Efficient Reproducible Floating-Point Reduction Operations on Large Scale Systems

James Demmel, **Hong Diep Nguyen**

ParLab - EECS - UC Berkeley

SIAM AN13 Jul 8-12, 2013

Plan

Plan

Floating-point arithmetic: defines a discrete subset of real values and suffers from *rounding errors*.

→ Floating-point operations $(+, \times)$ are commutative but not associative:

$$(-1 + 1) + 2^{-53} \neq -1 + (1 + 2^{-53}).$$

Consequence: results of floating-point computations depend on the order of computation.

Reproducibility: ability to obtain bit-wise identical results from run-to-run on the same input data, with different resources.

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. Ex: [1], Power State Estimation problem (spmv + dot product), after the 5th step the Euclidean norm of the residual vector differs up to 20% from one run to another.
- Contractual reasons (road type approval, drug design),
- ...

[1] Villa et al, *Effects of Floating-point non-Associativity on Numerical Computations on Massively Multithreaded Systems*, CUG 2009 Proceedings

Sources of non-reproducibility

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

- Changing Data Layouts:
  - Data alignment,
  - Data partitioning,
  - Data ordering,
- Changing Hardware Resources:
  - Fused Multiply-Adder 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,
  - ???

Sources of non-reproducibility

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

- Changing Data Layouts:
  - Data alignment,
  - Data partitioning,
  - Data ordering,
- Changing Hardware Resources:
  - Fused Multiply-Adder 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,
  - ???

Reproducibility at Large Scale

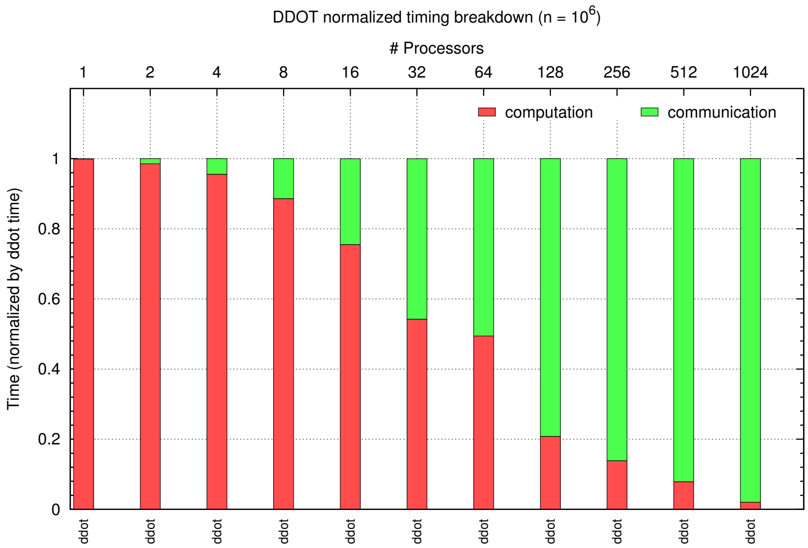
Large Scale: improve performance by increasing the number of processors.

- Highly dynamic scheduling,
- Network heterogeneity: reduction tree shape can vary,
- Drastically increased communication time

$$\begin{array}{llcl} \text{Cost} = & \text{Arithmetic} & + & \textbf{Communication} \\ & \textit{FLOPs} & & \#\text{words moved} + \#\text{messages} \end{array}$$

  - Communication-Avoiding algorithms change the order of computation on purpose, for ex. 2.5D Matmult, 2.5D LU, etc,
  - A little extra arithmetic cost is allowed so long as the communication cost is controlled.

Communication cost



State of the art

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 mode: *intolerably costly at large-scale systems*
  - fixed reduction tree: *substantial communication overhead*
- Eliminate/Reduce the rounding errors:
  - exact arithmetic (rounded at the end): *much more expensive in communication and very wide multi-word arithmetic*
  - fixed-point arithmetic: *limited range of values*
  - higher precision: *reproducible with high probability (not certain)*.
- Our proposed solution: deterministic errors.

