Cost of Floating-Point Reproducibility

James Demmel, **Hong Diep Nguyen**, Peter Ahrens

Numerical Reproducibility at Exascale 2015 - Austin, TX

Nov 20, 2015

Reproducibility

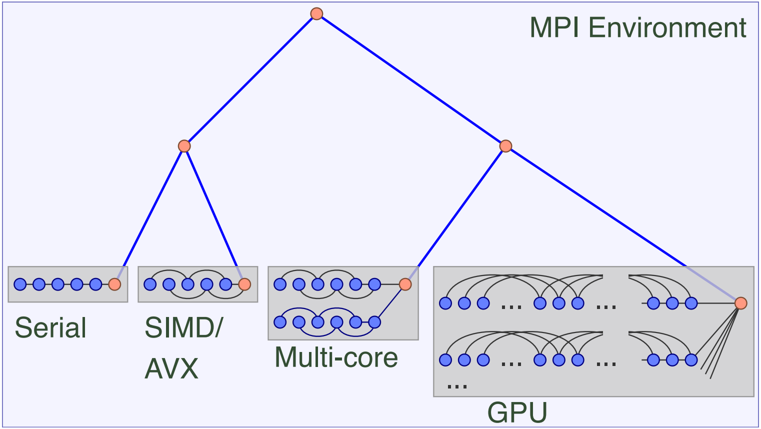
Reproducibility: obtaining bit-wise identical results from different runs of the program on the same input data, regardless of different available resources.
Reproducibility is needed for debugging and for understanding the reliability of the program.

Cause of nonreproducibility: *not* by roundoff error but by the *non-determinism* of accumulative roundoff error.

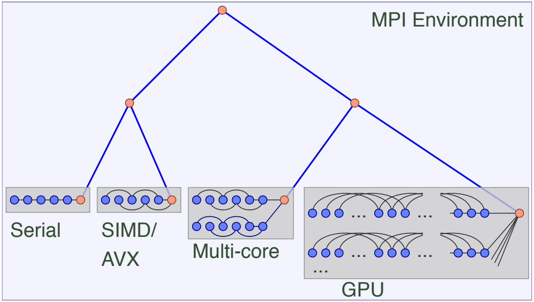
Due to the *non-associativity* of floating point addition, accumulative roundoff errors depend on the order of evaluation, and therefore depend on available computing resources.

$$\mathtt{fl}(\mathtt{fl}(a+b)+c) \neq \mathtt{fl}(a+\mathtt{fl}(b+c))$$

Sources of non-reproducibility

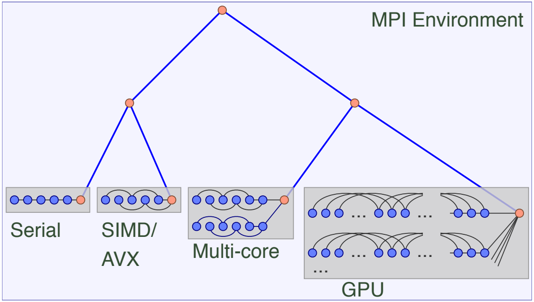


Sources of non-reproducibility



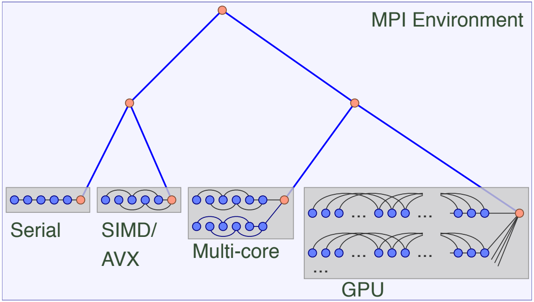
- number of MPI nodes

Sources of non-reproducibility



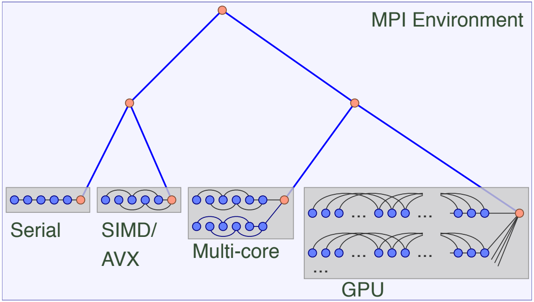
- number of MPI nodes
- MPI reduction tree shape

