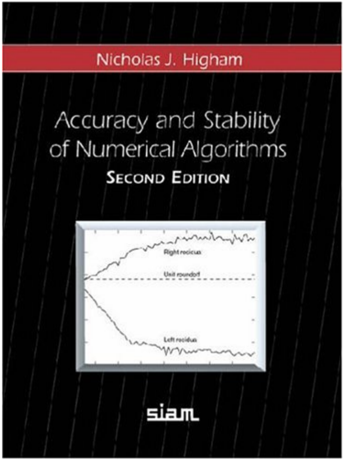
A register of length $L = k + 2emax + 2|emin| + 2n = 4288$ bits is sufficient (67 words of 64 bits)

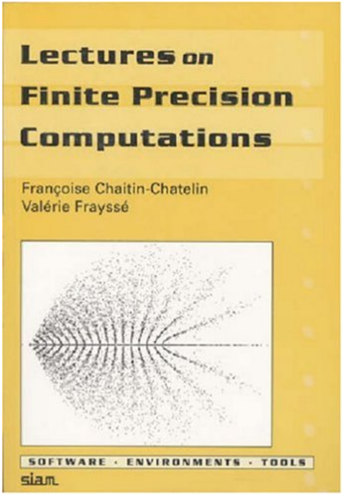
Kulisch accumulator



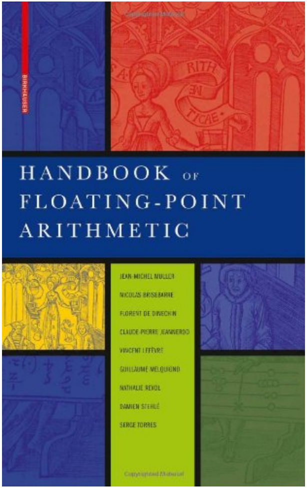
Source: Kulisch's papers

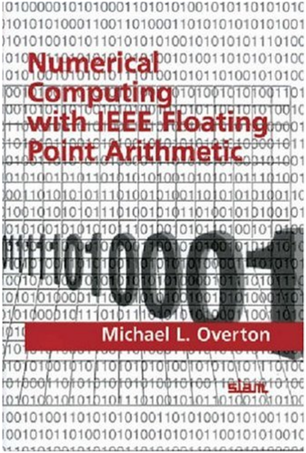
Bibliography





Bibliography



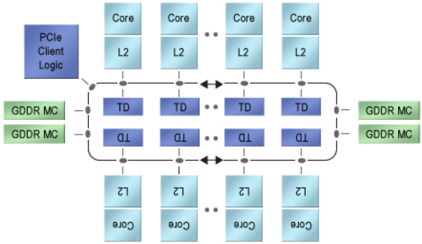


What every computer scientist should know about floating-point arithmetic. David Goldberg. ACM Computing Surveys, 23(1):5–48, 1991.

Outline of the talk

1. Introduction - motivations
2. Floating-point arithmetic
3. Numerical reproducibility and HPC

From multi-core to many-cores

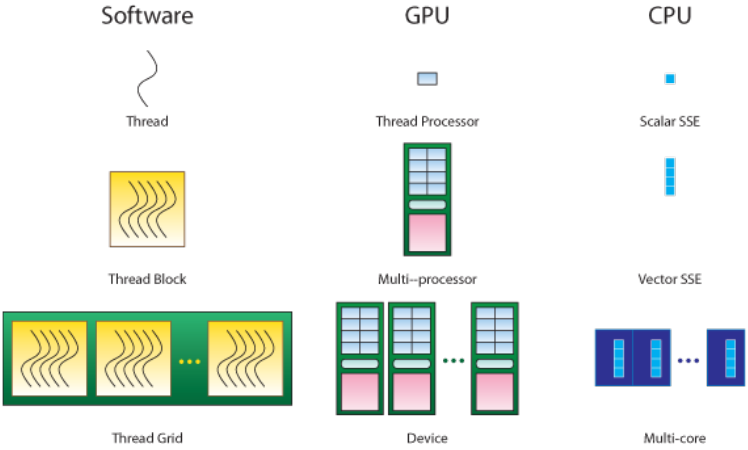


Intel Xeon Phi: 50 x86 cores

NVIDIA K20: 2496 CUDA cores

Source: http://www.altera.com/technology/system-design/articles/2012/multicore-many-core.html
http://wccftech.com/nvidia-tesla-k20-gk110-specifications-unveiled/

Execution on many-cores



Source: `http://www.pgroup.com/lit/articles/insider/v2n4a1.htm`

Numerical reproducibility

Floating-point operations suffers from rounding error

Floating-point operation $(+, \times)$ are commutatives but not 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.

Numerical reproducibility: ability to obtain bit-wise identical results from multiple runs of the same code on the same input data on different or even similar architectures.

Motivations

Demands for reproducible floating-point computations:

- 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 (security, liability, etc.)
- ...

Existing reproducibility failures for numerical simulations in energy, dynamical weather science, dynamical molecular, dynamical fluid

Sources of non-reproducibility

A performance-optimized floating-point library is prone to inconsistency for various reasons:

- Changing Data Layouts:
  - Data partitioning
  - Data alignment
- Changing Hardware Resources:
  - Number of threads
  - Fused Multiply-Add (FMA) 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
  - ...

Numerical reproducibility for Exascale

Exascale : ability to execute 10^{18} floating-point operations per second using $\mathcal{O}(10^9)$ processors

- Highly dynamic scheduling
- Network heterogeneity
- increased communication time

  Cost = Arithmetic + Communication

Numerical reproducibility for Exascale

ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems

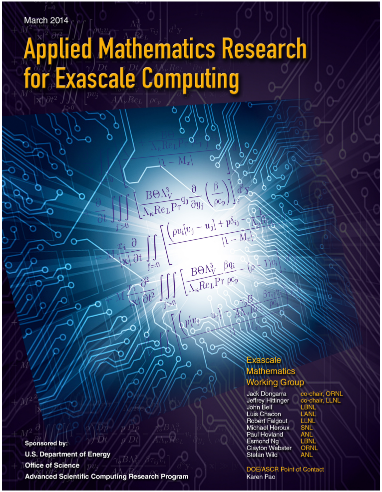
Peter Kogge, Editor & Study Lead
Keren Bergman
Shekhar Borkar
Dan Campbell
William Carlson
William Dally
Monty Denneau
Paul Franzon
William Harrod
Kerry Hill
Jon Hiller
Sherman Karp
Stephen Keckler
Dean Klein
Robert Lucas
Mark Richards
Al Scarpelli
Steven Scott
Allan Snavely
Thomas Sterling
R. Stanley Williams
Katherine Yelick

