AGATA computation control: from the crystal to the final measure

Roméo Molina[1 2]

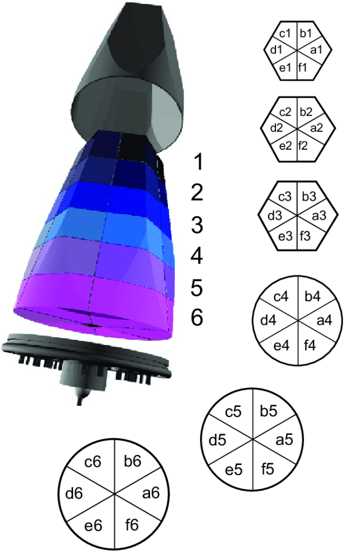
Joint work with David Chamont[1], Fabienne Jézéquel[2], Vincent Lafage[1]

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

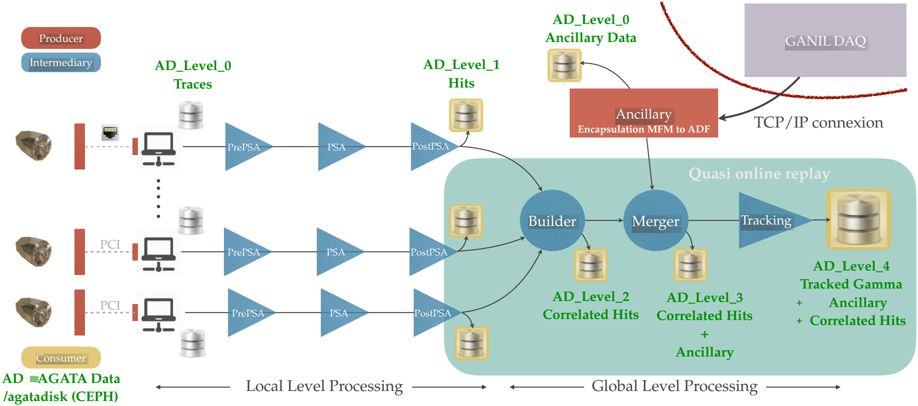
JI 2022

15 November 2022

AGATA Advanced GAmma Tracking Array

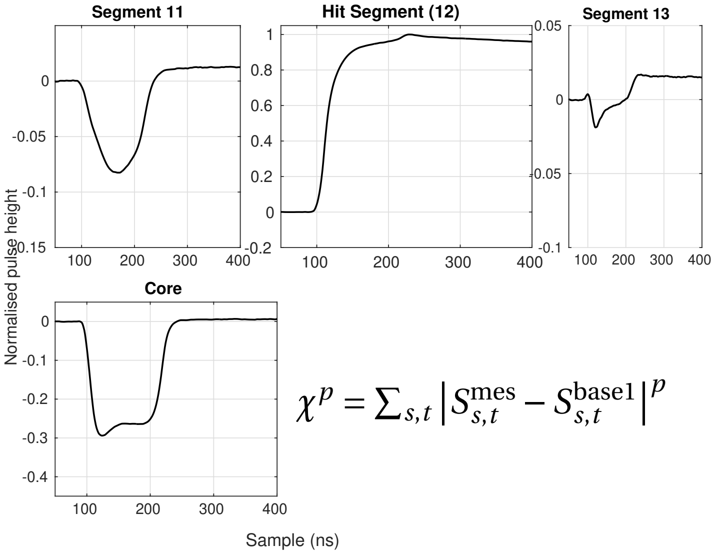


AGATA Data flow[1]



