



HAL
open science

Mixed precision for High Performance Computing, application to low energy gamma radiation measurements

Roméo Molina

► **To cite this version:**

Roméo Molina. Mixed precision for High Performance Computing, application to low energy gamma radiation measurements. Nuclear Experiment [nucl-ex]. Sorbonne Université, 2024. English. NNT : 2024SORUS340 . tel-04901698

HAL Id: tel-04901698

<https://theses.hal.science/tel-04901698v1>

Submitted on 20 Jan 2025

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Sorbonne Université

École Doctorale Informatique, Télécommunications et Electronique (ED130)

Mixed precision for High Performance Computing, application to low energy gamma radiation measurements

Précision mixte pour le calcul de haute performance, application aux mesures de radiation gamma de faible énergie

Par Roméo MOLINA


Thèse de doctorat d'Informatique

Dirigée par Fabienne JÉZÉQUEL

Présentée et soutenue publiquement le 30/09/2024

Devant un jury composé de :

Marc	BABOULIN	Rapporteur	Université Paris-Saclay - LMF
Alfredo	BUTTARI	Président du jury	CNRS - IRIT
David	CHAMONT	Encadrant	CNRS - IJCLab
Stef	GRAILLAT	Invité	Sorbonne Université - LIP6
Fabienne	JÉZÉQUEL	Directrice	Université Paris-Panthéon-Assas - LIP6
Vincent	LAFAGE	Encadrant	CNRS - IJCLab
Matthieu	MARTEL	Rapporteur	Université de Perpignan Via Domitia - LAMPS
Théo	MARY	Invité	CNRS - LIP6
Olivier	STEZOWSKI	Examineur	CNRS - IP2I

 Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by-nc-nd/3.0/>

Quelques remerciements non exhaustifs

Merci d'abord à mes encadrants : Fabienne, Vincent et David. Vous m'avez accompagné dans ce voyage intranquille de la thèse, toujours avec écoute et bienveillance, aussi et surtout quand je n'étais plus très sûr de continuer. Merci Théo et Stef, c'est beaucoup grâce à vous que j'ai fait cette thèse puisque tout à commencé par un stage dans lequel vous m'avez donné avec Fabienne le goût de l'arithmétique flottante et du contrôle d'erreurs. C'est aussi avec vous que j'ai persévéré parce que nous avons continué à travailler ensemble jusqu'au bout.

Merci à Quentin, avec qui j'ai partagé mon bureau pendant trois ans et aussi quelques parties d'échecs ou de pédantix, sinon ça aurait été bien plus dur. Merci à tous les collègues doctorant.e.s, postdocs, ou titulaires du LIP6 et d'IJCLab que j'ai croisé chaque jours ou seulement quelques fois.

Merci à mes camarades et ami.e.s de Jussieu et d'ailleurs, militant.e.s, syndicalistes, en colère contre l'état du monde. C'est grâce à vous qu'il ne va pas plus mal et qu'il ira mieux un jour.

Merci à mes parents de m'avoir amené jusque là et de me soutenir dans mes choix contradictoires.

Et puis merci Manon pour tes dernières relectures, ton soutien et toutes nos échappées au-delà du monde.

Dans le cadre de cette thèse, nous nous intéressons aux formats de précision numérique, au contrôle de la validité de résultats numériques et aux opportunités qu'ils offrent dans le cadre du calcul haute performance. Plus précisément, nous nous intéressons à la précision mixte, qui consiste à mêler plusieurs formats de précision dans un même code pour tirer partie à la fois des gains de performances apportés par les précisions faibles et de validité et de la stabilité des précisions élevées. Nous étudions ici deux approches visant à introduire de la précision mixte. D'une part le tuning de précision qui consiste à développer des outils d'aide à la décision permettant d'introduire des précisions plus faibles que celles d'origine, dans un code existant, tout en effectuant un contrôle sur la validité des résultats ainsi obtenus. D'autre part, le développement d'algorithmes d'algèbre linéaire intrinsèquement mixtes en termes de précision pouvant ensuite être utilisés comme des briques de bases pour des applications diverses.

Dans ce manuscrit, nous nous intéressons à une application en particulier : la recherche en physique nucléaire, c'est-à-dire l'étude des particules et des interactions qui régissent cette échelle. Cet objectif est souvent atteint par l'étude de valeurs extrêmes, en particulier sur des noyaux très instables à l'aide de détecteurs à haute résolution. AGATA est une collaboration européenne visant à mettre au point un détecteur de rayons gamma au Germanium de Haute Pureté. Celui-ci s'appuie sur deux nouvelles technologies : la segmentation électrique des cristaux de Germanium et la reconstruction du parcours complet d'un rayon dans le détecteur. Pour cela, une étape d'analyse de la forme des traces mesurées dans chaque segments avec ceux d'une base de donnée préalablement calculée ou obtenue par calibration du cristal permet d'identifier les points d'interaction et leurs énergies associées. La quantité de données mesurée implique un traitement en direct mais cette étape doit aussi être réalisée avec précision car une résolution de 5mm est requise. Nous avons donc cherché à accélérer cette étape en réduisant le volume des données en utilisant des formats de précisions réduite. Afin de vérifier le maintien de la validité des résultats obtenus, nous nous sommes appuyés sur l'arithmétique stochastique mais aussi sur une évaluation du nombre de points identifiés pareillement par les différentes méthodes. Nous avons ainsi mis en évidence que l'exécution de l'algorithme d'origine pouvait se faire sans perte de qualité en FP16 plutôt qu'en FP32. Nous avons également effectué une réécriture de l'algorithme pour l'adapter à l'architecture GPU, montrant des résultats positifs et incitant à choisir ce type de matériel pour effectuer

