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

Fabienne Jézéquel

LIP6, Sorbonne Université, France

“Optimising Floating-Point Precision” workshop
CERN
1-2 July 2025

Introduction

Floating-point arithmetic: | Sign | Exponent | Mantissa |

Various floating-point formats:

	#bits Mantissa (p)	Exp.	Range	$u = 2^{-p}$
bfloat16 (half)	8	8	$10^{\pm 38}$	$\approx 4 \times 10^{-3}$
fp16 (half)	11	5	$10^{\pm 5}$	$\approx 5 \times 10^{-4}$
fp32 (single)	24	8	$10^{\pm 38}$	$\approx 6 \times 10^{-8}$
fp64 (double)	53	11	$10^{\pm 308}$	$\approx 1 \times 10^{-16}$
fp128 (quad)	113	15	$10^{\pm 4932}$	$\approx 1 \times 10^{-34}$

↘ precision:

- ↘ execution time ☺
- ↘ volume of results exchanged ☺
- ↗ energy efficiency ☺

energy consumption proportional to p^2 [Jouppi et al'20]

energy ratio	
fp64/fp32	≈ 5
fp32/fp16	≈ 5
fp32/bfloat16	≈ 9

- But computed results may be invalid because of rounding errors ☹

Outline

In this talk we aim at answering the following questions.

1. How to control the validity of (mixed precision) floating-point results?
2. How to determine automatically the suitable format for each variable?

Rounding error analysis

Several approaches

- Interval arithmetic
  - guaranteed bounds for each computed result
  - the error may be overestimated
  - specific algorithms
  - ex: INTLAB [Rump'99]
- Static analysis
  - no execution, rigorous analysis, all possible input values taken into account
  - not suited to large programs
  - ex: FLUCTUAT [Goubault & al.'06], FLDLib [Jacquemin & al.'19]
- Probabilistic approach
  - estimates the number of correct digits of any computed result
  - requires no algorithm modification
  - can be used in HPC programs
  - ex: CADNA [Chesneaux'90], SAM [Graillat & al.'11], VERIFICARLO [Denis & al.'16], VERROU [Févotte & al.'17]

Discrete Stochastic Arithmetic (DSA) [Vignes'04]

Classic arithmetic

$A \oplus B \longrightarrow R$

R =3.14237654356891

DSA

Random rounding

$A_1 \oplus B_1 \longrightarrow R_1$

$A_2 \oplus B_2 \longrightarrow R_2$

$A_3 \oplus B_3 \longrightarrow R_3$

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

- each operation executed 3 times with a random rounding mode

Classic arithmetic

$A \oplus B \longrightarrow R$

$R =$3.14237654356891

DSA

Random rounding

$A_1 \oplus B_1 \longrightarrow R_1$

$A_2 \oplus B_2 \longrightarrow R_2$

$A_3 \oplus B_3 \longrightarrow R_3$

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

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

Discrete Stochastic Arithmetic (DSA) [Vignes'04]

Classic arithmetic

$A \oplus B \longrightarrow R$

$R =$3.14237654356891

DSA

Random rounding

$A_1 \oplus B_1 \longrightarrow R_1$

$A_2 \oplus B_2 \longrightarrow R_2$

$A_3 \oplus B_3 \longrightarrow R_3$

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

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

The CADNA library

cadna.lip6.fr

- implements stochastic arithmetic for C/C++ or Fortran codes
- provides stochastic types (3 floating-point variables and an integer)
- all operators and mathematical functions overloaded
  $\Rightarrow$ few modifications in user programs
- support for MPI, OpenMP, GPU codes
- in one CADNA execution: accuracy of any result, complete list of numerical instabilities

[Chesneaux'90], [Jézéquel & al'08], [Lamotte & al'10], [Eberhart & al'18],...