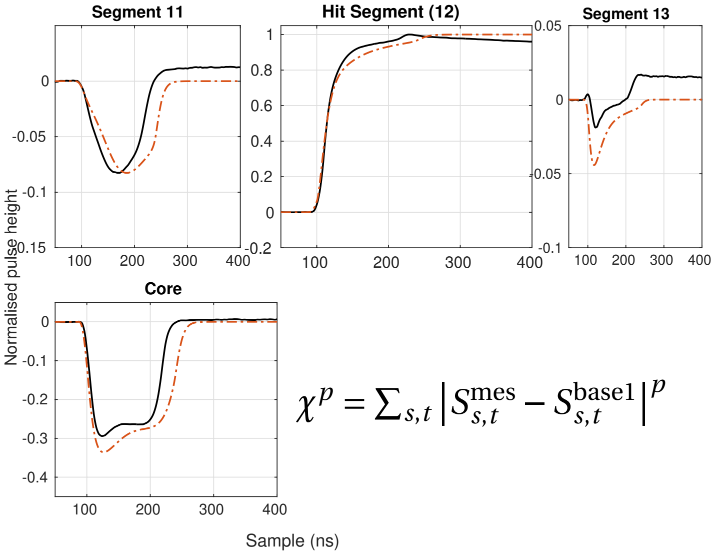
[1] merci O. Stézowski

PSA Pulse Shape Analysis



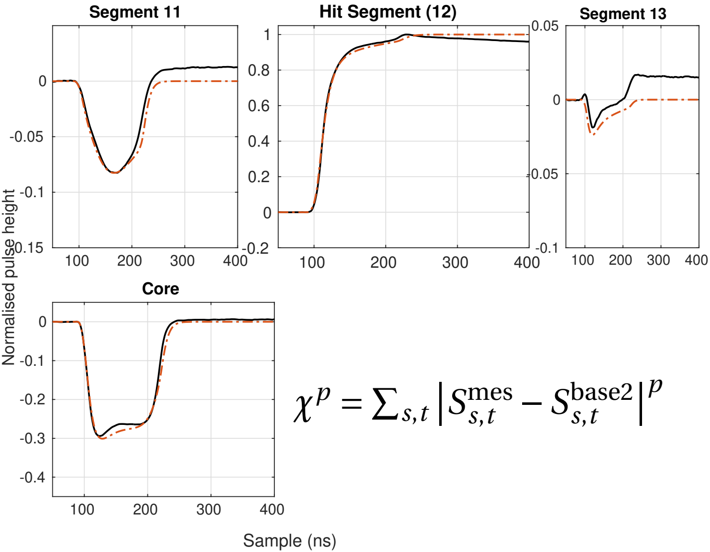
$$\chi^p = \sum_{s,t} \left| S_{s,t}^{\text{mes}} - S_{s,t}^{\text{base1}} \right|^p$$

PSA Pulse Shape Analysis



$$\chi^p = \sum_{s,t} \left| S_{s,t}^{\mathrm{mes}} - S_{s,t}^{\mathrm{base1}} \right|^p$$

PSA Pulse Shape Analysis



$$\chi^p = \sum_{s,t} \left| S_{s,t}^{\text{mes}} - S_{s,t}^{\text{base2}} \right|^p$$

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`!