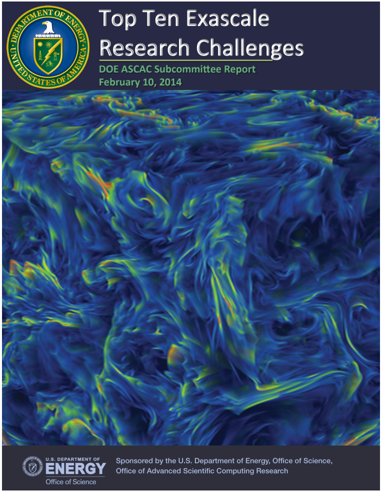
September 28, 2008

This work was sponsored by DARPA IPTO in the ExaScale Computing Study with Dr. William Harrod as Program Manager; AFRL contract number **FA8650-07-C-7724.** This report is published in the interest of scientific and technical information exchange and its publication does not constitute the Government's approval or disapproval of its ideas or findings

NOTICE

Numerical reproducibility for Exascale





Numerical reproducibility for Exascale

Top 10 Challenges to Exascale

3 Hardware, 4 Software, 3 Algorithms/Math Related

- **Energy efficiency:**
  - Creating more energy efficient circuit, power, and cooling technologies.
- **Interconnect technology:**
  - Increasing the performance and energy efficiency of data movement.
- **Memory Technology:**
  - Integrating advanced memory technologies to improve both capacity and bandwidth.
- **Scalable System Software:**
  - Developing scalable system software that is power and resilience aware.
- **Programming systems:**
  - Inventing new programming environments that express massive parallelism, data locality, and resilience
- **Data management:**
  - Creating data management software that can handle the volume, velocity and diversity of data that is anticipated.
- **Scientific productivity:**
  - Increasing the productivity of computational scientists with new software engineering tools and environments.
- **Exascale Algorithms:**
  - Reformulating science problems and refactoring their solution algorithms for exascale systems.
- **Algorithms for discovery, design, and decision:**
  - Facilitating mathematical optimization and uncertainty quantification for exascale discovery, design, and decision making.
- **Resilience and correctness:**
  - Ensuring correct scientific computation in face of faults, reproducibility, and algorithm verification challenges.

Source: Dongarra's slides, `http://www.scan2014.uni-wuerzburg.de/fileadmin/10030000/scan2014/talks/plenary_6.pdf`

Numerical reproducibility

Source of floating-point non-reproducibility: rounding errors lead to dependence of computed result on order of computations

To obtain reproducibility:

- Fix the order of computations:
  - sequential computations: high cost on parallel machines
  - fixed reduction tree: communication cost → Intel CNR
- Eliminate/Reduce the rounding errors:
  - fixed-point arithmetic: limited range of exponent
  - higher precision: higher probability but not always → Taufer et al.
  - computation without rounding-error (pre-rounding) → Demmel et al.
  - exact arithmetic (only one rounding at the end)

Numerical reproducibility for summation

Aim: compute $\sum_{i=1}^{n} x_i$ for some floating-point x_i.

Algorithm 5 (Recursive summation algorithm)

function $\texttt{res} = \texttt{Sum}(x)$
  $s = 0;$
  for $i = 1 : n$
    $s = s \oplus x_i$
  $\texttt{res} = s$

Reproducible reduction tree

Demmel et al. 2013

Idea: fix the reduction tree ahead of computing time so that its shape does not depends on available resources at runtime.

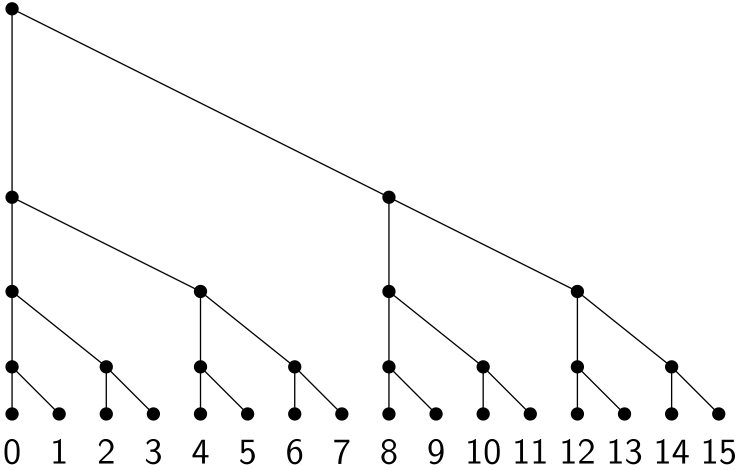
Strategy:

- Split input vectors into chunks of fixed size,
- Impose the reduction tree over chunks (not threads).

Intel Conditional Numerical Reproducibility (CNR) library for Intel MKL (Math Kernel Library)
$\rightarrow$ works only for the same version of MKL on the same hardware