CADNA performance

CADNA cost

- memory: 4
- run time $\approx$ 10

Efficient rounding mode change

- implicit change of the rounding mode thanks to

$$a \oplus_{+\infty} b = -(-a \oplus_{-\infty} -b) \qquad \text{(similarly for } \ominus\text{)}$$
$$a \otimes_{+\infty} b = -(a \otimes_{-\infty} -b) \qquad \text{(similarly for } \oslash\text{)}$$

$\bigcirc_{+\infty}$ (resp. $\bigcirc_{-\infty}$): floating-point operation rounded $\rightarrow +\infty$ (resp. $-\infty$)

$\Rightarrow$ no explicit change of the rounding mode

An example without/with CADNA

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

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

An example without/with CADNA

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

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

P1=2.00000000000000e+00

P2=8.02469135802469e–01

```
#include <iostream>

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

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

  return 0;
}
```

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

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

  return 0;
}
```

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

  return 0;
}
```

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

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

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

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

Remark: CADNAIZER available on `cadna.lip6.fr`

Results with CADNA

only correct digits are displayed

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

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

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

Stochastic types in CADNA

`half_st   float_st   double_st`

Half precision in CADNA

control of fp16 computation with

- emulated half precision thanks to the library developed by C. Rau (`http://half.sourceforge.net`)
- native half precision on e.g. GPUs or ARM CPUs ≥v8.2 (successful tests on Fugaku supercomputer)

Libraries for stochastic arithmetic in arbitrary precision

SAM: Stochastic Arithmetic in Multiprecision

- based on MPFR
  `mpfr.org`
- arbitrary mantissa length (exponent length not chosen)
- mantissa length “limited” by RAM
- `mp_st<M>` stochastic type
  Ex:
  mp_st<42>
  mp_st<500>

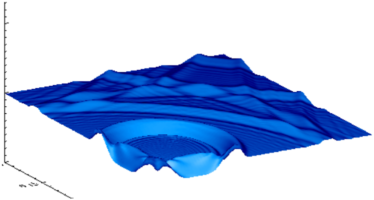
SAFE: Stochastic Arithmetic with Flexible Exponent

- based on FlexFloat
  `github.com/oprecomp/flexfloat`
- arbitrary mantissa length and arbitrary exponent length
- mantissa length limited by double (or quad) precision
- `flexfloat_st<E,M>` stochastic type
  Ex:
  flexfloat_st<8,7>⇒bf16
  flexfloat_st<5,10>⇒fp16
  flexfloat_st<5,2>⇒E5M2

- operator overloading ⇒ few modifications in user C/C++ programs
- control of arithmetic operations mixing several (non-native) formats
- accuracy estimation on FPGA

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

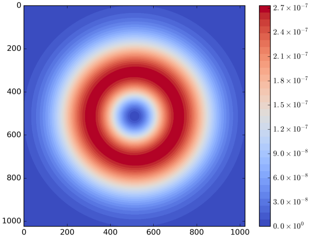
- Simulation of the evolution of water height and velocities in a 2D oceanic basin
- CUDA GPU code in double precision



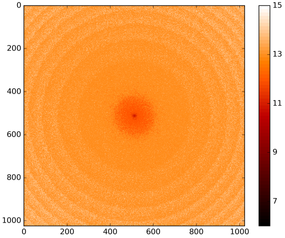
- Focusing on an eddy evolution:
  20 time steps (12 hours of simulated time) on a 1024 × 1024 grid

SW eddy simulation with CADNA-GPU

At the end of the simulation:



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



Number of correct digits estimated by CADNA-GPU

- at eddy center: great accuracy loss due to cancellations
- point at the very center: 9 digits lost
  ⇒ **no correct digits in single precision**
- fortunately, velocity values close to zero at eddy center
  → negligible impact on the output
  → **satisfactory overall accuracy**

Accuracy analysis... and then?

If the results accuracy is not satisfactory...

