A Reproducible Accurate Summation Algorithm for High-Performance Computing

Sylvain Collange\textsuperscript{1}, David Defour\textsuperscript{2}, Stef Graillat\textsuperscript{4}, and Roman Iakymchuk\textsuperscript{3,4}

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

roman.iakymchuk@lip6.fr

The SIAM EX14 Workshop
July 6th, 2014
Chicago, Illinois, USA
The Patriot Missile Failure

- The 1st Gulf War in 1991: an American Patriot missile battery failed to intercept an Iraqi Scud missile.
- The Scud missile hit a US garrison, killing 28 soldiers.

Analysis

- The Patriot HW clock delivers time in $\frac{1}{10}$ths of seconds.
- $0.1$ is not representable by a finite number of digits in basis 2. $0.1 = 0.0001100110011001100110011001100...$
- The Patriot system had been running for more than 100 hours. Time off was $10 \cdot 100 \cdot 3600 \cdot 5.96 \cdot 10^{-8} = 0.21$ secs.
- In this time, a Scud missile travels roughly 360 m.
1. Computer Arithmetic: Accuracy and Reproducibility
2. Existing Solutions
3. Multi-Level Reproducible and Accurate Algorithm
4. Conclusions and Future Work
Problems

- Floating-point arithmetic suffers from rounding errors.
- Floating-point operations (+, ×) are commutative but non-associative.

\[(−1 + 1) + 2^{-53} \neq −1 + (1 + 2^{-53})\] in double precision.
Floating-point arithmetic suffers from rounding errors.

Floating-point operations (+, ×) are commutative but non-associative.

\[ 2^{-53} \neq 0 \quad \text{in double precision} \]
Floating-point arithmetic suffers from rounding errors.

Floating-point operations (+, ×) are commutative but non-associative.

\((-1 + 1) + 2^{-53} \neq -1 + (1 + 2^{-53})\) 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 frequently inconsistent: each run returns a different result.
Reproducibility and ExaScale

Challenges

- **Increasing power** of current computers
  - GPU accelerators, Intel Phi processors, etc.

- Enable to solve more **complex problems**
  - Quantum field theory, supernova simulation, etc.

- A **high number** of floating-point **operations** performed
  - Each of them leads to round-off error
Reproducibility and ExaScale

Challenges

- **Increasing power** of current computers
  - GPU accelerators, Intel Phi processors, etc.

- Enable to solve more complex problems
  - Quantum field theory, supernova simulation, etc.

- A high number of floating-point operations performed
  - Each of them leads to round-off error

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, ...)

A performance-optimized floating-point library is 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
  - . . .
A performance-optimized floating-point library is 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
  - ...
Existing Solutions

To Obtain Reproducibility

- Fix the Order of Computations
  - Sequential mode: intolerably costly at large-scale systems
  - Fixed reduction trees: substantial communication overhead
  - Example: Intel **Conditional Numerical Reproducibility**
    (slow, no accuracy guarantees)
Existing Solutions

To Obtain Reproducibility

- **Fix the Order of Computations**
  - Sequential mode: intolerably costly at large-scale systems
  - Fixed reduction trees: substantial communication overhead
    → Example: Intel **Conditional Numerical Reproducibility**
      (slow, no accuracy guarantees)

- **Eliminate/Reduce the Rounding Errors**
  - Fixed-point arithmetic: limited range of values
  - Fixed **FP expansions with Error-Free Transformations** (EFT)
    → Example: double-double or quad-double (Briggs, Bailey, Hida, Li)
      (work well on a set of relatively close numbers)
Existing Solutions

To Obtain Reproducibility

- **Fix the Order of Computations**
  - Sequential mode: intolerably costly at large-scale systems
  - Fixed reduction trees: substantial communication overhead
    → Example: Intel Conditional Numerical Reproducibility (slow, no accuracy guarantees)

- **Eliminate/Reduce the Rounding Errors**
  - Fixed-point arithmetic: limited range of values
  - Fixed FP expansions with Error-Free Transformations (EFT)
    → Example: double-double or quad-double (Briggs, Bailey, Hida, Li) (work well on a set of relatively close numbers)
  - “Infinite” precision: reproducible independently from the inputs
    → Example: Kulisch accumulator (considered inefficient)
Our Approach

Algorithm 1 EFT of size 2
(Dekker and Knuth)

function \[ r, s \] = TwoSum(\( a, b \))

1: \( r \leftarrow a + b \)
2: \( z \leftarrow r - a \)
3: \( s \leftarrow (a - (r - z)) + (b - z) \)
Our Approach

Algorithm 1 EFT of size 2 (Dekker and Knuth)

function \[ r, s \] = TwoSum(\( a, b \))

1: \( r \leftarrow a + b \)
2: \( z \leftarrow r - a \)
3: \( s \leftarrow (a - (r - z)) + (b - z) \)

Algorithm 2 EFT of size \( n \) (init. by Priest and Shewchuk)

function = ExpansionAccumulate(\( x \))

1: \( \text{for } i = 0 \rightarrow n - 1 \text{ do} \)
2: \( (a_i, x) \leftarrow \text{TwoSum}(a_i, x) \)
3: \( \text{end for} \)
4: \( \text{if } x \neq 0 \text{ then} \)
5: \( \text{Superaccumulate}(x) \)
6: \( \text{end if} \)
Our Approach

Algorithm 1 EFT of size 2
(Dekker and Knuth)

function \([r, s] = \text{TwoSum}(a, b)\)
1: \(r \leftarrow a + b\)
2: \(z \leftarrow r - a\)
3: \(s \leftarrow (a - (r - z)) + (b - z)\)

Algorithm 2 EFT of size \(n\) (init. by Priest and Shewchuk)

function = ExpansionAccumulate\((x)\)
1: \(\text{for } i = 0 \rightarrow n - 1 \text{ do}\)
2: \((a_i, x) \leftarrow \text{TwoSum}(a_i, x)\)
3: \(\text{end for}\)
4: \(\text{if } x \neq 0 \text{ then}\)
5: \(\text{Superaccumulate}(x)\)
6: \(\text{end if}\)
Our Multi-Level Algorithm

Objective: To compute deterministic sums of floating-point numbers efficiently and with the best possible accuracy.

