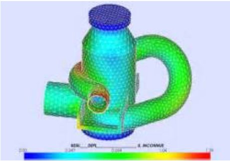
VERROU: a stochastic arithmetic evaluation without recompilation

SCAN 2016
September 28th, 2016

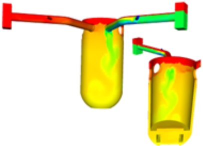
F. Févotte & B. Lathuilière

EDF R&D

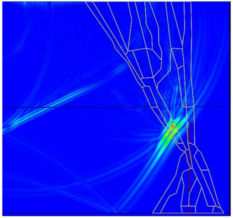
Need for computing codes
In-house development



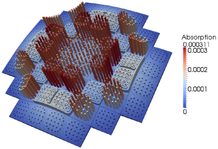
Structures



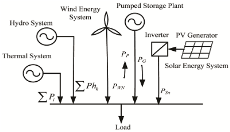
Fluid dynamics



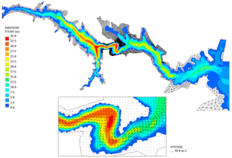
Wave propagation



Neutronics

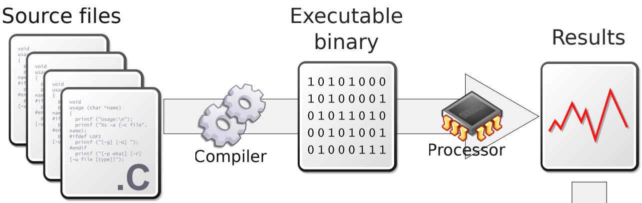


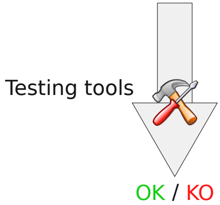
Power Systems



Free surface hydraulics

Development + V&V process

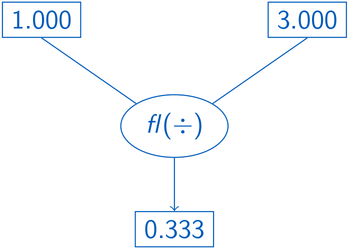




- Issues:
  - Numerical methods
  - **Floating-point arithmetic**
  - Bugs

Discrete Stochastic Arithmetic [1]

CADNA [2]

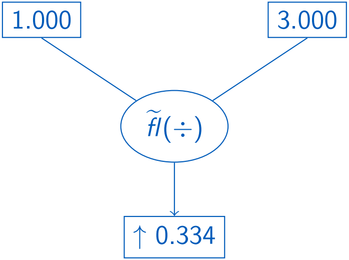


[1] J. Vignes, "A stochastic arithmetic for reliable scientific computation," *Mathematics and Computers in Simulation*, vol. 35, no. 3, 1993.

[2] J.-L. Lamotte, J.-M. Chesneaux and F. Jézéquel, "CADNA_C: A version of CADNA for use with C or C++ programs", *Computer Physics Communications*, vol. 181, no. 11, 2010.

Discrete Stochastic Arithmetic [1]

CADNA [2]

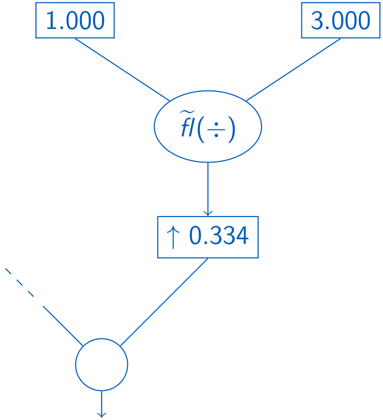


[1] J. Vignes, "A stochastic arithmetic for reliable scientific computation," *Mathematics and Computers in Simulation*, vol. 35, no. 3, 1993.

[2] J.-L. Lamotte, J.-M. Chesneaux and F. Jézéquel, "CADNA_C: A version of CADNA for use with C or C++ programs", *Computer Physics Communications*, vol. 181, no. 11, 2010.