Plan

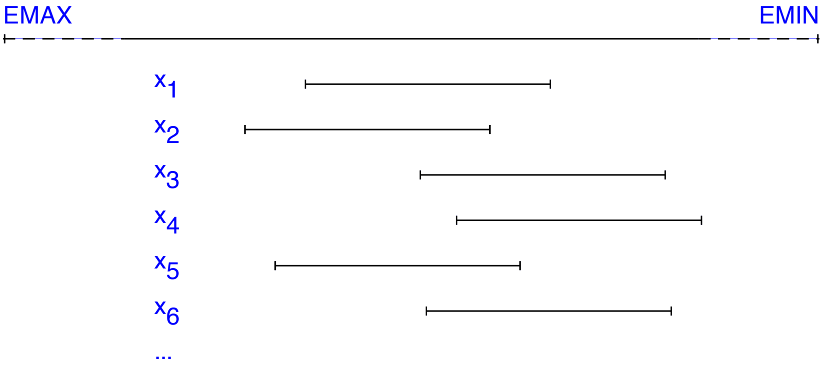
A proposed solution for global sum

Objectives:

- bit-wise identical results from run-to-run regardless of hardware heterogeneity, # processors, reduction tree shape, ...
- independent of data ordering,
- only 1 reduction per sum,
- no severe loss of accuracy.

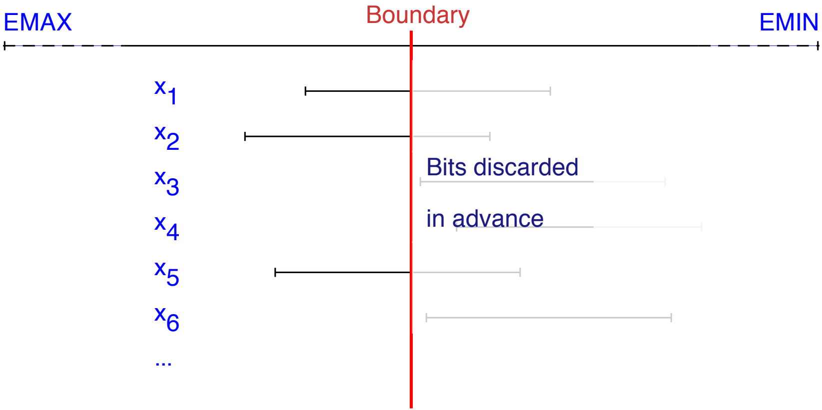
Idea: pre-rounding input values.

Pre-rounding technique



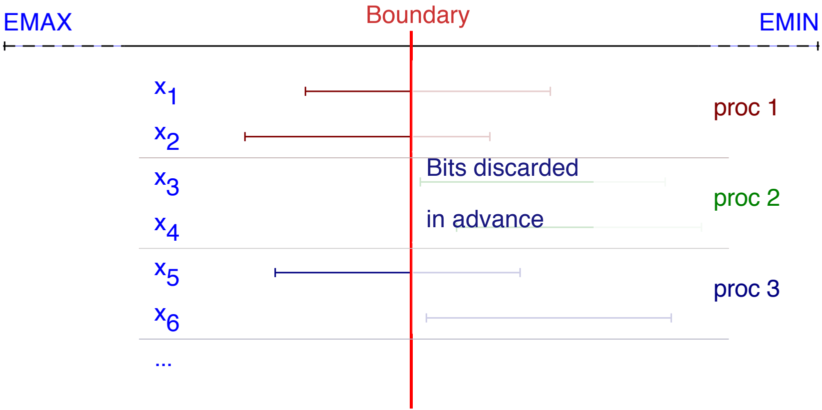
Rounding occurs at each addition. Computation's error depends on the intermediate results, which depend on the order of computation.

Pre-rounding technique



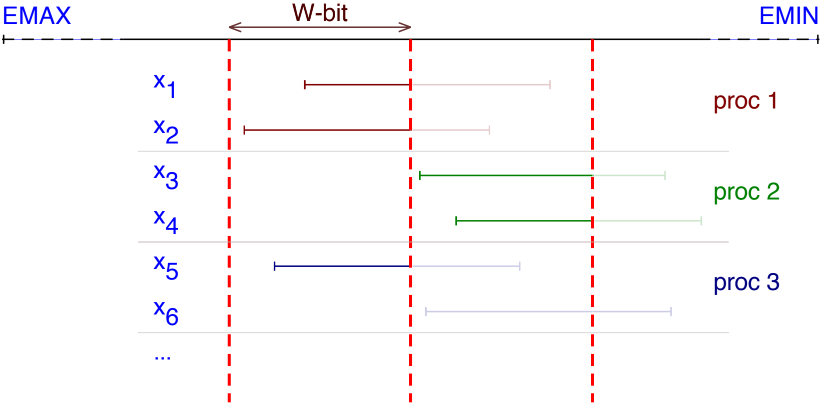
No rounding error at each addition. Computation's error depends on the Boundary, which depends on $\max|x_i|$, not on the ordering