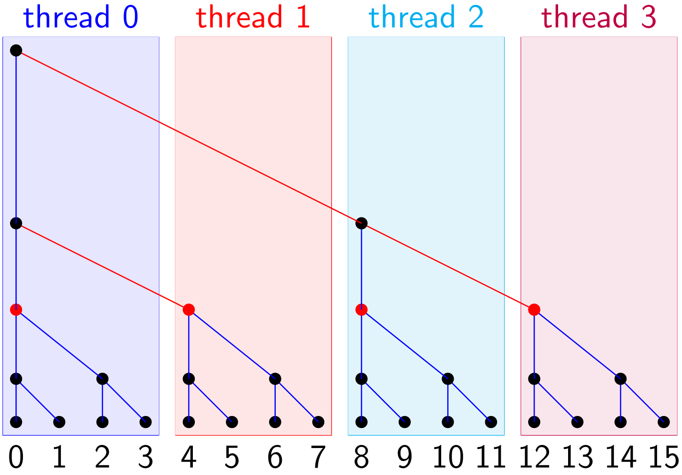
Reproducible reduction tree

Demmel et al. 2013



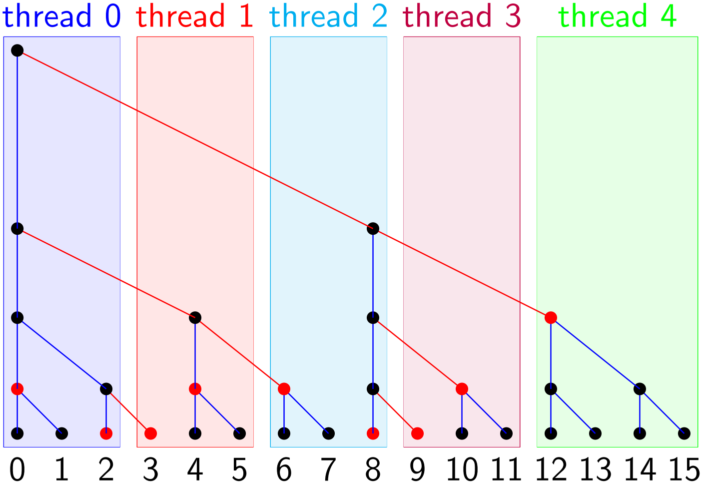
Reproducible reduction tree

Demmel et al. 2013



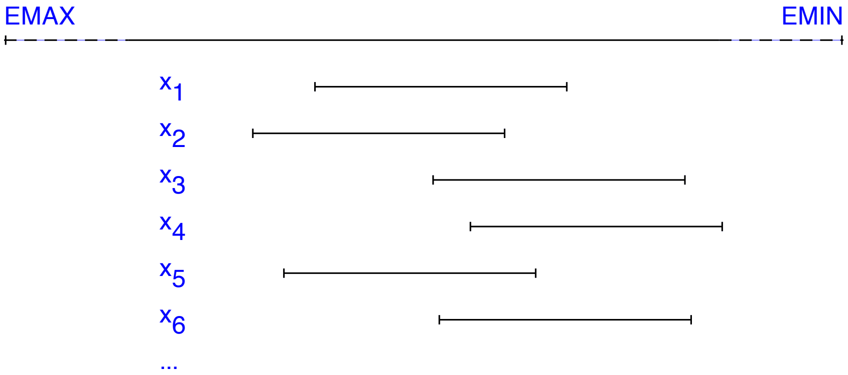
Reproducible reduction tree

Demmel et al. 2013



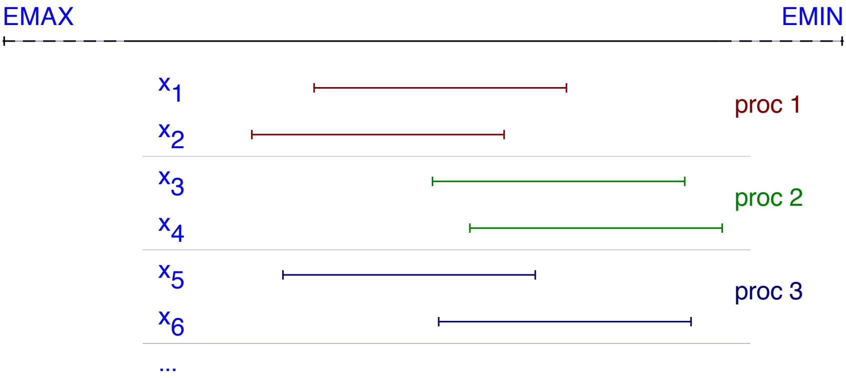
Pre-rounding technique

Demmel and al. 2013, 2014,2015



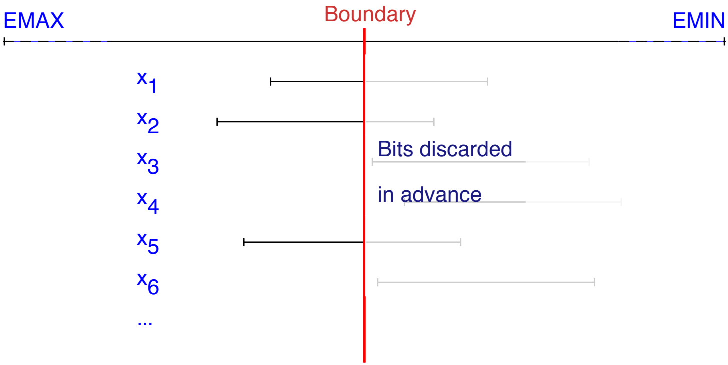
Rounding occurs at each addition. Computation's error depends on the intermediate results, which depend on the order of computation.

Pre-rounding technique



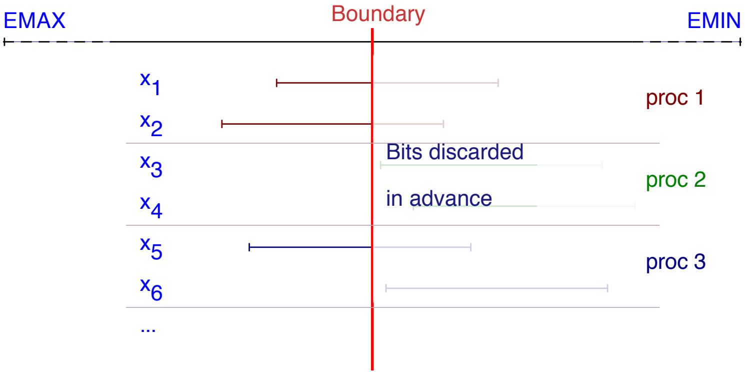
Rounding occurs at each addition. Computation's error depends on the intermediate results, which depend on the order of computation.

Pre-rounding technique



No rounding error at each addition. Computation's error depends on the boundary, which depends on $\max|x_i|$, not on the ordering.

Pre-rounding technique



No rounding error at each addition. Computation's error depends on the boundary, which depends on $\max |x_i|$, not on the ordering.