Discrete Stochastic Arithmetic [1]

CADNA [2]

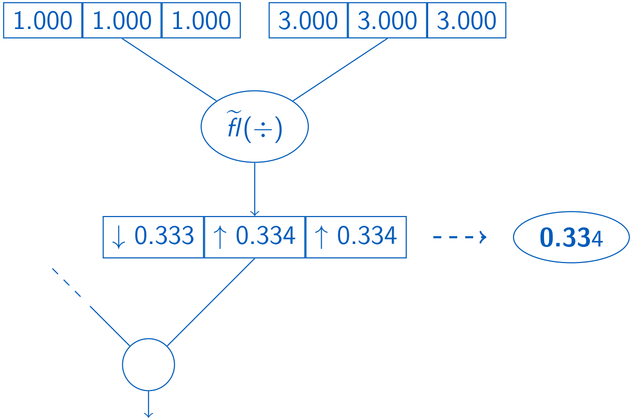


[1] J. Vignes, "A stochastic arithmetic for reliable scientific computation," *Mathematics and Computers in Simulation*, vol. 35, no. 3, 1993.

[2] J.-L. Lamotte, J.-M. Chesneaux and F. Jézéquel, "CADNA_C: A version of CADNA for use with C or C++ programs", *Computer Physics Communications*, vol. 181, no. 11, 2010.

Discrete Stochastic Arithmetic [1]

CADNA [2]

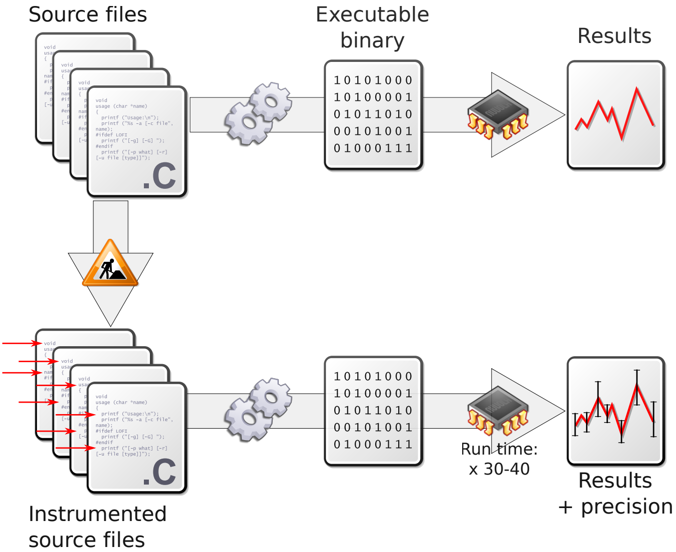


[1] J. Vignes, "A stochastic arithmetic for reliable scientific computation," *Mathematics and Computers in Simulation*, vol. 35, no. 3, 1993.

[2] J.-L. Lamotte, J.-M. Chesneaux and F. Jézéquel, "CADNA_C: A version of CADNA for use with C or C++ programs", *Computer Physics Communications*, vol. 181, no. 11, 2010.

Development + V&V process

CADNA: source code instrumentation

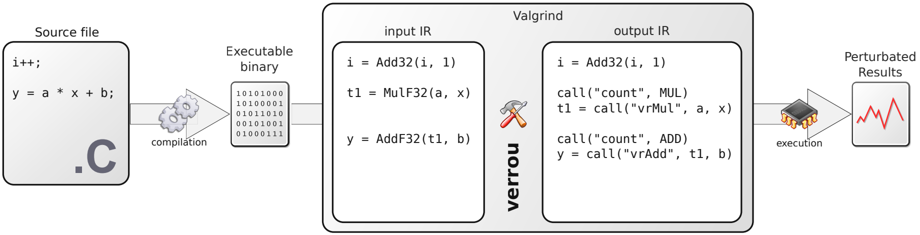


Verrou

	CADNA	VERROU
Method	sync.	async.
Instrumentation	sources	binary
Localization	fine	coarse

