Investigating mixed-precision for AGATA pulse-shape analysis

Roméo Molina[1,2,*], *Vincent* Lafage[1,**], *David* Chamont[1,***], and *Fabienne* Jézéquel[2,3,****]

[1]Université Paris-Saclay, CNRS/IN2P3, IJCLab, 91405 Orsay, France
[2]Sorbonne Université, CNRS, LIP6, Paris, 75005, France
[3]Université Paris-Panthéon-Assas, Paris, 75006, France

Abstract. The AGATA project aims at building a 4π gamma-ray spectrometer consisting of 180 germanium crystals, each crystal being divided into 36 segments. Each gamma ray produces an electrical signal within several neighbouring segments, which is compared with a data base of reference signals, enabling to locate the interaction. This step is called Pulse-Shape Analysis (PSA). In the execution chain leading to the PSA, we observe successive data conversions: the original 14-bit integers given by the electronics are finally converted to 32-bit floats. This made us wonder about the real numerical accuracy of the results, and investigate the use of shorter floats, with the hope to speedup the computation, and also reduce a major cache-miss problem. In this article, we first describe the numerical validation of the PSA code, thanks to the CADNA library. After the code being properly instrumented, CADNA performs each computation three times with a random rounding mode. This allows, for each operation, to evaluate the number of exact significant digits using a Student test with 95% confidence threshold. In a second step, we report our successes and challenges while refactoring the code so to mix different numerical formats, using high precision only when necessary, and taking benefit of hardware speedup elsewhere.

1 Introduction

AGATA[1] (Advanced GAmma-ray Tracking Array) is a European research project with the objective of building a detector with significantly higher resolution than existing ones. It will be used for experiments with intense stable and radioactive ion beams, to study the nucleus structure. In the present work we will deal with two problems: the uncertainty of the results in numerical simulations due to the floating point arithmetic that we will refer to as accuracy and the need to accelerate the processing of data coming from the detector. We will get interested in the way these both aspects are connected by the kind of precision formats that are used and from which depend both the performance and the accuracy of the computations.

*e-mail: romeo.molina@ijclab.in2p3.fr
**e-mail: vincent.lafage@ijclab.in2p3.fr
***e-mail: david.chamont@ijclab.in2p3.fr
****e-mail: fabienne.jezequel@lip6.fr

[1]https://www.agata.org

Today's computers are usually using floating-point operations to simulate calculations on real numbers. There exist several floating-point formats with varying numbers of bits. Until now, fp64 (called double precision) and fp32 (called single precision) have been the most common formats but we observe a rise of half precision formats in the last years, in particular on GPUs.

Applications requiring high definition and data processing, with numbers of very different magnitudes usually use the fp64 and fp32 formats, whose performance have been optimized in hardware and embedded in development practices. Those formats enable results with 7 or 15 significant digits and allow to deal with large ranges of values. In the counterpart, they are expensive in terms of execution time, energy and memory. Under the influence of low-precision applications, half (fp16, bf16) and even lower-precision formats are developing strongly, with dedicated hardware delivering excellent performance. On NVIDIA A100 GPUs, for example, fp16 or bfloat16 are 16 times faster than fp32. This performance is attracting interest in fields requiring both speed and high-accuracy results, such as particles and nuclear physics. To combine these two needs, mixed-precision approaches are being developed relying on the wise use of different kinds of precision formats in the same code to benefit from their respective advantages.

However, since floating-point arithmetic relies on approximations that accumulate as computations proceed, uncontrolled computations can lead to a completely false result. We therefore need to control the accuracy of our computations, in particular when we intend to mix several precisions. In the present paper we are using a software named `CADNA` which enables us to track the evolution of the number of exact significant digits in our code and detect numerical instabilities.

In this paper we will present our results on the stability of the code of the AGATA pulse-shape analysis by comparing the results obtained with fp64 format and the stochastic types from `CADNA`. We will also present the results obtained using a lower precision format fp16 on the naive full grid algorithm, and on the coarse-fine algorithm used in practice.

2 The AGATA experiment

Gamma-ray spectroscopy is a fast, non-destructive analysis technique for identifying radioactive isotopes in a sample. The energy of incident gamma rays is measured by a detector and compared with those known to be emitted by isotopes that allows to identify the isotope at the origin of the gamma ray.

High Purity Germanium (HPGe) detectors are the currently best solution for gamma and X rays spectroscopy. They are much more efficient than silicon ones that cannot be larger than few millimeters while germanium crystals can be few centimeters large and then absorb full gamma rays from a few tens of keV to beyond 10 MeV. Germanium crystals have to be very pure and work at really low temperatures around $-196\,^{\circ}\mathrm{C}$ that are achieved with nitrogen cryostats.