Approach with superaccumulator

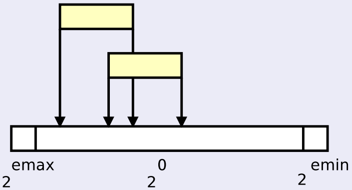
- Aims at benefiting from both FPEs and Kulisch long accumulators:
  - Fast and accurate computations with FPEs
  - ‘‘Infinite’’ precision of Kulisch long accumulators when needed

Algorithm 1 FPE of size n

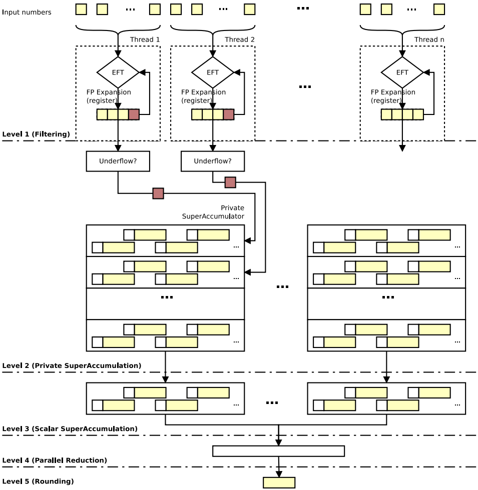
Function = ExpansionAccumulate(x)

for $i = 0 : n-1$ **do**
    $(a_i, x) \leftarrow \text{TwoSum}(a_i, x)$
end for
if $x \neq 0$ **then**
    Superaccumulate(x)
end if

Kulisch long accumulator



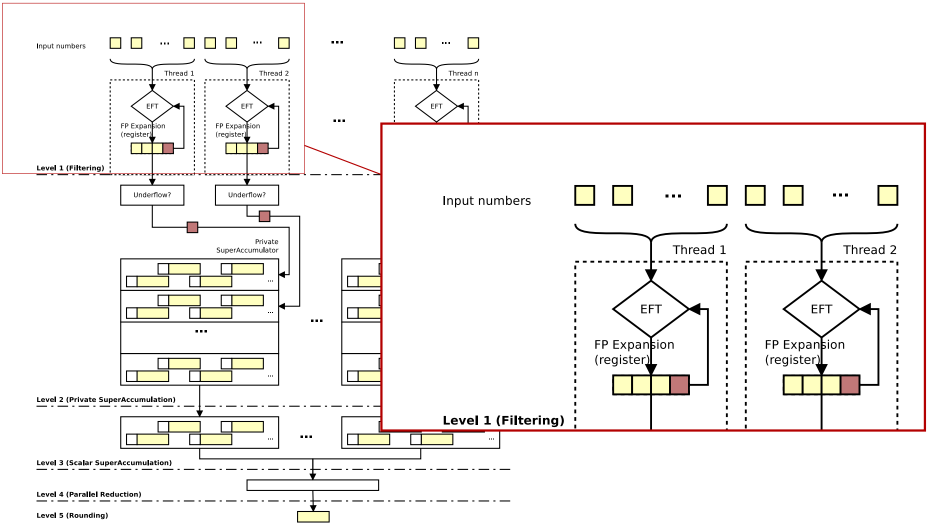
Multi-Level Reproducible Summation



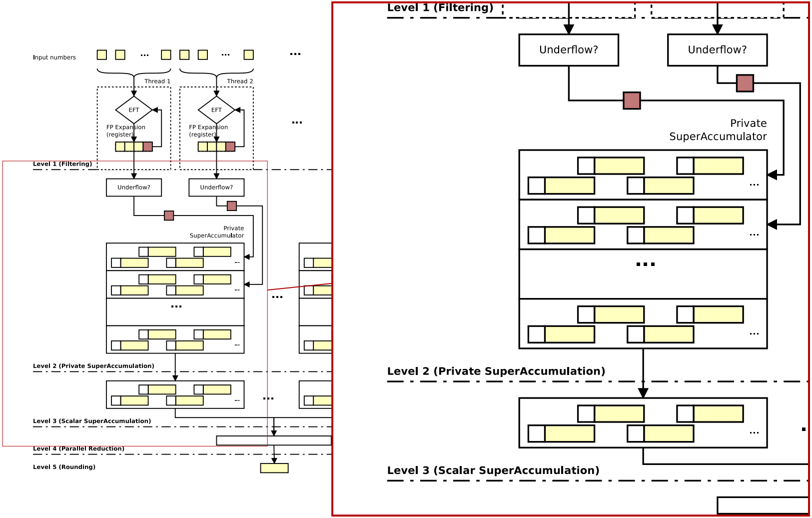
- Parallel algorithm with 5-levels
- Suitable for today's parallel architectures
- Based on FPE with EFT and Kulisch accumulator
- Guarantees "inf" precision