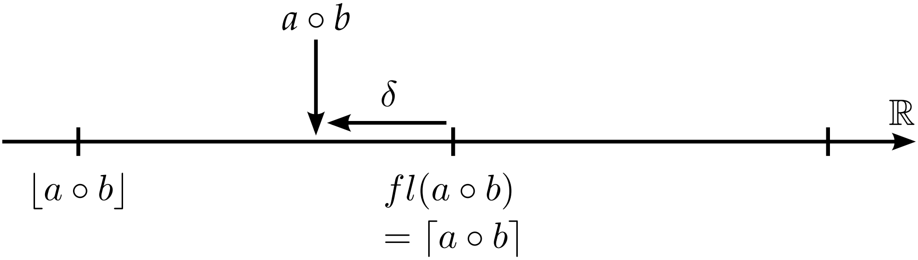
Dynamic binary analysis with valgrind

```
$ valgrind --tool=verrou --rounding-mode=random PROGRAM [ARGS...]
```



Verrou features

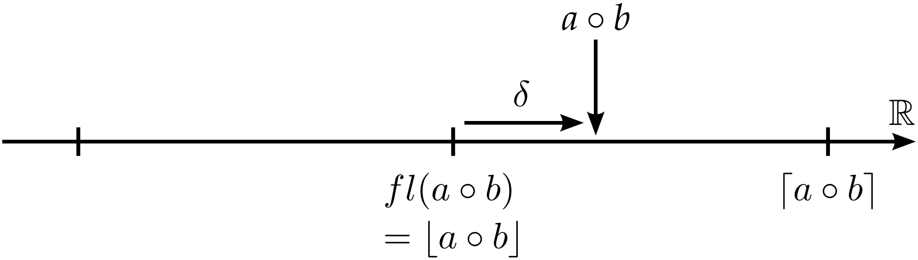
Change rounding modes (stochastic arithmetic)



- Error-free transformation
  (the division is a bit more involved):
  - $a \circ b = \sigma + \delta$,
  - $\sigma = fl(a \circ b)$
- If $\delta < 0$:
  - $\lfloor a \circ b \rfloor = fl(a \circ b) - ulp$,
  - $\lceil a \circ b \rceil = fl(a \circ b)$.

Verrou features

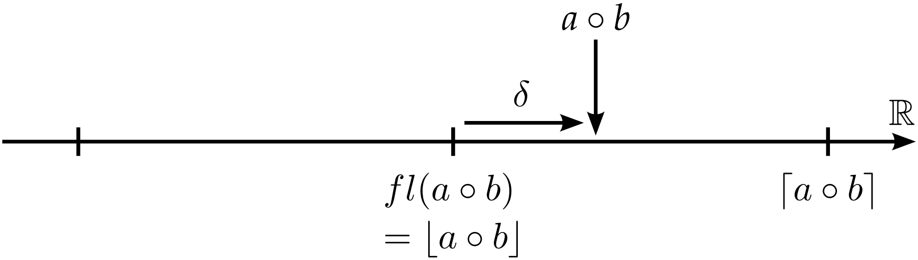
Change rounding modes (stochastic arithmetic)



- Error-free transformation
  (the division is a bit more involved):
  - $a \circ b = \sigma + \delta$,
  - $\sigma = fl(a \circ b)$
- If $\delta > 0$:
  - $\lfloor a \circ b \rfloor = fl(a \circ b)$,
  - $\lceil a \circ b \rceil = fl(a \circ b) + ulp$.

Verrou features

Change rounding modes (stochastic arithmetic)



- Random rounding mode:
  - $p(\lceil a \circ b \rceil) = 0.5$
  - $p(\lfloor a \circ b \rfloor) = 0.5$

Verrou: instrumented sections

Based on symbol name (or source file + line)

```
valgrind --tool=verrou --rounding-mode=random  python
```

```
» import math

» math.cos(42.)
-4.5847217124585136

» math.cos(42.)
-4.6689026578736614

» math.cos(42.)
-0.39998531498835133
```

Verrou: instrumented sections

Based on symbol name (or source file + line)

```
valgrind --tool=verrou --rounding-mode=random  \
         --exclude=libmath.exclude python
```

```
» import math
» math.cos(42.)
==17509== Using exclusion rule: * /lib/libm-2.11.3.so
-0.39998531498835127
» math.cos(42.)
-0.39998531498835127
» math.cos(42.)
-0.39998531498835127
```

- File `libmath.exclude`:

```
#sym  lib
*     /lib/libm-2.11.3.so
```

Verrou + Delta Debugging

Automated bisection of instrumented sections

- Delta Debugging [1]:
  - automatically isolate failure-inducing circumstances,
  - needs a way to check "deltas" → Verrou.
- Two passes:
  - at the function (symbol) level,
  - at the source file + line level if available (binary compiled with -g).
- Output (DDmax):
  - everything **not** listed works fine,
  - anything listed is unstable
    (randomly changing rounding modes produces large errors).