The processing of data from high-energy and nuclear physics experiments is historically divided into two steps. The first step, called *online*, takes place immediately on leaving the detector, and generally involves electronic processing to sort the data and eliminate noise. Today, it is performed using electronic components and FPGAs. The second step, called *offline*, consists in reconstructing the signal from the data obtained and stored, so that the computations can be performed several times, with the aim of refining them. However, the explosion of data volumes leads to more online computing. This is because physicists are seeking to identify increasingly rare and discreet phenomena hidden in what used to be considered as noise. The trend, therefore, is to introduce more software processing in the online part, to

cope with the influx of data. This calls for high-performance algorithmic processing, which is the goal of our work.

AGATA is a 4π array detector that will in time consist of 180 36-fold segmented HPGe detectors with optimized geometry grouped into triplets for a total of 6480 channels. It relies on the gamma-ray tracking. This technique consists in determining every point of interaction of a gamma-ray in the germanium crystal to track its whole path and then measuring the energy and the angle at which it has been emitted.

Each Germanium detector of AGATA is split into 36 segments (6 slices along the axis times 6 sectors around the axis). When a photon arrives into the Germanium crystal, it loses energy scattering electrons off atoms. This happens multiple times, producing a shower of electrons that generates an electric current and then an analog signal.

Analog signals are converted into digital signals using an analog-to-digital converter, sampling at 100 MHz over 14 bits. These data are then serialized and preprocessed by FPGAs, which reduce them by a factor of 100, retaining only those detectors that have recorded a relevant signal. In addition, a moving-window deconvolution algorithm determines the energy deposited in each detector. The raw data, energy information and time stamp are then transmitted to the computer system via optical fiber. From there, the NARVAL data acquisition system enables data to be stored in ADF (AGATA Data Flow) format and processed through a series of actors running as individual processes on a network of computer nodes.

The first actor is the crystal producer, which reads the measurements, casts it in fp32, which generates an increase in storage size, and passes it on to the preprocessing actor. This actor defines thresholds and corrects signal noise. At the output, it produces energy-calibrated and time-aligned data.

This is followed by the Pulse Shape Analysis (PSA), in which the signals obtained are compared with a signal database to determine the position of the interactions. This eliminates the need for all the data, and allows only the energy and position information to be transmitted to the next actor.

The post-PSA actor sets thresholds on core energy and segments that can be recalibrated and moved over time. It can also apply a correction for the effects of detector wear. The next step merges data from all AGATA crystals using time stamps. A filter can be used to set minimum event thresholds. Data from auxiliary detectors are then merged by the next actor, again using time stamps. The last actor is the tracking filter. It tracks the various events in the detector to determine how the gamma rays have interacted in the complete detector.

3 The `CADNA` tool for numerical validation

The representation of a floating-point number y, described by the IEEE754 standard [1], is $y = s \cdot m \cdot \beta^e$ with s the sign, β the base, e the exponent such that $e_{min} \leq e \leq e_{max}$ and m the mantissa which satisfies $1 \leq m < \beta$ for normalized numbers. Let us denote by t the precision, i.e. the number of bits in the mantissa plus an implicit bit set to 1. In the present paper, we are only dealing with floating point numbers in base $\beta = 2$. Thus we can define u the machine epsilon as $u = 2^{-t}$ in round-to-nearest mode. Depending on the format used, more or fewer bits are used to store the mantissa and exponent, as described in Table 1.

Elementary rounding errors accumulate and can lead to invalid results. This problem is perfectly illustrated by a polynomial introduced by S. Rump [2]. Let us consider $P(x, y) = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$ with $x = 77617$ and $y = 33096$. The results obtained with different working precisions are: $-1.85901\text{e+}30$ in fp32, $-1.18059\text{e+}21$ in fp64, 1.1726 in fp128, while the correct result is $\frac{54767}{66192} \approx -0.827396059946821368141165095479816292$.

