Fast Reproducible Floating-Point Summation

James Demmel, **Hong Diep Nguyen**

ParLab - EECS - UC Berkeley

ARITH 21 April 7-10, 2013

Plan

Plan

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

$\rightarrow$ 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,
  - ???

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

Objective

Algorithm 1 Standard Floating-Point Summation

Require: x is an array of n floating-point numbers

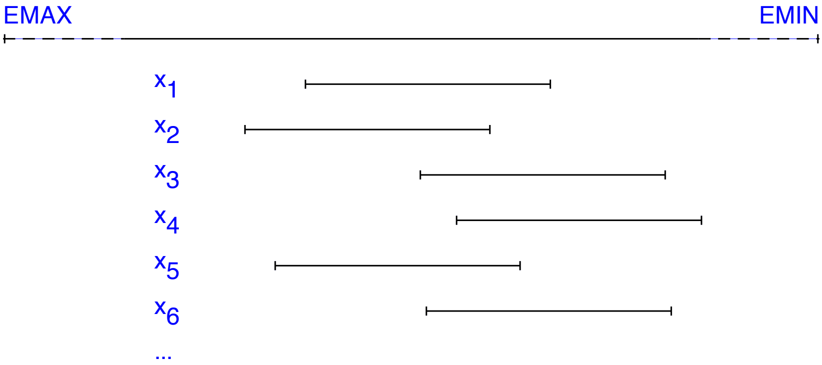
1: $T := 0$
2: **for** $i = 1$ **to** n **do** … in what order ?
3: $T := T + x_i$ … rounded
4: **end for**

Objectives:

- bit-wise identical results regardless of data assignment, # processors, reduction tree shape, … (*order-invariant*)
- of (almost) the same accuracy as the standard sum.

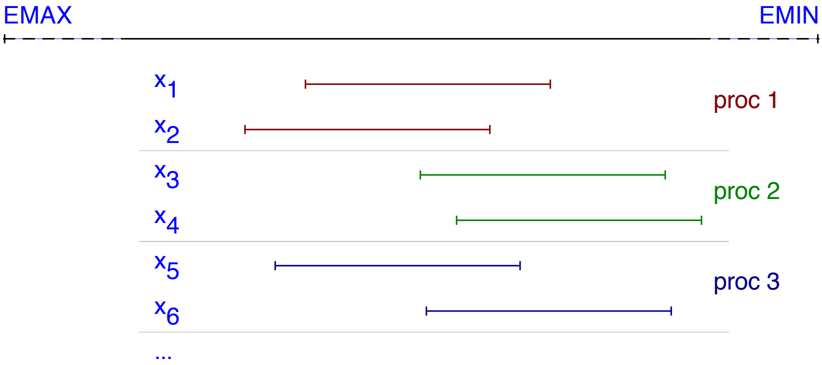
absolute error $\leq (n-1) \cdot \sum_{1}^{n} |x_i| \cdot \mathtt{u}$, $\mathtt{u}$ is the unit round-off error.

Pre-rounding technique



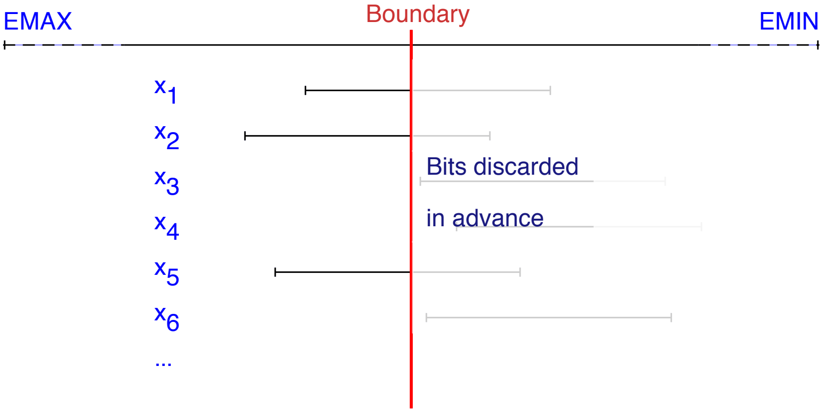
Rounding occurs after each addition. Computation's error depending on the intermediate results, which depend on the computation order.