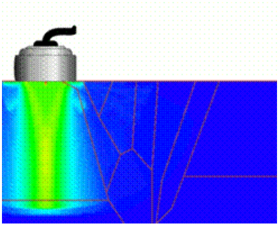
[1] A. Zeller and R. Hildebrandt, "Simplifying and isolating failure-inducing input," *IEEE Trans. Softw. Eng.*, vol. 28, no. 2, 2002.

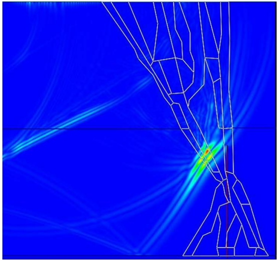
Real world examples

1. Ultra-sonic non-destructive evaluations
2. Unit commitment problem

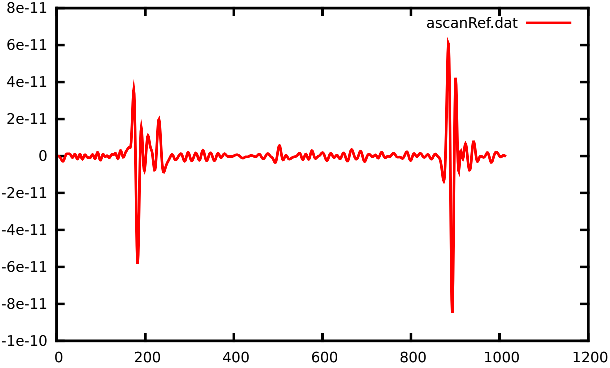
Example 1: Ultra-sonic non-destructive evaluations

Description





Result : "A-scan"



Example 1: Ultra-sonic non-destructive evaluations

Description

Computing code [1]

- Fortran 77+90
  - (2D) 36k lines
  - (3D) 70k lines
- dependencies:
  - (2D) BLAS, LAPACK,
  - (3D) BLAS, LAPACK, UmfPack, MPI

Objectives

- no identified problem
- "routine check"

[1] B. Chassignole, R. El Guerjouma, M.-A. Ploix and T. Fouquet, "Ultrasonic and structural characterization of anisotropic austenitic stainless steel welds: Towards a higher reliability in ultrasonic non-destructive testing", *NDT & E International*, vol. 43, no. 4, 2010.

Example 1: Ultra-sonic non-destructive evaluations

Non-regression tests under `verrou`

	random 1	random 2	random 3	random 4
case A				
ins1.dat	0	6.1e-06	6.1e-06	6.1e-06
ascan.dat	1.8e-12	5.9e-12	5.9e-12	5.9e-12
case B				
sismo.dat	7.9e-69	7.9e-69	4.3e-69	4.3e-69
ascan.dat	1.2e-10	2.0e-11	2.8e-10	1.1e-11
case C				
ins1.dat	4.6e-06	4.6e-06	4.6e-06	0
sismo.dat	8.0e-28	2.8e-28	8.0e-28	0
ascan.dat	2.0e-11	1.2e-11	1.8e-11	0
case D				
ins1.dat	1.5e-18	**4.1e-01**	**2.0e-01**	0
enerloc.dat	0	**2.3e-01**	**1.2e-01**	0
sismo.dat	0	**1.6e-01**	**3.2e-02**	0
ascan.dat	0	**1.5e-01**	**3.6e-01**	**6.5e-03**

