

Estimation d'arrondis, analyse de stabilité des grands codes de calcul numérique

Jean-Marie Chesneaux, Fabienne Jézéquel,
Jean-Luc Lamotte, Jean Vignes

Laboratoire d'Informatique de Paris 6,
P. and M. Curie University, Paris, France

Workshop quantification d'incertitude et calcul intensif

GDR MASCOT-NUM.

March 2013

- Floating-point arithmetic and round-off errors
- The CESTAC method and the stochastic arithmetic
- The CADNA software
- Contributions of CADNA in numerical methods
- CADNA for new architectures

High Performance Computing and uncertainties

Physics phenomena	→	measurement errors
Mathematical modelisation	→	modelisation errors
Modelisation discretisation	→	approximation errors
Numerical code	→	roundoff errors propagation

An example proposed by S. Rump (1)

Computation of $f(10864, 18817)$ and $f(\frac{1}{3}, \frac{2}{3})$ with $f(x, y) = 9x^4 - y^4 + 2y^2$

```
program ex1
  implicit double precision (a-h,o-z)
  x = 10864.d0
  y = 18817.d0
  write(*,*)'P(10864,18817) = ', rump(x,y)
  x = 1.d0/3.d0
  y = 2.d0/3.d0
  write(6,100) rump(x,y)
100 format('P(1/3,2/3) = ',e24.15)
end

function rump(x,y)
  implicit double precision (a-h,o-z)
  a=9.d0*x*x*x*x
  b=y*y*y*y
  c=2.d0*y*y
  rump = a-b+c
  return
end
```

An example proposed by S. Rump (2)

The results:

$$P(10864, 18817) = 2.0000000000000000$$

$$P(1/3, 2/3) = 0.802469135802469E+00$$

Rounding mode

Let \mathbb{F} be the set of real numbers which can be coded exactly on a computer: the set of floating point numbers.

Every real number x which is not a floating point number is approximated by a floating point number $X \in \mathbb{F}$.

Let X_{min} (resp. X_{max}) be the smallest (resp. the greatest) floating point number:

$$\forall x \in]X_{min}, X_{max}[, \exists \{X^-, X^+\} \in \mathbb{F}^2$$

such that

$$X^- < x < X^+ \text{ and }]X^-, X^+[\cap \mathbb{F} = \emptyset$$

To choose the rounding mode is to choose the algorithm that, according to x , gives X^- or X^+ .

The 4 rounding modes of the IEEE 754 standard

Rounding to zero: x is represented by the floating point number the nearest to x between x and 0.

Rounding to nearest: x is represented by the floating point number the nearest to x .

Rounding to plus infinity: x is represented by X^+ .

Rounding to minus infinity: x is represented by X^- .

The rounding operation is performed after each assignment and after every elementary arithmetic operation.

Inconsistency of the floating point arithmetic

On a computer, arithmetic operators are only approximations.

- commutativity
- no associativity
- no distributivity

On a computer, order relationships are the same as in mathematics

⇒ it leads to a global inconsistent behaviour.

$$X = Y \not\Rightarrow x = y \quad \text{and} \quad x = y \not\Rightarrow X = Y.$$

$$X \geq Y \not\Rightarrow x \geq y \quad \text{and} \quad x \geq y \not\Rightarrow X \geq Y.$$

Round-off error model

Let $r \in \mathbb{R}$ be the exact result of a computation of n elementary arithmetic operations.

On a computer, one obtains the result $R \in \mathbb{F}$ which is affected by round-off errors.

R can be modeled, at the first order with respect to 2^{-p} , by

$$R \approx r + \sum_{i=1}^n g_i(d).2^{-p}.\alpha_i$$

p is the number of bits used for the representation including the hidden bit, $g_i(d)$ are coefficients depending only on data and α_i are the round-off errors.

Remark: we have assumed that exponents and signs of intermediate results do not depend on α_j .

