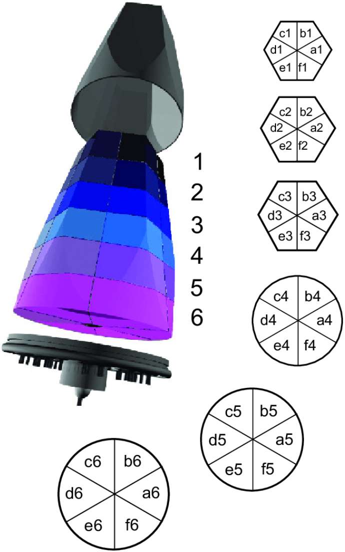
Investigating mixed-precision for AGATA pulse-shape analysis

Roméo Molina[1,2]
Joint work with David Chamont[1], Fabienne Jézéquel[2], Vincent Lafage[1]

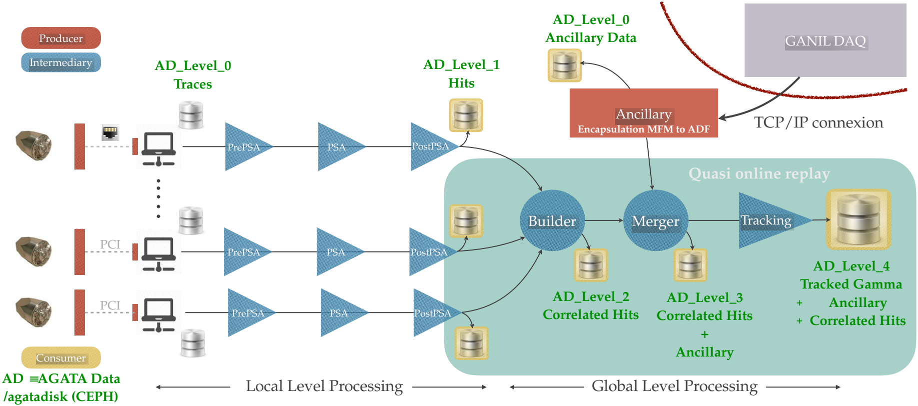
[1]IJCLab, Paris-Saclay, France
[2]LIP6, Sorbonne Université, France

AGATA IJCLAB 2023
3 July 2023

AGATA Advanced GAmma Tracking Array



AGATA Data flow[1]