Example 1: Ultra-sonic non-destructive evaluations

Non-regression tests under `verrou`

	random 1	random 2	random 3	random 4
case A				
ins1.dat	0	6.1e-06	6.1e-06	6.1e-06
ascan.dat	1.8e-12	5.9e-12	5.9e-12	5.9e-12
case B				
sismo.dat	7.9e-69	7.9e-69	4.3e-69	4.3e-69
ascan.dat	[illegible]	[illegible]	[illegible]	1.1e-11
case C				
ins1.dat	[illegible]	[illegible]	[illegible]	0
sismo.dat	8.0e-28	2.8e-28	8.0e-28	0
ascan.dat	2.0e-11	1.2e-11	1.8e-11	0
case D				
ins1.dat	1.5e-18	**4.1e-01**	**2.0e-01**	0
enerloc.dat	0	**2.3e-01**	**1.2e-01**	0
sismo.dat	0	**1.6e-01**	**3.2e-02**	0
ascan.dat	0	**1.5e-01**	**3.6e-01**	**6.5e-03**

Unstable algorithm for sensor placement

Example 1: Ultra-sonic non destructive evaluations

Verrou performance

	reference	random	average
case A	4.70s	83.23s (x17)	90.49s (x19)
case B	29.79s	969.54s (x32)	1042.02s (x34)
case C	21.15s	326.81s (x15)	358.08s (x16)
case D	1.99s	24.20s (x12)	25.87s (x12)
case E	0.46s	7.88s (x17)	8.88s (x19)
case F	0.38s	4.54s (x11)	4.95s (x12)
case G	6.16s	100.31s (x16)	109.70s (x17)
case H	14.09s	503.90s (x35)	549.50s (x39)
case I	1.48s	14.34s (x9)	14.85s (x10)

Slow-down between $\times 9$ **and** $\times 39$

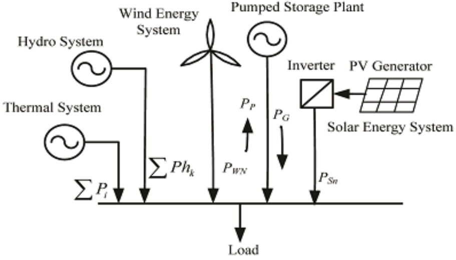
Example 2: Unit Commitment Problem
Description

Unit commitment problem

- ensure the production = consumption balance,
- minimize costs.

Computing code

- 300k+ lines of Fortran,
- “black box”:
  - no prior knowledge,
  - no access to sources;
- depends on IBM ILOG CPLEX.



Objectives

- non-reproducibility issue triggered by changing the numbering of power plants.

Example 2: Unit Commitment Problem

Verrou + Delta Debugging

Unstable symbols

```
couhyd_pi_
coutot_
decopt_pi_
ihyd_
iprit_
matctr_pi_
nzsv1_
opti_un_grth_
pildef_
prosca_
proxmyqn_
recrea_pi_
relax_vol_
remise_grad_
scale_hyd_
thpdyn_
```

Unstable file+lines

```
COUHYD_PI.f:196
COUHYD_PI.f:197
COUHYD_PI.f:198
COUHYD_PI.f:199
COUHYD_PI.f:200
COUHYD_PI.f:203
COUHYD_PI.f:204
COUHYD_PI.f:205
COUHYD_PI.f:206
COUHYD_PI.f:208
COUHYD_PI.f:211
COUHYD_PI.f:212
COUHYD_PI.f:213
COUHYD_PI.f:215

COUTOT.f:61
COUTOT.f:64
COUTOT.f:65
COUTOT.f:70
COUTOT.f:91
COUTOT.f:96
COUTOT.f:98
```

⋮

Example 2: Unit Commitment Problem

Verrou + Delta Debugging

- Unstable symbols

```
couhyd_pi_
coutot_
decopt_pi_
ihyd_
iprit_
matctr_pi_
nzsv1_
opti_un_grth_
pildef_
prosca_
proxmyqn_
recrea_pi_
relax_vol_
remise_grad_
scale_hyd_
thpdyn_
```

- Unstable file+lines

```
COUHYD_PI.f:196
COUHYD_PI.f:197
COUHYD_PI.f:198
COUHYD_PI.f:199
COUHYD_PI.f:200
COUHYD_PI.f:203
COUTOT.f:61
COUTOT.f:64
COUTOT.f:65
COUTOT.f:70
COUTOT.f:91
COUTOT.f:96
COUTOT.f:98
⋮
```

- 1/2 day to set things up
- 4 days for DD to complete (slow down = 9×)
- **instability found!**

Conclusions – Perspectives

Conclusions

Verrou seems to cover our needs (as industrials):

- practically no entry cost,
- CESTAC-like unstabilities quantification,
- coarse-grain localization of errors.