A theorem on numerical accuracy

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

$$C_R \approx -\log_2 \left| \frac{R-r}{r} \right| = p - \log_2 \left| \sum_{i=1}^n g_i(d) \cdot \frac{\alpha_i}{r} \right|$$

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**

- uses a random rounding mode
- estimates the number of exact significant digits of any computed result

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 \bar{R} of R_i , $i = 1, \dots, N$,
- estimating the number of exact significant decimal digits of \bar{R} with

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

where

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

τ_{β} is the value of Student's distribution for $N - 1$ degrees of freedom and a probability level β .

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 $C_{\bar{R}}$, 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.

Self-validation of the CESTAC method

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.

Therefore the CESTAC method requires a dynamical control of multiplications and divisions, during the execution of the code.

The problem of stopping criteria

Let a general iterative algorithm be: $U_{n+1} = F(U_n)$, U_0 being a data.

```
WHILE (ABS(X-Y) > EPSILON) DO  
  X = Y  
  Y = F(X)  
ENDDO
```

ε too low \implies a risk of infinite loop

ε too high \implies a too early termination.

The optimal choice from the computer point of view:

$X - Y$ **an insignificant value.**

New methods for numerical algorithms may be developed.

The concept of computed zero

J. Vignes, 1986

Definition

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

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

This means that 0 belongs to the confidence interval.

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

The stochastic definitions

Definition

Let X and Y be two computed results using the CESTAC method (N -sample), X is stochastically equal to Y , noted $X \text{ s} = Y$, if and only if

$$X - Y = @.0.$$

Definition

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

- X is stochastically strictly greater than Y , noted $X \text{ s} > Y$, if and only if

$$\bar{X} > \bar{Y} \text{ and } X \text{ s} \neq Y$$

- X is stochastically greater than or equal to Y , noted $X \text{ s} \geq Y$, if and only if

$$\bar{X} \geq \bar{Y} \text{ or } X \text{ s} = Y$$

DSA **Discrete Stochastic Arithmetic** is defined as the joint use of the CESTAC method, the computed zero and the relation definitions.

A few properties

- $x = 0 \implies X = @.0$.
- $X \text{ s}\neq Y \implies x \neq y$.
- $X \text{ s}\> Y \implies x > y$.
- $x \geq y \implies X \text{ s}\geq Y$.
- The relation $\text{s}\>$ is transitive.
- The relation $\text{s}=\text{}$ is reflexive, symmetric but not transitive.
- The relation $\text{s}\geq$ is reflexive, antisymmetric but not transitive.

The CADNA library

The CADNA library implements Discrete Stochastic Arithmetic.

CADNA allows to estimate round-off error propagation in any scientific program.

More precisely, CADNA enables one to:

- estimate the numerical quality of any result
- control branching statements
- perform a dynamic numerical debugging
- take into account uncertainty on data.

CADNA is a library which can be used with Fortran or C++ programs and also with MPI parallel programs.

CADNA can be downloaded from <http://www.lip6.fr/cadna>

The stochastic types

CADNA provides two new numerical types, the stochastic types (3 floating point variables x, y, z and a hidden variable `acc`):

- type `(single_st)` for stochastic variables in single precision stochastic type associated with real.
- type `(double_st)` for stochastic variables in double precision stochastic type associated with double precision.

All the operators and mathematical functions are overloaded for these types.

The cost of CADNA is about:

- 4 for memory
- 10 for run time.

