Verificarlo applied to ABINIT: detecting numerical instabilities

Yohan Chatelain, Pablo De Oliveira, Eric Petit, Jordan Bieder and Marc Torrent

Introduction

Objective:

- Numerical debugger and analyzer of the floating-point model

Context:

- Complex HPC environment: heterogeneous parallel architecture, compiler optimization, parallelization paradigm
- ABINIT: large program with millions line of code

Proposal:

- Automatically pinpoint the impact of the floating-point model on the numerical stability of regions of code

Outline

- Verificarlo
  - How does it works ?
  - Estimating output error
  - An example: Chebyshev polynomial
- Application on ABINIT
  - The Perovskite test case
  - Classify functions by numerical sensitiveness
  - Fine-grained analysis
  - Numerical improvement
- Conclusion & future prospects

Verificarlo

Verificarlo

Verificarlo

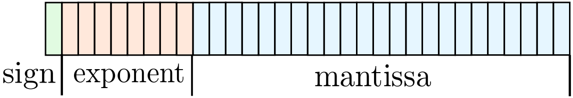
- Open Source Project under GPL licence, developped by University of Versailles and ENS Paris-Saclay
- Automatically analyses the numerical stability of applications
- Introduces a stochastic error on each floating-point operation

Verificarlo: reminder about floating-point

IEEE-754 Standard for Floating-Point Arithmetic

$$f = (-1)^s . b^e . \ x_0 x_1 ... x_{m-1}$$

$$0 \leq x_i \leq b - 1$$

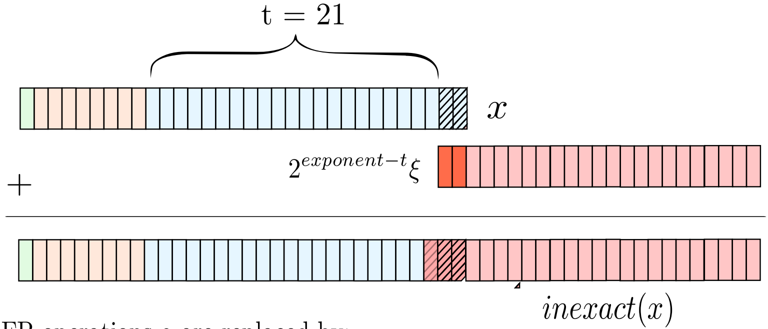


Format	sign	exponent	mantissa	Total bits
Half	1	5	10	16
Single	1	8	23	32
Double	1	11	52	64
Quad	1	15	112	128

Verificarlo: Monte Carlo Arithmetic (MCA)

Monte Carlo Arithmetic [parker1997monte]

$$inexact(x) = x + 2^{exponenent-t}\xi, \quad \xi \in [-\frac{1}{2}, \frac{1}{2}], \quad \text{t virtual precision}$$



FP operations $\circ$ are replaced by:

$$RR(x \circ y) = round(inexact(x \circ y))$$

Verificarlo: Estimating output error

Rounding errors distribution:

- Estimated by using N Monte Carlo samples

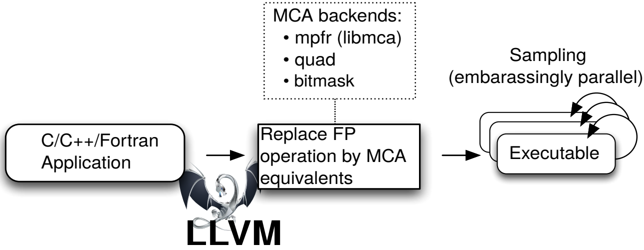
Significant digits number:

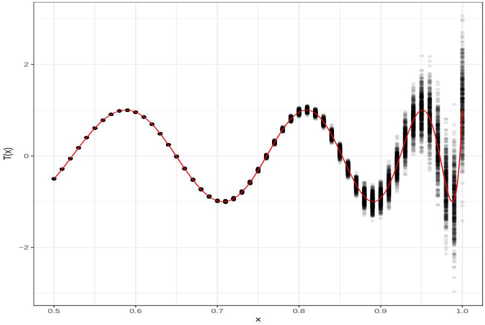
- $\tilde{s}(\chi) = -\log_{10}\left(\frac{\tilde{\sigma}}{\tilde{\mu}}\right) \xrightarrow[N\to\infty]{} s = -\log_{10}\left(\frac{\sigma}{\mu}\right)$

$\tilde{\mu}$: empirical expected value
$\tilde{\sigma}$: empirical standard deviation