→ bit-wise reproductibility

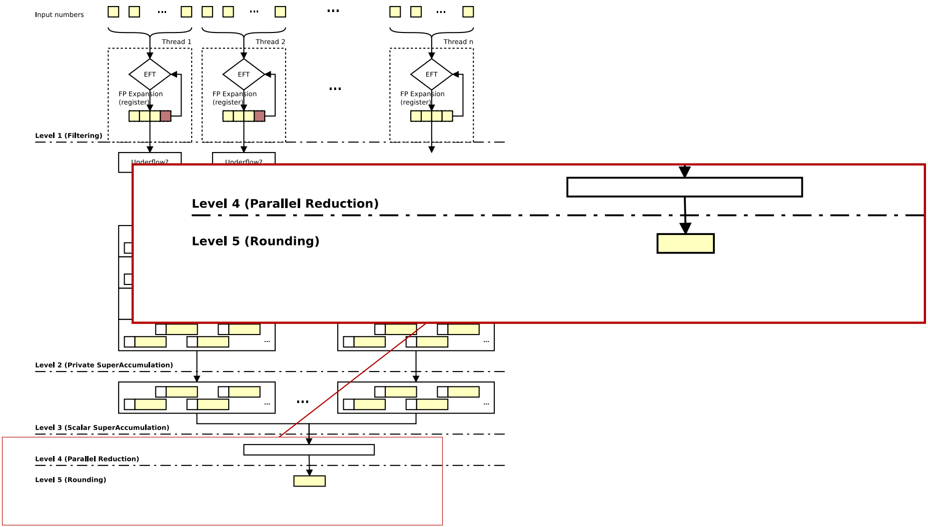
Level 1: Filtering



Level 2 and 3: Scalar Superaccumulator



Level 4 and 5: Reduction and Rounding



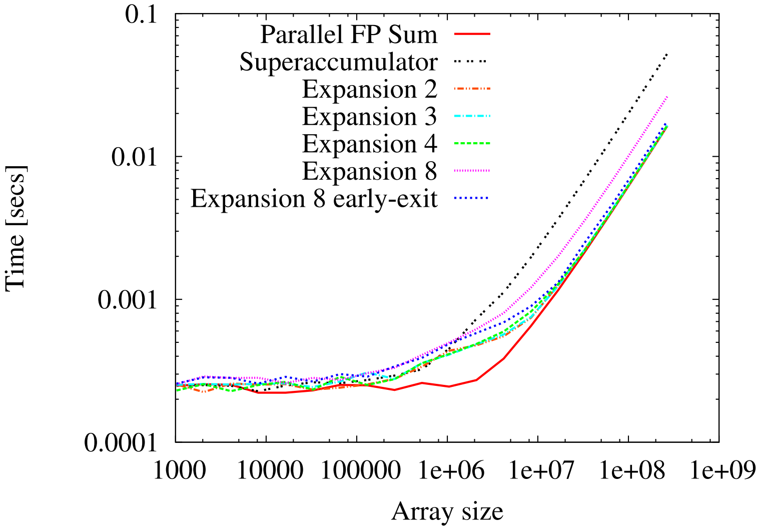
Experimental Environments

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

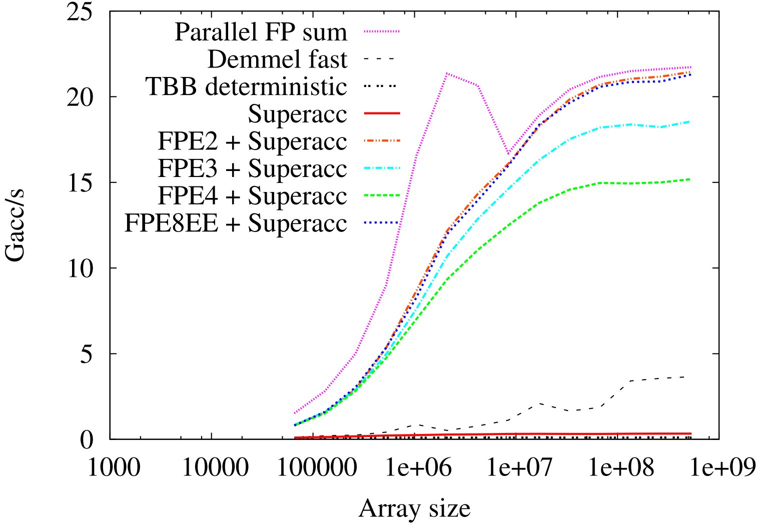
Parallel Summation

Performance Scaling on NVIDIA Tesla K20c



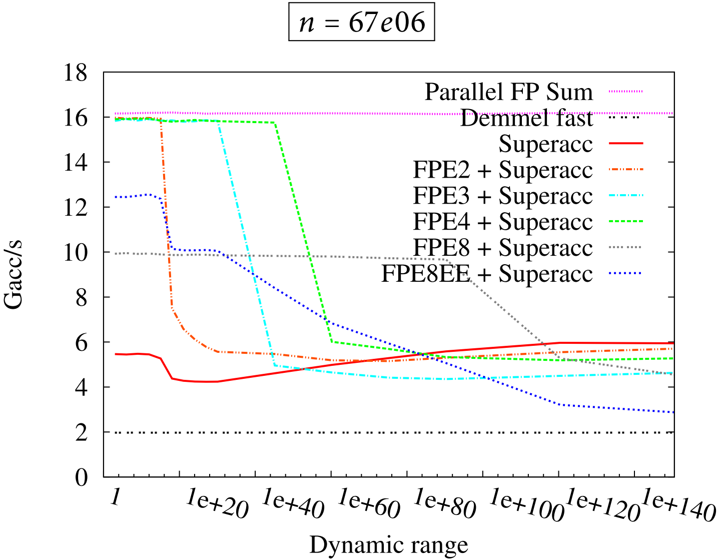
Parallel Summation

Performance Scaling on Intel Xeon Phi



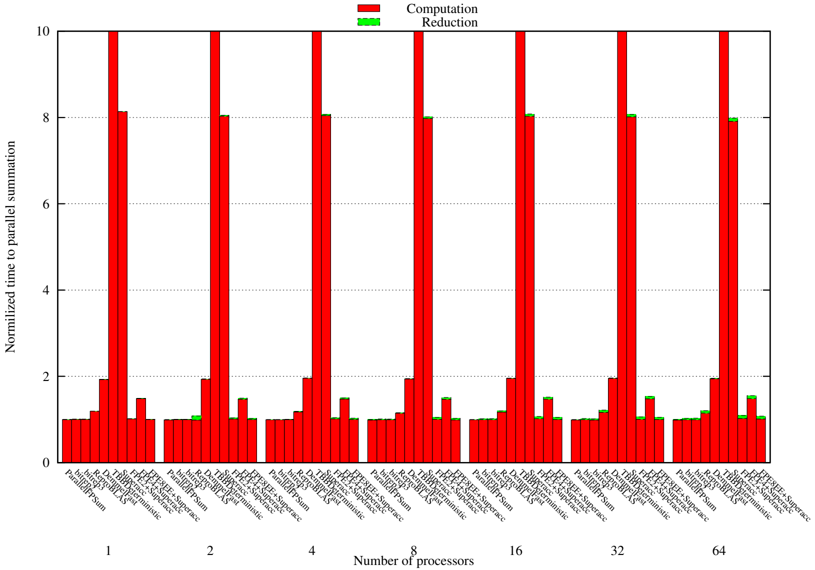
Parallel Summation

Data-Dependent Performance on NVIDIA Tesla K20c



Parallel Summation with MPI

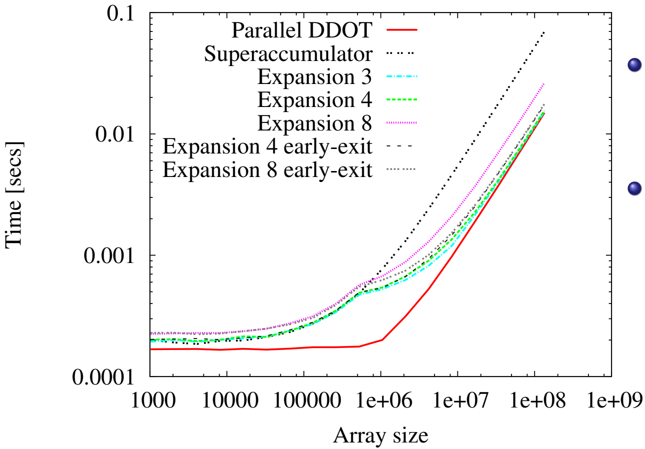
Performance Scaling on Mesu cluster; $n = 16e06$