Perspectives

- Handle all instructions:
  - AVX & single precision SSE vector instructions,
  - x87 scalar instructions ;
- Specifically handle functions from the libmath ;
- Couple Verrou and Cadna.

Thank you!

Get `verrou` on github:
`https://github.com/edf-hpc/verrou`

Questions?

1. Dynamic Binary Analysis with Valgrind
2. Verrou features
3. Division
4. Validation
5. V&V process

Dynamic binary analysis with valgrind

- C code:

```
i++;

y = a * x + b;
```

- Valgrind input:

```
i = Add32(i, 1)

t1 = MulF32(a, x)

y = AddF32(t1, b)
```

- Valgrind output:

```
i = Add32(i, 1)

Call("count", MUL)
t1 = MulF32(a, x)

Call("count", ADD)
Call("detectCancel", t1, b)
y = Call("myAdd", t1, b)
```

- Instrumentation choices:
  - Do nothing (copy the instruction as is)
  - Insert new instructions:
    - count things
    - detect errors
  - Replace instructions
    - **but Valgrind only handles rounding to** `NEAREST`

Verrou features

Count instructions

```
$ valgrind --tool=verrou --rounding-mode=random PROGRAM [ARGS...]
```

```
==4683== Verrou, Check floating-point rounding errors
==4683== Copyright (C) 2014, F. Fevotte & B. Lathuiliere.
...
==4683== First seed : 1430818339
==4683== Simulating AVERAGE rounding mode
==4683== Instrumented operations :
==4683==  add : yes
...
==4683==  ---------------------------------------------------------------------
==4683==  Operation                    Instructions count
==4683==   `- Precision       Instrumented                Total
==4683==  ---------------------------------------------------------------------
==4683==  add                 500869335                  500869335          (100%)
==4683==   `- flt                 400695468                  400695468      (100%)
==4683==   `- dbl                 100173867                  100173867      (100%)
==4683==  ---------------------------------------------------------------------
==4683==  sub                 763127658                  763127658          (100%)
==4683==   `- flt                 763127658                  763127658      (100%)
==4683==  ---------------------------------------------------------------------
==4683==  mul                1202086563                 1202086563          (100%)
==4683==   `- flt                1101912537                 1101912537      (100%)
==4683==   `- dbl                 100174026                  100174026      (100%)
==4683==  ---------------------------------------------------------------------
...
```

Verrou features

Count instructions

```
$ valgrind --tool=verrou --rounding-mode=random PROGRAM [ARGS...]
```

```
==4683== Verrou, Check floating-point rounding errors
==4683== Copyright (C) 2014, F. Fevotte & B. Lathuiliere.
...
==4683== First seed : 1430818339
==4683== Simulating AVERAGE rounding mode
==4683== Instrumented operations :
==4683==   add : yes
...
==4683==                                                        --------------------
==4683==
==4683==
==4683==                                                        --------------------
==4683==                                                        9335          (100%)
==4683==    `- flt                400695468               400695468          (100%)
==4683==    `- dbl                100173867               100173867          (100%)
==4683==   -----------------------------------------------------------------------
==4683==   sub               763127658               763127658              (100%)
==4683==    `- flt                763127658               763127658          (100%)
==4683==   -----------------------------------------------------------------------
==4683==   mul              1202086563              1202086563              (100%)
==4683==    `- flt               1101912537              1101912537          (100%)
==4683==    `- dbl                100174026               100174026          (100%)
==4683==   -----------------------------------------------------------------------
...
```

Normal program output

+ **Warnings for x87 scalar instructions**

Verrou features: instrumented sections

Using client requests

```
valgrind --tool=verrou --instr-atstart=no PROGRAM
```

```c
#include <valgrind/verrou.h>
#include <stdio.h>

float compute ();

int main () {
  VERROU_START_INSTRUMENTATION;
  float result = compute();
  VERROU_STOP_INSTRUMENTATION;

  fprintf (stdout, "result = %f", result);
}
```

- Need to recompile (part of) the code + re-link

Approximated transformation for the division

- What we would like to have:

$$\frac{a}{b} = q + r,$$

with $q = \mathrm{fl}(a/b)$.

