Fast Reproducible Floating-Point Summation

James Demmel, Hong Diep Nguyen

ParLab - EECS - UC Berkeley

ARITH 21  April 7-10, 2013
Plan

Introduction

Algorithms

Experimental results

Conclusions and Future work
Plan

Introduction

Algorithms

Experimental results

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

→ Floating-point operations (+, ×) 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 \((\text{spmv} + \text{dot product})\), 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),
- ...
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,
  - ???
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

Introduction

Algorithms

Experimental results

Conclusions and Future work
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
3: \( T := T + x_i \)
4: end for

Objectives:

- bit-wise identical results regardless of data assignment, \# processors, reduction tree shape, \ldots \textit{(order-invariant)}
- of (almost) the same accuracy as the standard sum.

\[
\text{absolute error} \leq (n-1) \cdot \sum_{1}^{n} |x_i| \cdot u, \quad u \text{ is the unit round-off error.}
\]
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:** \( M, x \in \mathbb{F}, \ M \gg |x| \)

1: \( S := M + x \) \quad ... rounded
2: \( q := S - M \) \quad ... exactly
3: \( r := x - q \) \quad ... exactly

\[ S = q + r \quad (Error-free transformation) \]

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

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

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

**Algorithm 3 Reproducible Sum**

**Require:** $x$ is an array of size $n$

1: $X := \max_i |x_i|$
2: $j := \lfloor \log_2(n \times X/(1 - 2 \times n \times u)) \rfloor$
3: $M := \text{pow}(2.0, j)$
4: $T := 0$
5: **for** $i = 1$ **to** $n$ **do** 
6: \hspace{1em} $S := M + x_i$ \hspace{1em} ... in any order
7: \hspace{1em} $q_i := S - M$ \hspace{1em} ... rounded
8: \hspace{1em} $x_i := x_i - q_i$ \hspace{1em} ... exactly
9: \hspace{1em} $T := T + q_i$ \hspace{1em} ... exactly
10: **end for**

**Requirement:**

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

\[
M = 2^j, j \in \mathbb{Z} \text{ where } 2^j \geq \frac{n \cdot \max |x_i|}{1 - 2n \cdot u} > \sum_1^n |q_i|
\]
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,
- \( T = \sum_1^n q_i \)
- \( |T - \sum_1^n x_i| \leq \sum_1^n |r_i| \leq 2n \cdot u \cdot M < \frac{4}{1-2mu} \cdot n^2 \cdot u \cdot \max|x_i| \)

Cost: \( 4n \) FLOPs (counting max as a FLOP).
Algorithm 4 Faster Reproducible sum

Require: \( x \) is an array of size \( n \)
use directed rounding

1: \( X := \max_i |x_i| \)
2: \( j := \lceil (\log_2(n \times X/(1 - 2 \times n \times u)) \rceil \)
3: \( M := 3 \times \text{pow}(2.0, j) \)
4: \( M_1 := M \)
5: \( \textbf{for} \ i = 1 \ \textbf{to} \ n \ \textbf{do} \) \( \ldots \) in any order
6: \( M_{i+1} := M_i + x_i \) \( \ldots \) rounded
7: \( q_i := M_{i+1} - M_i \) \( \ldots \) exactly
8: \( x_i := x_i - q_i \) \( \ldots \) rounded
9: \( \textbf{end for} \)
10: \( T := M_{n+1} - M \) \( \ldots \) 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: $M, x \in \mathbb{F}, \quad M \gg |x|$
1: $S := M + x$ \quad ... rounded
2: $q := S - M$ \quad ... exactly
3: $r := x - q$ \quad ... rounded

Theorem 3 (Splitting Algorithm in Directed Rounding)

Let $S, q, r$ be results computed by the Splitting Algorithm then:
- $q = \frac{1}{2} \cdot \text{ulp}(M) \cdot \mathbb{Z}$, \quad $r \leq 2 \cdot \text{ulp}(M) \leq 2 \cdot u \cdot M$,
- $S = M + q$,
- $|x - (q + r)| < 2 \cdot u^2 \cdot M$. 
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.

$|r| \leq 2 \cdot u \cdot M$: don’t have to recompute $\max |x_i|$

$M' \approx M \cdot 2n \cdot u$

<table>
<thead>
<tr>
<th></th>
<th>Algorithm $3_k$</th>
<th>Algorithm $4_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>error bound</td>
<td>$c \cdot n^{k+1} \cdot u^k \cdot \max</td>
<td>x_i</td>
</tr>
<tr>
<td>FLOPs</td>
<td>$4 \cdot k \cdot n$</td>
<td>$(3 \cdot k - 1) \cdot n$</td>
</tr>
</tbody>
</table>
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.

\[
\text{variation} = \max(\text{computed results}) - \min(\text{computed results}).
\]
Experimental results: Accuracy

For each input data size:

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

\[
\text{condition number} = \frac{\sum_1^n |x_i|}{|\sum_1^n x_i|}
\]

- 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 \((n = 10^6)\)

The graph shows the relationship between \(-\log_2(\text{relative error})\) and \(\log_2(\text{condition number})\) for various algorithms and error bounds.

- Normal sum error bound
- Normal sum error
- Normal sum variation
- Algorithm 3 error bound
- Algorithm 3 error
- Algorithm 4 error bound
- Algorithm 4 error
- Algorithm 4 (k=2) error bound
- Algorithm 4 (k=2) error
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 \texttt{dasum} (normalized to 1),
- Algorithm $4_k$ with $k = 2$, (\texttt{FastSum})
- Algorithm $3_k$ with $k = 2$, (\texttt{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

Normalized timing breakdown ($n = 2^{20}$)

- Summation
- Reduce (Sum)
- Max Abs
- Allreduce (Max)

Time (normalized by dasum time) vs. # Processors

- dasum
- FastSum
- AccSum
- 1-Reduce
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 $n$).