Estimate the number of correct significant digits

Verificarlo: An example





Chebyshev polynomial:

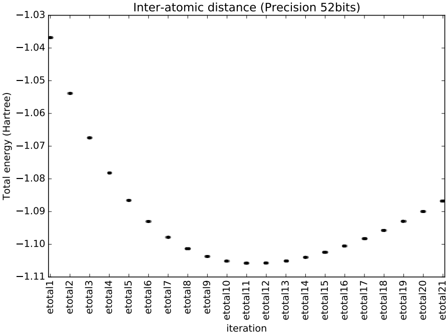
- $T_{20}(x), x \in [0,1]$
- 100 points across $[0,1]$
- 500 samples for each point
- Instability around 1
- $T_{20}(x) = cos(20cos^{-1}(x))$

Application on ABINIT

ABINIT: Presentation

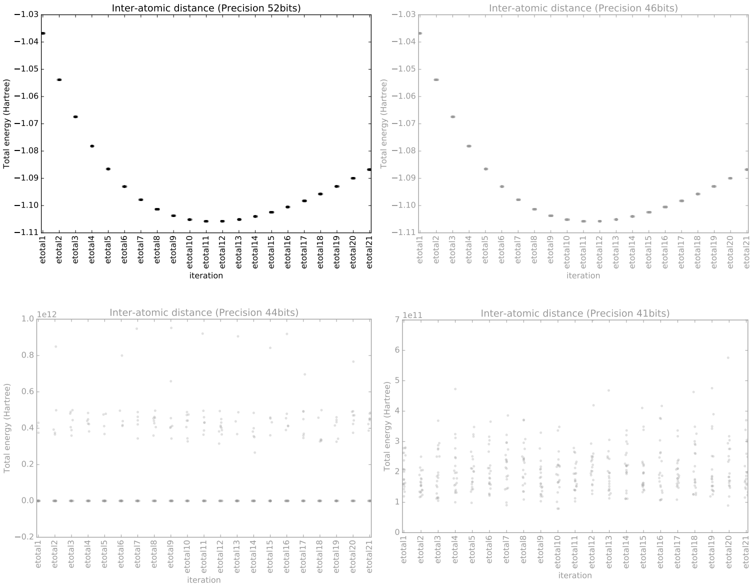
- Scientific computing code developed by an international community of industrial and academic scientifics
- Allows to find the total energy of a quantum system within Density Functional Theory (DFT)
- Code large and complex ($\sim$ 1.000.000 lines of Fortran)
- Suffering from numerical instabilities when vectorized

ABINIT: The hydrogen test case

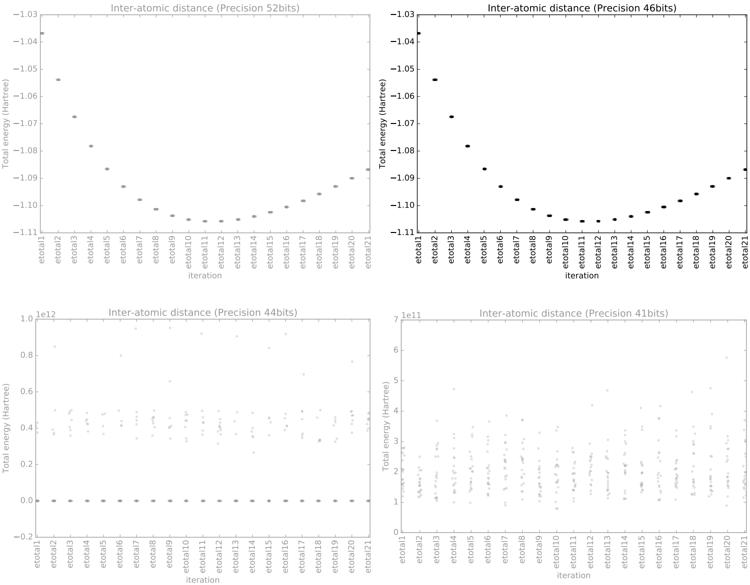


- The test case:
  - Find optimal inter-atomic distance for two hydrogen atoms
  - Simple example without non-local effects
  - Proof of concept to evaluate the cost of the method
  - Measure mean and standard deviation of MCA errors
  - Global analysis