cette étape.

Nous avons également réalisé un travail sur le produit matrice-vecteur creux en développant une version en précision mixte de cet algorithme. Celui-ci s'appuie sur une analyse rigoureuse qui permet de répartir les éléments en buckets et de les calculer dans une précision inversement proportionnelle à leur magnitude. Cet algorithme permet de garantir une précision cible mais aussi d'utiliser plusieurs formats de précision qu'ils soient natifs ou émulés. Cet algorithme permet d'obtenir des gains très importants en mémoire et en temps d'exécution mais se trouvait limité dans son utilisation des formats de précision non standards du fait de leur absence d'implémentation hardware. Nous avons donc décidé d'intégrer des accesseurs optimisés au sein du produit matrice-vecteur adaptatif et développé de nouveaux formats utilisant un exposant réduit tirant partie de la faible variation de magnitude au sein de chaque bucket.

In this thesis, we focus on numerical precision formats, on the control of numerical accuracy of the results and on the opportunities they offer in the context of High Performance Computing. More specifically, we are interested in mixed precision, which consists in mixing several precision formats in the same code to take advantage of both the performance gains of low precision and the validity and stability of high precision. Here, we examine two approaches to introduce mixed precision. On the one hand, precision tuning, which consists in developing decision-support tools to introduce lower precisions than the original ones, into an existing code, while checking the validity of the results thus obtained. On the other hand, the development of linear algebra algorithms that are intrinsically mixed in terms of precision, which can then be used as building blocks for various applications.

In this manuscript, we focus on one application in particular: nuclear physics research, i.e. the study of particles and the interactions that govern this scale. This is often achieved by studying extreme values, particularly on highly unstable nuclei, using high-resolution detectors. AGATA is a European collaboration to develop a High-Purity Germanium gamma-ray detector. It is based on the reconstruction of the complete path of a gamma-ray in the detector. To do this, there is a Pulse-Shape Analysis (PSA) that consists in comparing the traces measured in each segment with those of a database previously calculated or obtained by calibrating the crystal, to identify the interaction points and their associated energies. The quantity of data implies an on-line processing, but this step must also be carried out accurately, as a resolution of 5mm is required. We therefore sought to speed up this step by reducing the volume of data, using formats of reduced precision. To check the validity of the results obtained, we used stochastic arithmetic and evaluated the number of points identified by the different methods. In this way, we demonstrated that the original algorithm could be run without loss of quality when using FP16 rather than FP32. We also adapted the algorithm to GPU architecture, showing positive results and encouraging the choice of this kind of hardware for the PSA.

We also worked on the Sparse Matrix-Vector product (SpMV), developing a mixed-precision version of this algorithm. This is based on a rigorous analysis that divides elements into buckets and computes them with a precision inversely proportional to their magnitude. This algorithm not only guarantees target accuracy, but also allows the use

of several precision formats, both native and emulated. This algorithm delivers significant gains in memory and execution time, but was limited in its use of non-standard precision formats due to their lack of hardware implementation. We therefore decided to integrate optimized accessors within the adaptive matrix-vector product and developed new formats using a reduced exponent taking advantage of the small variation in magnitude within each bucket.

Contents

Résumé	v
Abstract	vii
Introduction	1
1 Context	2
1.1 Nuclear physics	2
1.2 Pulse-Shape Analysis of AGATA	3
1.3 Floating-point arithmetic	3
1.4 Towards a mixed precision paradigm	4
2 PhD objectives and contributions	5
3 Publications and presentations	6
4 Outline	7
1 Efficient and reliable floating-point computation	9
1 Reliable floating-point computation	9
1.1 Floating-point arithmetic	10
1.2 IEEE-754 1985 standard	13
1.3 Revisions and other formats	14
1.4 Error analysis	15
1.5 Error control	18
2 Benefits of mixed precision algorithms	23
2.1 Floating-point arithmetic context	23
2.2 Rise of low precision	24
2.3 The seek of mixing precisions	24
3 Conclusion	27
2 The Advanced GAMMA Tracking Array (AGATA)	29
1 Nuclear physics	29
1.1 Fundamental interactions	29
1.2 Nuclei behavior	30
1.3 Radioactive decay	30

