Synchronous stochastic arithmetic
The CADNA software

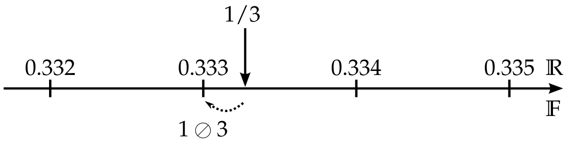
Fabienne Jézéquel

LIP6 (Laboratoire d'Informatique de Paris 6)
Équipe PEQUAN (PErformance et QUalité des Algorithmes Numériques)

Maison de la Simulation
28 June 2019

Floating-point arithmetic

- Finite precision of the floating-point representation
  - [our example] decimal, 3 significant digits: 42.0, 0.123
  - [`float`] binary, 24 significant bits ($\simeq 10^{-7}$)
  - [`double`] binary, 53 significant bits ($\simeq 10^{-15}$)



- Consequences: floating-point computation $\neq$ real computation
  - rounding $a \oplus b \neq a + b$
  - no more associativity $(a \oplus b) \oplus c \neq a \oplus (b \oplus c)$
    $\Rightarrow$ reproducibility problems

Impact of rounding errors: an example

$P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$
with $x = 77617$ and $y = 33096$

[S.M. Rump, 1988]

Impact of rounding errors: an example

$P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$
with $x = 77617$ and $y = 33096$

[S.M. Rump, 1988]

float: $P =$2.571784e+29

Impact of rounding errors: an example

$P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$
with $x = 77617$ and $y = 33096$

[S.M. Rump, 1988]

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

Impact of rounding errors: an example

$P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$
with $x = 77617$ and $y = 33096$

[S.M. Rump, 1988]

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

Impact of rounding errors: an example

$P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$
with $x = 77617$ and $y = 33096$

[S.M. Rump, 1988]

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

Impact of rounding errors: an example

$P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$
with $x = 77617$ and $y = 33096$

[S.M. Rump, 1988]

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

How to control rounding errors ?

Overview

- Rounding error analysis
- Discrete Stochastic Arithmetic and its implementations (CADNA, SAM)
- Contributions of Discrete Stochastic Arithmetic in numerical methods
- CADNA for HPC simulations

Round-off error model

$r \in \mathbb{R}$: exact result of n elementary arithmetic operations

The computed result R can be modeled, at the 1st order w.r.t. 2^{-p}, by

$$R \approx r + \sum_{i=1}^{s_n} g_i \, 2^{-p} \, \alpha_i$$

- p: number of mantissa bits including the hidden bit ($p = 24$ in $binary32$, $p = 53$ in $binary64$)
- the number of terms s_n depends on n (for $n = 1$, $s_n = 3$ if data are not exactly encoded)
- g_i are coefficients depending only on data and on the algorithm
- α_i are the round-off errors.

Remark: to simplify the equation, it is assumed that exponents and signs of intermediate results do not depend on α_i.

A theorem on numerical accuracy

The number of significant bits in common between R and r is

$$C_R \approx -\log_2 \left| \frac{R-r}{r} \right| = p - \log_2 \left| \sum_{i=1}^{s_n} g_i \frac{\alpha_i}{r} \right|$$

The last part corresponds to the accuracy which has been lost in the computation of R, we can note that it is independent of p.

Theorem

The loss of accuracy during a numerical computation is independent of the precision used.

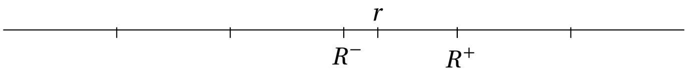
Round-off error analysis

Several approaches

- Inverse analysis
  based on the “ Wilkinson principle”: the computed solution is assumed to be the exact solution of a nearby problem
  - provides error bounds for the computed results
- Interval arithmetic
  The result of an operation between two intervals contains all values that can be obtained by performing this operation on elements from each interval.
  - guaranteed bounds for each computed result
  - the error may be overestimated
  - specific algorithms
- Probabilistic approach
  estimates the number of exact significant digits of any computed result

How to estimate rounding error propagation?

The exact result r of an arithmetic operation is approximated by a floating-point number R^- or R^+.



The random rounding mode

Approximation of r by R^- or R^+ with the probability 1/2

The CESTAC method [La Porte & Vignes 1974]

The same code is run several times with the random rounding mode.
Then different results are obtained.

Briefly, the part that is common to all the different results is assumed to be reliable and the part that is different is affected by round-off errors.

With the random rounding mode...

By running N times the code with the random rounding mode, one obtains an N-sample of the random variable modeled by

$$R \approx r + \sum_{i=1}^{s_n} g_i \, 2^{-p} \, \alpha_i$$

where the α_i's are modeled by independent identically distributed random variables. The common distribution of the α_i's is uniform on $[-1, +1]$.

$\Rightarrow$ the mathematical expectation of R is the exact result r,

$\Rightarrow$ the distribution of R is a quasi-Gaussian distribution.

Implementation of the CESTAC method

The implementation of the CESTAC method in a code providing a result R consists in:

- performing N times this code with the random rounding mode to obtain N samples R_i of R,
- choosing as the computed result the mean value $\overline{R}$ of R_i, $i = 1, ..., N$,
- estimating the number of exact significant decimal digits of $\overline{R}$ with

$$C_{\overline{R}} = \log_{10}\left(\frac{\sqrt{N}\left|\overline{R}\right|}{\sigma\tau_\beta}\right)$$

  where

$$\overline{R} = \frac{1}{N}\sum_{i=1}^{N} R_i \quad \text{and} \quad \sigma^2 = \frac{1}{N-1}\sum_{i=1}^{N}\left(R_i - \overline{R}\right)^2.$$

  τ_β is the value of Student's distribution for $N-1$ degrees of freedom and a probability level $1-\beta$.

In pratice, $N = 3$ and $\beta = 0.05$.

On the number of runs

2 or 3 runs are enough. To increase the number of runs is not necessary.