Pre-rounding technique



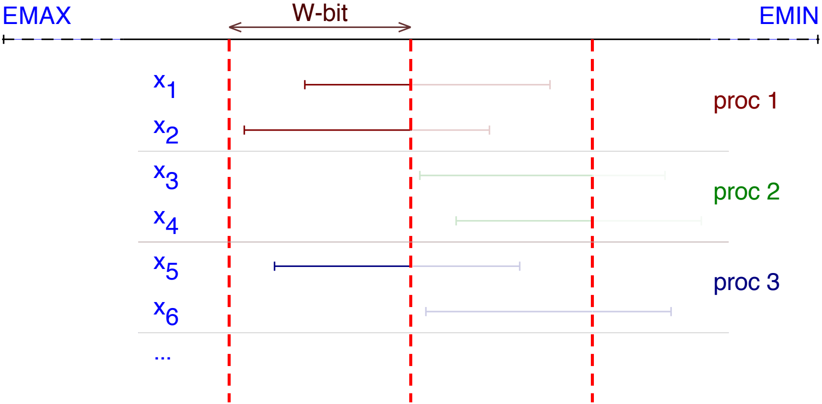
No rounding error at each addition. Computation's error depends on the Boundary, which depends on $\max|x_i|$, not on the ordering $\Rightarrow$ extra communication among processors.

1-Reduction technique



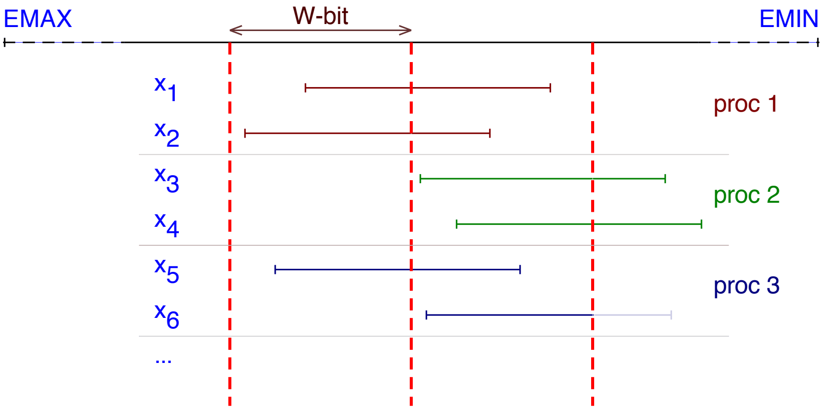
Boundaries are precomputed. Special Reduction Operator:
(`MAX` of boundaries combined `SUM` of corresponding partial sums)

1-Reduction technique



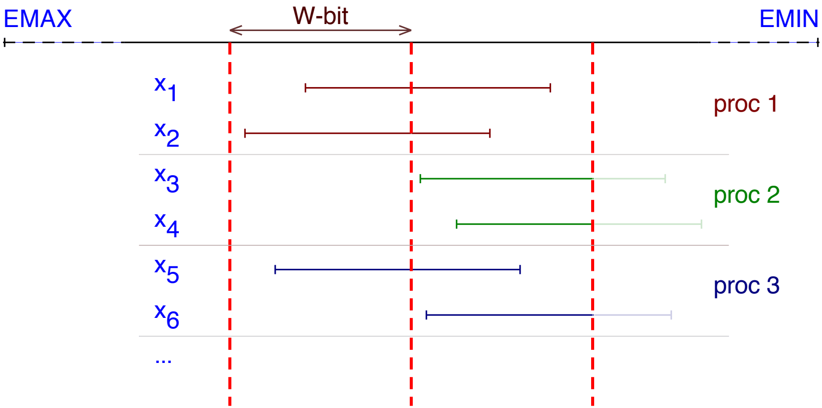
Boundaries are precomputed. Special Reduction Operator:
(`MAX` of boundaries combined `SUM` of corresponding partial sums)

k-fold Algorithm



Increase the number of bins to increase the accuracy.

k-fold Algorithm



Increase the number of bins to increase the accuracy.

k-fold Algorithm: Accuracy

k-fold algorithm has an error bound:

$$\textit{absolute error} \leq N \cdot \textit{Boundary}_k < N \cdot 2^{-(k-1)\cdot W} \cdot \max|x_i|.$$

In practice: $k = 3$, $W = 40$.

$$\begin{aligned} \textit{absolute error} < N \cdot 2^{-80} \cdot \max|x_i| = &\quad 2^{-27} \cdot N \cdot \epsilon &&\cdot \max|x_i| \\ \text{Standard sum's error bound} \leq &\quad (N-1) \cdot \epsilon &&\cdot \sum |x_i| \end{aligned}$$