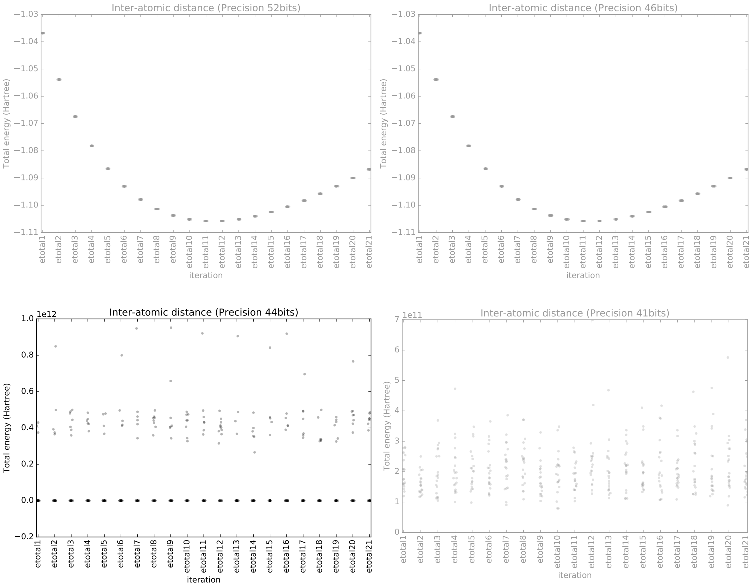
ABINIT Virtual precision exploration



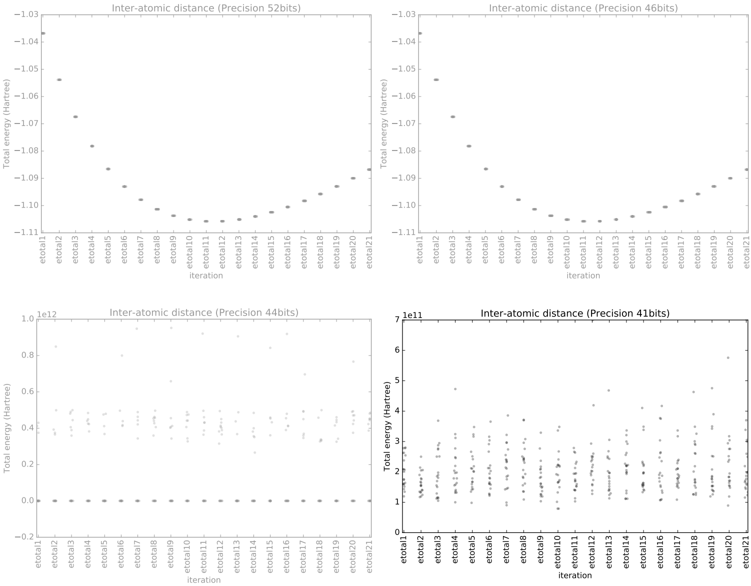
ABINIT Virtual precision exploration



ABINIT Virtual precision exploration



ABINIT: A virtual precision exploration



ABINIT: Perovskite, the realistic use case

Perovskite ($BaTiO_3$)

- Physic more complex
- Suffering from numerical instabilities when vectorized
- Not converging with Random Rounding mode

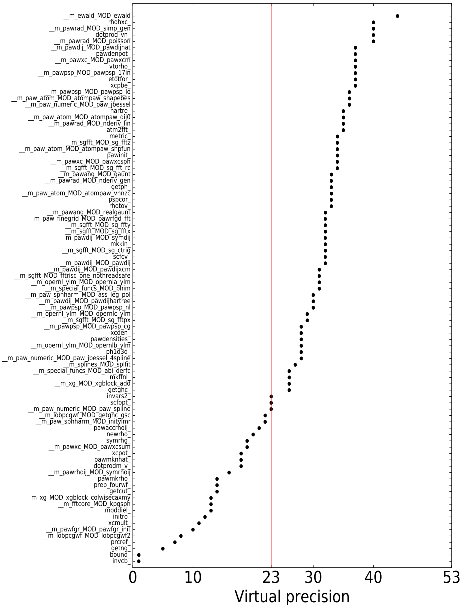
Problem: How to pinpoint the numerical unstabilities ?