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

Table 1. Various floating-point formats

Stochastic arithmetic [3] is a probabilistic method that involves performing each arithmetic operation several times with a random rounding mode: each result is rounded up or down with the same probability. Then the number of exact significant digits is estimated using a Student's test with a confidence level of 95 %. It has been shown [3] that, in some reasonable sense, the optimal sample size is $N = 3$. Indeed the estimation with $N = 3$ is more reliable than with $N = 2$ and increasing N does not significantly improve the quality of the estimation. `CADNA`[2] [4–7] is a library that implements stochastic arithmetic to estimate the number of exact significant digits of a floating-point number resulting from numerical computation. `CADNA` enables one to use new numerical types, called *stochastic types* on which numerical validation can be performed. `CADNA` also provides the definition of all arithmetic operations and mathematical functions involving stochastic variables. `CADNA` is available in `C/C++` and `Fortran` and is a relatively lightweight tool, requiring little code rewriting thanks to operator overloading, with a memory overhead of 4 and an execution time overhead of around 10. With `CADNA` each arithmetic operation is performed 3 times before the next arithmetic operation is executed, thereby propagating the rounding error differently each time. This synchronous execution enables the detection of numerical instabilities, usually due to results that have no exact significant digits. `CADNA` can be used for the numerical validation of HPC simulations: either CPU codes that rely on MPI and/or OpenMP or GPU codes written in CUDA. It makes it possible to control the numerical stability of a computation and to experiment with different precisions, including native or emulated half [8].

4 Profiling of AGATA computations

4.1 Performance analysis

The code of AGATA is almost fifteen years old, and suffers from various problems which put a strain on its performance. Furthermore, it is not adapted to take advantage of modern architectures capabilities. We first performed a static analysis that revealed the massive use of loops and sometimes arbitrary use of fp32 and fp64 formats. Using `perf`[3], a Linux-based performance analysis tool, we have explored its weaknesses and determined the functions that shall be targeted to improve its performance. This analysis clearly showed that the `Chi2InnerLoop` function concentrates 69 % of the CPU cycles, 80 % of the cache-references and more annoying, 73 % of the cache-misses. All this makes this function, and the PSA that includes it, a prime target for accelerating the execution of the code of AGATA. The number of memory accesses, due to the fact that we are processing a very large database, has led us to seek ways of reducing the amount of data accessed.

[2] `http://www.lip6.fr/cadna`
[3] `https://perf.wiki.kernel.org`

In addition, we have observed that in the execution chain leading to this PSA, there are successive data conversions. Data coming from the detector are originally sampled in 14 bits by the electronics, which correspond to a dozen bits of truthful information. These 14 bits are later extended into 16-bit integers to fit in computer format, and finally cast into 32-bit floats to perform floating-point operations. This made us wonder if lower floating-point precision can be used, while maintaining the accuracy.

Our work therefore has two concomitant objectives: reducing the cache-miss of the PSA by reducing the volume of data accessed and take advantage of the arithmetics of modern architectures, while controlling the numerical quality of our results.

4.2 Pulse-Shape Analysis

The PSA aims at comparing the obtained signal with a signal database to determine the position of the interaction. The naive algorithm consists in calculating a chi2 between the signals obtained in the interaction segment and its neighbors with those associated with each interaction point calibrated in a preliminary phase. Considering 9 neighboring segments, with 1200 sampled points each, over 60 time steps and 32 bits of data, we obtain 2.6 Mb, which exceeds the size of the cache.

Our work takes place in a context of numerous discussions with physicists to understand their needs and the logic behind their choices of implementation such as the use of a "chi2" using a power 0.3. These exchanges aim at defining what is an acceptable result from the PSA and therefore what room for maneuver we have in terms of accuracy.

A large amount of work has already been dedicated to this objective, with an algorithmic approach, denoted as *coarse-fine algorithm* from now, consisting in dividing the search into two phases: a first minimum search on a coarse grid and a second step that searches for the real minimum on a fine grid in the neighborhood of the coarse minimum. This approach, currently used in practice, considerably reduces the amount of data to be processed and accelerates the computation by a factor 5. However, it does not achieve exactly the same results as the naive *full grid algorithm*, as we explain hereafter. Another approach explored but not used in practice, consists in transforming the data to perform a wavelet analysis ; we do not develop this approach here.

We have noticed the use of a look-up table in the chi2 computation. This approach that reduces the accuracy, in particular for absolutely small numbers, also makes the use of `CADNA` inefficient, because data read from a table are considered exact, whereas loss of accuracy can be taken into account by `CADNA` in numbers resulting from numerical computation. Probably introduced to accelerate the code, it actually slows it down due to increased cache usage. We replaced it by a direct computation.