From the model, to increase by 1 the number of exact significant digits given by $C_{\overline{R}}$, we need to multiply the size of the sample by 100.

Such an increase of N will only point out the limit of the model and its error without really improving the quality of the estimation.

It has been shown that $N = 3$ is the optimal value. [Chesneaux & Vignes, 1988]

On the probability of the confidence interval

With $\beta = 0.05$ and $N = 3$,

- the probability of overestimating the number of exact significant digits of at least 1 is 0.054%
- the probability of underestimating the number of exact significant digits of at least 1 is 29%.

By choosing a confidence interval at 95%, we prefer to guarantee a minimal number of exact significant digits with a high probability (99.946%), even if we are often pessimistic by 1 digit.

Validity of the CESTAC method [Chesneaux & Vignes, 1988]

- The CESTAC method is based on a 1st order model.
    - A multiplication of two insignificant results
    - or a division by an insignificant result

  may invalidate the 1st order approximation.

  $\Rightarrow$ control of multiplications and divisions: *self-validation* of CESTAC.

- With CESTAC, rounding errors are assumed centered.

  Even if they are not rigorously centered, the accuracy estimation can be considered correct up to 1 digit.

The problem of stopping criteria

Let us consider a general iterative algorithm: $U_{n+1} = F(U_n)$.

```
while (fabs(X-Y) > EPSILON) {
    X = Y;
    Y = F(X);
}
```

ε too low $\Longrightarrow$ risk of infinite loop
ε too high $\Longrightarrow$ too early termination.

The problem of stopping criteria

Let us consider a general iterative algorithm: $U_{n+1} = F(U_n)$.

```
while (fabs(X-Y) > EPSILON) {
    X = Y;
    Y = F(X);
}
```

ε too low $\Longrightarrow$ risk of infinite loop
ε too high $\Longrightarrow$ too early termination.

It would be optimal to stop when $X - Y$ is an **insignificant value**.

Such a stopping criterion

- would enable one to develop new numerical algorithms
- is possible thanks to the concept of *computed zero*.

The concept of computed zero

[Vignes, 1986]

Definition

Using the CESTAC method, a result R is a computed zero, denoted by @.0, if

$$\forall i, R_i = 0 \;\text{ or }\; C_{\overline{R}} \leq 0.$$

It means that R is a computed result which, because of round-off errors, cannot be distinguished from 0.

The stochastic definitions

Let X and Y be two results computed using the CESTAC method (N-samples).

- X is stochastically equal to Y, noted $X \; s\!= \; Y$, iff
$$X - Y = @.0.$$
- X is stochastically strictly greater than Y, noted $X \; s\!> \; Y$, iff
$$\overline{X} > \overline{Y} \quad \text{and} \quad X \; s\!\neq \; Y$$
- X is stochastically greater than or equal to Y, noted $X \; s\!\geq \; Y$, iff
$$\overline{X} \geq \overline{Y} \quad \text{or} \quad X \; s\!= \; Y$$

Discrete Stochastic Arithmetic [Vignes, 2004]

Discrete Stochastic Arithmetic (DSA) is defined as the joint use of

- the CESTAC method
- the computed zero
- the stochastic relation definitions.

It implies a **simultaneous** computation of the different results.
Each arithmetic operation is performed N times before the next one is executed.

Implementation of DSA

- CADNA: for programs in half, single, double, and/or quadruple precision
  `http://cadna.lip6.fr`
  support for wide range of codes (vectorised, MPI, OpenMP, GPU)

- SAM: for arbitrary precision programs (based on MPFR)
  `http://www-pequan.lip6.fr/~jezequel/SAM`

The CADNA library `http://cadna.lip6.fr`

CADNA allows one to estimate round-off error propagation in any scientific program written in Fortran, C or C++.

CADNA enables one to:

- estimate the numerical quality of any result
- detect numerical instabilities (synchronous implementation of CESTAC)
- take into account uncertainty on data.

The CADNA library http://cadna.lip6.fr

CADNA allows one to estimate round-off error propagation in any scientific program written in Fortran, C or C++.

CADNA enables one to:

- estimate the numerical quality of any result
- detect numerical instabilities (synchronous implementation of CESTAC)
- take into account uncertainty on data.

CADNA provides new numerical types, the stochastic types, which consist of:

- 3 floating point variables
- an integer variable to store the accuracy.

All operators and mathematical functions are redefined for these types.
⇒ CADNA requires only a few modifications in user programs.

Cost of CADNA

The cost of CADNA is about 4 in memory, 10 in run time.

CADNA overhead:

Memory Bound Add	Compute Bound Add	Memory Bound Multiply	Compute Bound Multiply
$7.89\times$	$8.92\times$	$11.6\times$	$9.19\times$

(Intel Xeon E3-1275 at 3.5 GHz, gcc version 4.9.2, single precision, self-validation)

An example without/with CADNA

Computation of $P(x,y) = 9x^4 - y^4 + 2y^2$ [S.M. Rump, 1983]

```
#include <stdio.h>

double rump(double x, double y) {
  return 9.0*x*x*x*x - y*y*y*y + 2.0*y*y;
}
int main(int argc, char **argv) {
  double x, y;
  x = 10864.0;
  y = 18817.0;
  printf("P1=%.14e\n", rump(x, y));
  x = 1.0/3.0;
  y = 2.0/3.0;
  printf("P2=%.14e\n", rump(x, y));
  return 0;
}
```