Sources of non-reproducibility



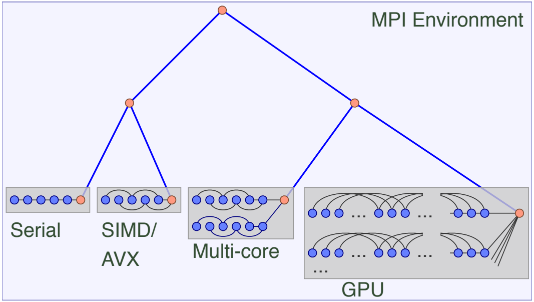
- number of MPI nodes
- MPI reduction tree shape
- number of cores per node

Sources of non-reproducibility



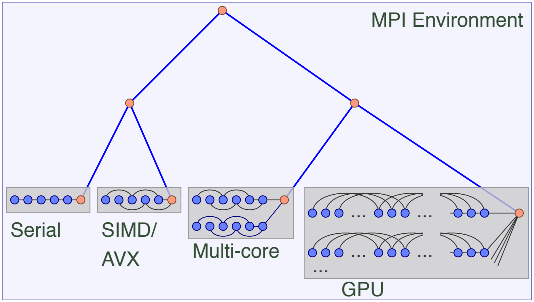
- number of MPI nodes
- MPI reduction tree shape
- number of cores per node
- data path (1,2,4,. . .)

Sources of non-reproducibility



- number of MPI nodes
- MPI reduction tree shape
- number of cores per node
- data path (1,2,4,. . .)
- data ordering

Sources of non-reproducibility

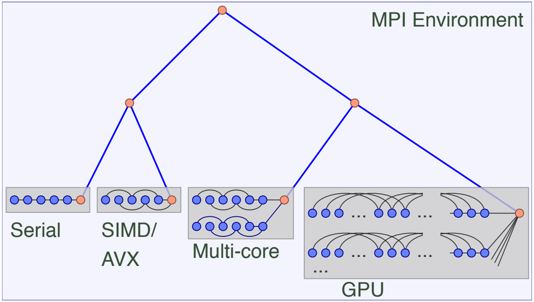


- number of MPI nodes
- MPI reduction tree shape
- number of cores per node
- data path (1,2,4,. . .)
- data ordering

Related work:

- Intel MKL 11.0 with CNR: reproducible with fixed number of processors, fixed instructions set, ~~and fixed data alignment~~,

Sources of non-reproducibility



- number of MPI nodes
- MPI reduction tree shape
- number of cores per node
- data path (1,2,4,...)
- data ordering

Related work:

- Intel MKL 11.0 with CNR: reproducible with fixed number of processors, fixed instructions set, ~~and fixed data alignment~~,
- Hardware (NIC) for reproducible reduction operator.

Solutions

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

Solutions

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

Solutions: fix evaluation order

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:

Solutions: fix evaluation order

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:

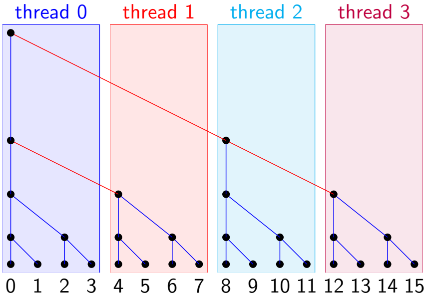
- sequential mode: *intolerably costly at ExaScale*

Solutions: fix evaluation order

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 ExaScale*
- fixed reduction tree: *substantial communication overhead*



Solutions: Eliminate rounding errors

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

To obtain reproducibility: Eliminate/Reduce rounding errors

Solutions: Eliminate rounding errors

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

To obtain reproducibility: Eliminate/Reduce rounding errors

- fixed-point arithmetic: *limited range of values*

Solutions: Eliminate rounding errors

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

To obtain reproducibility: Eliminate/Reduce rounding errors

- fixed-point arithmetic: *limited range of values*
- exact arithmetic (rounded at the end): *expensive in communication and arithmetic on long words*

Solutions: Eliminate rounding errors

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

To obtain reproducibility: Eliminate/Reduce rounding errors

- fixed-point arithmetic: *limited range of values*
- exact arithmetic (rounded at the end): *expensive in communication and arithmetic on long words*
- higher precision: *reproducible with high probability (not certain)*.