Parallel Dot Product

Performance Scaling on NVIDIA Tesla K20c

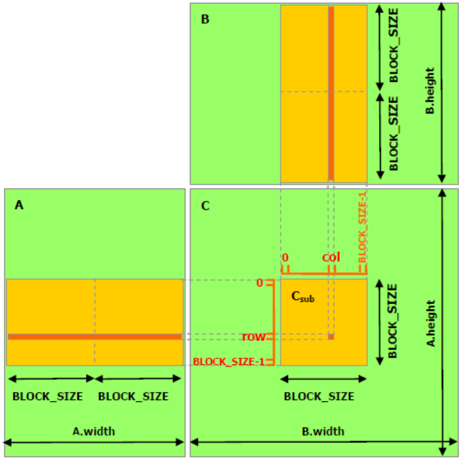
$$\boxed{\text{DDOT: } \alpha := x^T y = \sum_i^N x_i y_i}$$



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

Multi-Level Reproducible DGEMM

$$\boxed{\text{DGEMM: } C := \alpha AB + \beta C}$$

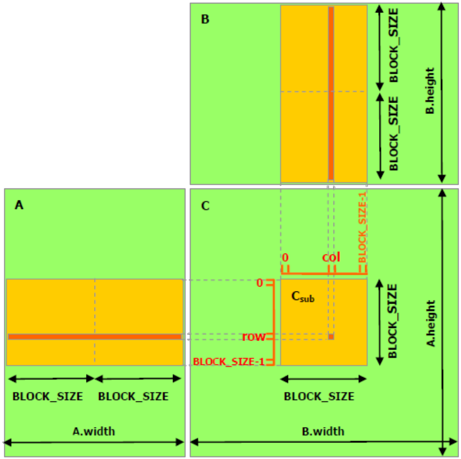


Source: CUDA C Programming Guide

- One FPE and Kulisch accumulator per thread
- Algorithm consists of 3 steps:
  - Filtering
  - Private SuperAccumulation
  - Rounding
- Each thread computes multiple elements of matrix C to reduce memory pressure

Multi-Level Reproducible DGEMM

$$\boxed{\text{DGEMM: } C := \alpha AB + \beta C}$$

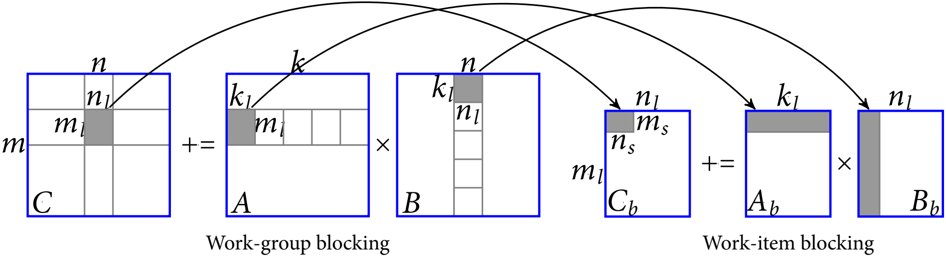


Source: CUDA C Programming Guide

- One FPE and Kulisch accumulator per thread
- Algorithm consists of 3 steps:
  - Filtering
  - Private SuperAccumulation
  - Rounding
- Each thread computes multiple elements of matrix C to reduce memory pressure

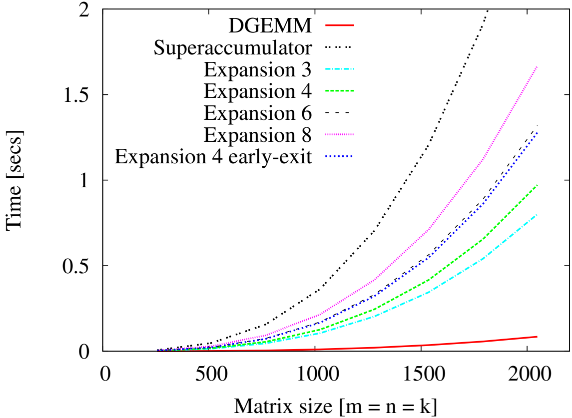
Parallel Matrix Multiplication

GEMM (General matrix multiplication): $C := \alpha AB + \beta C$



Partitioning of matrix-matrix multiplication

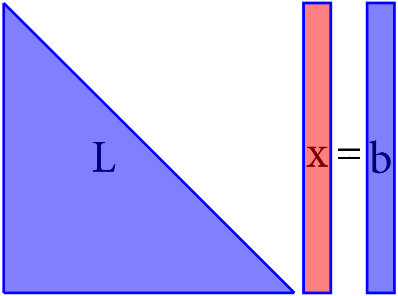
$$\boxed{\text{DGEMM: } C := \alpha AB + \beta C}$$



- $12dn^3$ flops, d is size of FPE
- Up to $76n^3$ more memory usage

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

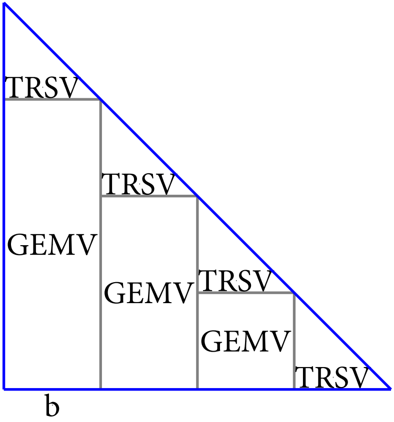


Figure : Partitioning of L in GotoBLAS

Triangular Solver
Matrix Partitioning

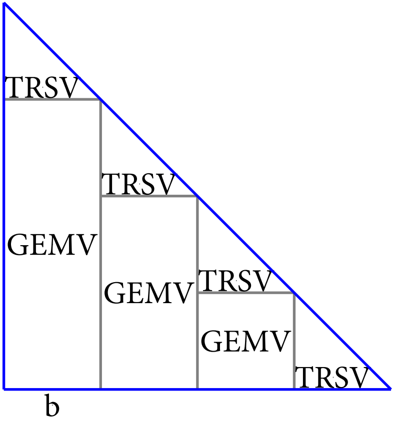
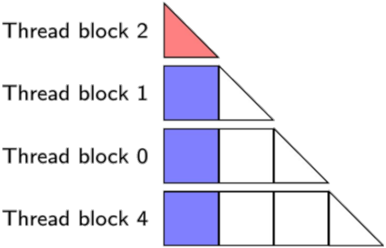