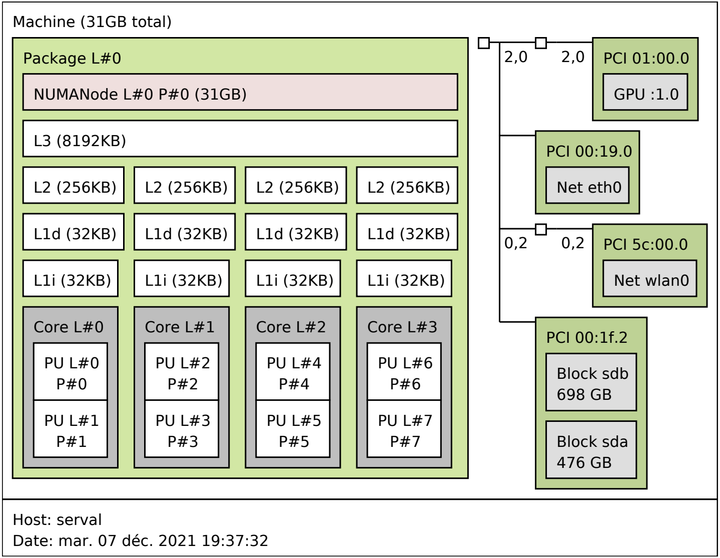
Cache-references

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
```

⇒ coherent with cycles analysis

Know your tool hwloc-ls



Know your tool

execute one typical instruction	1ns
read L1 cache memory	0.5ns
branch misprediction	5ns
read L2 cache memory	7ns
read main memory	100ns
read 2 Koctets from network 1 Gbps	20000ns
read 1 Moctets sequentially in memory	250000ns
read one new place on disk (seek)	8000000ns
read 1 Moctets sequentially on disk	20000000ns

Cache-misses

Cache-misses happen when the data is not in cache memory.
The application have 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
```

⇒ In `Chi2InnerLoop`: 38% of cache-misses !
⇒ Memory bound algorithm

How to reduce the number cache-misses

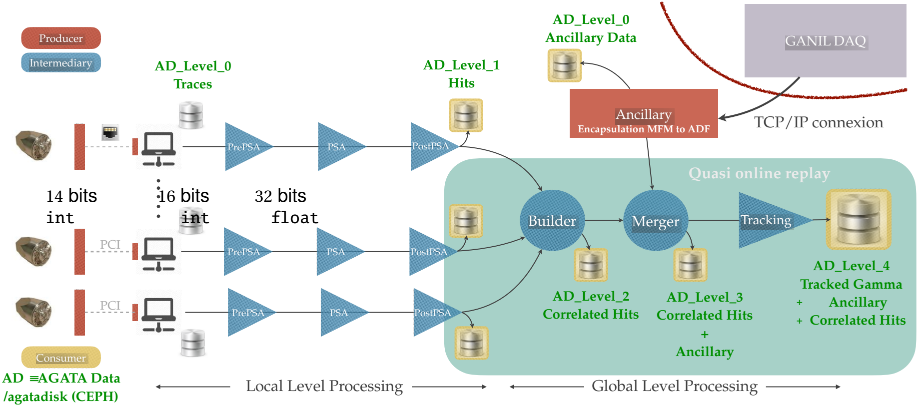
To reduce the cache-misses, our approach is to reduce the amount of data to make it fit in the cache.

To do so, we are interested in the data formats used in the PSA.

Is it possible to adopt a smaller format while maintaining the accuracy?

⇒ Credible because the material provides 14 bits of resolution, cast in 16 bits integers that are themselves cast in fp32...
The precision is artificially extended!

Increase precision for free



Using low precisions is promising

Floating-point arithmetic:

Sign	Exponent	Mantissa

	Bits Mantissa (t)	Exp.	Range	$u = 2^{-t}$
bfloat16	8	8	$10^{\pm 38}$	4×10^{-3}
fp16	11	5	$10^{\pm 5}$	5×10^{-4}
fp32	24	8	$10^{\pm 38}$	6×10^{-8}
fp64	53	11	$10^{\pm 308}$	1×10^{-16}
fp128	113	15	$10^{\pm 4932}$	1×10^{-34}

- 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** on emerging hardware ($16\times$ on A100 from fp32 to fp16/bfloat16)
- **Some limitations too:**
  - Low accuracy (large u)
  - Narrow range

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)

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

Various approaches: Mixed precision, Multiprecision, Adaptive precision, Variable precision, Transprecision, Dynamic precision, . . .

How to select the right precision for the right variable/operation

- **Precision tuning:** autotuning based on the source code, my thesis area: CADNA / PROMISE...
  - Does not need any understanding of what the code does

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

Various approaches: Mixed precision, Multiprecision, Adaptive precision, Variable precision, Transprecision, Dynamic precision, . . .

