Precision auto-tuning and control of accuracy in high performance simulations

Fabienne Jézéquel

Laboratoire d'Informatique de Paris 6 (LIP6), Sorbonne Université, France

45e Forum ORAP : Quelles précisions pour le HPC ?

24-26 Nov. 2020

Floating-point arithmetic

Floating-point computation $\neq$ mathematical evaluation

- rounding $a \oplus b \neq a + b$
- no more associativity $(a \oplus b) \oplus c \neq a \oplus (b \oplus c)$

Consequences:

- invalid results
- non reproducibility
- performance issue (useless iterations)

Rounding error analysis

Several approaches

- Condition number estimates
  - provides error bounds for the computed results
  - implemented in Lapack
- Interval arithmetic
  - guaranteed bounds for each computed result
  - the error may be overestimated
  - specific algorithms
  - ex: INTLAB [Rump'99]
- Static analysis
  - no execution, rigorous analysis, all possible input values taken into account
  - not suited to large programs
  - ex: Fluctuat [Goubault et al.'06]
- Probabilistic approach
  - estimates the number of correct digits of any computed result
  - ex: CADNA [Chesneaux'90], VerifiCarlo [Denis & al.'16], Verrou [Févotte & al.'17]

Stochastic arithmetic [Vignes'04]

Classic arithmetic

$$A \oplus B \longrightarrow R$$

R =3.14237654356891

Stochastic arithmetic

Random rounding

$$A_1 \oplus B_1 \longrightarrow R_1$$

$$A_2 \oplus B_2 \longrightarrow R_2$$

$$A_3 \oplus B_3 \longrightarrow R_3$$

R_1 =**3.14**1354786390989
R_2 =**3.14**3689456834534
R_3 =**3.14**2579087356598

- each operation executed 3 times with a random rounding mode

Stochastic arithmetic [Vignes'04]

Classic arithmetic

$A \oplus B \longrightarrow R$

$R =$3.14237654356891

Stochastic arithmetic

Random rounding

$A_1 \oplus B_1$  $\longrightarrow R_1$

$A_2 \oplus B_2$  $\longrightarrow R_2$

$A_3 \oplus B_3$  $\longrightarrow R_3$

$R_1 =$**3.14**1354786390989
$R_2 =$**3.14**3689456834534
$R_3 =$**3.14**2579087356598

- each operation executed 3 times with a random rounding mode
- number of correct digits in the results estimated using Student's test with the confidence level 95%

Stochastic arithmetic [Vignes'04]

Classic arithmetic

$A \oplus B \longrightarrow R$

R =3.14237654356891

Stochastic arithmetic

Random rounding

$A_1 \oplus B_1$  $\longrightarrow R_1$

$A_2 \oplus B_2$  $\longrightarrow R_2$

$A_3 \oplus B_3$  $\longrightarrow R_3$

R_1 =**3.14**1354786390989
R_2 =**3.14**3689456834534
R_3 =**3.14**2579087356598

- each operation executed 3 times with a random rounding mode
- number of correct digits in the results estimated using Student's test with the confidence level 95%
- operations executed synchronously
  - ⇒ detection of numerical instabilities
    Ex: `if (A>B)` with `A-B` numerical noise
  - ⇒ optimization of stopping criteria

The CADNA library

`cadna.lip6.fr`

- implements stochastic arithmetic for C/C++ or Fortran codes
- provides stochastic types (3 floating-point variables and an integer)

  `half_st   float_st   double_st   quad_st`
- all operators and mathematical functions overloaded
  $\Rightarrow$ few modifications in user programs
- support for MPI, OpenMP, GPU, vectorised codes
- In one CADNA execution: accuracy of any result, complete list of numerical instabilities

The CADNA library

cadna.lip6.fr

- implements stochastic arithmetic for C/C++ or Fortran codes
- provides stochastic types (3 floating-point variables and an integer)

  `half_st` `float_st` `double_st` `quad_st`
- all operators and mathematical functions overloaded

  ⇒ few modifications in user programs
- support for MPI, OpenMP, GPU, vectorised codes
- In one CADNA execution: accuracy of any result, complete list of numerical instabilities

Recent improvement: control of half precision computation

- emulated
- native (deployment on ARM v8.2)