[1] O. Stézowski, AGATA Meeting 2022

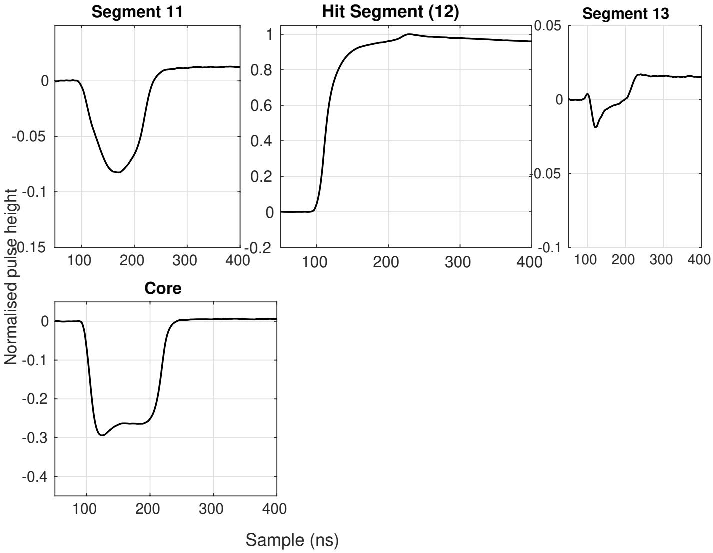
Performance analysis with `perf`

`perf` allows to count the number of CPU cycles per function

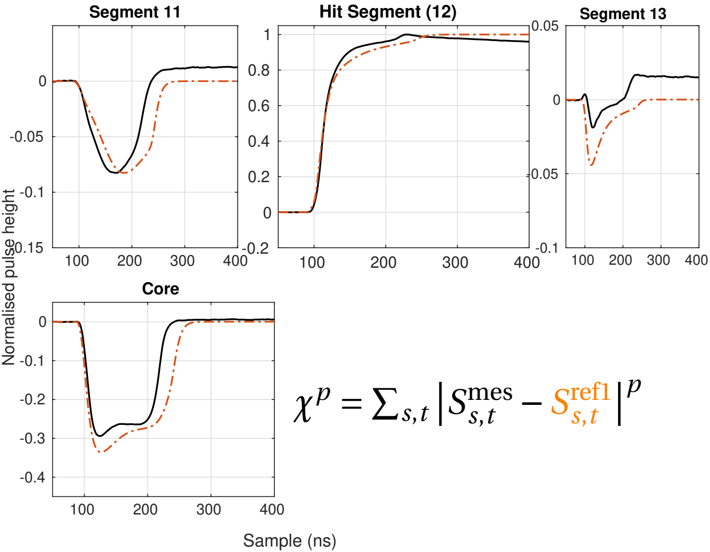
```
Samples: 58K of event 'cycles', Event count (approx.): 68053389580
Overhead  Command  Shared Object           Symbol
  68,56%  femul    libPSAFilter.so         [.] PSAFilterGridSearch::Chi2InnerLoop
   5,69%  femul    libPSAFilter.so         [.] PSAFilterGridSearch::SearchAdaptive1
   4,83%  femul    libPSAFilter.so         [.] pointPsa::convDeltaToExp
   2,80%  femul    libPSAFilter.so         [.] pointPsa::add
   1,93%  femul    libPSAFilter.so         [.] pointExp::AddBaseTrace
   1,67%  femul    libPSAFilter.so         [.] SignalBasis::ReadBasisFormatBartB
   1,59%  femul    libPSAFilter.so         [.] pointPsa::addXT
   1,12%  femul    libPSAFilter.so         [.] SignalBasis::FindNeighbours
   1,05%  femul    libPSAFilter.so         [.] SignalBasis::CalcPtPtDistance
   0,87%  femul    libPSAFilter.so         [.] PSAFilterGridSearch::FitT0AfterPSA
   0,76%  femul    libPSAFilter.so         [.] PSAFilterGridSearch::ShiftMoveTrace
   0,73%  femul    libPSAFilter.so         [.] pointPsa::sumOfSignals
   0,71%  femul    libPSAFilter.so         [.] PSAFilterGridSearch::AddToSolution
```

⇒ We shall optimize `Chi2InnerLoop`!

PSA Pulse Shape Analysis

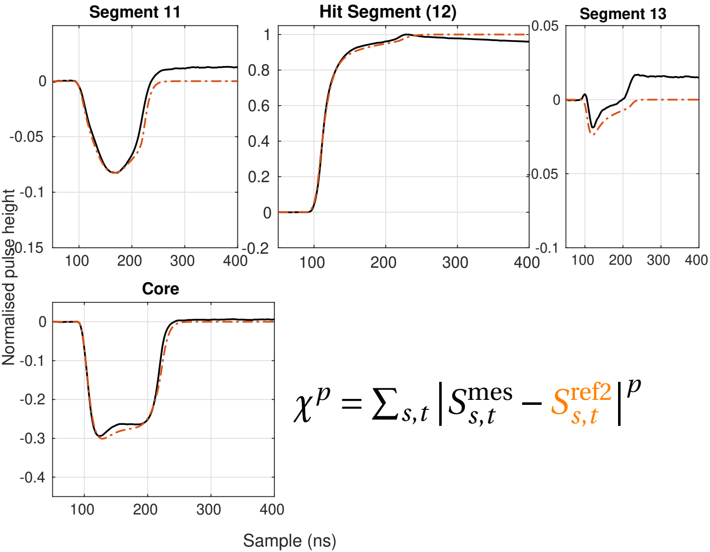


PSA Pulse Shape Analysis



$$\chi^p = \sum_{s,t} \left| S_{s,t}^{\text{mes}} - S_{s,t}^{\text{ref1}} \right|^p$$

PSA Pulse Shape Analysis



$$\chi^p = \sum_{s,t} \left| S_{s,t}^{\text{mes}} - S_{s,t}^{\text{ref2}} \right|^p$$