- Verificarlo is time consuming
- Exhaustive analyse of the coupling of functions is impractible: $2^{\#Functions}$ functions to evaluate $\times$ t precisions $\times$ N samples

Idea:

- Reduce the set of function to test: some function does not impact the final result
- Modify the introduction of error: MCA is costly, low-order model (BITMASK backend)

Numerical sensitive functions



Functions that impact the final result:

- For each function, plot the minimal precision required to reach machine accuracy
- Only 88 functions among 3400 functions

Functions < 23:

- 1/3 of the functions are below 23bits
- Possible transformation:

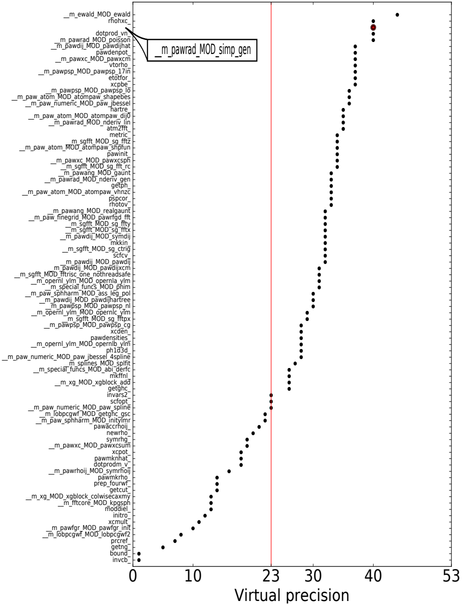
double precision $\rightarrow$ single precision
memory scaled down, computation faster

Functions ≥ 23:

- Sensitive functions
- Require a fine-grained analysis

! No coupling effects

Numerical sensitive functions



Functions that impact the final result:

- For each function, plot the minimal precision required to reach machine accuracy
- Only 88 functions among 3400 functions

Functions $<$ 23:

- 1/3 of the functions are below 23bits
- Possible transformation:

double precision $\rightarrow$ single precision
memory scaled down, computation faster

Functions $\geq$ 23:

- Sensitive functions
- Require a fine-grained analysis

! No coupling effects

ABINIT: *simp_gen*

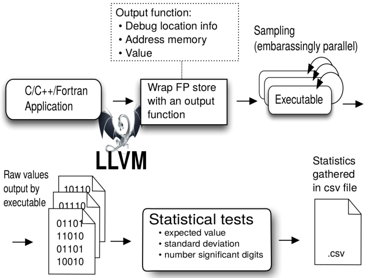
Simpson's integral:

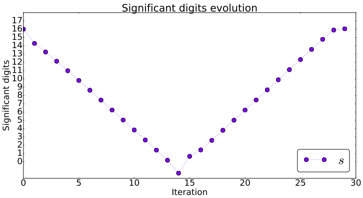
- Compute the integral over a generalized grid using Simpson's rule
- One of the most numerical sensitive function

Fine-grained analysis:

- Trace the numerical quality over time
- Dectect parts of code causing the numerical unstabilites
- Find the errors propagation accross variables

VerificarloTracer





VerificarloTracer:

- Allows one to trace the numerical quality of a variable over time
- Automatically instrument store instruction to output values
- Plot information gracefully

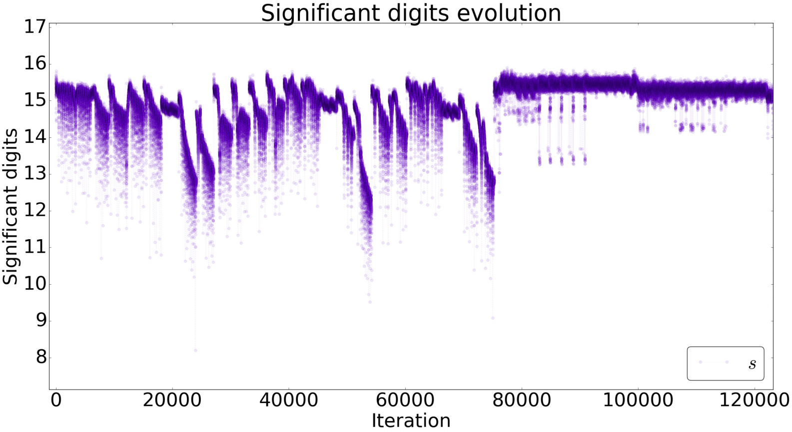
Example [bajard1996introduction]:

$$u_0 = 1, \ u_1 = -4,$$

$$u_n = 111 - \frac{1130}{u_{n-1}} + \frac{3000}{u_{n-1}u_{n-2}}$$

- Numerical limit $\neq$ Theoritical limit
- Need to track the evolution of the numerical quality

ABINIT: Numerical quality of *simp_gen*



- Evolution of the number significant digits.
- 24 samples, virtual precision $t = 53$ in Random Rounding mode (RR)
- Running on Occigen GENCI cluster