The SAM library

www-pequan.lip6.fr/~jezequel/SAM

SAM (Stochastic Arithmetic in Multiprecision) [Graillat & al.'11]

- implements stochastic arithmetic in arbitrary precision (based on MPFR[1])
  mp_st stochastic type
- operator overloading ⇒ few modifications in user C/C++ programs

[1] www.mpfr.org

The SAM library

www-pequan.lip6.fr/~jezequel/SAM

SAM (Stochastic Arithmetic in Multiprecision) [Graillat & al.'11]

- implements stochastic arithmetic in arbitrary precision (based on MPFR[1])
  `mp_st` stochastic type
- operator overloading ⇒ few modifications in user C/C++ programs

Recent improvement: control of operations mixing different precisions

Ex: mp_st<23> A; mp_st<47>B; mp_st<35> C;

$$C = A \oplus B$$

35 bits   23 bits   47 bits

⇒ accuracy estimation on FPGA

[1] www.mpfr.org

An example without/with CADNA

Computation of $P(x,y) = 9x^4 - y^4 + 2y^2$ [Rump'83]

```
#include <iostream>
using namespace std;
double rump(double x, double y) {
  return 9.0*x*x*x*x - y*y*y*y + 2.0*y*y;
}
int main() {
  cout.precision(15);
  cout.setf(ios::scientific,ios::floatfield);
  double x, y;
  x = 10864.0;
  y = 18817.0;
  cout<<"P1="<<rump(x, y)<< endl;
  x = 1.0/3.0;
  y = 2.0/3.0;
  cout<<"P2="<<rump(x, y)<< endl;
  return 0;
}
```

An example without/with CADNA

Computation of $P(x,y) = 9x^4 - y^4 + 2y^2$ [Rump'83]

```
#include <iostream>
using namespace std;
double rump(double x, double y) {
  return 9.0*x*x*x*x - y*y*y*y + 2.0*y*y;
}
int main() {
  cout.precision(15);
  cout.setf(ios::scientific,ios::floatfield);
  double x, y;
  x = 10864.0;
  y = 18817.0;
  cout<<"P1="<<rump(x, y)<< endl;
  x = 1.0/3.0;
  y = 2.0/3.0;
  cout<<"P2="<<rump(x, y)<< endl;
  return 0;
}
```

P1=2.00000000000000e+00

P2=8.02469135802469e–01

```
#include <iostream>

using namespace std;
double   rump(double   x, double   y) {
  return 9.0*x*x*x*x-y*y*y*y+2.0*y*y;
}
int main() {
  cout.precision(15);
  cout.setf(ios::scientific,ios::floatfield);

  double   x, y;
  x=10864.0; y=18817.0;
  cout«"P1="«rump(x, y)«endl;
  x=1.0/3.0; y=2.0/3.0;
  cout«"P2="«rump(x, y)«endl;

  return 0;
}
```

```
#include <iostream>
#include <cadna.h>
using namespace std;
double   rump(double   x, double   y) {
  return 9.0*x*x*x*x-y*y*y*y+2.0*y*y;
}
int main() {
  cout.precision(15);
  cout.setf(ios::scientific,ios::floatfield);

  double   x, y;
  x=10864.0; y=18817.0;
  cout«"P1="«rump(x, y)«endl;
  x=1.0/3.0; y=2.0/3.0;
  cout«"P2="«rump(x, y)«endl;

  return 0;
}
```

```
#include <iostream>
#include <cadna.h>
using namespace std;
double   rump(double   x, double   y) {
  return 9.0*x*x*x*x-y*y*y*y+2.0*y*y;
}
int main() {
  cout.precision(15);
  cout.setf(ios::scientific,ios::floatfield);
  cadna_init(-1);
  double   x, y;
  x=10864.0; y=18817.0;
  cout<<"P1="<<rump(x, y)<<endl;
  x=1.0/3.0; y=2.0/3.0;
  cout<<"P2="<<rump(x, y)<<endl;

  return 0;
}
```

```
#include <iostream>
#include <cadna.h>
using namespace std;
double   rump(double   x, double   y) {
  return 9.0*x*x*x*x-y*y*y*y+2.0*y*y;
}
int main() {
  cout.precision(15);
  cout.setf(ios::scientific,ios::floatfield);
  cadna_init(-1);
  double   x, y;
  x=10864.0; y=18817.0;
  cout«"P1="«rump(x, y)«endl;
  x=1.0/3.0; y=2.0/3.0;
  cout«"P2="«rump(x, y)«endl;
  cadna_end();
  return 0;
}
```

```
#include <iostream>
#include <cadna.h>
using namespace std;
double   rump(double   x, double   y) {
  return 9.0*x*x*x*x-y*y*y*y+2.0*y*y;
}
int main() {
  cout.precision(15);
  cout.setf(ios::scientific,ios::floatfield);
  cadna_init(-1);
  double   x, y;
  x=10864.0; y=18817.0;
  cout<<"P1="<<rump(x, y)<<endl;
  x=1.0/3.0; y=2.0/3.0;
  cout<<"P2="<<rump(x, y)<<endl;
  cadna_end();
  return 0;
}
```

```
#include <iostream>
#include <cadna.h>
using namespace std;
double_st rump(double_st x, double_st y) {
  return 9.0*x*x*x*x-y*y*y*y+2.0*y*y;
}
int main() {
  cout.precision(15);
  cout.setf(ios::scientific,ios::floatfield);
  cadna_init(-1);
  double_st x, y;
  x=10864.0; y=18817.0;
  cout<<"P1="<<rump(x, y)<<endl;
  x=1.0/3.0; y=2.0/3.0;
  cout<<"P2="<<rump(x, y)<<endl;
  cadna_end();
  return 0;
}
```