- Proposed algorithm:

Input : a, b
Output : $\tilde{r}$ such that
$a/b \simeq \mathrm{fl}(a/b) + \tilde{r}$.

$q \leftarrow \mathrm{fl}(a/b)$
$(p, s) \leftarrow \mathrm{twoprod}(b, q)$
$t \leftarrow \mathrm{fl}(a - p)$
$u \leftarrow \mathrm{fl}(t - s)$
$\tilde{r} \leftarrow \mathrm{fl}(u/b)$

- Idea of the proof

$$q = \frac{a}{b}\,(1 + \epsilon_1)$$

$$\begin{aligned} p &= b\,q\,(1 + \epsilon_2) \\ &= a\,(1 + \epsilon_1)\,(1 + \epsilon_2) \end{aligned}$$

$$t = a - p \qquad \text{(Sterbenz' lemma)}$$

$$\begin{aligned} u &= (t - s)\,(1 + \epsilon_3) \\ &= \Big(a - (p + s)\Big)\,(1 + \epsilon_3) \\ &= (a - bq)\,(1 + \epsilon_3) \\ &= b\,r\,(1 + \epsilon_3) \end{aligned}$$

$$\begin{aligned} \tilde{r} &= \frac{u}{b}\,(1 + \epsilon_4) \\ &= r\,(1 + \epsilon_3)\,(1 + \epsilon_4)\,. \end{aligned}$$

Verrou: validation

Kahan polynomial [1]

- Roots of $7169\,x^2 - 8686\,x + 2631$:

Computation method	r_1	r_2
exact (rounded)	0.6062438663	0.6053616575
IEEE-754 (`float`, nearest)	0.6061973	0.6054083
error	*0.0000466*	*0.0000466*
`verrou` average (5 samples)	**0.6062**421	**0.6053**803
standard deviation	*0.0000397*	*0.0000228*
average error	*0.0000018*	*0.0000186*
`cadna` (`float_st`)	**0.6062**	**0.6053**
error	*0.0000439*	*0.0000617*

[1] W. Kahan, "The improbability of probabilistic error analyses for numerical computations", *UC Berkeley Stastistics Colloquium*, 1996.

Verrou: validation

Reccurrent sequence [1]

- Iterates of u_k:

$$u_0 = 2,$$
$$u_1 = -4,$$

$\forall k > 0,$

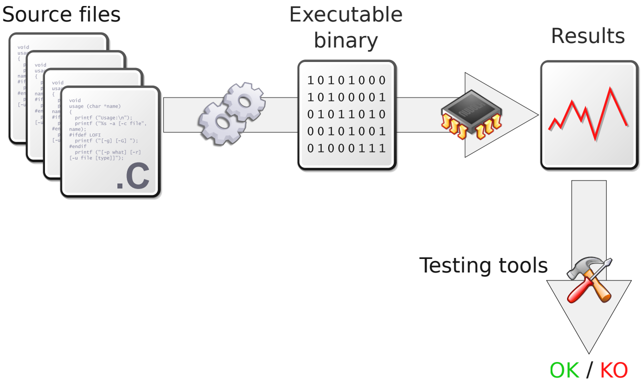
$$u_{k+1} = 111 - \frac{1130}{u_k} + \frac{3000}{u_k \; u_{k-1}},$$

k	u_k average
0	**2.000000**
1	**-4.000000**
2	**18.500000**
3	**9.378378**
4	**7.801148**
5	**7.154356**
6	**6.805962**
7	**6.580517**
8	**6.265057**
9	3.400501
10	-83.174968
11	**1**14.316190
12	**100**.777983
13	**100.0**47690
14	**100.00**2459

[1] J.C. Bajard, D. Michelucci, J.M. Moreau and J.M. Muller, "Introduction to the Special Issue "Real Numbers and Computers"", *Journal of Universal Computer Science*, 1996.

Development + V&V process

Verrou



Development + V&V process

Verrou