Towards High Performance Stochastic Arithmetic

Pacôme Eberhart
Julien Brajard Pierre Fortin Fabienne Jézéquel

Université Pierre et Marie Curie, LIP6 & LOCEAN - France

SCAN2014
September 23, 2014
Würzburg, Germany

Estimation of rounding error propagation

Evaluating the accuracy of numerical results

- Accumulation of rounding errors $\Rightarrow$ numerical results different from mathematical results
- Measure of the reliability and reproducibility of the computation
- Particularly important in HPC environments and future exascale supercomputers

Some methods

- Backward error analysis: low oberhead, but not suited to every type of code
- Interval arithmetic: 100% accurate but usually needs code rewriting
- Stochastic arithmetic: probabilistic approach easy to use in real-life applications

Stochastic arithmetic and HPC

High performance numerical validation

- New hardware architectures
- More computing resources available for scientific computing
- Need for high performance numerical validation

SIMD (Single Instruction Multiple Data)

- Instruction executed over different data at the same time
- Instruction set extensions for CPUs
  - SSE (128-bit wide vector)
  - AVX (256-bit wide vector)
- Dedicated hardware
  - Intel Xeon Phi (512-bit wide vector)
  - GPUs

Stochastic arithmetic

CESTAC method

- Each arithmetic operation is performed N times
- Randomly rounded towards $+\infty$ or $-\infty$ with probability 0.5
- Computed result R is the set of N samples R_i
- Value $\overline{R}$ of the computed result is the mean of $\{R_i\}$
- Number of exact significant digits, $C_{\overline{R}}$, estimated within a 95% confidence interval

Validity of $C_{\overline{R}}$

- Compromised if both operands in a multiplication are not significant
- Likewise if a divisor is not significant

The CADNA library

Implementation

- Implementation of stochastic arithmetic in C/C++
- Classes and operator overloading for ease of use

Data types

- Classes `float_st` and `double_st` for single and double precision
- Contain $N = 3$ floating-point values ($\{R_i\}$) of the corresponding type
- Arithmetic and relational operators overloaded with stochastic ones

The CADNA library: self-validation and anomaly detection

Anomaly detection

- Self-validation to ensure validity of stochastic arithmetic
- Anomaly detection for numerical analysis of the code

Warning types

- **Self-validation**: both operands in a multiplication or a divisor not significant
- **Cancellation detection**: sudden loss in accuracy on addition or subtraction
- **Mathematical instability**: instability in a mathematical function
- **Branching instability**: undeterminism in a branching test

Overhead

Computation time

- Depends on the program and the level of detection
- Is usually one order of magnitude higher on real-life applications
- Even higher on highly optimised routines

Causes

- Cost of anomaly detection
- Cost of stochastic operations

Cost of anomaly detection

Detection types

- Self-validation and branching instability: relatively low cost test ($C_{\overline{R}} \leq 0$)
- Mathematical instability: inexpensive compared to the cost of mathematical function calls
- Cancellation detection: computing the number of exact significant digits of both operands and the result

Calculating the number of exact significant digits

- Uses the mean value and the standard deviation of $\{R_i\}$
- Relies on a costly logarithmic evaluation

Cost of stochastic operations

FPU (Floating Point Unit) rounding modes

- Stochastic operations frequently change the rounding mode of the FPU
- Pipeline flushed when rounding mode changed, hence hindering performance
- Prevents vectorisation as rounding mode is the same for all lanes

Overloaded operators

- Operators replaced by functions, compiled in the library
- FPU instructions replaced by function calls, causing performance overhead, especially in arithmetic intensive codes

Cancellation detection

Logarithm approximation

- Cancellation detection: number of exact significant digits computed with log_{10}
- Using the base 2 exponent (multiplied by $log_{10}(2)$) as a fast approximation for logarithm
- Easily obtained from binary representation of floating point numbers