Cache-references

`perf` allows to analyse the memory usage

```
Samples: 52K of event 'cache-references', Event count (approx.): 742466595
Overhead  Command  Shared Object             Symbol
  80,89%  femul    libPSAFilter.so           [.] PSAFilterGridSearch::Chi2InnerLoop
   3,02%  femul    [unknown]                 [k] 0xffffffffa005e23e
   2,60%  femul    libPSAFilter.so           [.] PSAFilterGridSearch::SearchAdaptive1
   1,69%  femul    libPSAFilter.so           [.] pointPsa::add
   0,85%  femul    libPSAFilter.so           [.] pointPsa::convDeltaToExp
   0,85%  femul    libPSAFilter.so           [.] pointPsa::sumOfSignals
   0,72%  femul    libPSAFilter.so           [.] pointPsa::addXT
   0,64%  femul    libPSAFilter.so           [.] pointExp::AddBaseTrace
   0,62%  femul    libc-2.31.so              [.] __memmove_avx_unaligned_erms
   0,55%  femul    libPSAFilter.so           [.] PSAFilterGridSearch::AddToSolution
   0,55%  femul    libc-2.31.so              [.] __memset_avx2_erms
   0,51%  femul    libPSAFilter.so           [.] SignalBasis::ReadBasisFormatBartB
```

⇒ consistant with cycles analysis

Cache-misses

Cache-misses

Cache-misses happen when the data is not in cache memory.
The application has to attempt to find the data in slower memory that causes massive performance reduction.

```
Samples: 49K of event 'cache-misses', Event count (approx.): 311766482
Overhead  Command  Shared Object             Symbol
  73,26%  femul    libPSAFilter.so           [.] PSAFilterGridSearch::Chi2InnerLoop
   6,44%  femul    [unknown]                 [k] 0xffffffffa005e23e
   4,49%  femul    libPSAFilter.so           [.] PSAFilterGridSearch::SearchAdaptive1
   1,16%  femul    libPSAFilter.so           [.] pointPsa::add
   1,07%  femul    libPSAFilter.so           [.] SignalBasis::ReadBasisFormatBartB
   1,05%  femul    libc-2.31.so              [.] __memmove_avx_unaligned_erms
   0,98%  femul    libc-2.31.so              [.] __memset_avx2_erms
   0,67%  femul    libPSAFilter.so           [.] pointPsa::sumOfSignals
   0,57%  femul    [unknown]                 [k] 0xffffffffa005e240
   0,56%  femul    libPSAFilter.so           [.] pointPsa::convDeltaToExp
   0,51%  femul    libPSAFilter.so           [.] PSAFilterGridSearch::AddToSolution
```

⇒ Memory bound algorithm

How to reduce the number cache-misses

⇒ reduce the amount of data to make it fit in the cache

⇒ use smaller formats while maintaining the same accuracy

⇒ what gains for what risks?

How to reduce the number cache-misses

⇒ reduce the amount of data to make it fit in the cache
⇒ use smaller formats while maintaining the same accuracy
⇒ what gains for what risks?

How to reduce the number cache-misses

⇒ reduce the amount of data to make it fit in the cache
⇒ use smaller formats while maintaining the same accuracy
⇒ what gains for what risks?

Using low precisions is promising

		Number of bits Signif. (t)	Exp.	Range	$u = 2^{-t}$
fp128	quadruple	113	15	$10^{\pm 4932}$	1×10^{-34}
fp64	double	53	11	$10^{\pm 308}$	1×10^{-16}
fp32	single	24	8	$10^{\pm 38}$	6×10^{-8}
fp16	half	11	5	$10^{\pm 5}$	5×10^{-4}
bfloat16		8	8	$10^{\pm 38}$	4×10^{-3}
fp8 (e4m3)	quarter	4	4	$10^{\pm 2}$	6×10^{-2}
fp8 (e5m2)		3	5	$10^{\pm 5}$	1×10^{-1}

- Low precision increasingly supported by hardware
- **Great benefits:**
  - Reduced **storage**, data movement, and communications
  - Reduced **energy** consumption ($5\times$ with fp16, $9\times$ with bfloat16)
  - Increased **speed** ($16\times$ on A100 from fp32 to fp16/bfloat16)

Floating-point arithmetic

Floating-point computation $\neq$ mathematical evaluation

- rounding $a \oplus b \neq a + b$
- no more associativity $(a \oplus b) \oplus c \neq a \oplus (b \oplus c)$

Consequences:

- invalid results
- non reproducibility
- performance issue (useless iterations)

Some limitations to the low precisions:

- Low accuracy (large u)
- Narrow range

Assess the accuracy

cadna.lip6.fr

- implements stochastic arithmetic for C/C++ or Fortran codes
- all operators and mathematical functions overloaded ⇒ little code rewriting
- support for MPI, OpenMP, GPU, vectorised codes
- supports emulated ou native half precision
- in one CADNA execution: accuracy of any result, complete list of numerical instabilities

CADNA cost

- memory: 4
- run time ≈ 10

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

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 to avoid useless iterations

CADNA validates fp32 results

- PSA performed natively in fp32
- minimum search in a 504-dimensional space
- risk to accumulate catastrophic cancellations
- requires instrumentation to assess the accuracy results

⇒ code sensitive to perturbations?

- $0.02\,\%$ among points matched differently between fp64 version and original version
- $0.02\,\%$ between CADNA version and original version

⇒ Satisfactory original fullgrid PSA results!

CADNA validates fp32 results

- PSA performed natively in fp32
- minimum search in a 504-dimensional space
- risk to accumulate catastrophic cancellations
- requires instrumentation to assess the accuracy results

⇒ code sensitive to perturbations?

- $0.02\,\%$ among points matched differently between fp64 version and original version
- $0.02\,\%$ between CADNA version and original version

⇒ Satisfactory original fullgrid PSA results!

CADNA validates fp32 results

- PSA performed natively in fp32
- minimum search in a 504-dimensional space
- risk to accumulate catastrophic cancellations
- requires instrumentation to assess the accuracy results

⇒ code sensitive to perturbations?

- $0.02\,\%$ among points matched differently between fp64 version and original version
- $0.02\,\%$ between CADNA version and original version

⇒ Satisfactory original fullgrid PSA results!

Turn it into half computation

- $7.76\,\%$ differences between original and fp16 version
- too much?
- need to find another way to exploit low precision

Mixed precision algorithms

Mix several precisions in the same code with the goal of

- Getting the performance benefits of low precisions
- While preserving the accuracy and stability of high precision

⇒ **Why does it make sense to make the precision vary?**

- Because not all computations are equally “important”!
  Example:

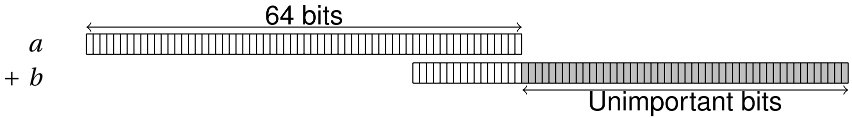
Mixed precision algorithms

Mix several precisions in the same code with the goal of

- Getting the performance benefits of low precisions
- While preserving the accuracy and stability of high precision

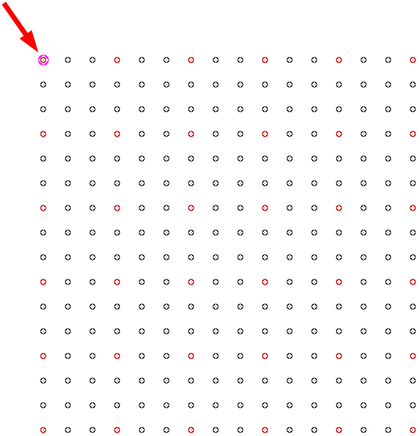
⇒ **Why does it make sense to make the precision vary?**

- Because not all computations are equally "important"!
  Example:



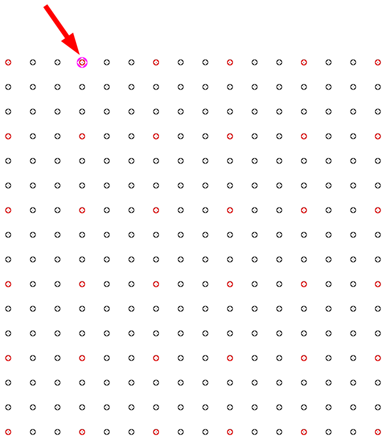
How to slice computations

- the algorithm used in practice is smarter than previously presented
- coarse-fine algorithm