Results with CADNA

only correct digits are displayed

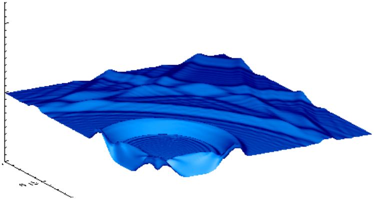
CADNA_C software
Self-validation detection: ON
Mathematical instabilities detection: ON
Branching instabilities detection: ON
Intrinsic instabilities detection: ON
Cancellation instabilities detection: ON

P1= @.0 (no more correct digits)
P2= 0.802469135802469E+000

There are 2 numerical instabilities
2 LOSS(ES) OF ACCURACY DUE TO CANCELLATION(S)

Numerical validation of a shallow-water (SW) simulation on GPU

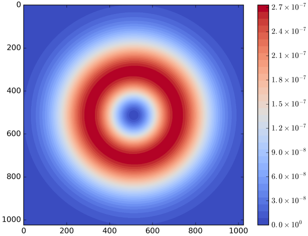
- Numerical model (combination of finite difference stencils) simulating the evolution of water height and velocities in a 2D oceanic basin



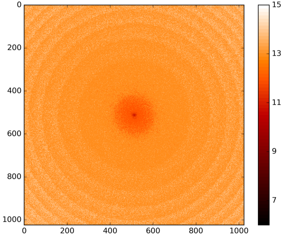
- Focusing on an eddy evolution:
  - 20 time steps (12 hours of simulated time) on a 1024 × 1024 grid
  - CUDA GPU deployment
  - in double precision

SW eddy simulation with CADNA-GPU

At the end of the simulation:



Square of water velocity in $m^2.s^{-2}$



Number of exact significant digits estimated by CADNA-GPU

- at eddy center: great accuracy loss
  equilibrium between several forces (pressure, Coriolis)
  ⇒ **possible cancellations**
- point at the very center: 9 exact significant digits lost
  ⇒ **no correct digits in SP**
- fortunately, velocity values close to zero at eddy center
  → negligible impact on the output
  → **satisfactory overall accuracy**

Tools related to CADNA

available on `cadna.lip6.fr`

- CADNAIZER
  - automatically transforms C codes to be used with CADNA
- CADTRACE
  - identifies the instructions responsible for numerical instabilities

    Example:

    There are 12 numerical instabilities.

    10 LOSS(ES) OF ACCURACY DUE TO CANCELLATION(S).
    5 in <ex> file "ex.f90" line 58
    5 in <ex> file "ex.f90" line 59

    1 INSTABILITY IN ABS FUNCTION.
    1 in <ex> file "ex.f90" line 37

    1 UNSTABLE BRANCHING.
    1 in <ex> file "ex.f90" line 37

Accuracy analysis... and then?

accurate results?