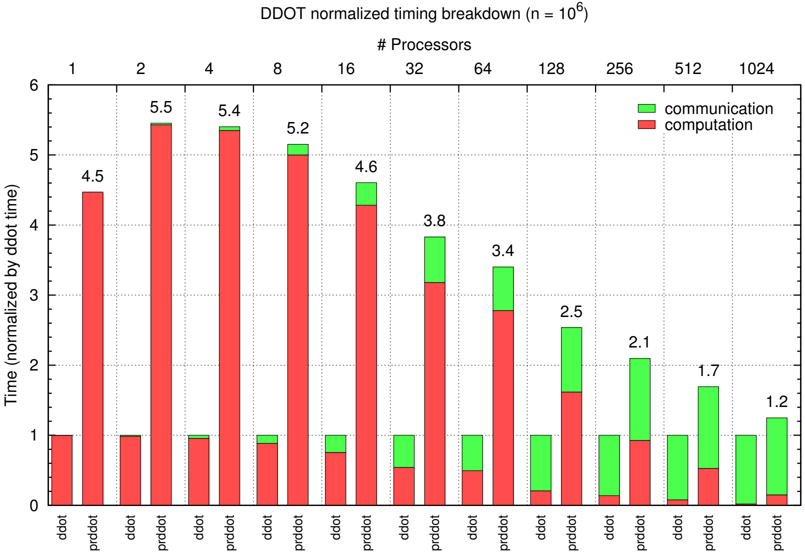
Plan

Experimental results: Accuracy

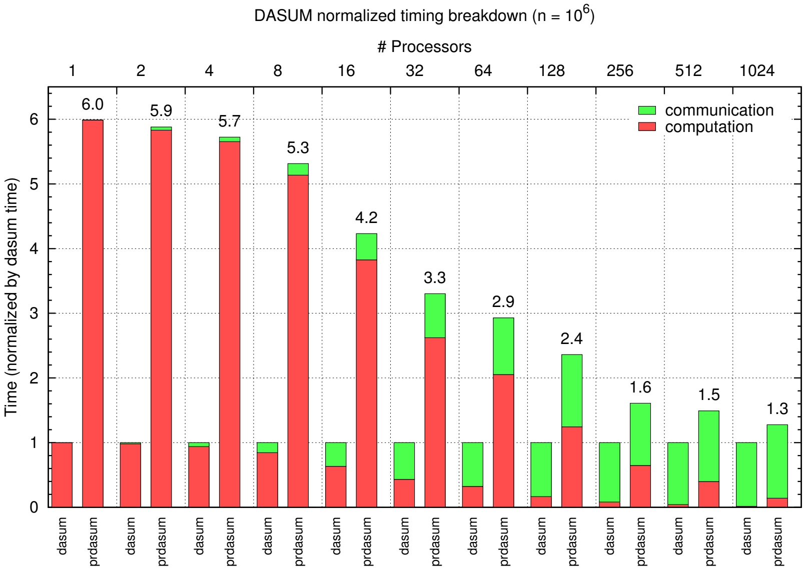
Summation of $n = 10^6$ floating-point numbers. Computed results of both reproducible summation and standard summation (with different ordering: ascending value, descending value, ascending magnitude, descending magnitude) are compared with result computed using quad-double precision.

Generator x_i	reproducible	standard
`drand48()`	0	-8.5e-15 ÷ 1.0e-14
`drand48()` $- 0.5$	1.5e-16	$-1.7e-13 \div 1.8e-13$
$\sin(2.0 * \pi * i/n)$	1.5e-15	$-1.0 \div 1.0$
$\sin(2.0 * \pi * i/n) * 2^{-5}$	1.0	$-1.0 \div 1.0$