- **higher precision**: `single` → `double` → `quad` → arbitrary precision
  ⚠ numerical validation

- **compensated algorithms**
  - for sum, dot product, polynomial evaluation,...
  - results ≈ as accurate as with twice the working precision

  [Kahan'87], [Priest'92], [Ogita & al.'05], [Graillat & al.'09], ...

- **accurate and reproducible BLAS**
  - ExBLAS
    https://github.com/riakymch/exblas
  - Repro-BLAS
    https://bebop.cs.berkeley.edu/reproblas
  - OzBLAS
    https://github.com/mukunoki/ozblas

When the results accuracy is satisfactory...

Can we use reduced or mixed precision
to improve performance and energy efficiency?

- **mixed precision linear algebra algorithms**
  design of a mixed precision version of specific algorithms
  with an error analysis

  Surveys: [Abdelfattah et al'21, Higham & Mary'22, Kashi et al'24]

- **precision autotuning**
  automatic search for a valid mixed precision types configuration
  whatever the user algorithms

Precision autotuning

Static tools

- FPTaylor/FPTuner [Solovyev & al'15] symbolic Taylor expansions
- DAISY [Darulova & al'18] mixed-precision with rewriting
- TAFFO [Cherubin & al'19] auto-tuning for floating to fixed-point optimization
- POP [Ben Khalifa & al'19] error analysis by constraint generation

not suited to large scale programs ☹

Precision autotuning

Dynamic tools

intend to deal with large codes

- CRAFT [Lam & al'13] binary modifications on the operations
- Precimonious [Rubio-Gonzàlez & al'13] source modification with LLVM
- Blame Analysis [Nguyen & al'15] improves Precimonious
- HiFPTuner [Guo & al'18] based on a hierarchical search algorithm
- ADAPT [Menon & al'18] based on algorithmic differentiation
- FloatSmith [Lam & al'19] combination of CRAFT & ADAPT
- FPLearner [Wang & Rubio-Gonzàlez'24] improves precision tuners using ML
- Tools dedicated to GPUs (that pay attention to casts):
  - AMPT-GA [Kotipalli & al'19]
  - GPUMixer [Laguna & al'19]
  - GRAM [Ho & al'21]

Precision autotuning

Dynamic tools rely on comparisons with the highest precision result.

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

float: $P =$2.571784e+29

Precision autotuning

Dynamic tools rely on comparisons with the highest precision result.

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

float: $P =$2.571784e+29

double: $P =$1.17260394005318

Precision autotuning

Dynamic tools rely on comparisons with the highest precision result.

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

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

Precision autotuning

Dynamic tools rely on comparisons with the highest precision result.

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

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

PROMISE (PRecision OptiMISE) [Graillat & al'19], [F.J. & al'21], ...

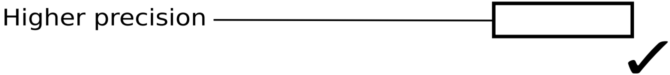
promise.lip6.fr

PROMISE

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

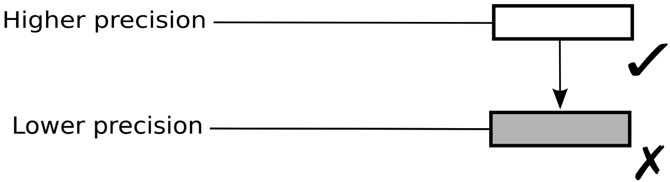
Searching for a valid configuration with 2 types

Method based on the Delta Debug algorithm [Zeller'09]



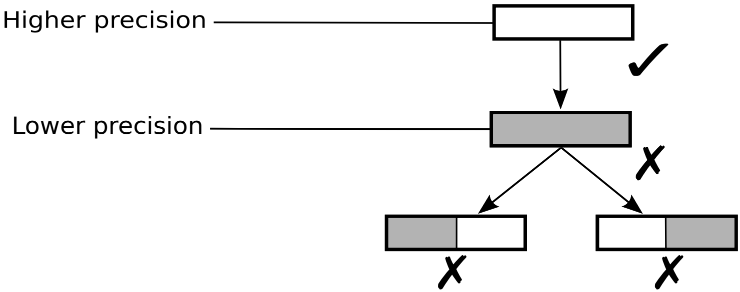
Searching for a valid configuration with 2 types