Pre-rounding technique



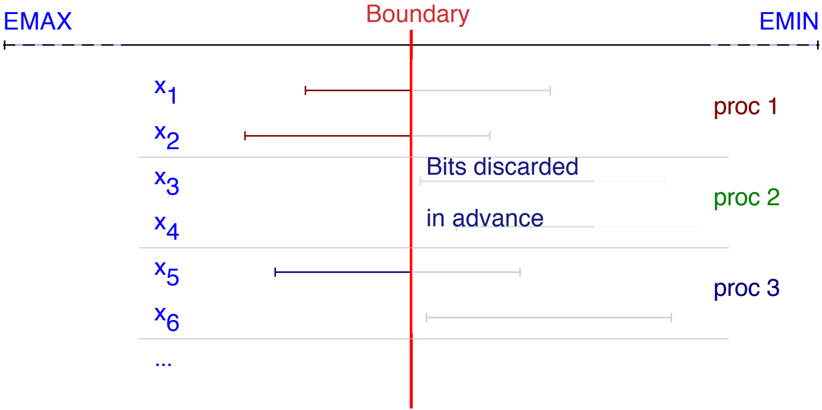
Rounding occurs after each addition. Computation's error depending on the intermediate results, which depend on the computation order.

Pre-rounding technique



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

Pre-rounding technique: splitting algorithm

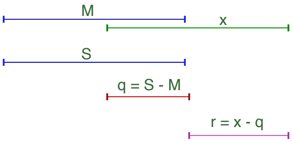
Algorithm 2 Splitting Algorithm

Require: $M, x \in \mathbb{F}, \quad M \gg |x|$

$S := M + x$ ... rounded

$q := S - M$ ... exactly

$r := x - q$ ... exactly



Theorem 1 (Splitting Algorithm in Rounding-to-nearest-even)

Let S, q, r be results computed by the Splitting Algorithm then:

- $q = \frac{1}{2} \cdot \mathtt{ulp}(M) \cdot \mathbb{Z}, \quad r \leq \mathtt{ulp}(M) \leq 2 \cdot \mathtt{u} \cdot M,$
- $S = M + q,$
- $x = q + r$ *(Error-free transformation)*

Reproducible sum

Algorithm 3 Reproducible Sum

```
Require: x is an array of size n
  X := max_i |x_i|
  j := ⌈(log_2(n * X/(1 − 2 * n * u))⌉
  M := pow(2.0, j)
  T := 0
  for i = 1 to n do            … in any order
      S := M + x_i             … rounded
      q_i := S − M             … exactly
      x_i := x_i − q_i         … exactly
      T := T + q_i             … exactly
  end for
```

Requirement:

- q_i is reproducible
- $\sum_1^n q_i$ is exact,

$M = 2^j, j \in \mathbb{Z}$ where

$$2^j \geq \frac{n \cdot \max|x_i|}{1 - 2n \cdot \mathrm{u}} > \sum_1^n |q_i|$$

Reproducible sum: Error bound

Theorem 2 (Reproducibility of Algorithm 3)

Let $T \in \mathbb{F}$ be the result computed by Algorithm 3 then:

- *T is always reproducible,*
- $T = \sum_1^n q_i$
- $|T - \sum_1^n x_i| \leq \sum_1^n |r_i| \leq 2n \cdot \mathtt{u} \cdot M < \frac{4}{1-2n\mathtt{u}} \cdot n^2 \cdot \mathtt{u} \cdot max|x_i|$

Cost: $4n$ `FLOPs` (counting max as a `FLOP`).

Faster Reproducible sum

Algorithm 4 Faster Reproducible sum

Require: x is an array of size n

use directed rounding

$X := \max_i |x_i|$

$j := \lceil (\log_2(n * X/(1 - 2 * n * u)) \rceil$

$M := 3 * \texttt{pow}(2.0, j)$

$M_1 := M$

for $i = 1$ **to** n **do** … in any order

$\quad M_{i+1} := M_i + x_i$ … rounded

$\quad q_i := M_{i+1} - M_i$ … exactly

$\quad x_i := x_i - q_i$ … rounded

end for

$T := M_{n+1} - M$ … exactly

Requirement: The rounding of $M_i + x_i$ is reproducible

- M_i have the same `ulp`: $M = 2^{j+1} + 2^j$ where

$$2^j \geq \frac{n \cdot \max |x_i|}{1 - 2n \cdot u}$$

- must be performed in directed rounding to avoid ambiguous half-way case.