Difference with the previous evaluation

- Estimated number of exact significant digits can vary
- However, since $log_{10}(2) < 0.31$, at most a 1 digit difference
- Approximation gives a more pessimistic estimation for number of digits

Stochastic operations

Sign handling

- As $X + Y = -(-X + -Y)$ (likewise for subtraction),
- And $X \times Y = -(X \times -Y)$ (likewise for division),
- Obtain rounded up value from rounded down operations (or conversely) by changing signs

Efficient implementation for random sign change

- Multiplying operands and result by -1 creates branching (from the 0.5 random probability), possibly inefficient to vectorise
- Random flip of the bit sign of the IEEE binary representation removes branching

Inlining the functions

- Minimise the cost of function calls
- Recompile CADNA library with application

Vectorising CADNA

Prerequisites

- FPU rounding mode changes not necessary anymore
- Sequential random generator changed to support vectorisation
- Some variables duplicated for each lane (anomaly counters, ...)

Vectorising methods

- Using intrinsics: tedious and difficult to use due to data types
- icc automatic vectorisation and OpenMP 4.0 problematic due to random generator

SPMD (Single Program Multiple Data)

- Scalar programming with simple C-like syntax
- Compiler generates SIMD instructions
- ispc (Intel SPMC Program Compiler) supports operator overloading, chosen over OpenCL

Performance setup

Hardware

- Intel Xeon E3-1275 (with AVX)
- 3.5GHz, 1 core used only

Example of test code: term by term addition

```
for(i = 0; i < size; i++){
  for(j = 0; j < intensity; j++){
    a[i] = a[i] + b[i]
  }
}
```

