# Auto-ajustement de la précision grâce au logiciel PROMISE

Quentin Ferro, Stef Graillat, Thibault Hilaire, Fabienne Jézéquel
LIP6, Sorbonne Université, France

MS "Calcul en précision variable et multi-précision"
CANUM 2024
Ile de Ré, France

# 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 $p^2$

| energy ratio | |
|---|---|
| fp64/fp32 | $\approx$ 5 |
| fp32/fp16 | $\approx$ 5 |
| fp32/bfloat16 | $\approx$ 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/part of a code?

# 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]

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

# 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%

# 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
  ⇒ 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],...

# Stochastic types

Various stochastic types that can be mixed together or with classic types:

`half_st` `float_st` `double_st` `float128_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. NVIDIA GPUs or recent ARM processors (successful tests on Fugaku supercomputer)

# Numerical validation... and then?

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

- mixed precision linear algebra algorithms
  - matrix-matrix and matrix-vector multiplication
  - LU and QR matrix factorizations
  - iterative refinement
  - Krylov solvers
  - least squares problems

  survey: [Higham & Mary'22]

- precision autotuning

# 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
- 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]

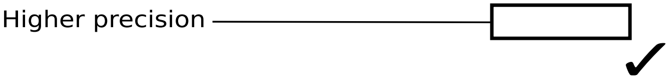
promise.lip6.fr

PROMISE

- provides a mixed precision code (half, single, double) 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.

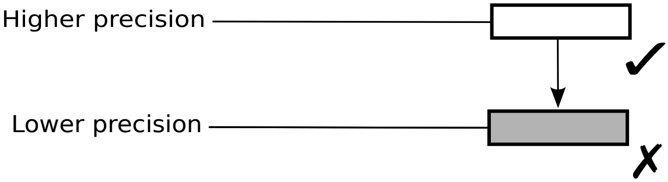
# Searching for a valid configuration with 2 types

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



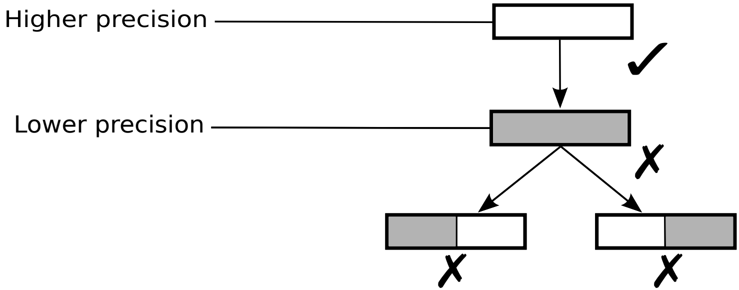
# Searching for a valid configuration with 2 types

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



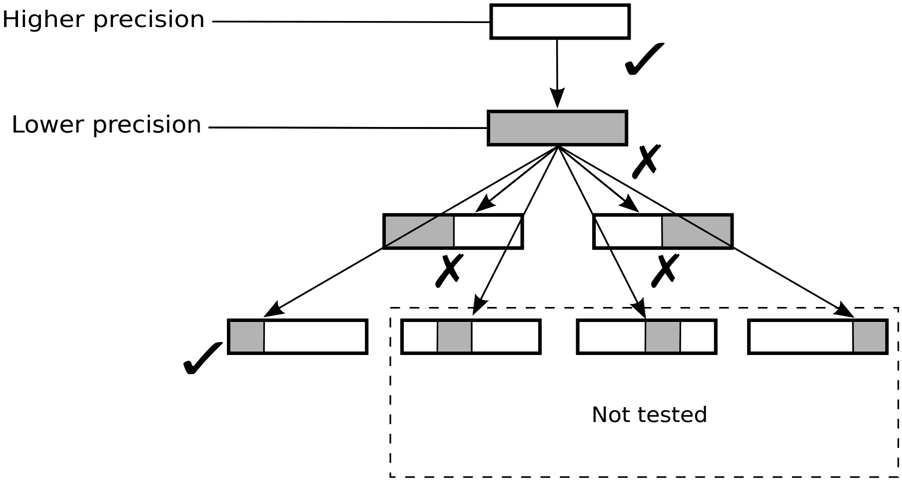
# Searching for a valid configuration with 2 types