No ☹

- increase precision: `single` → `double` → `quad` → arbitrary precision
- compensated algorithms
  [Kahan'87], [Priest'92], [Ogita & al.'05], [Graillat & al.'09]
  - for sum, dot product, polynomial evaluation,...
  - results ≈ as accurate as with twice the working precision
- accurate and reproducible BLAS
  - ExBLAS [Collange & al.'15]
  - RARE-BLAS [Chohra & al.'16]
  - OzBLAS [Mukunoki & al.'19]
- symbolic computation

Accuracy analysis... and then?

accurate results?

No ☹

- increase precision: `single` → `double` → `quad` → arbitrary precision
- compensated algorithms
  [Kahan'87], [Priest'92], [Ogita & al.'05], [Graillat & al.'09]
  - for sum, dot product, polynomial evaluation,...
  - results ≈ as accurate as with twice the working precision
- accurate and reproducible BLAS
  - ExBLAS [Collange & al.'15]
  - RARE-BLAS [Chohra & al.'16]
  - OzBLAS [Mukunoki & al.'19]
- symbolic computation

Yes ☺

performance improvement thanks to mixed precision?

Precision autotuning

- floating-point autotuning tools that intend to deal with large codes:
  - Precimonious [Rubio-Gonzàlez & al.'13]
    - source modification with LLVM
  - CRAFT [Lam & al.'13]
    - binary modifications on the operations
  - ADAPT [Menon & al.'18]
    - based on algorithmic differentiation
  - CRAFT & ADAPT now combined in FloatSmith [Lam & al.'19]

They rely on comparisons with the highest precision result.

Precision autotuning

- floating-point autotuning tools that intend to deal with large codes:
  - Precimonious [Rubio-Gonzàlez & al.'13]
    - source modification with LLVM
  - CRAFT [Lam & al.'13]
    - binary modifications on the operations
  - ADAPT [Menon & al.'18]
    - based on algorithmic differentiation
  - CRAFT & ADAPT now combined in FloatSmith [Lam & al.'19]

  They rely on comparisons with the highest precision result.

⚠ [Rump'88] $P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$
with $x = 77617$ and $y = 33096$

Precision autotuning

- floating-point autotuning tools that intend to deal with large codes:
  - Precimonious [Rubio-Gonzàlez & al.'13]
    - source modification with LLVM
  - CRAFT [Lam & al.'13]
    - binary modifications on the operations
  - ADAPT [Menon & al.'18]
    - based on algorithmic differentiation
  - CRAFT & ADAPT now combined in FloatSmith [Lam & al.'19]

  They rely on comparisons with the highest precision result.

⚠ [Rump'88] $P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$
with $x = 77617$ and $y = 33096$
float: $P =$2.571784e+29

Precision autotuning

- floating-point autotuning tools that intend to deal with large codes:
  - Precimonious [Rubio-Gonzàlez & al.'13]
    - source modification with LLVM
  - CRAFT [Lam & al.'13]
    - binary modifications on the operations
  - ADAPT [Menon & al.'18]
    - based on algorithmic differentiation
  - CRAFT & ADAPT now combined in FloatSmith [Lam & al.'19]

  They rely on comparisons with the highest precision result.

⚠ [Rump'88] $P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$
with $x = 77617$ and $y = 33096$

float: $P =$ 2.571784e+29

double: $P =$ 1.17260394005318

Precision autotuning

- floating-point autotuning tools that intend to deal with large codes:
    - Precimonious [Rubio-Gonzàlez & al.'13]
        - source modification with LLVM
    - CRAFT [Lam & al.'13]
        - binary modifications on the operations
    - ADAPT [Menon & al.'18]
        - based on algorithmic differentiation
    - CRAFT & ADAPT now combined in FloatSmith [Lam & al.'19]

  They rely on comparisons with the highest precision result.

⚠ [Rump'88] $P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$
with $x = 77617$ and $y = 33096$

float: $P =$2.571784e+29
double: $P =$1.17260394005318
quad: $P =$1.17260394005317863185883490452018

Precision autotuning

- floating-point autotuning tools that intend to deal with large codes:
  - Precimonious [Rubio-Gonzàlez & al.'13]
    - source modification with LLVM
  - CRAFT [Lam & al.'13]
    - binary modifications on the operations
  - ADAPT [Menon & al.'18]
    - based on algorithmic differentiation
  - CRAFT & ADAPT now combined in FloatSmith [Lam & al.'19]

  They rely on comparisons with the highest precision result.