Performance setup

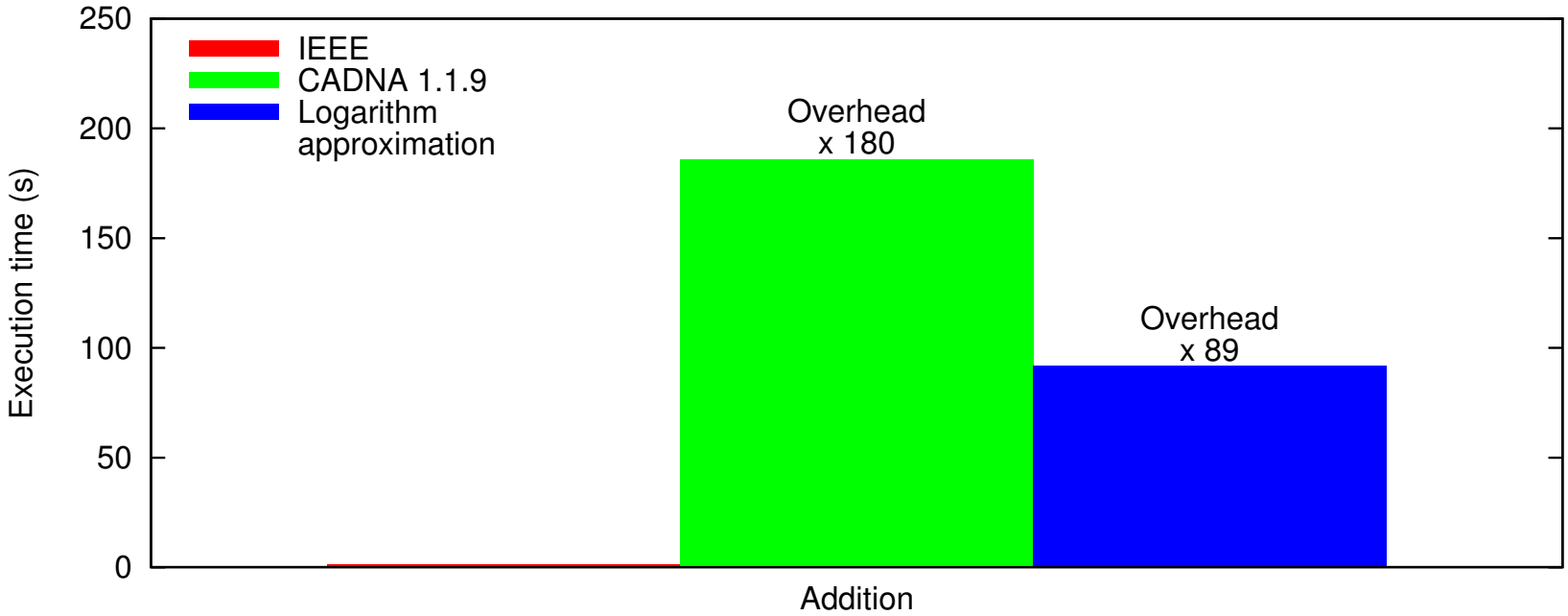
Hardware

- Intel Xeon E3-1275 (with AVX)
- 3.5GHz, 1 core used only

Test programs

- Arrays chosen large enough to not fit inside CPU cache
- Single precision floating point and stochastic numbers
- Compiled with gcc -O2
- Execution time measured by wall-clock

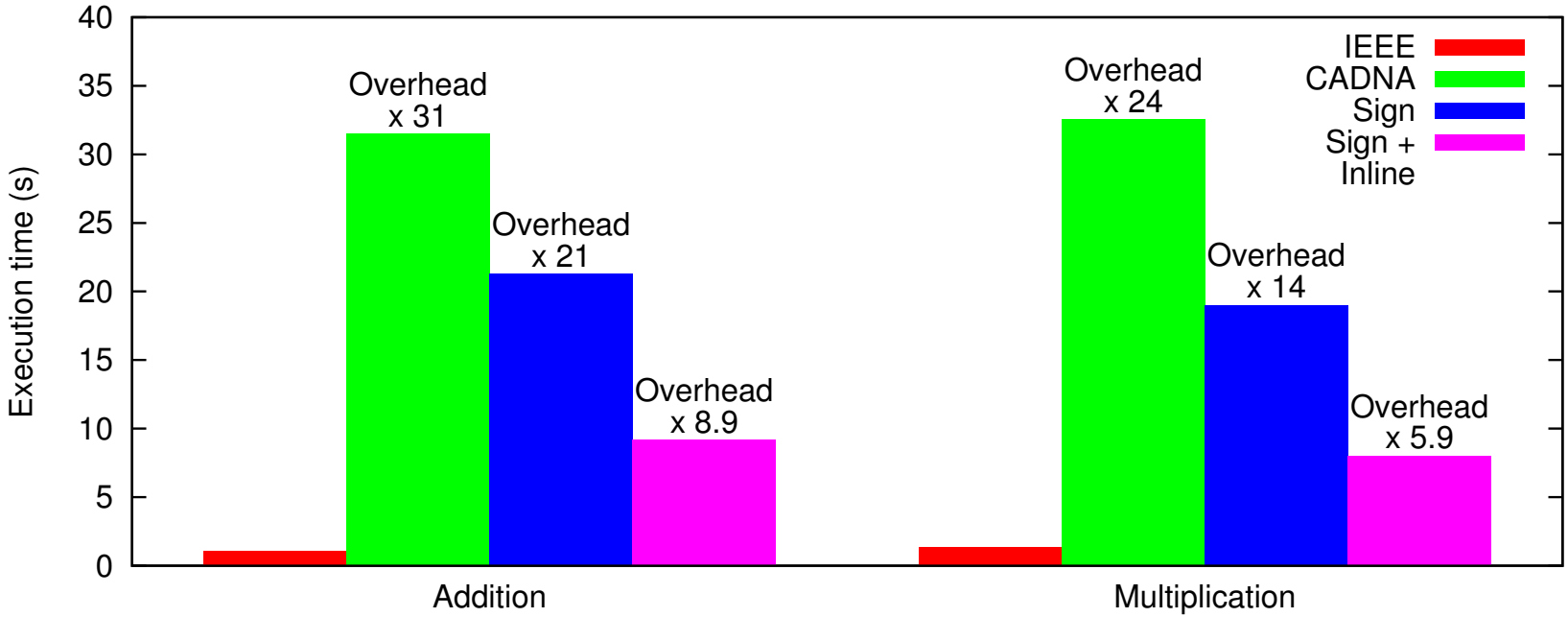
Cancellation detection



Analysis

- Addition only, cancellation only applies to addition
- All detections activated
- Overhead divided by 2