An example without/with CADNA

Computation of $P(x,y) = 9x^4 - y^4 + 2y^2$ [S.M. Rump, 1983]

```
#include <stdio.h>

double rump(double x, double y) {
  return 9.0*x*x*x*x - y*y*y*y + 2.0*y*y;
}
int main(int argc, char **argv) {
  double x, y;
  x = 10864.0;
  y = 18817.0;
  printf("P1=%.14e\n", rump(x, y));
  x = 1.0/3.0;
  y = 2.0/3.0;
  printf("P2=%.14e\n", rump(x, y));
  return 0;
}
```

```
P1=2.00000000000000e+00
P2=8.02469135802469e–01
```

```
#include <stdio.h>

double   rump(double   x, double   y) {
  return 9.0*x*x*x*x-y*y*y*y+2.0*y*y;
}
int main(int argc, char **argv) {

  double   x, y;
  x=10864.0; y=18817.0;
  printf("P1=%.14e\n",       rump(x, y) );"
  x=1.0/3.0; y=2.0/3.0;
  printf("P2=%.14e\n",       rump(x, y) );"

  return 0;
}
```

```
#include <stdio.h>
#include <cadna.h>
double   rump(double   x, double   y) {
  return 9.0*x*x*x*x-y*y*y*y+2.0*y*y;
}
int main(int argc, char **argv) {

  double   x, y;
  x=10864.0; y=18817.0;
  printf("P1=%.14e\n",      rump(x, y) );"
  x=1.0/3.0; y=2.0/3.0;
  printf("P2=%.14e\n",      rump(x, y) );"

  return 0;
}
```

```
#include <stdio.h>
#include <cadna.h>
double   rump(double   x, double   y) {
  return 9.0*x*x*x*x-y*y*y*y+2.0*y*y;
}
int main(int argc, char **argv) {
  cadna_init(-1);
  double   x, y;
  x=10864.0; y=18817.0;
  printf("P1=%.14e\n",      rump(x, y) );"
  x=1.0/3.0; y=2.0/3.0;
  printf("P2=%.14e\n",      rump(x, y) );"

  return 0;
}
```

```
#include <stdio.h>
#include <cadna.h>
double   rump(double   x, double   y) {
  return 9.0*x*x*x*x-y*y*y*y+2.0*y*y;
}
int main(int argc, char **argv) {
  cadna_init(-1);
  double   x, y;
  x=10864.0; y=18817.0;
  printf("P1=%.14e\n",      rump(x, y) );"
  x=1.0/3.0; y=2.0/3.0;
  printf("P2=%.14e\n",      rump(x, y) );"
  cadna_end();
  return 0;
}
```

```
#include <stdio.h>
#include <cadna.h>
double   rump(double   x, double   y) {
  return 9.0*x*x*x*x-y*y*y*y+2.0*y*y;
}
int main(int argc, char **argv) {
  cadna_init(-1);
  double   x, y;
  x=10864.0; y=18817.0;
  printf("P1=%.14e\n",      rump(x, y) );"
  x=1.0/3.0; y=2.0/3.0;
  printf("P2=%.14e\n",      rump(x, y) );"
  cadna_end();
  return 0;
}
```

```
#include <stdio.h>
#include <cadna.h>
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(int argc, char **argv) {
  cadna_init(-1);
  double_st x, y;
  x=10864.0; y=18817.0;
  printf("P1=%.14e\n",       rump(x, y) );"
  x=1.0/3.0; y=2.0/3.0;
  printf("P2=%.14e\n",       rump(x, y) );"
  cadna_end();
  return 0;
}
```

```
#include <stdio.h>
#include <cadna.h>
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(int argc, char **argv) {
  cadna_init(-1);
  double_st x, y;
  x=10864.0; y=18817.0;
  printf("P1=%.14e\n",      rump(x, y) );"
  x=1.0/3.0; y=2.0/3.0;
  printf("P2=%.14e\n",      rump(x, y) );"
  cadna_end();
  return 0;
}
```

```
#include <stdio.h>
#include <cadna.h>
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(int argc, char **argv) {
  cadna_init(-1);
  double_st x, y;
  x=10864.0; y=18817.0;
  printf("P1=%s\n", strp(rump(x, y)));"
  x=1.0/3.0; y=2.0/3.0;
  printf("P2=%s\n", strp(rump(x, y)));"
  cadna_end();
  return 0;
}
```

Results with CADNA

only correct digits are displayed

```
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)
```

Contributions of Discrete Stochastic Arithmetic

- In direct methods:
  - estimate the numerical quality of the results
  - control branching statements
- In iterative methods:
  - optimize the number of iterations
  - check if the computed solution is satisfactory
- In approximation methods:
  - optimize the integration step

In direct methods - Example

$$0.3x^2 - 2.1x + 3.675 = 0$$

Without CADNA, in single precision with rounding to nearest:

```
d = -3.8146972E-06
Two complex roots
z1 = 0.3499999E+01 + i * 0.9765625E-03
z2 = 0.3499999E+01 + i * -.9765625E-03
```

With CADNA:

```
d = @.0
The discriminant is null
The double real root is 0.3500000E+01
```

Contribution of CADNA in iterative methods

$$U_{n+1} = F(U_n)$$

Without / with CADNA

```
while (fabs(X-Y) > EPSILON) {
    X = Y;
    Y = F(X);
}
```

With CADNA

```
while (X != Y) {
    X = Y;
    Y = F(X);
}
```

☺ optimal stopping criterion

Iterative methods - Example

The linear system $AX = B$ is solved using Jacobi method.

$$x_i^{(k)} = -\frac{1}{a_{ii}} \sum_{j=1, j\neq i}^{n} a_{ij} x_j^{(k-1)} + \frac{b_i}{a_{ii}}$$

Without CADNA

- Stop when $\max_{i=1}^{n} |x_i^{(k)} - x_i^{(k-1)}| < \varepsilon$
- Compute $R = B - AX^{(k)}$.