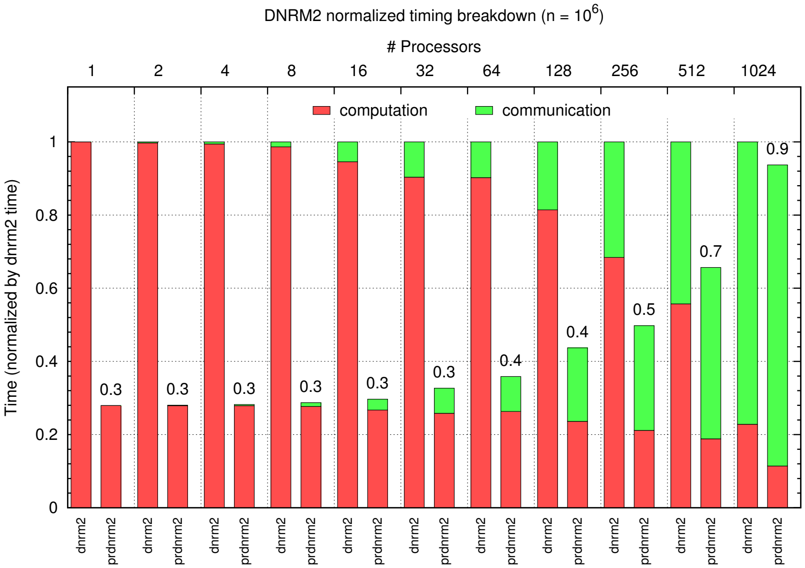
Experimental results: Performance



Experimental results: Performance



Experimental results: Performance



Conclusions

The proposed 1-Reduction pre-rounding technique

- provides bit-wise identical reproducibility, regardless of
  - data permutation, data assignment,
  - processor count, reduction tree shape,
  - hardware heterogeneity, etc.
- obtains better error bound than the standard sum's,
- can be done in on-the-fly mode,
- requires only ONE reduction for the global parallel summation,
- is suitable for very large scale systems (ExaScale),
- can be applied to Cloud computing environment,
- can be applied to other operations which use summation as the reduction operator.

Future works

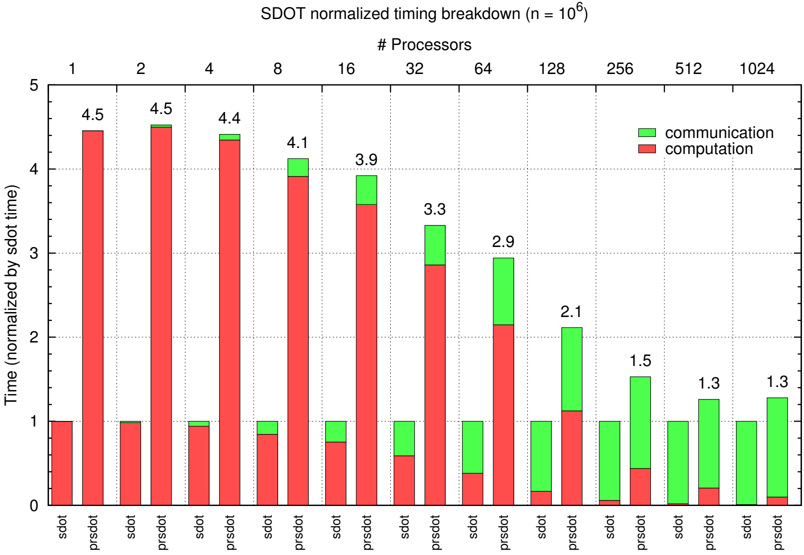
In Progress

- Reproducible Blas level 1,
- Parallel Prefix Sum,
- Matrix-vector / Matrix-matrix multiplication,

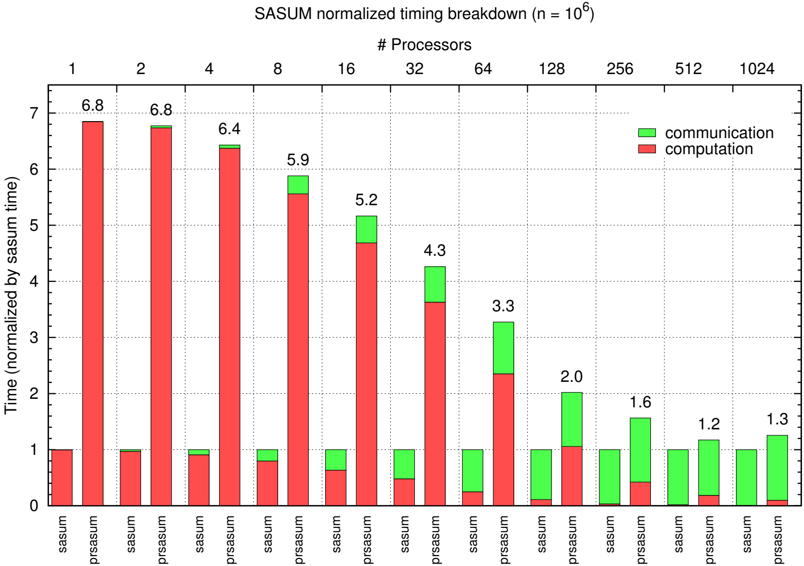
TODO

- Higher level driver routine: `trsm`, factorizations like LU, ...
- n.5D algorithms (2.5D Matmult, 2.5D LU),
- `spMV`,
- Other associative operations
- Real-world applications ?

Experimental results: Performance (single precision)



Experimental results: Performance (single precision)



Experimental results: Performance (single precision)