ABINIT: *simp_gen* function

```
subroutine simp_gen(intg, func, radmesh)
...
nn=radmesh%int_meshsz
simp=zero
do i=1,nn
  simp=simp+func(i)*radmesh%simfact(i)
end do
...
intg=simp+resid
end subroutine
```

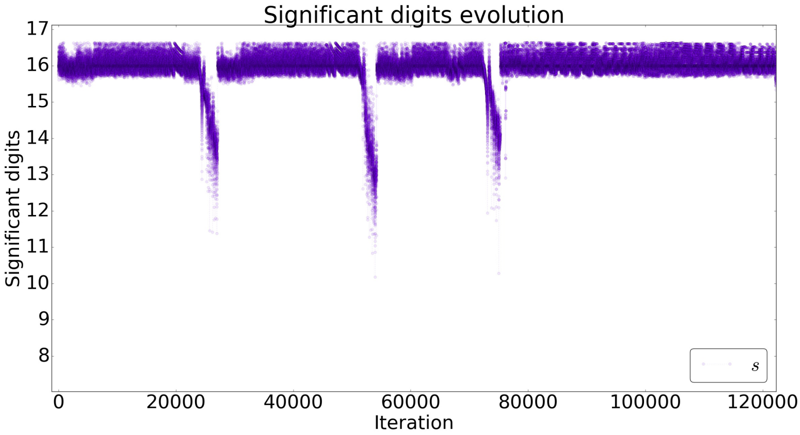
simp_gen function, can be seen as a dot product

ABINIT: A first approach

```
subroutine simp_gen(intg, func, radmesh)
...
nn=radmesh%int_meshsz
simp=zero
!Dot product in twice the working precision from
Ogita, Rump and Oisha [ogita2005accurate]
!Using library libeft github.com/ffevotte/libeft
call Dot2(simp, func, radmesh%simfact)
...
intg=simp+resid
end subroutine
```

simp_gen function, can be seen as a dot product

ABINIT: A first approach



- Evolution of the number significant digits.
- Compensated version of simp_gen using Dot2 [ogita2005accurate].

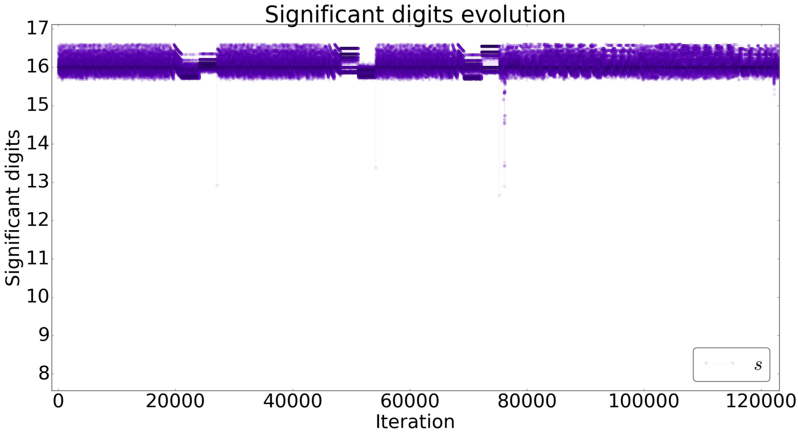
ABINIT: Data flow error propagation 1/2

```
subroutine pawpsp_calc(...)
...
if (testval) then
  nhat(1:msz)=tnvale(1:msz)*vale_mesh%rad(1:msz)**2
  call simp_gen(qq,nhat,vale_mesh)
  qq=zion/four_pi-qq
end if
call atompaw_shpfun(0,vale_mesh,intg,pawtab,nhat)
nhat(1:msz)=qq*nhat(1:msz)
tnvale(1:msz)=tnvale(1:msz)+nhat(1:msz)
...
call pawpsp_cg(..., tnvale, ...)
```

ABINIT: Data flow error propagation 2/2

```
subroutine pawpsp_cg(..., nr, ...)
...
do ir=1,mesh_size
  rnr(ir)=radmesh%rad(ir)*nr(ir)
end do
...
do ir=1,mesh_size
  if (abs(rnr(ir))>1.d-20)
    ff(ir)=sin(arg*radmesh%rad(ir))*rnr(ir)
end do
...
call simp_gen(r1torm,ff,radmesh)
```

ABINIT: Compensated version



- Evolution of the number significant digits.
- Compensated version + compensated parts outside simp_gen