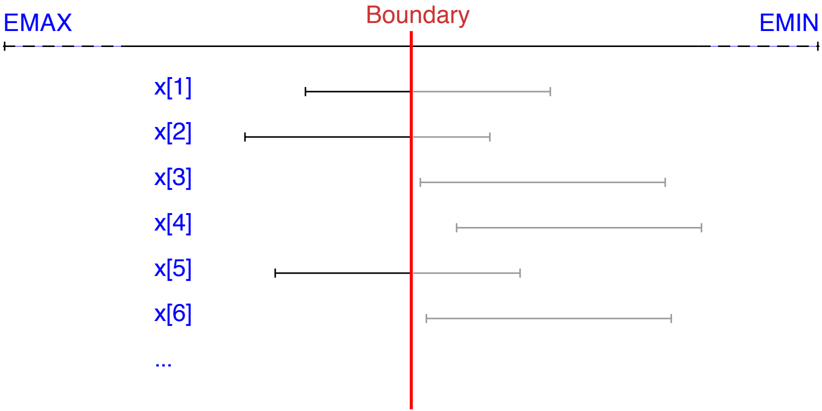
A proposed solution for global sum

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.

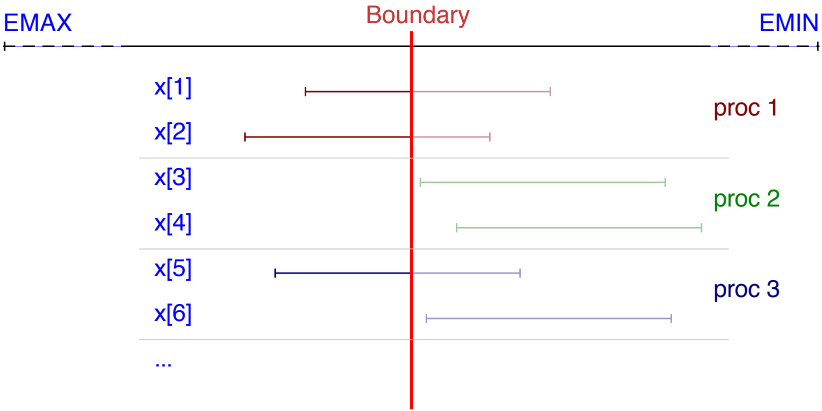
Idea: pre-rounding input values.

Pre-rounding technique



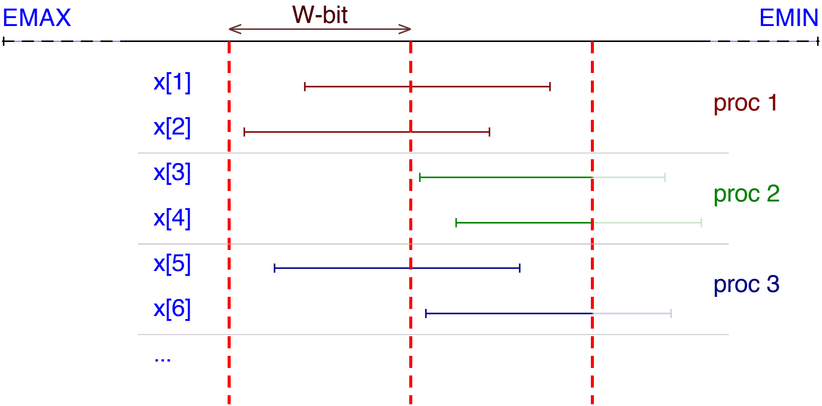
No rounding error at each addition. Computation's error depends on the Boundary, which depends on max $|x_i|$

Pre-rounding technique



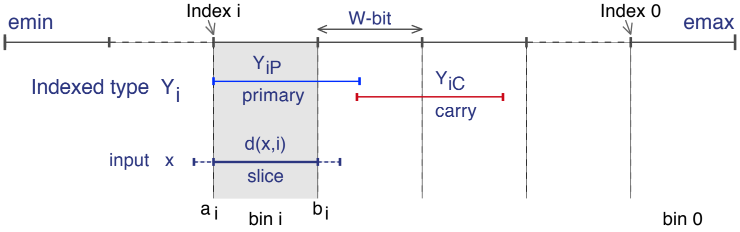
No rounding error at each addition. Computation's error depends on the Boundary, which depends on $\max |x_i|$

Pre-rounding technique



No rounding error at each addition. Computation's error depends on the Boundary, which depends on $\max |x_i|$

Indexed Floating-Point format



- The exponent range is divided into bins of contiguous bits.
- Each input is split into several bins.
- Values in each bin are summed correctly.
- Only a number of greatest bins are kept.