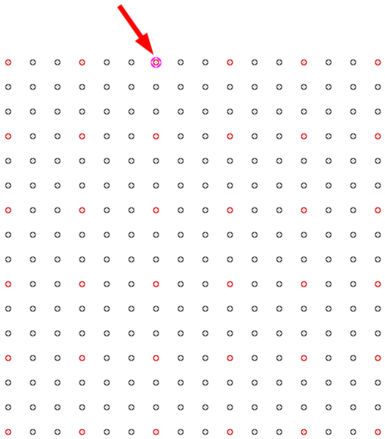
How to slice computations

- the algorithm used in practice is smarter than previously presented
- coarse-fine algorithm



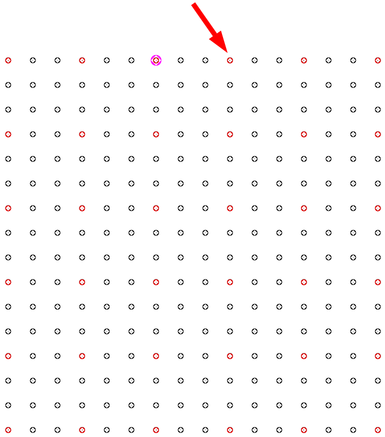
How to slice computations

- the algorithm used in practice is smarter than previously presented
- coarse-fine algorithm



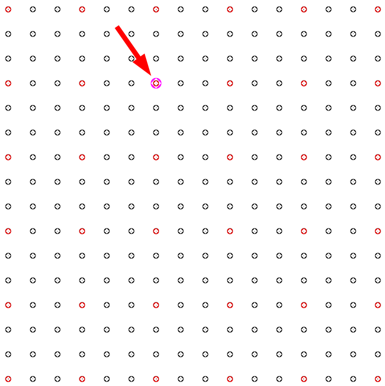
How to slice computations

- the algorithm used in practice is smarter than previously presented
- coarse-fine algorithm



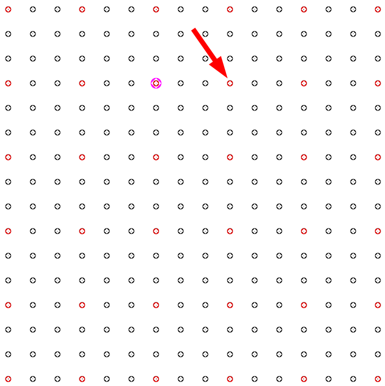
How to slice computations

- the algorithm used in practice is smarter than previously presented
- coarse-fine algorithm



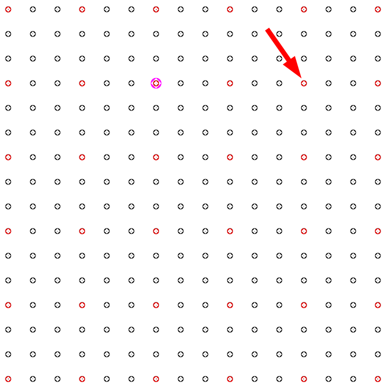
How to slice computations

- the algorithm used in practice is smarter than previously presented
- coarse-fine algorithm



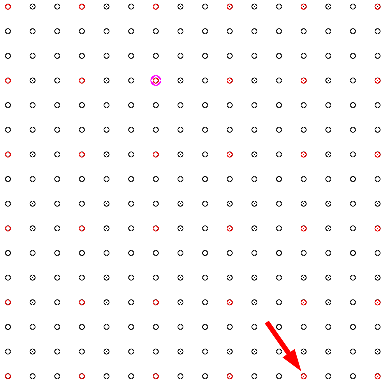
How to slice computations

- the algorithm used in practice is smarter than previously presented
- coarse-fine algorithm



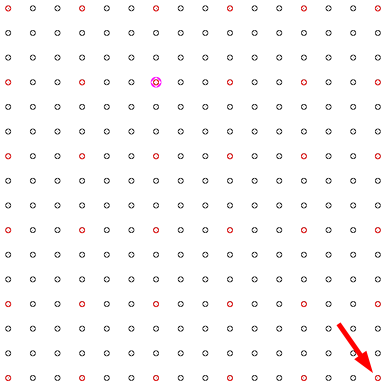
How to slice computations

- the algorithm used in practice is smarter than previously presented
- coarse-fine algorithm