Accurate and Reproducible Parallel Summation:

- Based on FP expansions with EFT and Kulisch accumulator
- Parallel algorithm with 5-levels
- Suitable for today’s parallel architectures
- Guarantees “infinite” precision = bit-wise reproducibility
Level 1: Filtering

Input numbers

Thread 1

- EFT
- FP Expansion (register)

UnderFlow?

Thread 2

- EFT
- FP Expansion (register)

UnderFlow?

Thread n

- EFT
- FP Expansion (register)

Level 1 (Filtering)

Input numbers

Thread 1

- EFT
- FP Expansion (register)

Thread 2

- EFT
- FP Expansion (register)

Level 1 (Filtering)
Level 2 and 3: Scalar Superaccumulator

Roman Iakymchuk (ICS & LIP6, UPMC)

Reproducible Accurate Summation

July 6th, 2014
Level 4 and 5: Reduction and Rounding

Input numbers

Thread 1
  EFT
  FP Expansion (register)
  Underflow?

Thread 2
  EFT
  FP Expansion (register)
  Underflow?

... 

Thread n
  EFT
  FP Expansion (register)
  Underflow?

Level 1 (Filtering)

Level 3 (Scalar SuperAccumulation)

Level 4 (Parallel Reduction)

Level 5 (Rounding)
## Experimental Environments

Table: Hardware platforms employed in the experimental evaluation\(^a\).

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Intel Core i7-4770 (Haswell)</td>
<td>4 cores with HT</td>
</tr>
<tr>
<td>B</td>
<td>Intel Xeon E5-2450 (Sandy Bridge-EN)</td>
<td>2 × 8 cores</td>
</tr>
<tr>
<td>C</td>
<td>Intel Xeon Phi 3110P</td>
<td>60 cores × 4-way MT</td>
</tr>
<tr>
<td>D</td>
<td>NVIDIA Tesla K20c</td>
<td>13 SMs × 192 CUDA cores</td>
</tr>
<tr>
<td>E</td>
<td>AMD Radeon HD 7970</td>
<td>32 CUs × 64 units</td>
</tr>
</tbody>
</table>

\(^a\)S. Collange, D. Defour, S. Graillat and R. Iakymchuk. Full-Speed Deterministic Bit-Accurate Parallel Floating-Point Summation on Multi- and Many-Core Architectures, Feb, 2014. HAL-ID: hal-00949355
Performance Results on Intel Phi

Parallel Summation: Performance Scaling

![Graph showing performance results on Intel Phi, comparing Parallel FP sum, TBB deterministic, Superaccumulator, Expansion 2, Expansion 3, Expansion 4, and Expansion 8 early-exit.](image-url)
Performance Results on Intel Phi

Parallel Summation: Data-Dependent Performance

Roman Iakymchuk (ICS & LIP6, UPMC)
Performance Results on NVIDIA Tesla
Parallel Summation: Performance Scaling

![Graph showing performance results for different summation methods.](image-url)

- **Parallel FP Sum**
- **Superaccumulator**
- **Expansion 2**
- **Expansion 3**
- **Expansion 4**
- **Expansion 8**
- **Expansion 8 early-exit**
Conclusions

The Proposed Multi-Level Summation Algorithm

- Computes the results with no errors due to rounding
- Provides bit-wise identical reproducibility, regardless of
  - Data permutation, data assignment
  - Thread scheduling, etc.
Conclusions

The Proposed Multi-Level Summation 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
- Scale perfectly with the increase of the problem size or the number of cores
Conclusions

The Proposed Multi-Level Summation 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
- Scale perfectly with the increase of the problem size or the number of cores
- Can be applied to other operations which use summation or dot product
- Is suitable for very large scale systems (ExaScale) with one more reduction step between nodes
Future Work

**ExBLAS – Exact BLAS**

- ExBLAS-1: ExSCAL, ExDOT, ExAXPY, ...
- ExBLAS-2: ExGER, ExGEMV, ExSYR, ...
- ExBLAS-3: ExGEMM, ExTRMM, ExSYR2K, ...
Future Work

**DDOT:** \( \alpha := x^T y = \sum_{i}^{N} x_i y_i \)
Future Work

ExBLAS – Exact BLAS

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

Distributed architectures

- Parallelization with MPI
- Computation on network cards
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).
S. Collange, D. Defour, S. Graillat, and R. Iakymchuk
Numerical Reproducibility for the Summation Problem on Multi- and Many-Core Architectures. Submitted to the Parallel Computing Journal

S. Collange, D. Defour, S. Graillat, and R. Iakymchuk
Full-Speed Deterministic Bit-Accurate Parallel Floating-Point Summation on Multi- and Many-Core Architectures, Tech report, Feb, 2014. HAL-ID: hal-00949355

J. Demmel and H.D. Hguyen

J. Demmel and H.D. Hguyen

J. Demmel and H.D. Hguyen
Numerical reproducibility and accuracy at ExaScale (invited talk), the 21st IEEE Symposium on Computer Arithmetic, Austin, Texas, USA, 2013

A. Arteaga, O. Fuhrer, and T.Hoefler
Designing Bit-Reproducible Portable High-Performance Applications. Proceedings of the 28th IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2014

G. Michelogiannakis, X.S. Li, D.H. Bailey, and J. Shalf