$\varepsilon = 10^{-3}$

```
niter =            35
x( 1)= 0.1699924E+01 (exact: 0.1700000E+01), r( 1)= 0.3051758E-03
x( 2)=-0.4746889E+04 (exact:-0.4746890E+04), r( 2)= 0.1953125E-02
x( 3)= 0.5023049E+02 (exact: 0.5023000E+02), r( 3)= 0.1464844E-02
x( 4)=-0.2453197E+03 (exact:-0.2453200E+03), r( 4)=-0.7324219E-03
x( 5)= 0.4778290E+04 (exact: 0.4778290E+04), r( 5)=-0.4882812E-03
x( 6)=-0.7572980E+02 (exact:-0.7573000E+02), r( 6)= 0.9765625E-03
x( 7)= 0.3495430E+04 (exact: 0.3495430E+04), r( 7)= 0.3173828E-02
x( 8)= 0.4350277E+01 (exact: 0.4350000E+01), r( 8)= 0.0000000E+00
x( 9)= 0.4529804E+03 (exact: 0.4529800E+03), r( 9)= 0.9765625E-03
x(10)=-0.2759901E+01 (exact:-0.2760000E+01), r(10)= 0.9765625E-03
x(11)= 0.8239241E+04 (exact: 0.8239240E+04), r(11)= 0.7568359E-02
x(12)= 0.3459919E+01 (exact: 0.3460000E+01), r(12)=-0.4882812E-03
x(13)= 0.1000000E+04 (exact: 0.1000000E+04), r(13)= 0.9765625E-03
x(14)=-0.4999743E+01 (exact:-0.5000000E+01), r(14)= 0.1464844E-02
x(15)= 0.3642400E+04 (exact: 0.3642400E+04), r(15)=-0.1953125E-02
x(16)= 0.7353594E+03 (exact: 0.7353600E+03), r(16)=-0.3662109E-03
x(17)= 0.1700038E+01 (exact: 0.1700000E+01), r(17)= 0.1464844E-02
x(18)=-0.2349171E+04 (exact:-0.2349170E+04), r(18)= 0.1953125E-02
x(19)=-0.8247521E+04 (exact:-0.8247520E+04), r(19)=-0.8728027E-02
x(20)= 0.9843570E+04 (exact: 0.9843570E+04), r(20)= 0.0000000E+00
```

$\varepsilon = 10^{-4}$

```
niter =          1000
x( 1)= 0.1699924E+01 (exact: 0.1700000E+01), r( 1)= 0.1831055E-03
x( 2)=-0.4746890E+04 (exact:-0.4746890E+04), r( 2)=-0.4882812E-03
x( 3)= 0.5022963E+02 (exact: 0.5023000E+02), r( 3)=-0.9765625E-03
x( 4)=-0.2453193E+03 (exact:-0.2453200E+03), r( 4)= 0.1464844E-02
x( 5)= 0.4778290E+04 (exact: 0.4778290E+04), r( 5)=-0.1464844E-02
x( 6)=-0.7573022E+02 (exact:-0.7573000E+02), r( 6)=-0.1953125E-02
x( 7)= 0.3495430E+04 (exact: 0.3495430E+04), r( 7)= 0.5126953E-02
x( 8)= 0.4350277E+01 (exact: 0.4350000E+01), r( 8)=-0.4882812E-03
x( 9)= 0.4529798E+03 (exact: 0.4529800E+03), r( 9)=-0.9765625E-03
x(10)=-0.2760255E+01 (exact:-0.2760000E+01), r(10)=-0.1953125E-02
x(11)= 0.8239240E+04 (exact: 0.8239240E+04), r(11)= 0.3173828E-02
x(12)= 0.3459731E+01 (exact: 0.3460000E+01), r(12)=-0.1464844E-02
x(13)= 0.1000000E+04 (exact: 0.1000000E+04), r(13)=-0.1953125E-02
x(14)=-0.4999743E+01 (exact:-0.5000000E+01), r(14)= 0.1953125E-02
x(15)= 0.3642400E+04 (exact: 0.3642400E+04), r(15)= 0.0000000E+00
x(16)= 0.7353599E+03 (exact: 0.7353600E+03), r(16)=-0.7324219E-03
x(17)= 0.1699763E+01 (exact: 0.1700000E+01), r(17)=-0.4882812E-03
x(18)=-0.2349171E+04 (exact:-0.2349170E+04), r(18)= 0.0000000E+00
x(19)=-0.8247520E+04 (exact:-0.8247520E+04), r(19)=-0.9155273E-03
x(20)= 0.9843570E+04 (exact: 0.9843570E+04), r(20)=-0.3906250E-02
```

With CADNA

```
 niter =           29
x( 1)= 0.170E+01     (exact: 0.1699999E+01), r( 1)=@.0
x( 2)=-0.4746888E+04 (exact:-0.4746888E+04), r( 2)=@.0
x( 3)= 0.5023E+02    (exact: 0.5022998E+02), r( 3)=@.0
x( 4)=-0.24532E+03   (exact:-0.2453199E+03), r( 4)=@.0
x( 5)= 0.4778287E+04 (exact: 0.4778287E+04), r( 5)=@.0
x( 6)=-0.75729E+02   (exact:-0.7572999E+02), r( 6)=@.0
x( 7)= 0.349543E+04  (exact: 0.3495428E+04), r( 7)=@.0
x( 8)= 0.435E+01     (exact: 0.4349999E+01), r( 8)=@.0
x( 9)= 0.45298E+03   (exact: 0.4529798E+03), r( 9)=@.0
x(10)=-0.276E+01     (exact:-0.2759999E+01), r(10)=@.0
x(11)= 0.823923E+04  (exact: 0.8239236E+04), r(11)=@.0
x(12)= 0.346E+01     (exact: 0.3459999E+01), r(12)=@.0
x(13)= 0.10000E+04   (exact: 0.9999996E+03), r(13)=@.0
x(14)=-0.5001E+01    (exact:-0.4999999E+01), r(14)=@.0
x(15)= 0.364239E+04  (exact: 0.3642398E+04), r(15)=@.0
x(16)= 0.73536E+03   (exact: 0.7353597E+03), r(16)=@.0
x(17)= 0.170E+01     (exact: 0.1699999E+01), r(17)=@.0
x(18)=-0.234917E+04  (exact:-0.2349169E+04), r(18)=@.0
x(19)=-0.8247515E+04 (exact:-0.8247515E+04), r(19)=@.0
x(20)= 0.984356E+04  (exact: 0.9843565E+04), r(20)=@.0
```

Approximation methods

Approximation of a limit $L = \lim_{h \to 0} L(h)$

If $h \searrow$, truncation error $\searrow$, but rounding error $\nearrow$

How to estimate the optimal step?