Method based on the Delta Debug algorithm [Zeller'09]



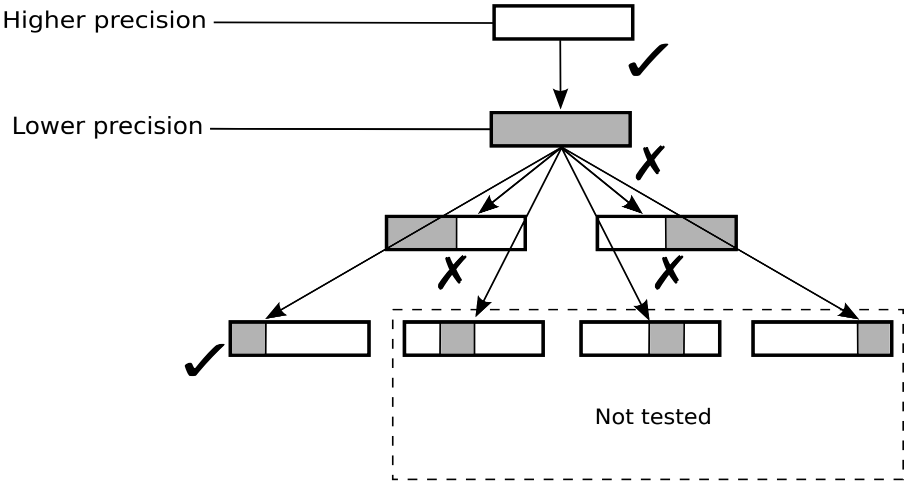
Searching for a valid configuration with 2 types

Method based on the Delta Debug algorithm [Zeller'09]



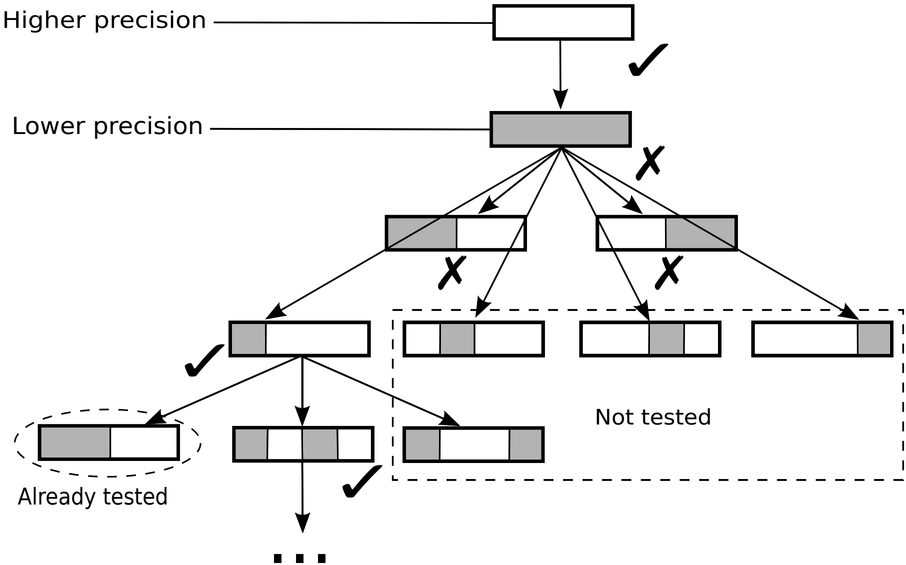
Searching for a valid configuration with 2 types

Method based on the Delta Debug algorithm [Zeller'09]