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



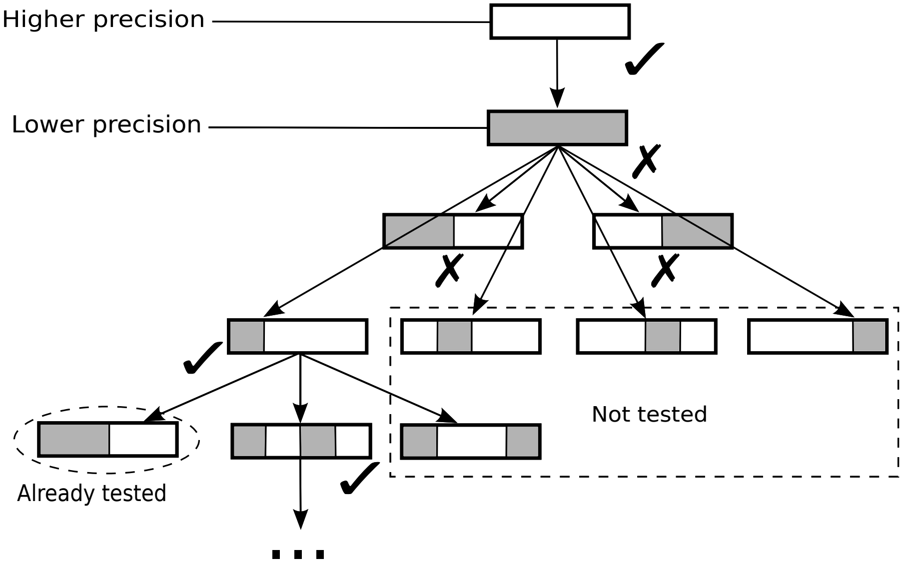
# Searching for a valid configuration with 2 types

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

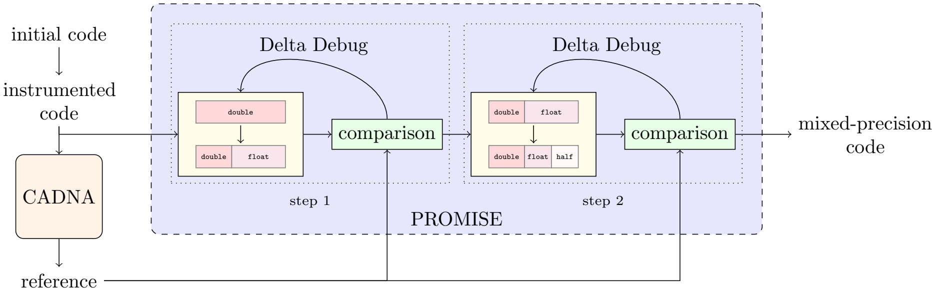


# Searching for a valid configuration with 2 types

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

# 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<br>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

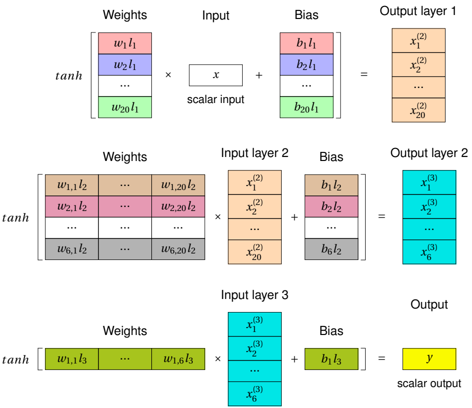
# Precision autotuning of neural networks

- neural networks created and trained with Keras or PyTorch
- automatically transformed into C++ codes to be used with PROMISE

Sine NN: interpolation network approximating the sine function

- Scalar input/ouput
- 3 dense layers with tanh activation function:
  - 20 neurons → 21 types to set
  - 6 neurons → 7 types to set
  - 1 neuron → 2 types to set