Figure : Partitioning of L in GotoBLAS

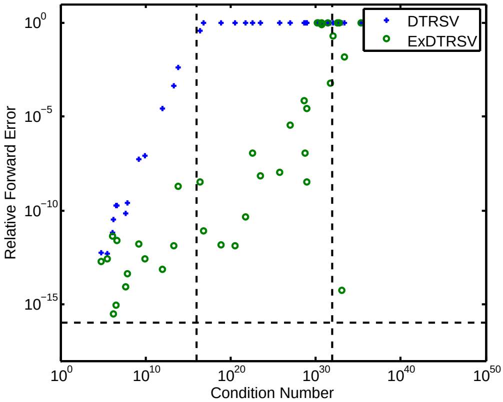


Source: A fast triangular solve on GPUs by Hogg

Triangular Solver

Accuracy

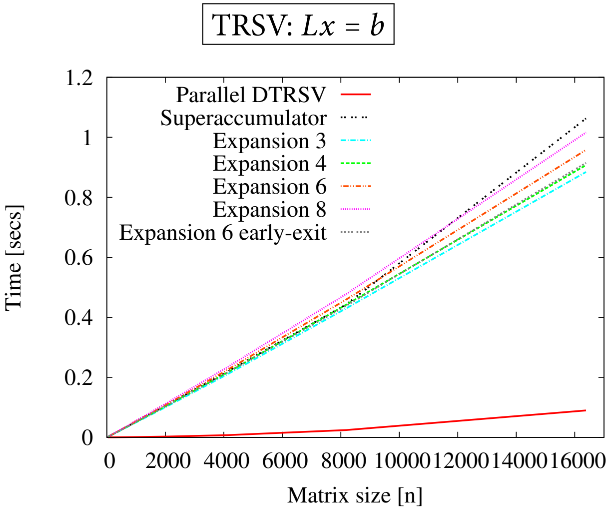
$$\frac{\|x-\widehat{x}\|}{\|x\|} \leq n \cdot \mathbf{u} \cdot \mathrm{cond}(T, x) + \mathrm{O}(u^2)$$



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: $\quad x_i \leftarrow fl(RN(s)/l_{ii})$
8: **end for**

Multi-Level Reproducible TRSV

Performance Scaling on NVIDIA Quadro K5000



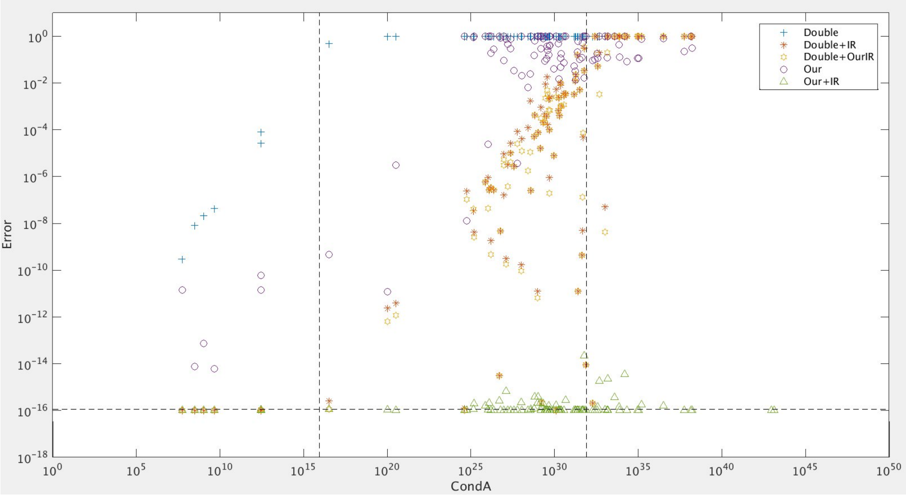
- Use of $n \times b$ threads and superaccumulators
- Higher usage of memory and switches to accumulators → lower performance
- But, it is reproducible

Reproducible TRSV with iterative refinement

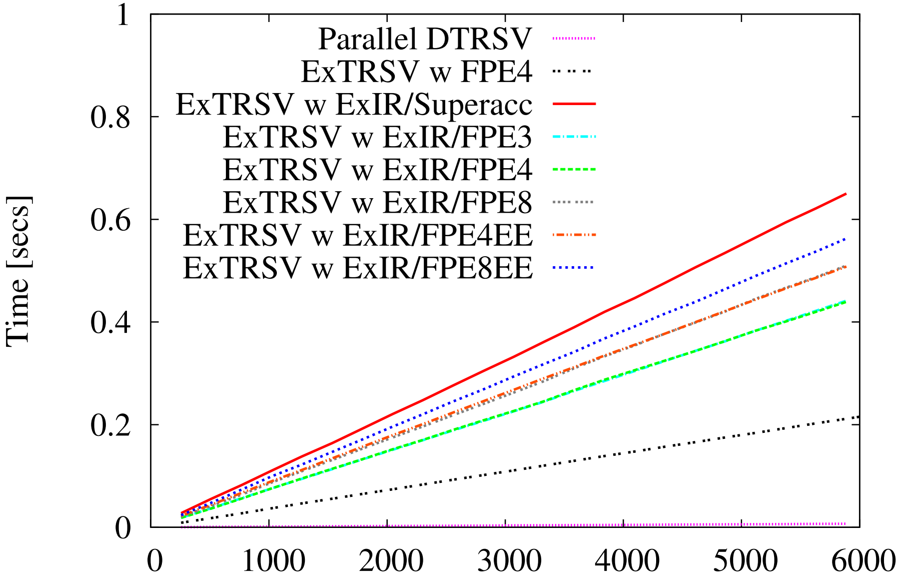
Algorithm 3 Reproducible TRSV with iterative refinement