Conclusion & Future Prospects

Future prospects

- Reproduce with stochastic errors “realife” bugs such as when vectorizing
- Explore benefits of optimizations such as mixed-precision and approximate computing

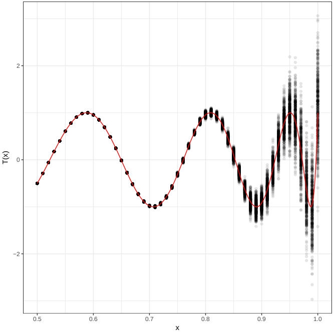
Conclusion

Verificarlo

https://github.com/verificarlo/verificarlo

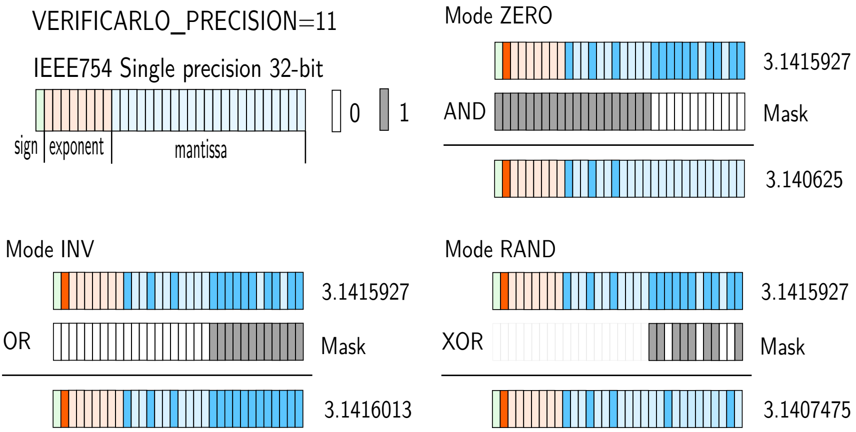
- Verificarlo: a tool for automatically detecting numerical instability
- Visualises the numerical quality of a variable over time
- Pinpoits functions causing global errors
- Reveals possible optimizations such as precision reduction

Chebyshev



$$
\begin{aligned}
T_{20}(x) &= \cos(20\cos^{-1}(x)) \\
&= 524288x^{20} - 2621440x^{18} \\
&+ 5570560x^{16} - 6553600x^{14} \\
&+ 4659200x^{12} - 2050048x^{10} \\
&+ 549120x^{8} - 84480x^{6} \\
&+ 6600x^{4} - 200x^{2} + 1
\end{aligned}
$$

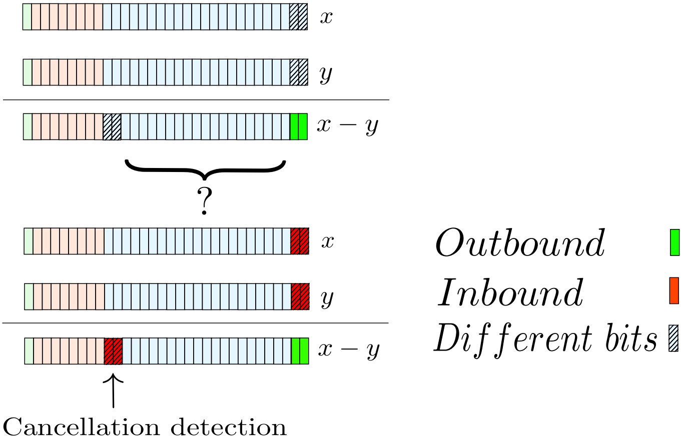
Backend BITMASK



Inbound / Outbound

$$x \sim y$$

$$mca(x) = round(inexact(inexact(x) - inexact(y)))$$



Verificarlo overhead

version	samples	total time (s)	time sample (s)
original program	1	.056	.056
Verificarlo MASK	128	12.45	.097
Verificarlo MPFR	128	834.57	6.52
Verificarlo QUAD	128	198.58	1.55
Verificarlo MPFR 16 thds.	128	54.39	.42
Verificarlo QUAD 16 thds.	128	12.54	.098

Figure 1: Verificarlo overhead on a compensated sum algorithm (double) on a 16-core 2-socket Xeon E5@2.70GHz.

- Monte Carlo Arithmetic requires additional precision which is costly
- No size fits all
  - MASK backend is cheap (x2 per iteration) but imprecise
  - QUAD backend implements exact MCA model but costly (x27 per iteration)
  - MPFR used only for validation
- Embarrassingly parallel across executions