⚠ [Rump'88] $P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$
with $x = 77617$ and $y = 33096$

float:	P =2.571784e+29
double:	P =1.17260394005318
quad:	P =1.17260394005317863185883490452018
exact:	$P \approx$-0.827396059946821368141165095479816292

PROMISE (PRecision OptiMISE) [Graillat & al.'19]

promise.lip6.fr

- provides a mixed precision code (half, single, double, quad) taking into account a required accuracy
- uses CADNA to validate a type configuration
- uses the Delta Debug algorithm `[Zeller'09]` to search for a valid type configuration with a mean complexity of $O(n \log(n))$ for n variables.

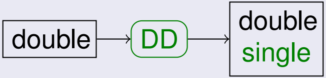
Recent improvements:

- complete rewriting (more user friendly, performance improved)
- half precision

Searching for a valid type configuration

PROMISE with 2 types (ex: double & single precision)

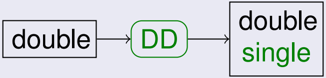
From a code in double, the Delta Debug (DD) algorithm finds which variables can be relaxed to single precision.



Searching for a valid type configuration

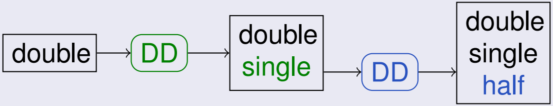
PROMISE with 2 types (ex: double & single precision)

From a code in double, the Delta Debug (DD) algorithm finds which variables can be relaxed to single precision.



PROMISE with 3 types (ex: double, single & half precision)

The Delta Debug algorithm is applied twice.



Precision auto-tuning using PROMISE

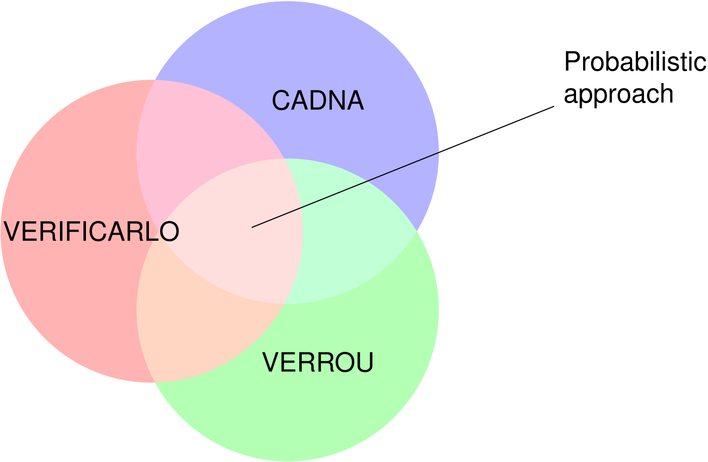
MICADO: simulation of nuclear cores (EDF)

- neutron transport iterative solver
- 11,000 C++ code lines

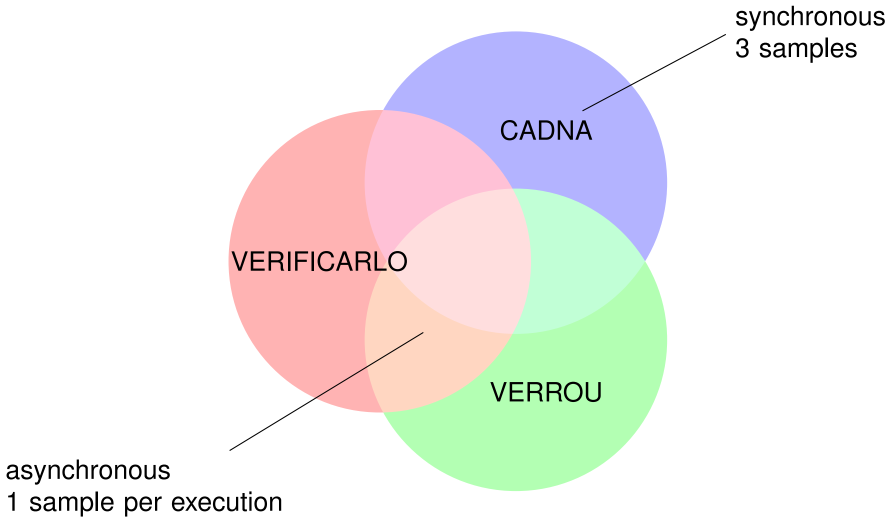
# Digits	# double - # float	Speed up	memory gain
10	19-32	1.01	1.00
8	18-33	1.01	1.01
6	13-38	1.20	1.44
5 4	0-51	1.32	1.62