Dynamical control of approximation methods

Theorem [FJ, 2006]

Let us consider a numerical method which provides an approximation $L(h)$ of order p to an exact value L:

$$L(h) - L = Kh^p + \mathcal{O}\left(h^q\right) \text{ with } 1 \leq p < q,\ K \in \mathbb{R}.$$

If L_n is the approximation computed with the step $\frac{h_0}{2^n}$, then

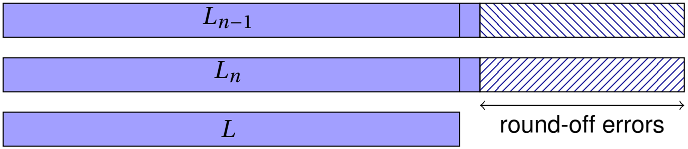
$$C_{L_n,L_{n+1}} = C_{L_n,L} + \log_{10}\left(\frac{2^p}{2^p - 1}\right) + \mathcal{O}\left(2^{n(p-q)}\right).$$

If the convergence zone is reached, the digits common to two successive iterates are also common to the exact result, up to one.

Approximation methods with the CADNA library

If iterations are stopped when $L_n - L_{n-1} = @.0$, the result L_n is optimal.

Furthermore its significant digits which are not affected by round-off errors are in common with the exact result L, up to one.



Approximation methods with the CADNA library

Example: approximations of an integral using Simpson's method

```
n= 1 Ln= 0.532202672142964E+002 err= 0.459035794670113E+002
n= 2 Ln=-0.233434428466744E+002 err= 0.306601305939595E+002
n= 3 Ln=-0.235451792663099E+002 err= 0.308618670135950E+002
...
n=13 Ln= 0.73166877473053E+001 err= 0.202E-010
n=14 Ln= 0.73166877472864E+001 err= 0.1E-011
n=15 Ln= 0.73166877472852E+001 err= 0.1E-012
n=16 Ln= 0.73166877472851E+001 err=@.0
```

The exact solution is: `7.316687747285081429939`.

Approximation methods with the CADNA library

Example: approximations of an integral using Simpson's method

```
n= 1 Ln= 0.532202672142964E+002 err= 0.459035794670113E+002
n= 2 Ln=-0.233434428466744E+002 err= 0.306601305939595E+002
n= 3 Ln=-0.235451792663099E+002 err= 0.308618670135950E+002
...
n=13 Ln= 0.73166877473053E+001 err= 0.202E-010
n=14 Ln= 0.73166877472864E+001 err= 0.1E-011
n=15 Ln= 0.73166877472852E+001 err= 0.1E-012
n=16 Ln= 0.73166877472851E+001 err=@.0
```

The exact solution is: `7.316687747285081429939`.

Also theoretical results for combined sequences
⇒ dynamical control of infinite integrals, multidimensional integrals

Deployment of CADNA on CPU-GPU

Rounding mode change

- **CPU**:
  implicit change of the rounding mode thanks to

  $a \oplus_{+\infty} b = -(-a \oplus_{-\infty} -b)$ (similarly for $\ominus$)
  $a \otimes_{+\infty} b = -(a \otimes_{-\infty} -b)$ (similarly for $\oslash$)
  $\bigcirc_{+\infty}$ (resp. $\bigcirc_{-\infty}$): floating-point operation rounded $\rightarrow +\infty$ (resp. $-\infty$)

Rounding mode change

- **CPU**:
  implicit change of the rounding mode thanks to

  $a \oplus_{+\infty} b = -(-a \oplus_{-\infty} -b)$ (similarly for $\ominus$)
  $a \otimes_{+\infty} b = -(a \otimes_{-\infty} -b)$ (similarly for $\oslash$)
  $\bigcirc_{+\infty}$ (resp. $\bigcirc_{-\infty}$): floating-point operation rounded $\rightarrow +\infty$ (resp. $-\infty$)

- **GPU**:
  Arithmetic operations with CUDA can be performed with a specified rounding mode.

  Ex:
  __fmul_ru ($\otimes_{+\infty}$)
  __fmul_rd ($\otimes_{-\infty}$)

Instability detection

- **CPU**:
  - dedicated counters are incremented
  - the occurrence of each kind of instability is given at the end of the run.

Instability detection

- **CPU**:
  - dedicated counters are incremented
  - the occurrence of each kind of instability is given at the end of the run.
- **GPU**:
  - No counter: would need a lot of atomic operations
  - An unsigned char is associated with each result (each bit associated with a type of instability).

CADNA overhead on GPU

Overhead in single (SP) and double (DP) precision on NVIDIA K20c:

	Memory Bound Add	Compute Bound Add	Memory Bound Multiply	Compute Bound Multiply
SP	$7.25\times$	$19.0\times$	$19.3\times$	$58.7\times$
DP	$6.39\times$	$12.5\times$	$18.6\times$	$49.2\times$

- Higher overheads than on CPU: warp divergence (random rounding)
- memory-bound benchmarks: lower overheads
  For an initially memory-bound code, the additional computation induced by CADNA is more easily absorbed by the GPU.