4.3 Accuracy control

We have then been particularly interested in checking the accuracy of computations carried out using the `CADNA` tool and experimenting the use of different precision settings. To do so, we have instrumented the PSA code so that we can easily use different precisions and also stochastic types of `CADNA`. We needed to experiment using `CADNA` to control the accuracy because we are performing a minimum search in a 504-dimensional space that carries a risk to accumulate catastrophic cancellations. We have first analyzed the version of the code implementing the full grid search and observed that the code is slightly sensitive to perturbations. Indeed, using stochastic fp32, stochastic fp64, or classic fp64, instead of the original classic fp32, the code gives 0.02 % difference in the points found. This first analysis validates the original results and allows us to envisage a further reduction of the precision used.

5 Using reduced precision formats

5.1 Half-precision computation

Having confirmed the numerical quality of the results obtained by the PSA, we sought to use reduced-precision formats within it. We therefore used a library[4] that emulates fp16 half precision. This format, introduced in the 2008 revision of the IEEE754 standard, enables one to perform floating-point operations on 16 bits only, and is becoming increasingly widely available, in particular on recent GPUs. It does not only reduce storage and energy consumption, but also delivers significant speed-up. For example, a computation performed in fp16 instead of fp32 can be accelerated by up to $\times 16$. This format, driven by the dynamism and massive investment in neural network computation that often uses reduced precision, is set to continue developing, and thus appears to be a central element in the performance gains for years to come. We therefore felt it necessary to explore the possibilities of exploiting such a format, in a context requiring a certain level of accuracy, namely nuclear physics, and in this case the execution of AGATA PSA.

Thanks to our initial instrumentation, we have been able to easily perform the full grid algorithm in fp16 which resulted in determining 7.76 % points differently from the full grid fp32 version. We have also been able to perform the coarse-fine algorithm in fp16 that resulted in 14.04 % of the points found differently.

The acceptability of those differences is to be discussed with physicists and is analysed more closely in section 5.3.

5.2 Mixed-precision computation

Mixed-precision computation is an emerging paradigm that mixes 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. It is based on the idea that not all computations are equally important, considering for example that in an addition between two numbers of different magnitudes, most of the significant digits of the smaller one may be eliminated. The main problem for this paradigm is to decide in which precision each operation shall be performed. Our work takes place in the area of precision tuning that aims at using a tool that evaluates the number of correct digits in a result for different precision configurations.

We therefore looked for a way of using low precision while remaining close to the original results. We turned to the algorithm used in production, which performs a coarse-fine search. Such an algorithm naturally lends itself to a mixed-precision approach, in which the coarse search is performed with low precision and the fine search with higher precision. Running the coarse-fine algorithm in fp32 yields 8.22 % different points with respect to the direct algorithm, while with the mixed-precision version we obtain 8.55 %, which seems comparable and then acceptable. Of course, we could also perform both steps in low precision, but under the same conditions, this would result in 14.04 % of differences from the fp32 full grid algorithm as already mentioned in section 5.1. In the case of the coarse-fine algorithm, the coarse part represents around 20 % of the computation time, so this is a limit to the time savings we can expect if we do not change the grid size to relatively increase the coarse part. These preliminary results confirm the interest of mixed precision for applications to improve the performance while maintaining the accuracy and strengthen our determination to continue in this direction.

[4] https://half.sourceforge.net

5.3 Evaluation of error acceptability

In this work, we were confronted with different ways of analyzing computation errors. Firstly, we were interested in numerical accuracy in the sense of the number of exact significant digits, an analysis enabled by CADNA and stochastic arithmetic. Then, we looked at the number of interaction points found differently using the possible combinations of methods and precision settings. In this section we try to measure how different the points found are and in which energy class they lie. To this end, we have produced the following figures. In Figure 1 we can observe that most of the points found differently from the full grid fp32 method are less than 3 mm far from the original points and that the number of points found differently from the full grid fp32 points decreases with distance. Moreover, as we can observe on Figure 2, most of the unfaithful points rely, as any point, in the smaller energy class that requires less accuracy. Also, we can observe that the distribution of unfaithful points is the same for the classic coarse-fine algorithm performed in fp32 and the mixed precision one using half precision for the coarse search. All in all, these data confirm that the mixed precision solution can be trusted as well as the uniform precision fp32 coarse-fine algorithm.