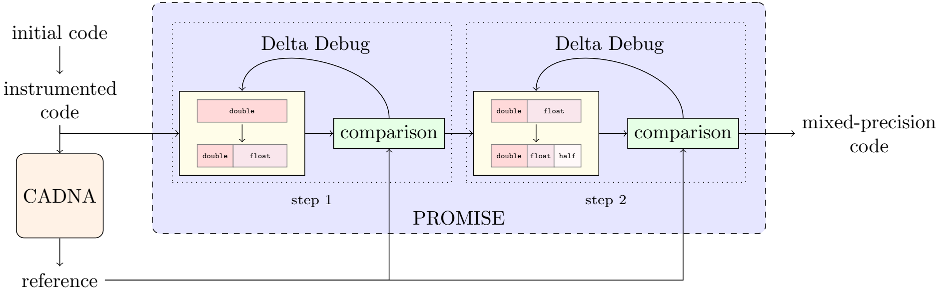


Searching for a valid configuration with 2 types

Method based on the Delta Debug algorithm [Zeller'09]



PROMISE in double, single and half precision



- step 1: code in double → variables relaxed to single precision
- step 2: single precision variables → variables relaxed to half precision

Control application

Active controller of vehicle longitudinal oscillations:

$$\left\{ \begin{array}{rcl} x(k+1) & = & Ax(k)+Bu(k) \\ y(k) & = & Cx(k)+Du(k) \end{array} \right.$$

A, B, C and D are given matrices, k is time.

Control application

Active controller of vehicle longitudinal oscillations:

$$\begin{cases} x(k+1) &= Ax(k)+Bu(k) \\ y(k) &= Cx(k)+Du(k) \end{cases}$$

After 100 iterations:

# req. digits	# exec	# half-# single-# double	time (s)
1-2	58	6-12-0	58.16
3	52	0-18-0	51.47
4	55	0-15-3	47.53
5	62	0-11-7	50.92
6	67	0-9-9	53.76
7	66	0-7-11	50.89
8	63	0-4-14	47.36
9-11	52	0-1-17	38.10

time: total execution time of PROMISE (compilations, executions, and time spent in PROMISE routines)

Active controller of vehicle longitudinal oscillations:

$$\begin{cases} x(k+1) & = & Ax(k)+Bu(k) \\ y(k) & = & Cx(k)+Du(k) \end{cases}$$

After 100 iterations:

# req. digits	# exec	# half-# single-# double	time (s)
1-2	58	6-12-0	58.16
3	52	0-18-0	51.47
4	55	0-15-3	47.53
5	62	0-11-7	50.92
6	67	0-9-9	53.76
7	66	0-7-11	50.89
8	63	0-4-14	47.36
9-11	52	0-1-17	38.10

time: total execution time of PROMISE (compilations, executions, and time spent in PROMISE routines)

MICADO: simulation of nuclear cores

code developed by EDF (French energy supplier)

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

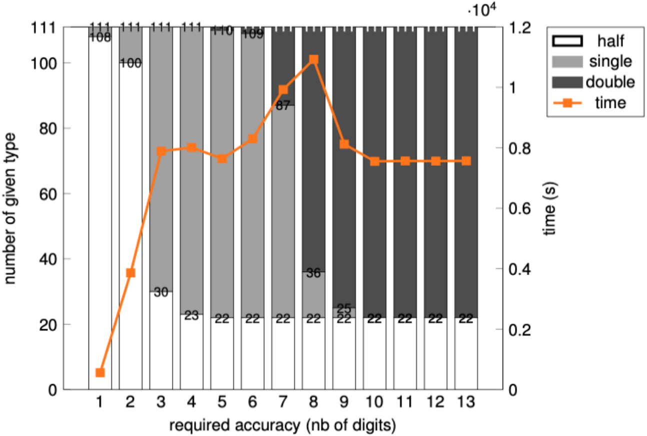
# req. digits	# single - # double	speed up	memory gain
10	32-19	1.01	1.00
8	33-18	1.01	1.01
6	38-13	1.20	1.44
5 4	51-0	1.32	1.62