Example: matrix multiplication

```
#include "cadna.h"
#include "cadna_gpu.cu"

__global__ void matMulKernel(
                float_gpu_st* mat1,
                float_gpu_st* mat2,
                float_gpu_st* matRes,
                int dim) {

  unsigned int x = blockDim.x*blockIdx.x+threadIdx.x;
  unsigned int y = blockDim.y*blockIdx.y+threadIdx.y;

  cadna_init_gpu();

  if (x < dim && y < dim){
    float_gpu_st temp;
    temp=0;
    for(int i=0; i<dim;i++){
      temp = temp + mat1[y * dim + i] * mat2[i * dim + x];
    }
    matRes[y * dim + x] = temp;
  }
}
```

Example: matrix multiplication

```
...
float_st mat1[DIMMAT][DIMMAT], mat2[DIMMAT][DIMMAT],
res[DIMMAT][DIMMAT];
...
cadna_init(-1);
int size = DIMMAT * DIMMAT * sizeof(float_st);
cudaMalloc((void **) &d_mat1, size);
cudaMalloc((void **) &d_mat2, size);
cudaMalloc((void **) &d_res,  size);
cudaMemcpy(d_mat1, mat1, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_mat2, mat2, size, cudaMemcpyHostToDevice);

dim3 threadsPerBlock(16,16);
int nbbx = (int)ceil((float)DIMMAT/(float)16);
int nbby = (int)ceil((float)DIMMAT/(float)16);
dim3 numBlocks(nbbx , nbby);
matMulKernel<<< numBlocks , threadsPerBlock>>>
(d_mat1, d_mat2, d_res, DIMMAT);
cudaMemcpy(res, d_res,  size, cudaMemcpyDeviceToHost);
...
cadna_end();
```

Output

```
mat1=
0.0000000E+000  0.1000000E+001  0.2000000E+001  0.3000000E+001
0.4000000E+001  0.5000000E+001  0.6000000E+001  0.6999999E+001
0.8000000E+001  @.0             0.1000000E+002  0.1099999E+002
0.1199999E+002  0.1299999E+002  0.1400000E+002  0.1500000E+002

mat2=
0.1000000E+001  0.1000000E+001  0.1000000E+001  0.1000000E+001
0.1000000E+001  @.0             0.1000000E+001  0.1000000E+001
0.1000000E+001  0.1000000E+001  0.1000000E+001  0.1000000E+001
0.1000000E+001  0.1000000E+001  0.1000000E+001  0.1000000E+001

res=
0.5999999E+001  @.0                        0.5999999E+001  0.5999999E+001
0.2199999E+002  @.0                        0.2199999E+002  0.2199999E+002
@.0             @.0       MUL              @.0             @.0
0.5399999E+002  @.0                        0.5399999E+002  0.5399999E+002
----------------------------------------------------------------
No instability detected on CPU
```

Results are printed with the instabilities that have affected them.

Reproducibility failures in a wave propagation code

For oil exploration, the 3D acoustic wave equation

$$\frac{1}{c^2}\frac{\partial^2 u}{\partial t^2} - \sum_{b\in x,y,z} \frac{\partial^2}{\partial b^2} u = 0$$

where u is the acoustic pressure, c is the wave velocity and t is the time

is solved using a finite difference scheme

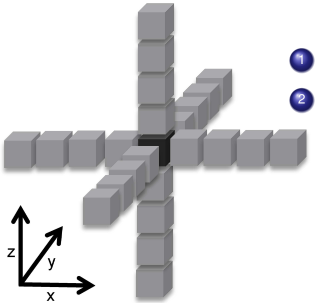
- time: order 2
- space: order p (in our case $p = 8$).

2 implementations of the finite difference scheme

1. $$u_{ijk}^{n+1} = 2u_{ijk}^{n} - u_{ijk}^{n-1} + \frac{c^2 \Delta t^2}{\Delta h^2} \sum_{l=-p/2}^{p/2} a_l \left(u_{i+ljk}^{n} + u_{ij+lk}^{n} + u_{ijk+l}^{n} \right) + c^2 \Delta t^2 f_{ijk}^{n}$$

2. $$u_{ijk}^{n+1} = 2u_{ijk}^{n} - u_{ijk}^{n-1} + \frac{c^2 \Delta t^2}{\Delta h^2} \left(\sum_{l=-p/2}^{p/2} a_l u_{i+ljk}^{n} + \sum_{l=-p/2}^{p/2} a_l u_{ij+lk}^{n} + \sum_{l=-p/2}^{p/2} a_l u_{ijk+l}^{n} \right) + c^2 \Delta t^2 f_{ijk}^{n}$$

where u_{ijk}^{n} (resp. f_{ik}^{n}) is the wave (resp. source) field in (i,j,k) coordinates and n^{th} time step and $a_{l \in -p/2, p/2}$ are the finite difference coefficients



1. nearest neighbours first
2. dimension 1, 2 then 3

Reproducibility problems

Results depend on :

- the implementation of the finite difference scheme
- the compiler / architecture (various CPUs and GPUs used)

In *binary32*, for $64 \times 64 \times 64$ space steps and 1000 time iterations:

- any two results at the same space coordinates have 0 to 7 common digits
- the average number of common digits is about 4.

Results computed at 3 different points

scheme	point in the space domain		
	$p_1 = (0, 19, 62)$	$p_2 = (50, 12, 2)$	$p_3 = (20, 1, 46)$
AMD Opteron CPU with gcc			
1	**-1.11**0479E+0	**5.454**238E+1	**6.1410**38E+2
2	**-1.11**0426E+0	**5.454**199E+1	**6.1410**35E+2
NVIDIA C2050 GPU with CUDA			
1	**-1.11**0204E+0	**5.454**224E+1	**6.1410**46E+2
2	**-1.10**9869E+0	**5.454**244E+1	**6.1410**47E+2
NVIDIA K20c GPU with OpenCL			
1	**-1.10**9953E+0	**5.454**218E+1	**6.1410**44E+2
2	**-1.11**1517E+0	**5.454**185E+1	**6.1410**24E+2
AMD Radeon GPU with OpenCL			
1	**-1.10**9940E+0	**5.454**317E+1	**6.1410**38E+2
2	**-1.11**0111E+0	**5.454**170E+1	**6.1410**44E+2
AMD Trinity APU with OpenCL			
1	**-1.11**0023E+0	**5.454**169E+1	**6.1410**62E+2
2	**-1.11**0113E+0	**5.454**261E+1	**6.1410**49E+2