Splitting algorithm in directed rounding

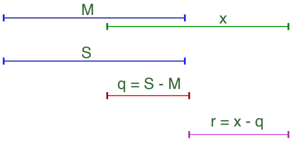
Algorithm 5 Splitting Algorithm

Require: $M, x \in \mathbb{F}, \quad M \gg |x|$

$S := M + x$ … rounded

$q := S - M$ … exactly

$r := x - q$ … rounded



Theorem 3 (Splitting Algorithm in Directed Rounding)

Let S, q, r be results computed by the Splitting Algorithm then:

- $q = \frac{1}{2} \cdot \mathtt{ulp}(M) \cdot \mathbb{Z}, \quad r \leq 2 \cdot \mathtt{ulp}(M) \leq 2 \cdot u \cdot M,$
- $S = M + q,$
- $|x - (q + r)| < 2 \cdot u^2 \cdot M.$

Faster Reproducible sum: Error bound

Theorem 4 (Reproducibility of Algorithm 4)

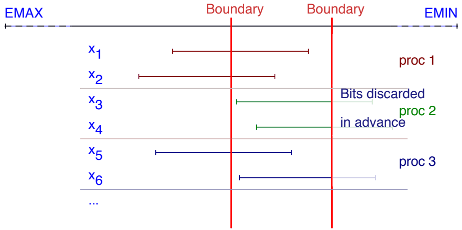
Let $T \in \mathbb{F}$ be the result computed by Algorithm 4 then:

- *all M_i have the same unit in the last place,*
- *T is always reproducible,*
- $|T - \sum_1^n x_i| < \frac{4}{1-2nu} \cdot n^2 \cdot u \cdot max|x_i|$

Cost: $2n$ `FLOPs` (counting max as a `FLOP`).

K-fold algorithms

Idea: use k extraction steps to reduce error.



$|r| \leq 2 \cdot \mathtt{u} \cdot M$: don't have to recompute $\max|x_i|$

$$M' \approx M \cdot 2n \cdot \mathtt{u}$$

	Algorithm 3_k	Algorithm 4_k
error bound	$c \cdot n^{k+1} \cdot \mathtt{u}^k \cdot max\|x_i\|$	$c \cdot n^{k+1} \cdot u^k \cdot max\|x_i\|$ $+c_2 \cdot n^2 \cdot u^2 \cdot \max\|x_i\|$
FLOPs	$4 \cdot k \cdot n$	$(3 \cdot k - 1) \cdot n$

Plan

Introduction

Algorithms

Experimental results

Conclusions and Future work

Experimental results: Reproducibility

For each input data:

- split into blocks of b summands ($b = 2^l, 32 \leq b \leq n$),
- sum each block sequentially,
- reduce the final result in different order using different tree (flat, binary).

Results computed by Algorithms 3 and 4 are always reproducible.

Results computed by standard non-reproducible summation algorithm fluctuate around the exact result with different ordering.

$$variation = \max(computed\ results) - \min(computed\ results).$$

Experimental results: Accuracy

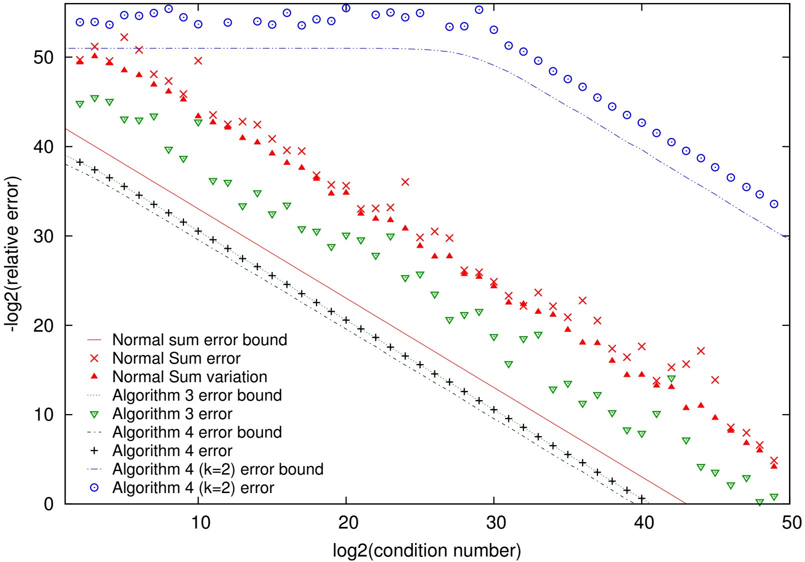
For each input data size:

- Generate input data of varying condition number $(2^0, \ldots, 2^{50})$

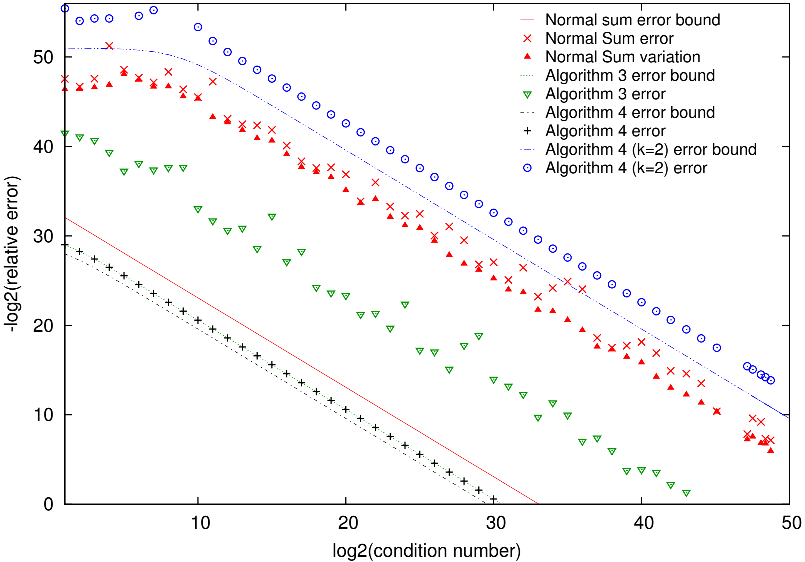
$$\textit{condition number} = \frac{\sum_1^n |x_i|}{|\sum_1^n x_i|}$$

- For each input array, sums are computed using
  - normal sum (in red),
  - Algorithm 3 (in green),
  - Algorithm 4 (in black),
  - Algorithm 4_k with $k = 2$ (in blue).
- For each algorithm, both relative error and error bound are computed.

Experimental results: Accuracy ($n = 10^3$)



Experimental results: Accuracy ($n = 10^6$)



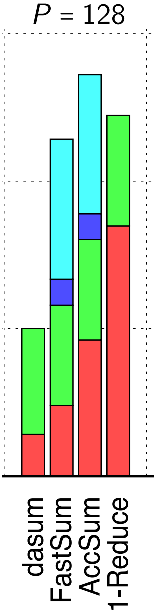
Experimental results: Performance

Sum 2^{20} floating-point numbers on Hopper machine (Cray XE6) for varying number of processors ($P = 1, 2, \ldots, 2048$)

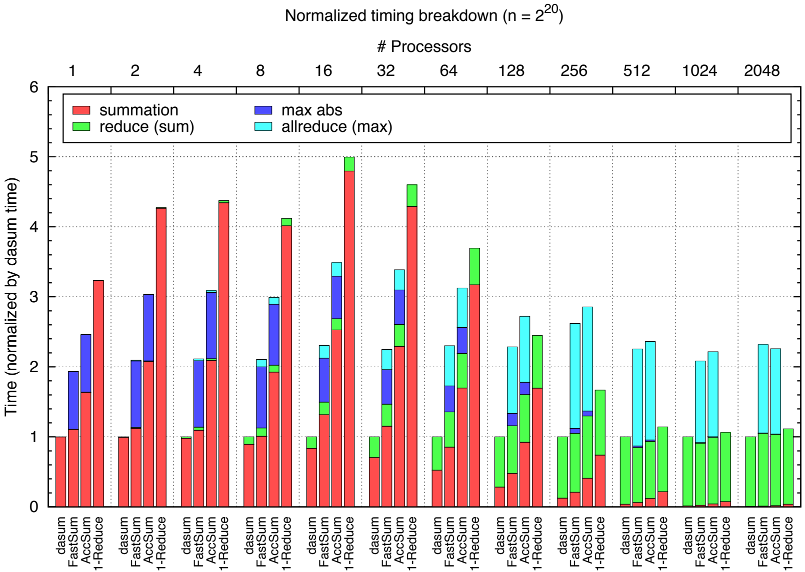
For each P, measure the running time of:

- The optimized `dasum` (normalized to 1),
- Algorithm 4_k with $k = 2$, (`FastSum`)
- Algorithm 3_k with $k = 2$, (`AccSum`)
- The 1-Reduction algorithm.

- running time to compute local summations
- communication time to reduce final result
- running time to compute local $\max |x_i|$
- communication time to reduce global $\max |x_i|$



Experimental results: Performance



Conclusions

Two reproducible summation algorithms (Reproducible Sum, Fast Reproducible Sum):

- provide bit-wise reproducibility, regardless of computation order,
- require TWO reductions
  (can be reduced to ONE using precomputed Ms),
- can be applied to other operations which use summation as the reduction operator.

Future works

In Progress

- Modify Fast Reproducible Sum for rounding-to-nearest-even,
- Parallel Prefix Sum,
- Matrix-vector / Matrix-matrix multiplication,

TODO

- Higher level driver routine: trsm, factorizations like LU, ...
- n.5D algorithms (2.5D Matmul, 2.5D LU),
- spMV,
- Other associative operations,
- Changing rounding-mode in multi-thread environment,
- Single precision operations (limit n).