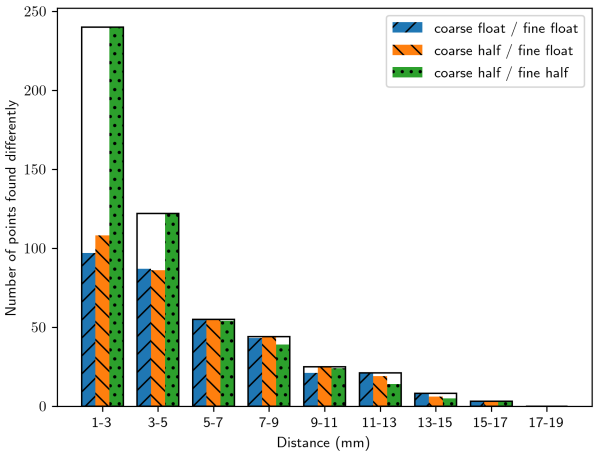

Figure 1. Distances between points found by the full grid fp32 algorithm and alternative methods

6 Conclusion and perspectives

We have demonstrated the benefits of using low precision in the AGATA PSA code, and specified the error control procedures required to obtain satisfactory results. To this end, we instrumented the AGATA code to allow the use of different combinations of precisions, using the CADNA tool which, with the aid of stochastic arithmetic, estimates the number of exact significant digits in a result. We faced a difficulty, since the result we were seeking was not a floating-point number, but a series of discrete points which we wanted to be as close as possible to the actual points of interaction. Finally, we opted for a mixed-precision computation based on the existing coarse-fine minimum search algorithm used in practice, with fp16 in the coarse step and fp32 in the fine step. This strategy enabled us to obtain very similar results to those provided by the coarse-fine algorithm in uniform precision fp32. We

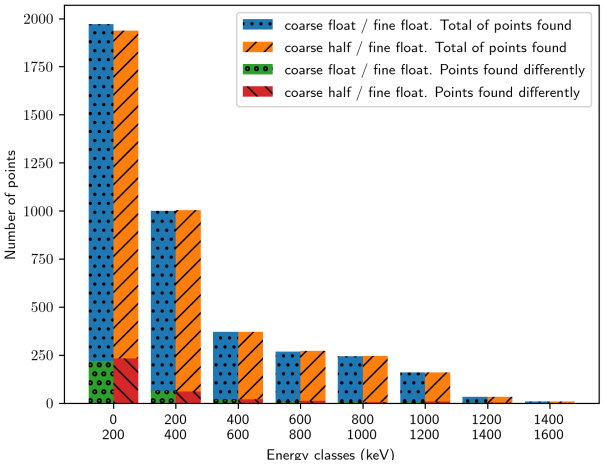

Figure 2. Energy classes in which the points found by various methods belong to

were also able to verify that points found differently from the full grid fp32 algorithm were at a small distance and very similar to those found by the coarse-fine fp32 algorithm. Finally, we checked the energy classes in which the different points were found and observed the same similarities and a distribution essentially in the low energies.

Following on from this work, it would appear necessary to implement our mixed-precision solution on hardware really supporting half-precision, and check the expected performance benefit. Another interesting lines of research would consist in refining the hierarchy of coarse and fine grids, either by adding intermediate grid sizes using various precisions, or by varying the respective weights of coarse and fine grids. Indeed, our strategy shows the expected benefit on the coarse part, but this remains limited due to the relative weight of the coarse search, which currently represents only 20 % of the total computation.

References

[1] IEEE 754 Committee, IEEE Std 754-2008, Microprocessor Standards Committee of the IEEE Computer Society, New York, NY (2008), doi:10.1109/IEEESTD.2008.4610935

[2] S.M. Rump, in *Reliability in computing* (Elsevier, 1988), pp. 109–126

[3] J. Vignes, Mathematics and Computers in Simulation **35**, 233 (1993)

[4] J.M. Chesneaux, *Study of the computing accuracy by using probabilistic approach*, in *Contribution to Computer Arithmetic and Self-Validating Numerical Methods*, edited by C. Ullrich (IMACS, New Brunswick, New Jersey, USA, 1990), pp. 19–30

[5] F. Jézéquel, J.M. Chesneaux, Computer Physics Communications **178**, 933 (2008)

[6] G. Flegar, H. Anzt, T. Cojean, E.S. Quintana-Ortí, ACM Trans. Math. Softw. **47** (2021)

[7] P. Eberhart, J. Brajard, P. Fortin, F. Jezequel, Reliable Computing **21** (2015)

[8] F. Jézéquel, S. Sadat Hoseininasab, T. Hilaire, *Numerical validation of half precision simulations*, in *1st Workshop on Code Quality and Security (CQS 2021) in conjunction with WorldCIST'21 (9th World Conference on Information Systems and Technologies)* (Terceira Island, Azores, Portugal, 2021), https://hal.science/hal-03138494