How to implement CADNA

The use of the CADNA library involves six steps:

- declaration of the CADNA library for the compiler,
- initialization of the CADNA library,
- substitution of the type REAL or DOUBLE PRECISION by stochastic types in variable declarations,
- possible changes in the input data if perturbation is desired, to take into account uncertainty in initial values,
- change of output statements to print stochastic results with their accuracy,
- termination of the CADNA library.

```

program ex1

implicit double precision (a-h,o-z)

x = 10864.d0
y = 18817.d0
write(*,*)'P(10864,18817) = ', rump(x,y)
x = 1.d0/3.d0
y = 2.d0/3.d0
write(*,*)'P(10864,18817) = ', rump(x,y)

end

function rump(x,y)

implicit double precision (a-h,o-z)
a = 9.d0*x*x*x*x
b = y*y*y*y
c = 2.d0*y*y
rump = a-b+c
return
end

```

```

program ex1
  use cadna
  implicit double precision  (a-h,o-z)

  x = 10864.d0
  y = 18817.d0
  write(*,*)'P(10864,18817) = ', rump(x,y)
  x = 1.d0/3.d0
  y = 2.d0/3.d0
  write(*,*)'P(10864,18817) = ', rump(x,y)

end

function rump(x,y)
  use cadna
  implicit double precision  (a-h,o-z)
  a = 9.d0*x*x*x*x*x
  b = y*y*y*y
  c = 2.d0*y*y
  rump = a-b+c
  return
end

```

```

program ex1
use cadna
implicit double precision (a-h,o-z)
call cadna_init(-1)
x = 10864.d0
y = 18817.d0
write(*,*)'P(10864,18817) = ', rump(x,y)
x = 1.d0/3.d0
y = 2.d0/3.d0
write(*,*)'P(10864,18817) = ', rump(x,y)

end

function rump(x,y)
use cadna
implicit double precision (a-h,o-z)
a = 9.d0*x*x*x*x*x
b = y*y*y*y
c = 2.d0*y*y
rump = a-b+c
return
end

```



```
program ex1
use cadna
implicit double precision (a-h,o-z)
call cadna_init(-1)
x = 10864.d0
y = 18817.d0
write(*,*)'P(10864,18817) = ', rump(x,y)
x = 1.d0/3.d0
y = 2.d0/3.d0
write(*,*)'P(10864,18817) = ', rump(x,y)
call cadna_end()
end
```

```
function rump(x,y)
use cadna
implicit double precision (a-h,o-z)
a = 9.d0*x*x*x*x*x
b = y*y*y*y
c = 2.d0*y*y
rump = a-b+c
return
end
```

```
program ex1
use cadna
implicit double precision (a-h,o-z)
call cadna_init(-1)
x = 10864.d0
y = 18817.d0
write(*,*)'P(10864,18817) = ', rump(x,y)
x = 1.d0/3.d0
y = 2.d0/3.d0
write(*,*)'P(10864,18817) = ', rump(x,y)
call cadna_end()
end
```

```
function rump(x,y)
use cadna
implicit double precision (a-h,o-z)
a = 9.d0*x*x*x*x*x
b = y*y*y*y
c = 2.d0*y*y
rump = a-b+c
return
end
```

```
program ex1
use cadna
implicit type(double_st) (a-h,o-z)
call cadna_init(-1)
x = 10864.d0
y = 18817.d0
write(*,*)'P(10864,18817) = ', rump(x,y)
x = 1.d0/3.d0
y = 2.d0/3.d0
write(*,*)'P(10864,18817) = ', rump(x,y)
call cadna_end()
end
```

```
function rump(x,y)
use cadna
implicit type(double_st) (a-h,o-z)
a = 9.d0*x*x*x*x*x
b = y*y*y*y
c = 2.d0*y*y
rump = a-b+c
return
end
```

```
program ex1
use cadna
implicit type(double_st) (a-h,o-z)
call cadna_init(-1)
x = 10864.d0
y = 18817.d0
write(*,*)'P(10864,18817) = ', rump(x,y)
x = 1.d0/3.d0
y = 2.d0/3.d0
write(*,*)'P(10864,18817) = ', rump(x,y)
call cadna_end()
end
```

```
function rump(x,y)
use cadna
implicit type(double_st) (a-h,o-z)
a = 9.d0*x*x*x*x*x
b = y*y*y*y
c = 2.d0*y*y
rump = a-b+c
return
end
```

```
program ex1
use cadna
implicit type(double_st) (a-h,o-z)
call cadna_init(-1)
x = 10864.d0
y = 18817.d0
write(*,*)'P(10864,18817) = ',str(rump(x,y))
x = 1.d0/3.d0
y = 2.d0/3.d0
write(*,*)'P(10864,18817) = ',str(rump(x,y))
call cadna_end()
end
```

```
function rump(x,y)
use cadna
implicit type(double_st) (a-h,o-z)
a = 9.d0*x*x*x*x
b = y*y*y*y
c = 2.d0*y*y
rump = a-b+c
return
end
```