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

Precision autotuning of neural networks

Generation of mixed precision C++ codes from Pytorch or Keras models

Ex: classification network for CIFAR (5 layers, 111 types to set)



Conclusion/Perspectives

To optimize precision

- numerical validation tools such as CADNA
- precision autotuning tools such as PROMISE
- mixed precision algorithms

Work in progress: extension of PROMISE to arbitrary precision

- relies on FloatX (similar to FlexFloat, completely in C++)
  `https://github.com/oprecomp/FloatX`
- From a list of numerical formats (e.g. fp64, fp32, bf16, E4M3), several Delta Debug executions identify which variables/parts can be relaxed to lower precisions

Perspectives

- extension of PROMISE to GPUs
- combination of mixed precision algorithms and floating-point autotuning

References

J. Vignes, Discrete Stochastic Arithmetic for Validating Results of Numerical Software, Num. Algo., 37, 1–4, p. 377–390, 2004.

P. Eberhart, J. Brajard, P. Fortin, and F. Jézéquel, High Performance Numerical Validation using Stochastic Arithmetic, Reliable Computing, 21, p. 35–52, 2015.

https://hal.science/hal-01254446

S. Graillat, F. Jézéquel, R. Picot, F. Févotte, and B. Lathuilière, Auto-tuning for floating-point precision with Discrete Stochastic Arithmetic, J. Computational Science, 36, 2019.

https://hal.science/hal-01331917

F. Jézéquel, S. sadat Hoseininasab, T. Hilaire, Numerical validation of half precision simulations, 1st Workshop on Code Quality and Security (CQS 2021), WorldCIST'21, 2021.

https://hal.science/hal-03138494

Q. Ferro, S. Graillat, T. Hilaire, F. Jézéquel, B. Lewandowski, Neural Network Precision Tuning Using Stochastic Arithmetic, 15th Int. Workshop on Numerical Software Verification, 2022.

https://hal.science/hal-03682645

Q. Ferro, S. Graillat, T. Hilaire, F. Jézéquel, Performance of precision auto-tuned neural networks, POAT-2023, within MCSoC-2023.

https://hal.science/hal-04149501

- CADNA: http://cadna.lip6.fr
- SAM: https://perso.lip6.fr/Fabienne.Jezequel/SAM
- SAFE: https://perso.lip6.fr/Fabienne.Jezequel/SAFE
- PROMISE: http://promise.lip6.fr

Thanks to the CADNA/SAM/SAFE/PROMISE contributors:
Julien Brajard, Romuald Carpentier, Xinye Chen, Jean-Marie Chesneaux, Patrick Corde, Pacôme Eberhart, Marek Felšoci, Quentin Ferro, François Févotte, Pierre Fortin, Stef Graillat, Thibault Hilaire, Sara Hoseininasab, Jean-Luc Lamotte, Baptiste Landreau, Bruno Lathuilière, Ivan Lucas Romain Picot, Antoine Quedeville, Enzo Roaldes, Jonathon Tidswell, Su Zhou, ...

Thanks to the CADNA/SAM/SAFE/PROMISE contributors:
Julien Brajard, Romuald Carpentier, Xinye Chen, Jean-Marie Chesneaux, Patrick Corde, Pacôme Eberhart, Marek Felšoci, Quentin Ferro, François Févotte, Pierre Fortin, Stef Graillat, Thibault Hilaire, Sara Hoseininasab, Jean-Luc Lamotte, Baptiste Landreau, Bruno Lathuilière, Ivan Lucas Romain Picot, Antoine Quedeville, Enzo Roaldes, Jonathon Tidswell, Su Zhou, ...

Thank you for your attention!