Results computed at 3 different points

scheme	point in the space domain		
	$p_1 = (0, 19, 62)$	$p_2 = (50, 12, 2)$	$p_3 = (20, 1, 46)$
AMD Opteron CPU with gcc			
1	**-1.11**0479E+0	**5.454**238E+1	**6.1410**38E+2
2	**-1.11**0426E+0	**5.454**199E+1	**6.1410**35E+2
NVIDIA C2050 GPU with CUDA			
1	**-1.11**0204E+0	**5.454**224E+1	**6.1410**46E+2
2	**-1.10**9869E+0	**5.454**244E+1	**6.1410**47E+2
NVIDIA K20c GPU with OpenCL			
1	**-1.10**9953E+0	**5.454**218E+1	**6.1410**44E+2
2	**-1.11**1517E+0	**5.454**185E+1	**6.1410**24E+2
AMD Radeon GPU with OpenCL			
1	**-1.10**9940E+0	**5.454**317E+1	**6.1410**38E+2
2	**-1.11**0111E+0	**5.454**170E+1	**6.1410**44E+2
AMD Trinity APU with OpenCL			
1	**-1.11**0023E+0	**5.454**169E+1	**6.1410**62E+2
2	**-1.11**0113E+0	**5.454**261E+1	**6.1410**49E+2

How to estimate the impact of rounding errors?

The acoustic wave propagation code examined with CADNA

The code is run on:

- an AMD Opteron 6168 CPU with gcc
- an NVIDIA C2050 GPU with CUDA.

With both implementations of the finite difference scheme, the number of exact digits varies from 0 to 7 (single precision).

Its mean value is:

- 4.06 with both schemes on CPU
- 3.43 with scheme 1 and 3.49 with scheme 2 on GPU.

$\Rightarrow$ consistent with our previous observations

Instabilities detected: > 270 000 cancellations

The acoustic wave propagation code examined with CADNA

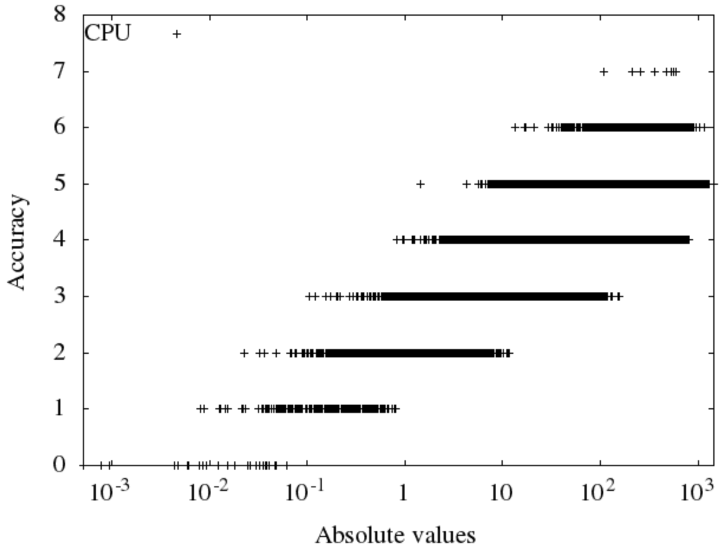
Results computed at 3 different points using scheme 1:

	Point in the space domain		
	$p_1 = (0, 19, 62)$	$p_2 = (50, 12, 2)$	$p_3 = (20, 1, 46)$
IEEE CPU	-1.110479E+0	5.454238E+1	6.141038E+2
IEEE GPU	-1.110204E+0	5.454224E+1	6.141046E+2
CADNA CPU	-1.1E+0	5.454E+1	6.14104E+2
CADNA GPU	-1.11E+0	5.45E+1	6.1410E+2
Reference	-1.108603879E+0	5.454034021E+1	6.141041156E+2

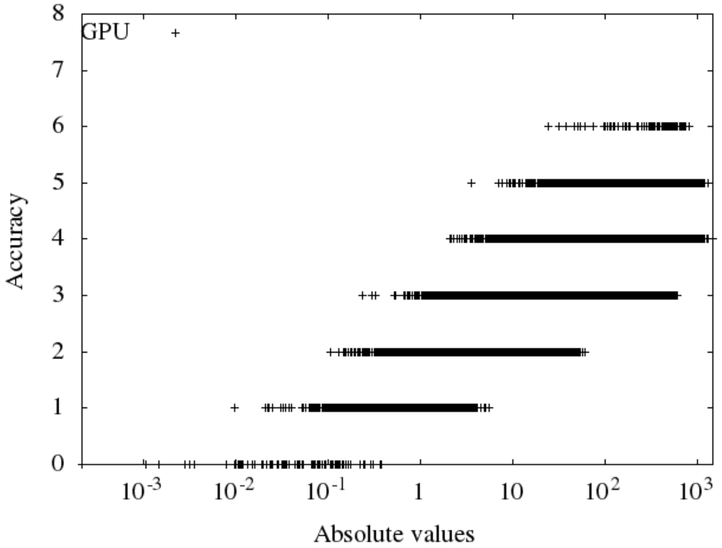
Despite differences in the estimated accuracy, the same trend can be observed on CPU and on GPU.

- Highest round-off errors impact negligible results.
- Highest results impacted by low round-off errors.