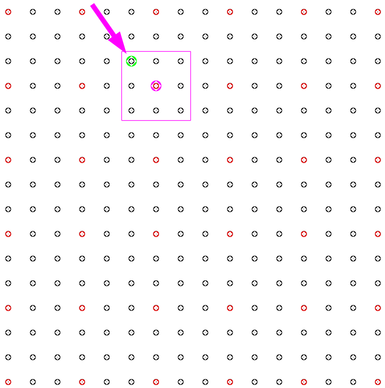
How to slice computations

- the algorithm used in practice is smarter than previously presented
- coarse-fine algorithm



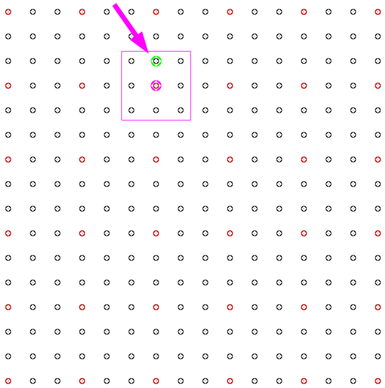
How to slice computations

- the algorithm used in practice is smarter than previously presented
- coarse-fine algorithm



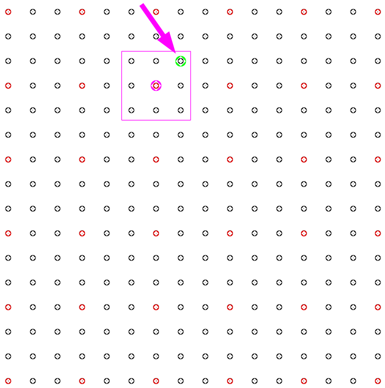
How to slice computations

- the algorithm used in practice is smarter than previously presented
- coarse-fine algorithm



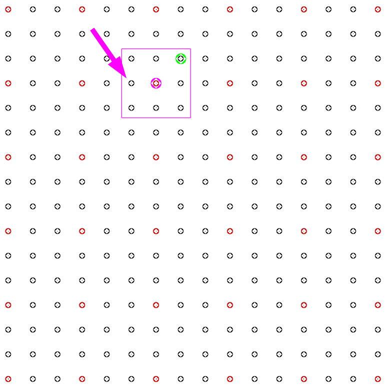
How to slice computations

- the algorithm used in practice is smarter than previously presented
- coarse-fine algorithm



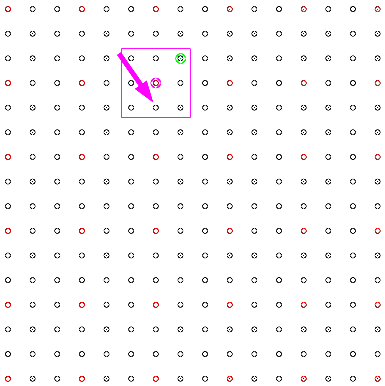
How to slice computations

- the algorithm used in practice is smarter than previously presented
- coarse-fine algorithm



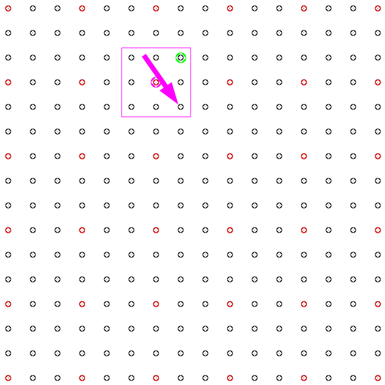
How to slice computations

- the algorithm used in practice is smarter than previously presented
- coarse-fine algorithm



How to slice computations

- the algorithm used in practice is smarter than previously presented
- coarse-fine algorithm



How to slice computations

- the algorithm used in practice is smarter than previously presented
- coarse-fine algorithm

- $8.22\,\%$ differences with fullgrid fp32 version, validated by the physicists

⇒ provides an opportunity for mixed precision

Coarse in half, fine in float

- first step in half
- second step in float
- $8.55\,\%$ differences with fullgrid fp32 version
- under the same conditions, half-half produces $14.04\,\%$ differences!

Conclusion

- low precision is beneficial (speed, energy, storage) but you should be careful
- accuracy control is mandatory
- CADNA is well designed to do so
- mixed-precision is a way to benefit from low precision in fields that require high accuracy

Perspectives

- varying the coarse/fine gridsize allocation
- introducing a hierarchy of intermediate grids
- implement it on GPUs to improve performance

Thank you for your attention!