Reproducibility of parallel Floating-point computations

James Demmel, Nguyen Hong Diep

ParLab - EECS - UC Berkeley

IWASEP 9, June 7, 2012
Motivations (1)

Floating-point arithmetic: defines a discontinuous 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.

Results computed by performance-optimized parallel floating-point libraries may be frequently inconsistent: each run returns a different result.
Motivations (2)

Demands for consistent 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: \(^1\) Gauss-Seidel iterations (spmv + dot product), after the 5th step the Euclidean norm of the residual vector differs up to 20% from one run to another.

Sources of inconsistency

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

- Changing Data Layouts:
  - Data alignment,
  - Data partitioning,

- Changing Hardware Resources:
  - Number of threads,
  - Fused Multiply-Adder support,
  - Intermediate precision (64bits, 80 bits, 128 bits, etc),
  - SIMD register size,
  - Cache line size,
  - ???
Sources of inconsistency

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

- **Changing Data Layouts:**
  - *Data alignment*,
  - *Data partitioning*,

- **Changing Hardware Resources:**
  - *Number of threads*,
  - Fused Multiply-Adder support,
  - Intermediate precision (64bits, 80 bits, 128 bits, etc),
  - SIMD register size,
  - Cache line size,
  - ???

Our goal is to obtain consistent result from run-to-run on the same machine (**repeatability**)

Non-repeatability of the MKL library

Case study: MKL v10.3 ddot function.

Test configurations: Mac OS, Intel Core i7 2.2 GHz, 4 cores, 8 GB Memory, 32 KB L1 Cache, 256 KB L2 Cache (per Core), 6 MB L3 Cache.

Input vectors $a$ and $b$ are generated randomly. For each pair $a, b$, compute $v_j = a \times b$ for varying numbers of threads ($j = 1, 2, 3, 4$). Non-repeatability is measured by:

$$\text{variation} \quad = \quad \max(v_j) - \min(v_j) \quad \text{(ulps}^1 \text{ in } \max(|v_j|))$$

$$\text{relative error} \quad = \quad \frac{\max(v_j) - \min(v_j)}{\max(|v_j|)}$$

---

$^1$ulp: units in the last place
MKL ddot: no cancellation, varying numbers of threads

size: $10^5$  $a, b$ uniform distribution $[0.0, 1.0)$

histogram of variation in ulps over 25000 input data.
MKL ddot: orthogonal vectors, varying numbers of threads

size: $10^5$  \[ a = \text{rand}(n), \ b = \text{rand}(n), \ a = a - (a' \ast b) \ast b / (b' \ast b) \]

histogram of relative error over 25000 input data.
MKL ddot: orthogonal vectors, varying numbers of threads

size: $10^5$  \[ a = \text{rand}(n), \ b = \text{rand}(n), \ a = a - (a' \ast b) \ast b / (b' \ast b) \]

histogram of relative error over 25000 input data.

different signs
MKL library: consistent results of computations

(MKL’s user guide) To better assure identical results from run-to-run, do the following:

- Align input arrays on 16-byte boundaries,
- Run Intel MKL in the sequential mode.

What if we want to run it fast?
Solutions

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

To obtain repeatability:
Source of floating-point inconsistency: rounding errors lead to dependence of computed result on *order of computations*.

To obtain repeatability:

- Fix the order of computations:
  - sequential order,
  - reproducible reduction tree,
Solutions

Source of floating-point inconsistency: *rounding errors* lead to dependence of computed result on *order of computations*.

To obtain repeatability:

- Fix the order of computations:
  - sequential order,
  - reproducible reduction tree,
- Eliminate/Reduce the rounding errors:
  - exact arithmetic,
  - correct rounding for the overall operation,
  - higher precision,
Solutions

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

To obtain repeatability:

- Fix the order of computations:
  - sequential order,
  - reproducible reduction tree,
- Eliminate/Reduce the rounding errors:
  - exact arithmetic,
  - correct rounding for the overall operation,
  - higher precision,
Reproducible reduction tree

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

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

- Tree height is of order \( \log(n) \), \( n \) is number of chunks
- No need to be stored explicitly,
- Total number of data transfers (intra/inter-node): \( \approx n \).
Best case: nb of inter-node communications is $p - 1$
Reproducible reduction tree: Properties (2)

- Best case: nb of inter-node communications is $p - 1$
- Wost case: nb of inter-node communications is $\mathcal{O}(p \log(n))$
- Additional memory storage: $\log(n/p)$
Reproducible reduction tree: ddot experiments

Configurations:
- Nehalem machine (Xeon X5550) 8 cores, 32KB L1 cache, 256KB L2 cache, 8MB L3 cache, 12GB DRAM,
- Intel Compiler icc v.11.1, OpenMP 3.0,
- Chunk size: 128,
- Max sequential size: 2432,
- Medium size: ≤ 64 chunks,
- Compared with Intel MKL v.10.3.
Reproducible reduction tree: ddot experiments
Higher precision

**Idea**: Use higher precision for intermediate results.

![Diagram showing 106 and 53]

Summation: Kahan’s algorithm, Knuth’s algorithm, extended precision (for ex. double-double), etc.

**Problem**: higher precision is *expensive*.

**Solution**: split data into fixed-size chunks, use working precision to perform computation of each chunk, only use higher precision to accumulate intermediate results.
Higher precision: ddot experiments

dot product of vectors of size 100000, chunk size 256.
Higher precision: matrix multiplication (sketch)

\[ T_{i,j} = A_{i,k} \times B_{k,j} \]

\[ T_{i,j} \text{ working precision} += T_{i,j} \text{ higher precision} \]
Use of correctly rounded / exact arithmetic.
More operations: BLAS, SpMV, factorizations like LU, . . .
Implementation for distributed system,
n.5D algorithms,
Debugging tool: automatically locates anomalies and sources of non-consistency of the program \(^2\).

\(^2\)David H. Bailey, James Demmel, William Kahan, Guillaume Revy, and Koushik Sen, *Techniques for the automatic debugging of scientific floating-point programs*, SCAN'2010