How to select the right precision for the right variable/operation

- **Precision tuning:** autotuning based on the source code, my thesis area: CADNA / PROMISE...
  - ▲ Does not need any understanding of what the code does

Adaptive precision algorithms

- Given an algorithm and a prescribed accuracy, adaptively select the minimal precision for each computation

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

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

and small elements produce small errors :

$$|\mathrm{fl}(a \,\mathrm{op}\, b) - a \,\mathrm{op}\, b| \le u|a \,\mathrm{op}\, b|, \qquad \mathrm{op} \in \{+, -, *, \div\}$$

⇒ **Opportunity for mixed precision:** adapt the precisions to the data at hand by storing and computing “less important” (usually smaller) data in lower precision

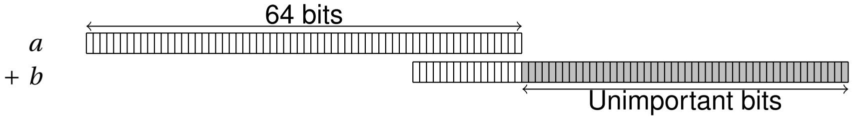
Before modifying the precisions used, we want to explore the current accuracy.

Adaptive precision algorithms

- Given an algorithm and a prescribed accuracy, adaptively select the minimal precision for each computation

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

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



and small elements produce small errors :

$$|\mathrm{fl}(a \,\mathrm{op}\, b) - a \,\mathrm{op}\, b| \le u|a \,\mathrm{op}\, b|, \qquad \mathrm{op} \in \{+, -, *, \div\}$$

⇒ **Opportunity for mixed precision:** adapt the precisions to the data at hand by storing and computing “less important” (usually smaller) data in lower precision

Before modifying the precisions used, we want to explore the current accuracy.

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]

The CADNA library

cadna.lip6.fr

- implements stochastic arithmetic for C/C++ or Fortran codes
- few code rewriting
- all operators and mathematical functions overloaded
- 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 $\approx$ 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

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

Executing CADNA on the PSA

To execute CADNA on the PSA, we essentially change the types.

Executing CADNA on the PSA

To execute CADNA on the PSA, we essentially change the types.

This execution exposes multiple numerical instabilities that hide potential massive loss of accuracy.

```
----------------------------------------------------------------
CADNA_C 3.1.11 software

CRITICAL WARNING: the self-validation detects major problem(s).
The results are NOT guaranteed.

There are 2803679 numerical instabilities
124420 UNSTABLE MULTIPLICATION(S)
127753 UNSTABLE BRANCHING(S)
323243 UNSTABLE INTRINSIC FUNCTION(S)
266 UNSTABLE MATHEMATICAL FUNCTION(S)
2227997 LOSS(ES) OF ACCURACY DUE TO CANCELLATION(S)
----------------------------------------------------------------
```

CADNA tricks

What if we use CADNA on peculiar physics codes?

CADNA tricks

What if we use CADNA on peculiar physics codes?

What if I truncate floats to int and use this int as an index for a Look Up Table that stores cbrt?

CADNA tricks

What if we use CADNA on peculiar physics codes?

What if I truncate floats to int and use this int as an index for a Look Up Table that stores cbrt?

Magical world: accuracy exiting the LUT is full!

CADNA tricks

What if we use CADNA on peculiar physics codes?

What if I truncate floats to int and use this int as an index for a Look Up Table that stores cbrt?

Magical world: accuracy exiting the LUT is full!

Without the LUT, we obtain around 6 exact correct digits

Perspectives

- Identify and fix numerical instabilities
- Explore further the actual accuracy of the PSA
- Explore smarter minimum research algorithms
- Experiment the use of fp16 and bf16
- Find the faster, enough accurate cbrt
- Implement mixed-precision in the PSA

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.archives-ouvertes.fr/hal-01254446

- CADNA: http://cadna.lip6.fr

Thank you for your attention!