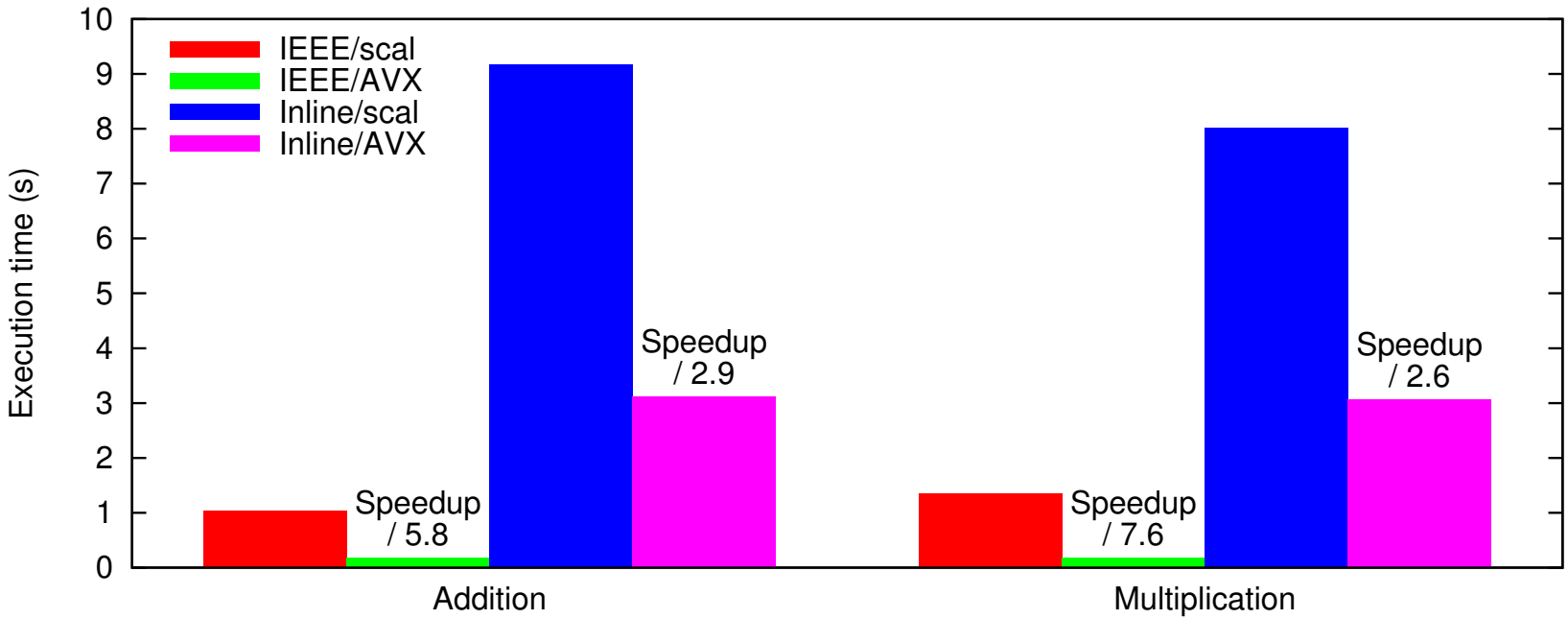
Stochastic operations



Analysis

- No detection activated
- Overhead reduced by more than 3

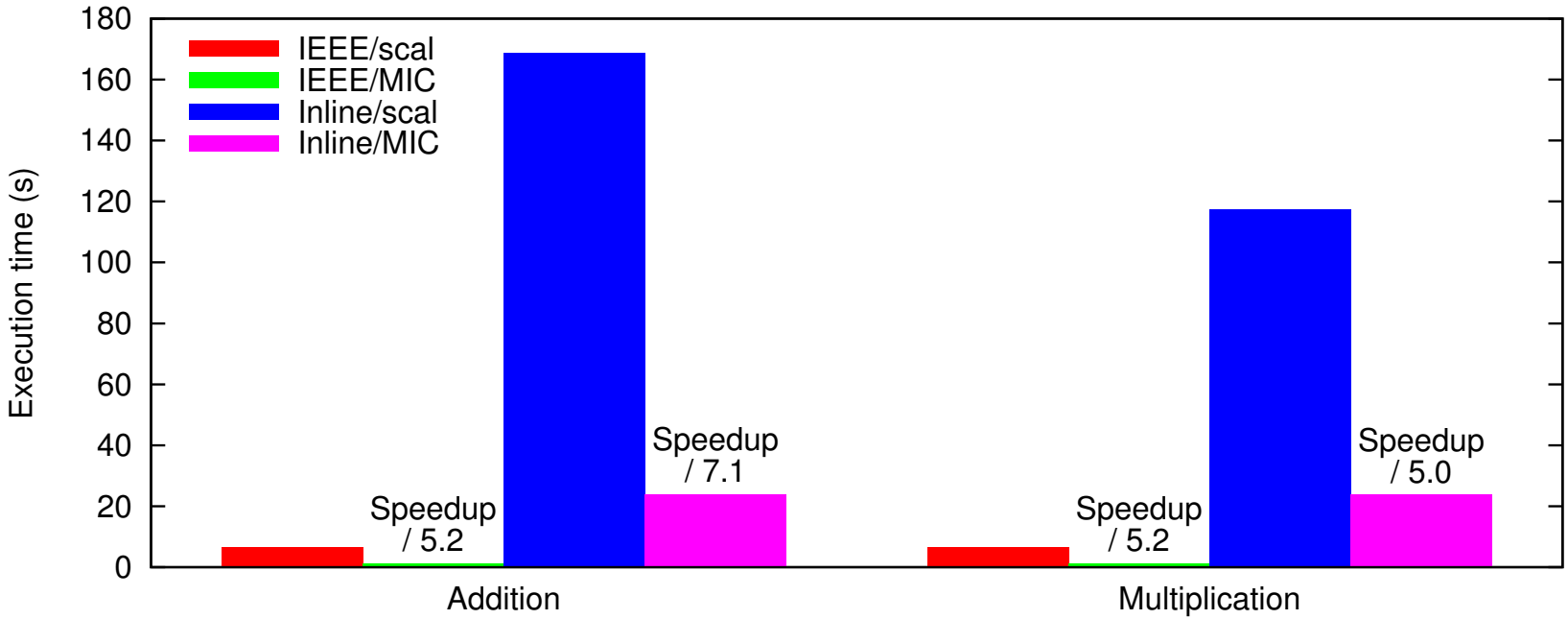
Stochastic operations on AVX



Analysis

- No detection activated
- AVX instruction set extension
- Vectorisation achieved (was impossible with original CADNA)
- CADNA speedup lower than IEEE speedup
  - probably due to inefficient scatter/gather implementation

Stochastic operations on Xeon Phi



Analysis

- No detection activated
- Compiled for Xeon Phi with icc -O2
- CADNA speedup similar to IEEE speedup
  - due to better scatter/gather implementation

Conclusion

Scalar improvements

- Logarithm approximation enables considerable gain when detecting cancellations
- Sign manipulations and inlining achieve large performance improvements on arithmetic operations

Vectorising

- Enabling SIMD computing with stochastic aritmetic
- Higher overhead than scalar
- However, faster execution

Future prospects

- Integrating these improvements in a new CADNA version
- Testing performance for real-life applications