The run with CADNA

CADNA software — University P. et M. Curie — LIP6

Self-validation detection: ON

Mathematical instabilities detection: ON

Branching instabilities detection: ON

Intrinsic instabilities detection: ON

Cancellation instabilities detection: ON

$P(10864,18817) = @.0$

$P(1/3,2/3) = 0.802469135802469E+000$

CADNA software — University P. et M. Curie — LIP6

There are 2 numerical instabilities

0 UNSTABLE DIVISION(S)

0 UNSTABLE POWER FUNCTION(S)

0 UNSTABLE MULTIPLICATION(S)

0 UNSTABLE BRANCHING(S)

0 UNSTABLE MATHEMATICAL FUNCTION(S)

0 UNSTABLE INTRINSIC FUNCTION(S)

2 UNSTABLE CANCELLATION(S)

- 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 the 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 $0.3500000E+01$

Iterative methods: which strategy to adopt?

- problems with a solution that cannot be controlled (sequence computation):
The following stopping criterion should be used

IF $(x(k).eq.x(k + 1))$ THEN

- problems with a solution that can be controlled:
the solution x_s satisfies $\Psi(x_s) = 0$.
The optimal stopping criterion should be used

IF $(\Psi(x(k)).eq.0)$ THEN

Iterative methods: the solution cannot be controlled

$$S_n(x) = \sum_{i=1}^n \frac{x^i}{i!}$$

Stopping criterion

- IEEE: $|S_n - S_{n-1}| < 10^{-15}|S_n|$
- CADNA: $S_n == S_{n-1}$

	IEEE		CADNA	
x	iter	$S_n(x)$	iter	$S_n(x)$
-5.	37	6.737946999084039E-003	38	0.673794699909E-002
-10.	57	4.539992962303130E-005	58	0.45399929E-004
-15.	76	3.059094197302006E-007	77	0.306E-006
-20.	94	5.621884472130416E-009	95	@.0
-25.	105	-7.129780403672074E-007	106	@.0

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

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

Without CADNA

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

eps=1.E-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
```

eps=1.E-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
```

Specific implementations

- Specific implementation using SSE for X86 architecture
- SAM : extended precision of CADNA
- parallel codes using MPI
 - need to define new MPI types for the stochastic types
 - works as for sequential codes
- parallel codes using OpenMP
 - No implementation
 - problem with the reduction operation (see OpenMP 4)
- GPU applications

- C++ : enough functionalities in CUDA 3.0 to overload operators and functions
- Random functions and the random rounding mode
- Instability detection and trace

The random rounding mode

CPU

```
if (RANDOM) rnd_switch();  
res.x=a.x*b.x;  +∞
```

```
if (RANDOM) rnd_switch();  
res.y=a.y*b.y;  -∞  
rnd_switch();  
res.z=a.z*b.z;  +∞
```

GPU

```
if (RANDOMGPU())  
    res.x=__fmul_ru(a.x,b.x);  
else  
    res.x=__fmul_rd(a.x,b.x);
```

```
if (RANDOMGPU()) {  
    res.y=__fmul_rd(a.y,b.y);  
    res.z=__fmul_ru(a.z,b.z);  
}  
else {  
    res.y=__fmul_ru(a.y,b.y);  
    res.z=__fmul_rd(a.z,b.z);  
}
```

2 types `float_st` for CPU computation and `float_gpu_st` for GPU computation.