- Speedup, memory gain: w.r.t. the initial configuration (in double precision).
- Speed-up up to 1.32 and memory gain 1.62
- Mixed precision approach successful: speed-up 1.20 and memory gain 1.44

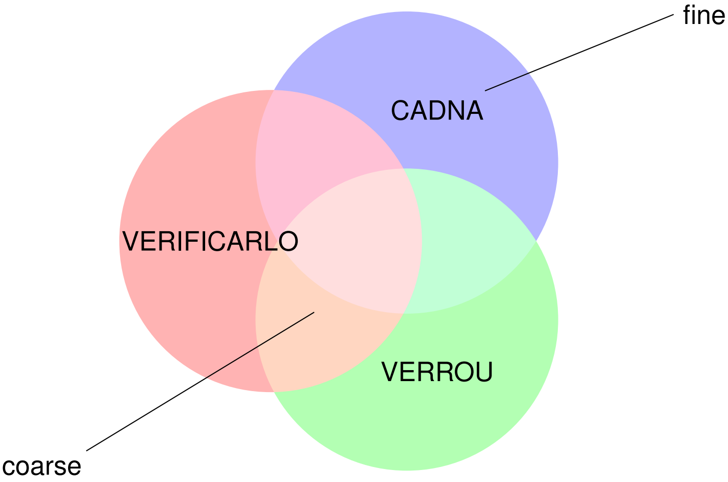
Comparison of CADNA, VERIFICARLO and VERROU



Method:

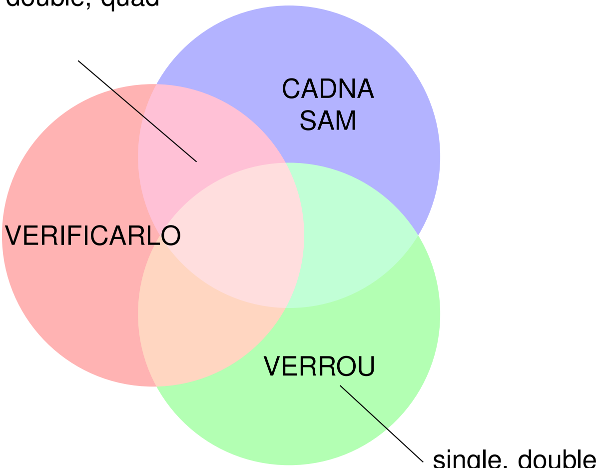

synchronous
3 samples
CADNA
VERIFICARLO
VERROU
asynchronous
1 sample per execution

Instability localization:

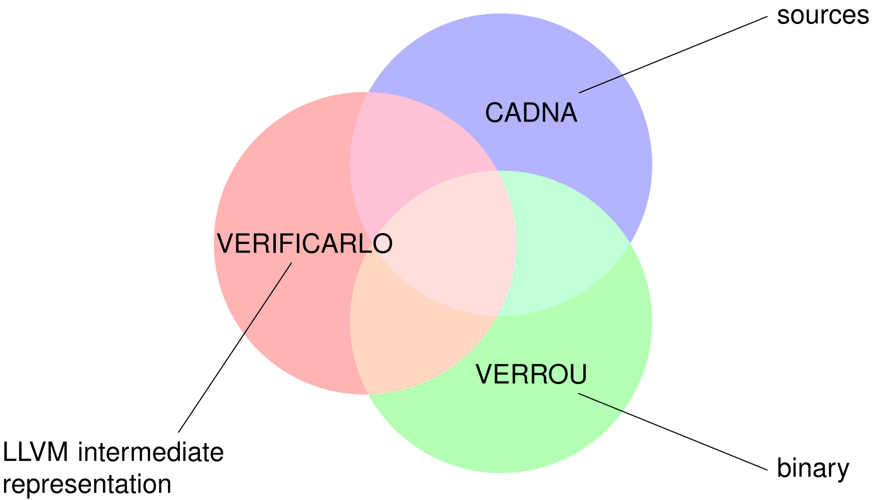


Supported precisions:

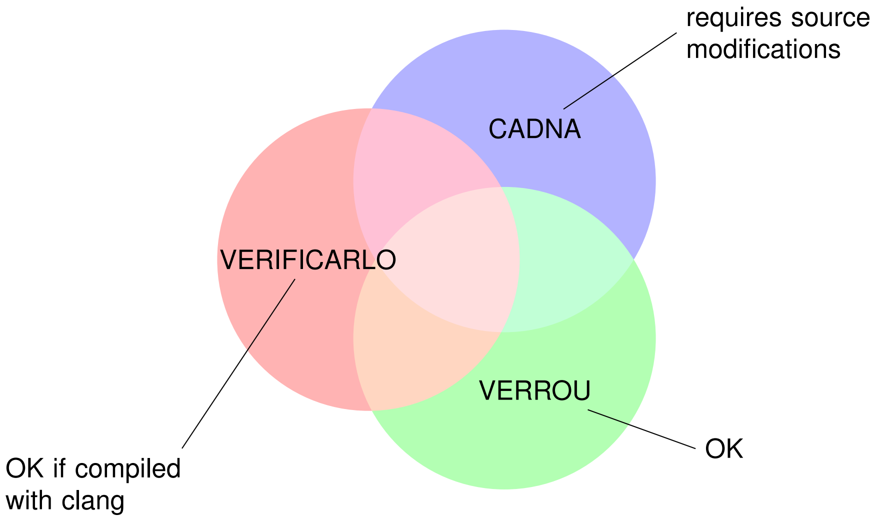
- half, single, double, quad
- arbitrary



Instrumentation:



Control of external libraries?



Cost comparison

C++ arithmetic benchmarks (compute/memory bound) [Picot'18]

	3 samples w.r.t classic exec.
CADNA	≈ 5 to 8
VERIFICARLO	≈ 300 to 600
VERROU	≈ 30

Supported languages

	C/C++	Fortran	Python	assembly
CADNA	✓	✓	POC	✗
VERIFICARLO	✓	✓	docker image	✗
VERROU	✓	✓	✓	✓

Supported HPC codes

	vecto.	MPI	OpenMP	GPU
CADNA	✓	✓	✓	cuda
VERIFICARLO	✓	✓	in progress	✗
VERROU	✓	✓	✓	✗

Perpectives: Interflop

the Interflop project

- recently accepted ANR project led by David Defour
- with Aneo, CEA, EDF, Intel, Sorbonne Univ., TriScale innov, Univ. Perpignan, Univ. Versailles
- aims at proposing a unified platform for the numerical validation of large codes

Perpectives: Interflop

the Interflop project

- recently accepted ANR project led by David Defour
- with Aneo, CEA, EDF, Intel, Sorbonne Univ., TriScale innov, Univ. Perpignan, Univ. Versailles
- aims at proposing a unified platform for the numerical validation of large codes

Among our goals...

- combine a set of error analyzes that cover a large number of possible inputs
- propose new floating-point formats
- improve precision auto-tuning
- provide original solutions to visualise and interpret results

Thanks to the CADNA/SAM/PROMISE contributors:
Julien Brajard, Romuald Carpentier, Jean-Marie Chesneaux, Patrick Corde, Pacôme Eberhart, François Févotte, Pierre Fortin, Stef Graillat, Thibault Hilaire, Sara Hoseininasab, Jean-Luc Lamotte, Baptiste Landreau, Bruno Lathuilière, Romain Picot, Jonathon Tidswell, Su Zhou, ...

Thanks to the CADNA/SAM/PROMISE contributors:
Julien Brajard, Romuald Carpentier, Jean-Marie Chesneaux, Patrick Corde, Pacôme Eberhart, François Févotte, Pierre Fortin, Stef Graillat, Thibault Hilaire, Sara Hoseininasab, Jean-Luc Lamotte, Baptiste Landreau, Bruno Lathuilière, Romain Picot, Jonathon Tidswell, Su Zhou, ...

Thank you for your attention!