Accuracy distribution on CPU



Accuracy distribution on GPU



Numerical validation of half precision codes on GPU

Half precision (*binary16*)

- mantissa precision 11 bits ⇒ maximal accuracy: 3 decimal digits
- available for instance on Nvidia GPU P100, V100
- `half` or `half2` computation

Numerical validation of half precision codes on GPU

Half precision (*binary16*)

- mantissa precision 11 bits ⇒ maximal accuracy: 3 decimal digits
- available for instance on Nvidia GPU P100, V100
- `half` or `half2` computation

CADNA and half precision

- Extension of CADNA-GPU for half precision codes
- Application to a tiny neural network trained with backpropagation
  `https://cognitivedemons.wordpress.com/2017/09/02/`
  `a-neural-network-in-10-lines-of-cuda-c-code`
  simplified set (4 samples) from Fisher's Iris data set [Fisher, 1936]
  - input: flower characteristics (sepal length, sepal width, petal length, petal width)
  - output: Iris flower class (Iris Setosa (0) or Iris Virginica (1))

Numerical results

	Prediction	True value
float CADNA	6.099681E-02	0
	7.619311E-02	0
	9.275507E-01	1
	9.182625E-01	1
float IEEE	6.09968**2**E-02	0
	7.619311E-02	0
	9.27550**8**E-01	1
	9.18262**6**E-01	1
half CADNA	6.1E-02	0
	7.6E-02	0
	9.2E-01	1
	9.1E-01	1
half IEEE	6.09**4360**E-02	0
	7.6**29395**E-02	0
	9.27**7344**E-01	1
	9.18**4570**E-01	1

Remark: cast to single precision for printing on CPU

Numerical results

	Prediction	True value
float CADNA	6.099681E-02	0
	7.619311E-02	0
	9.275507E-01	1
	9.182625E-01	1
float IEEE	6.09968**2**E-02	0
	7.619311E-02	0
	9.27550**8**E-01	1
	9.18262**6**E-01	1
half CADNA	6.1E-02	0
	7.6E-02	0
	9.2E-01	1
	9.1E-01	1
half IEEE	6.09**4360**E-02	0
	7.6**29395**E-02	0
	9.27**7344**E-01	1
	9.18**4570**E-01	1

Remark: cast to single precision for printing on CPU

Perspective: numerical validation of larger half precision codes

CADNA for parallel codes using OpenMP and/or MPI

CADNA for OpenMP parallel codes

- In `cadna_init()`: **all threads set to rounding** $\rightarrow +\infty$
- **distinct random generator** in each thread (via `threadprivate`)
- **Detection of numerical instabilities:**
  - counters concurrently incremented by multiple threads
  - OpenMP **`atomic`** constructs for safe updates
- **Extension of OpenMP reductions** to stochastic variables:
  - **`declare reduction`** construct (OpenMP 4.0) along with the redefinition of all arithmetic operators for stochastic types
  - +, - and * operators currently supported
- **Atomic constructs:**
  - cannot be applied to CADNA stochastic (non scalar) types
  - each **`atomic` construct replaced by a `critical` block** in the user code
  - it is the **only OpenMP-CADNA modification required** in the user code: all previous modifications internal to the CADNA library

CADNA for MPI parallel codes

- Enabling **exchange of stochastic variables**

  new MPI data types:
  - MPI_FLOAT_ST
  - MPI_DOUBLE_ST
- New **reduction operators** for stochastic types (available for *, +, min, max)
- **Instability counting**:
  - count of each kind of instability for each process
  - global count of each kind of instability
- CADNA **specific functions** in MPI codes
  - cadna_mpi_init
  - cadna_mpi_end
- CADNA allows the numerical validation of **MPI-OpenMP codes.**

Tools related to CADNA

- CADNAIZER `http://cadna.lip6.fr`
  - automatically transforms C codes to be used with CADNA

Tools related to CADNA

- CADNAIZER http://cadna.lip6.fr
  - automatically transforms C codes to be used with CADNA
- CADTRACE http://cadna.lip6.fr
  - identifies in a code the instructions responsible for numerical instabilities

Example:

```
There are 11 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
```

Tools related to CADNA

- CADNAIZER `http://cadna.lip6.fr`
  - automatically transforms C codes to be used with CADNA
- CADTRACE `http://cadna.lip6.fr`
  - identifies in a code the instructions responsible for numerical instabilities

Example:

```
There are 11 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
```

- PROMISE (PRecision OptiMISE) `http://promise.lip6.fr`
  - optimises the precision of variables (quad, double, float) taking into account a target accuracy
  - checks the validity of the type configurations with CADNA

Tools related to CADNA

- CADNAIZER `http://cadna.lip6.fr`
  - automatically transforms C codes to be used with CADNA
- CADTRACE `http://cadna.lip6.fr`
  - identifies in a code the instructions responsible for numerical instabilities

Example:

```
There are 11 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
```

- PROMISE (PRecision OptiMISE) `http://promise.lip6.fr`
  - optimises the precision of variables (quad, double, float) taking into account a target accuracy
  - checks the validity of the type configurations with CADNA
- SAM (Stochastic Arithmetic in Multiprecision)

  `http://www-pequan.lip6.fr/~jezequel/SAM`
  - same functionalities as CADNA for arbitrary precision codes

Conclusion

Discrete Stochastic Arithmetic can estimate which digits are affected by round-off errors and possibly explain reproducibility failures.

- In one execution: 3 runs of the program, accuracy of any result, complete list of numerical instabilities.
- Relatively low overhead
- Support for wide range of codes (vectorised, GPU, MPI, OpenMP)
- Numerical instabilities sometimes difficult to understand in a large code
- Easily applied to real life applications

  CADNA has been successfully used for the numerical validation of academic and industrial simulation codes in various domains (astrophysics, atomic physics, chemistry, climate science, fluid dynamics, geophysics,...)

Thanks to Julien Brajard, Romuald Carpentier, Jean-Marie Chesneaux, Patrick Corde, Pacôme Eberhart, François Févotte, Pierre Fortin, Stef Graillat, Jean-Luc Lamotte, Baptiste Landreau, Bruno Lathuilière, Romain Picot, Issam Saïd, Su Zhou, ...

Thanks to Julien Brajard, Romuald Carpentier, Jean-Marie Chesneaux, Patrick Corde, Pacôme Eberhart, François Févotte, Pierre Fortin, Stef Graillat, Jean-Luc Lamotte, Baptiste Landreau, Bruno Lathuilière, Romain Picot, Issam Saïd, Su Zhou, ...

Thanks for your attention!