Indexed Floating-Point: Accuracy

Tunable with:

- $\mathtt{K}$: number of bins,
- $\mathtt{W}$: number of bits in each bin.

Absolute error bound:

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

In practice, for double precision, $\mathtt{K} = 3$, $\mathtt{W} = 40$:

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

Experimental results: Accuracy

Summation of $n = 10^6$ floating-point numbers. Computed results of both reproducible summation and standard summation are compared with result computed using quad-double precision.

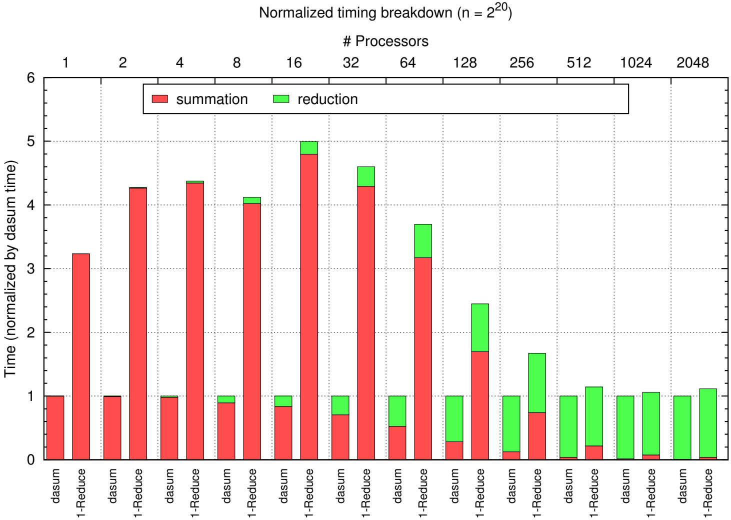
Condition number: $\kappa = \sum_1^n |x[i]| / |\sum_1^n x[i]|$.

Generator $x[i]$	reproducible	standard	κ
`drand48()`	0	3.0×10^{-15}	1
`drand48()` $- 0.5$	1.5×10^{-16}	1.3×10^{-13}	1.8×10^{3}
$\sin(2.0 * \pi * i/n)$	1.5×10^{-15}	1.0	1.9×10^{19}

Table : Relative error of summation algorithms

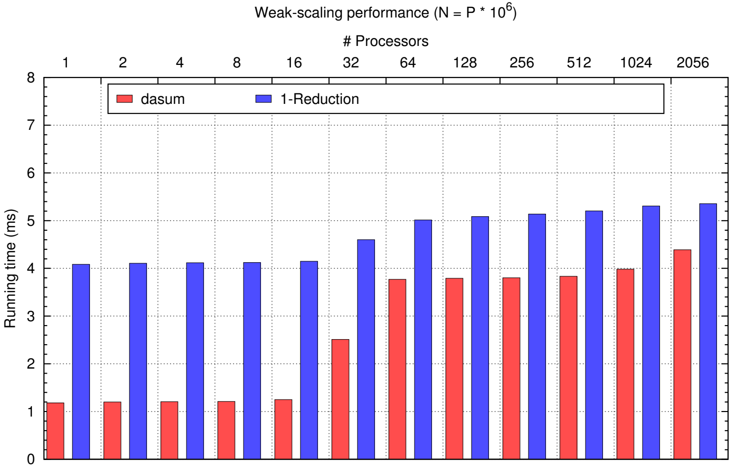
Experimental results: strong-scaling

On Edison, Cray XC30 machine at NERSC (5576 x 12-core Intel "Ivy Bridge" processors)



Experimental results: weak-scaling

On Edison, Cray XC30 machine at NERSC (5576 x 12-core Intel "Ivy Bridge" processors)



Praticality of Reproducibility

`ReproBLAS` / Exact arithemtic:

- provides rigorous reproducibility for random evaluation order
- requires only 1 reduction operation: performs well for the case of parallel summation.