1: $\widehat{x} \leftarrow T^{-1}b$ ExTRSV
2: **for** $i = 1 \rightarrow nbiter$ **do**
3: $r \leftarrow b - T\widehat{x}$ ExGEMV
4: $d \leftarrow T^{-1}r$ ExTRSV
5: $\widehat{x} \leftarrow \widehat{x} + d$ ExAXPY
6: **end for**

Reproducible TRSV with iterative refinement



Reproducible TRSV with iterative refinement



Reproducible LU factorization

Partition

$$A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$$

where A_{TL} is 0×0

While $size(A_{TL}) < size(A)$ **do**

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$$

where α_{11} is 1×1

$$\begin{array}{lr} a_{10}^T := a_{10}^T U_{00}^{-1} & \texttt{(TRSV)} \\ \alpha_{11} := \alpha_{11} - a_{10}^T a_{01} & \texttt{(DOT)} \\ a_{12}^T := a_{12}^T - a_{10}^T A_{02} & \texttt{(GEMV)} \end{array}$$

Continue with

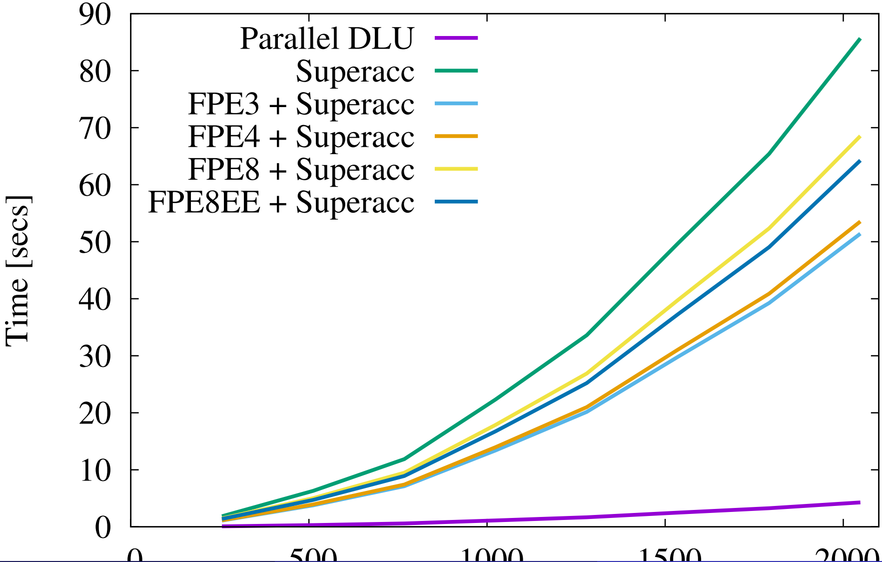
$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$$

endwhile

$$\begin{array}{c|c|c|c|} & i & 1 & n-i-1 \\ \hline i & A_{00} & a_{01} & A_{02} \\ \hline 1 & a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline m-i-1 & A_{20} & a_{21} & A_{22} \\ \hline \end{array}$$

Reproducible LU factorization

Performance of ExLU on NVIDIA K20c.



Reproducible linear algebra libraries

- ReproBLAS : `http://bebop.cs.berkeley.edu/reproblas/`

  developed at University of California, Berkeley by Jim Demmel and Hong Diep Nguyen

- ExBLAS : `https://exblas.lip6.fr/`

  developed at LIP6, UPMC by Sylvain Collange, Stef Graillat, David Defour and Roman Iakymchuk

Conclusions

The Proposed Multi-Level Algorithm

- Computes the results with no errors due to rounding
- Provides bit-wise identical reproducibility, regardless of
  - Data permutation, data assignment
  - Thread scheduling, etc.
- Is efficient – delivers comparable performance to the standard parallel summation and dot product
- Scales with the increase of the problem size or the number of cores
- The ExGEMM and ExLU performances need to be enhanced

Conclusions

The Proposed Multi-Level Algorithm

- Computes the results with no errors due to rounding
- Provides bit-wise identical reproducibility, regardless of
  - Data permutation, data assignment
  - Thread scheduling, etc.
- Is efficient – delivers comparable performance to the standard parallel summation and dot product
- Scales with the increase of the problem size or the number of cores
- The ExGEMM and ExLU performances need to be enhanced

Conclusions

The Proposed Multi-Level Algorithm

- Computes the results with no errors due to rounding
- Provides bit-wise identical reproducibility, regardless of
  - Data permutation, data assignment
  - Thread scheduling, etc.
- Is efficient – delivers comparable performance to the standard parallel summation and dot product
- Scales with the increase of the problem size or the number of cores
- The ExGEMM and ExLU performances need to be enhanced

Future Work

- ExBLAS on more architectures (Intel Phi and Intel CPUs)
- ExBLAS for large scale systems (ExaScale) with several nodes
- Use of Communication-Avoiding Algorithms

ExBLAS – Exact BLAS

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

References I

- J. Demmel, H. D. Nguyen.
  Fast reproducible floating-point summation.
  Proceedings of the 21st IEEE Symposium on Computer Arithmetic, Austin, Texas, USA, 2013, pp. 163–172.
- A. Arteaga, O. Fuhrer, T. Hoefler.
  Designing bit-reproducible portable high-performance applications.
  Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium, IPDPS '14, IEEE Computer Society, Washington, DC, USA, 2014, pp. 1235–1244.
- J. Demmel and H. D. Nguyen.
  Parallel Reproducible Summation.
  IEEE Transactions on Computers, 64(7):2060–2070, 2015.

References II

S. Collange, D. Defour, S. Graillat, and R. Iakymchuk.
Numerical Reproducibility for the Parallel Reduction on Multi- and Many-Core Architectures.
Parallel Computing, 49:83–97, 2015.

M. Taufer, M. Becchi.
The Numerical Reproducibility Fair Trade: Facing the Concurrency Challenges at the Extreme Scale
Challenges in 21st Century Experimental Mathematical Computation. Institute for Computational and Experimental Research in Mathematics (ICERM). Providence, RI, USA, 2014