Implemented solutions

- No counter for the numerical instabilities.
 - need more memory (shared)
 - need a lot of atomic operations
 - instabilities are associated to a result.

CPU + GPU

```
class float_st {  
protected:  
float x,y,z;  
private:  
mutable unsigned int accuracy;  
unsigned char accuracy;  
mutable unsigned char error;  
unsigned char pad1, pad2;  
}
```

GPU

```
class float_gpu_st {  
public:  
float x,y,z;  
public:  
mutable unsigned char accuracy;  
mutable unsigned char error;  
unsigned char pad1, pad2; }
```

Implemented solutions

- No counter for the numerical instabilities.
 - need more memory (shared)
 - need a lot of atomic operations
 - instabilities are associated to a result.

CPU + GPU

```
class float_st {  
protected:  
float x,y,z;  
private:  
mutable unsigned int accuracy;  
unsigned char accuracy;  
mutable unsigned char error;  
unsigned char pad1, pad2;  
}
```

GPU

```
class float_gpu_st {  
public:  
float x,y,z;  
public:  
mutable unsigned char accuracy;  
mutable unsigned char error;  
unsigned char pad1, pad2; }
```

Implemented solutions

- No counter for the numerical instabilities.
 - need more memory (shared)
 - need a lot of atomic operations
 - instabilities are associated to a result.

CPU +GPU

```
class float_st {
protected:
float x,y,z;
private:
mutable unsigned int accuracy;
unsigned char accuracy;
mutable unsigned char error;
unsigned char pad1, pad2;
}
```

GPU

```
class float_gpu_st {
public:
float x,y,z;
public:
mutable unsigned char accuracy;
mutable unsigned char error;
unsigned char pad1, pad2; }
```

Computation of 2D Slater integrals

2DRMP is a suite of 2D R-matrix propagation programs simulating electron scattering from H-like atoms and ions at intermediate energies. We focus on the NEWRD program which involves the computation of a large number of Slater integrals:

$$I_\lambda = J_{1,\lambda} + J_{2,\lambda}$$

with

$$\begin{aligned} J_{1,\lambda} &= \int_a^b \int_a^y f_\lambda(x, y) dx dy, \\ f_\lambda(x, y) &= \frac{P_{n_1, l_1}(y) P_{n_3, l_3}(y)}{y^{\lambda+1}} x^\lambda P_{n_2, l_2}(x) P_{n_4, l_4}(x), \quad x \in [a, y], \\ J_{2,\lambda} &= \int_a^b \int_y^b \phi_\lambda(x, y) dx dy, \\ \phi_\lambda(x, y) &= P_{n_1, l_1}(y) P_{n_3, l_3}(y) y^\lambda \frac{P_{n_2, l_2}(x) P_{n_4, l_4}(x)}{x^{\lambda+1}}, \quad x \in [y, b] \end{aligned}$$

P_{n_i, l_i} : eigenfunctions of the Schrödinger equation

- Intel Core 2 Quad Processor Q8200 (4M Cache, 2.33 GHz, 1333 MHz FSB)
- NVIDIA C1060
- NVIDIA C2050 FERMI, CUDA Core 448, Mem 3GB GDDR5

Architecture	time without CADNA	time with CADNA	ratio
CPU	501 sec		
C1060 GPU without texture	26 sec	7 min 22	17
C1060 GPU with texture	21 sec	7 min 15	20.7
C2050 GPU without texture	22 sec	3 min 20	9
C2050 GPU with texture	22 sec	3 min 20	9

All the results are correct with 7 significant digits.

Conclusion

- Efficient method but time and memory consuming
- Can be used on real life applications
- Difficulties to understand the numerical instabilities in large codes
- solution for parallel programs (MPI and GPU)
- difficult to use with the libraries (BLAS, LAPACK ...)
- project : a compiler for CADNA