Obstacle: they still run much slower (**8x** for `ReproBLAS`) than performance-optimized nonreproducible counterparts for local computation. The slowdown is more prominent for higher order operations such as matrix multiplication.

Question:

- Do we need to obtain high accuracy? **Not always**.
- Do we need to accommodate totally random evaluation order? **Not neccesarily**, for example matrix computation.

On-going work

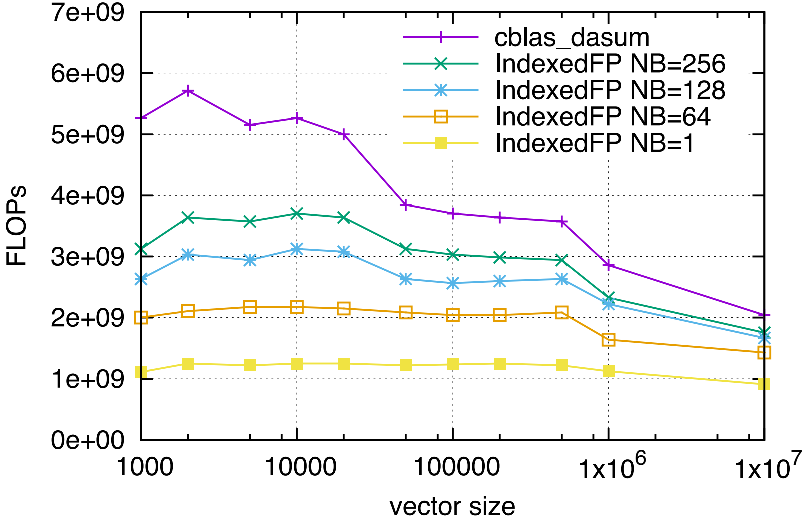
Idea: relax the requirement to accomodate random order of evaluation.

- Input data are split into blocks of fixed size. Data in each block are fixed.
- Local computation of each block can be done using performance-optimized operations.
- Only use reproducible techniques to reduce final result.

Caveats

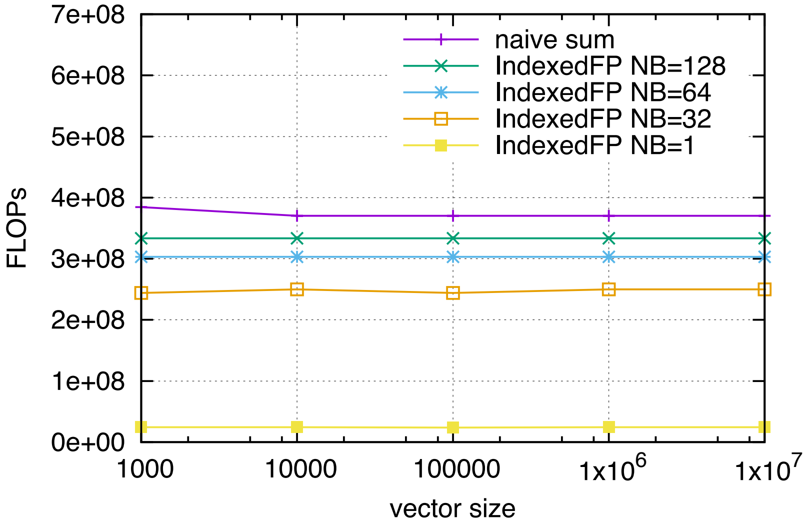
- Computed results can be at most as accurate as the corresponding standard floating-point operations. `ReproBLAS` can be tuned to obtain certain accuracy.
- Floating-point operations are not well optimized for small size (`libxsmm`).

Example: performance optimized summation



Machine: Sandy Bridge, Intel(R) Core(TM) i7-2600 @ 3.4GHz.

Example: non-optimized summation



Machine: Sandy Bridge, Intel(R) Core(TM) i7-2600 @ 3.4GHz.

On-going work

Applicable to:

- gemv
- gemm

Questions

- What would be the best block-size
- What would be the impact if the input are not well aligned.