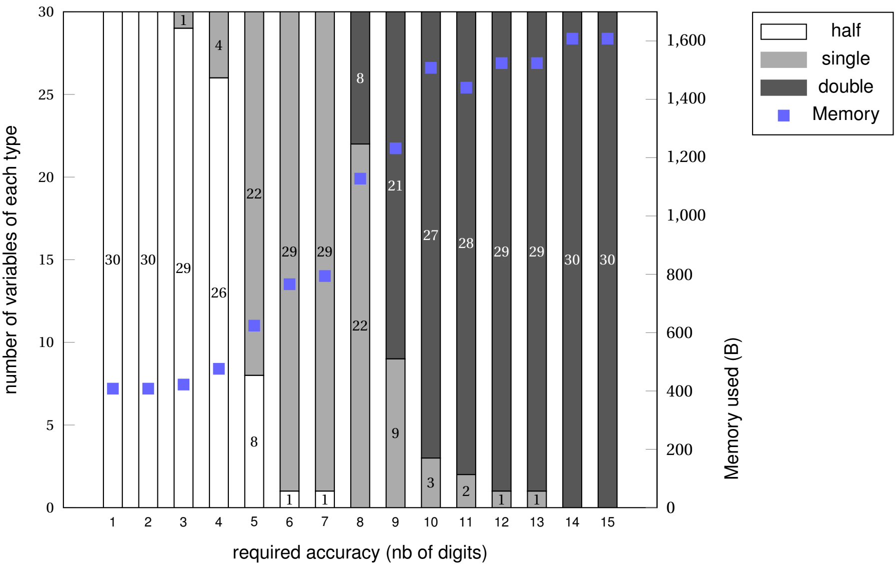
⇒ 30 types to set in total



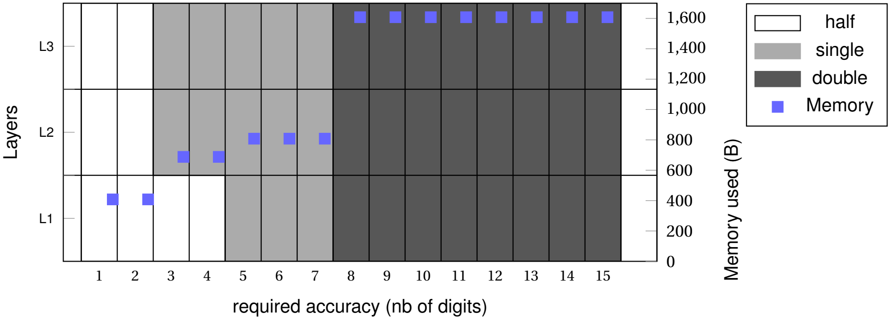

2 approaches:

- one type per neuron
- one type per layer

# Sine NN, one type per neuron



# Sine NN, one type per layer




In this talk, input=0.5
similar trends observed with different input values

# MNIST NN

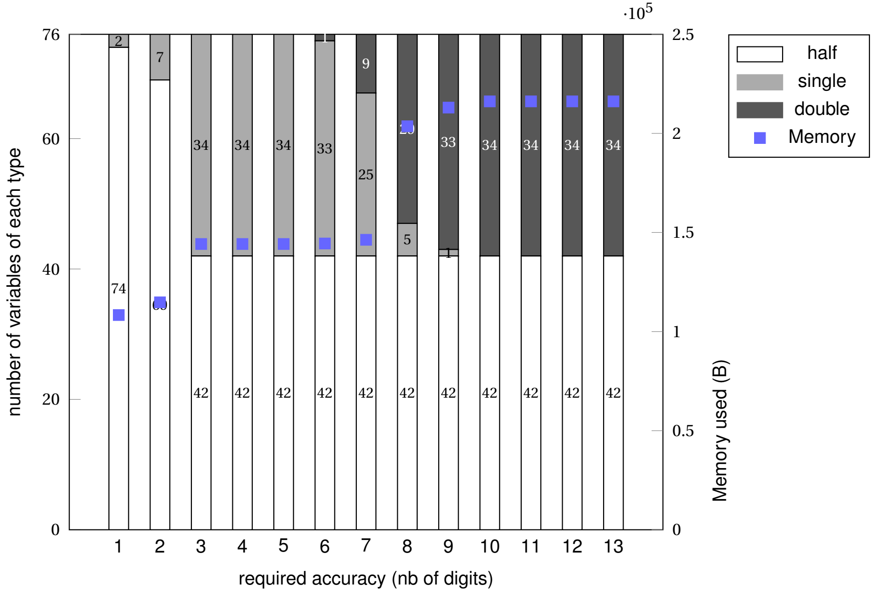
Classification of handwritten digits:

- input: vector of size 784 (flatten image)
- output : vector of size 10, probability distribution for the 10 different classes
- 2 dense layers:
  - 64 neurons and ReLU activation function → 65 types to set
  - 10 neurons and softmax activation function → 11 types to set

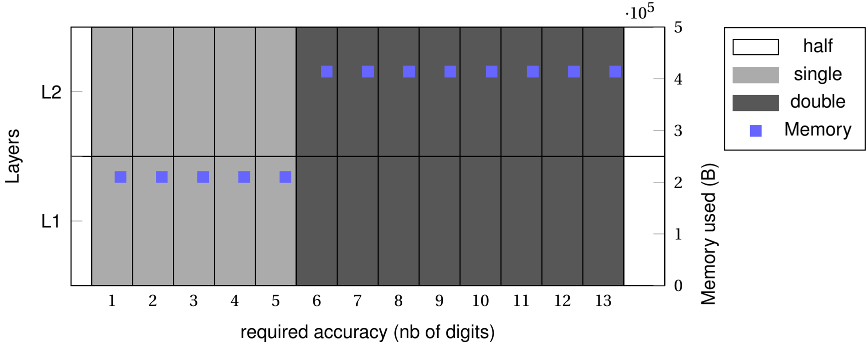
wikipedia.org

⇒ 76 types to set in total

# MNIST NN, one type per neuron



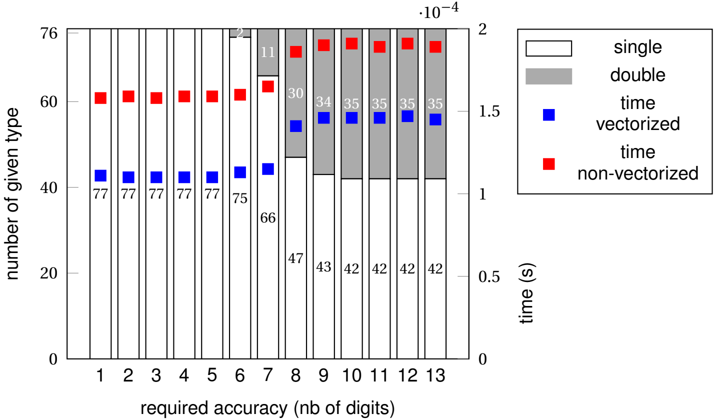
# MNIST NN, one type per layer




In this talk, input image test_data[61] from MNIST data base
similar trends observed with different input images

# Time gain with MNIST NN, one type per neuron

matrix-vector products vectorized using OpenMP SIMD on AVX2
⇒ 8 fp32 (or 4 fp64) operations in parallel

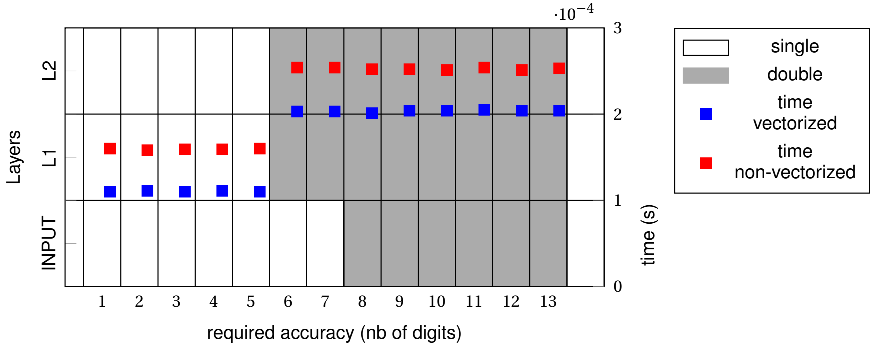



time ratio fp64 / mixed precision:

- up to 1.30 for non-vectorized codes
- up to 1.41 for vectorized codes

theoretical time ratio fp64 / fp32 = 2

# Time gain with MNIST NN, one type per layer




time ratio fp64 / fp32:

- up to 1.60 for non-vectorized codes
- up to 1.86 for vectorized codes

time ratio non-vectorized / vectorized: up to 1.45

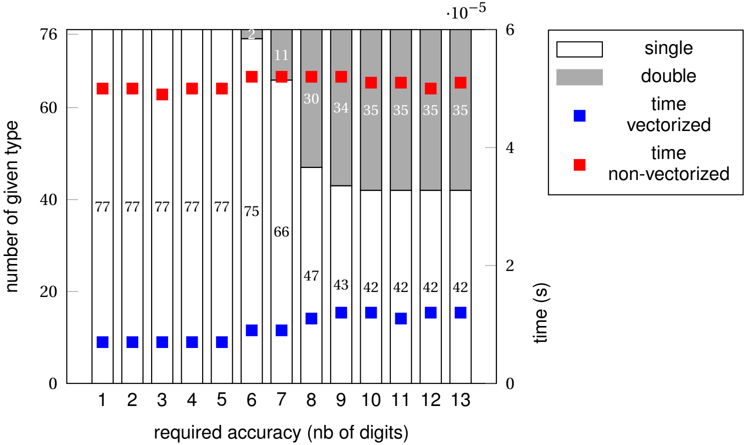
theoretical time ratio:

fp64 / fp32 = 2

non-vect. / vect. = 8 in fp32, 4 in fp64

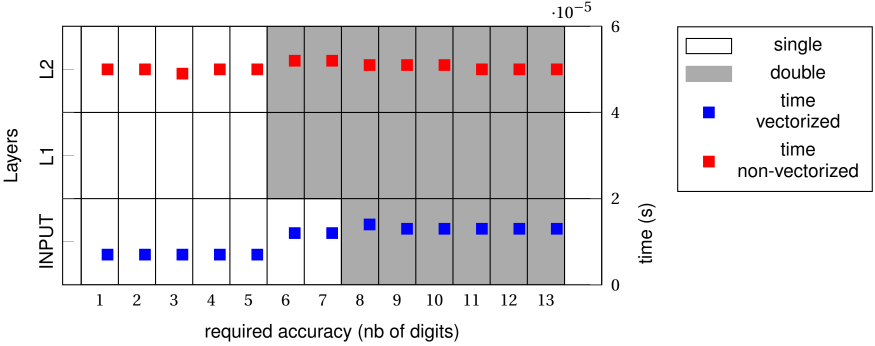
# Matrix-vector products in MNIST NN

## one type per neuron



# Matrix-vector products in MNIST NN

one type per layer

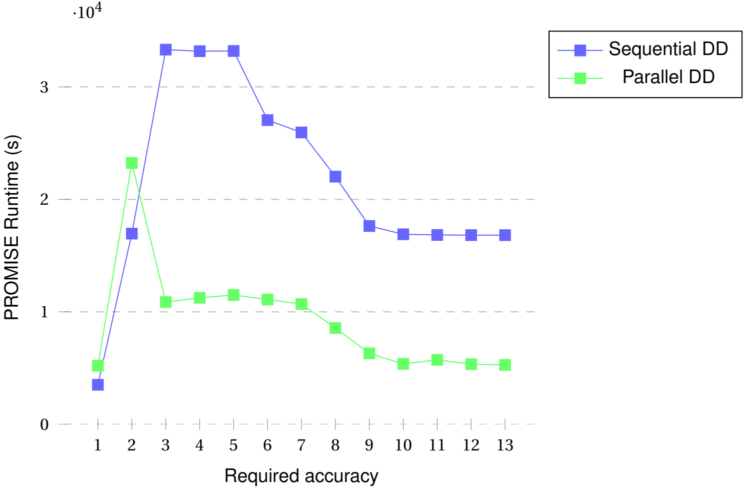



time ratio fp64 / fp32:

- no time difference for non-vectorized codes
- 2 for vectorized codes

time ratio non-vectorized / vectorized: up to 7.2 in fp32 and 3.9 in fp64
☺ close to theoretical ratio

# Execution time of PROMISE for MNIST NN




Parallel version of the Delta Debug from [Hodován & Kiss'16]

Speedup up to 3.2 on a CPU having 6 cores with 16 GB RAM

# Conclusion/Perspectives

## Conclusion

in user codes, suitable configuration types provided by precision autotuning

- $\searrow$ memory consumption
- $\searrow$ execution time, in particular in vectorized codes

## Perspectives

- extension of CADNA/PROMISE to other formats such as bf16
- extension of PROMISE to GPUs
- floating-point autotuning in arbitrary precision
- combination of mixed precision algorithms and floating-point autotuning

PostDoc offer in LIP6!

# 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.
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

Thank you for your attention!
Any question?