1.4	Gamma-ray spectroscopy	33
1.5	Online and offline computing	34
2	The AGATA experiment	35
2.1	The geometry of the detector	35
2.2	AGATA data processing	36
3	The pulse-shape analysis of AGATA	38
3.1	AGATA Data Library	38
3.2	Gridsearch algorithm	39
3.3	Metric selection	41
3.4	Multiple interactions	41
3.5	A time consuming step	41
4	Conclusion	42
3	Adaptive SpMV and application to Krylov solvers	43
1	Introduction	43
2	Uniform precision matrix–vector product	44
3	Adaptive precision matrix–vector product: error analysis	46
3.1	A more practical componentwise bucket criteria	49
4	Adaptive precision SpMV: numerical experiments	50
4.1	Implementation	50
4.2	Experimental setting	52
4.3	Main results	53
4.4	Effect of dropping	56
4.5	Parallel scaling analysis	57
5	Application to Krylov solvers	57
5.1	Adaptive precision Krylov solvers	57
5.2	Iterative refinement	59
5.3	Adaptive GMRES-IR convergence analysis	60
5.4	Performance comparison for different Krylov solvers	61
6	Conclusion	65
4	Reduced-precision and reduced-exponent formats for accelerating adaptive SpMV	73
1	Introduction	73
2	Methods	75
2.1	Adaptive precision SpMV	75
2.2	Custom reduced-precision formats	75
2.3	Reduced-precision formats for adaptive precision SpMV	76
2.4	Reduced-exponent formats for adaptive precision SpMV	77
3	Evaluation	80
3.1	Performance of adaptive precision SpMV with RFPF	82
3.2	Performance of adaptive precision SpMV with RPRE and RPREU	83
4	Conclusion	83

5	Mixed precision for AGATA Pulse-Shape Analysis	87
1	Introduction	87
2	Profiling of AGATA computations	88
	2.1 Gridsearch configuration	88
	2.2 Performance analysis	89
	2.3 Accuracy control	90
3	Reduced precision formats	91
	3.1 Half-precision computation	91
	3.2 Mixed precision computation	92
	3.3 Error acceptability	93
4	Adapting the code to modern hardware	94
	4.1 The PSA test environment	94
	4.2 Experiments on CPU	95
	4.3 GPU deployment	96
	4.4 Use of native FP16	97
5	Conclusion and perspectives	97
	Conclusion & Perspectives	99
6	Conclusion	99
7	Perspectives	101
	Références	103

Introduction

Computers are first and foremost formidable computation machines. It is tempting to forget this point because this aspect is far from our everyday use of various computers. But this is where their power lies, and that is why we compare computers according to their computing power. Indeed, by enabling us to perform computations far more complex than we could ever hope to do by hand, computers open up impressive prospects, like in engineering or research.

However, despite they are everywhere in our lives, computers remain strange objects whose operation are for the most of us inaccessible. Indeed, using a computer is completely different from knowing how it works. Computer scientists know more about them, but there are many different specialties. A PhD thesis topic is usually situated in a small subset of a discipline. Ours is at the intersection of two subsets: accuracy issues in High Performance Computing (HPC) and nuclear physics experiments.

The main goal of HPC is to maximize computers' computing power. This involves both powerful machines using specific capabilities possibly connected to form computing clusters, and algorithmic efforts to make the most of the available hardware. In this context, every aspect of computing is a target for optimization, and so are the numerical formats with which operations are performed.

The accuracy of a result can be defined quite naturally as its closeness to the exact result. In the field of computer science we are talking about numerical accuracy to designate the quality of the results provided by the computer. While taking into account the error bounds of a result, whether due to limited-accuracy input data or numerical methods, has long been a major concern in physics, errors produced directly by the computer tend to be neglected. Yet, the computer does not always provide an exact result and, most of the time it does not, in particular when it operates on real numbers. Actually, as finite boxes, computers are not able to deal with real numbers and they use approximate representations called fixed-point or, more usually, floating-point numbers. Both of these representations exist in different precisions i.e. number of bits. But this number of bits is always finite, making it necessary to perform a rounding at each operation. The accumulation of these roundings may lead to completely false results. Then if it is not possible to achieve the exact results, the objective becomes obtaining a sufficiently accurate result, sufficiently close to the exact one and being able to control this closeness. This is the objective of any computer scientist wishing to produce meaningful

results as remind by [Goldberg, 1991]. In the context of HPC, the challenge is no longer only to obtain a sufficiently accurate result, but also to calibrate this accuracy to be the least expensive as possible. Indeed, the formats with which each operation is carried out determine not only the validity of the result, but also its cost in terms of time, energy and memory.

Why do computations have to be carried out in such a short time, and why must each operation be accelerated? The answer depends on the application. In this thesis, we focus on the field of nuclear physics, i.e. the study of atomic nuclei and the particles that interact with them. Current knowledge of the energies involved at this scale comes from the study of electromagnetic radiation emitted when the system releases energy through the emission of a photon. To observe these reactions, research requires two complementary tools: particle accelerators and collisioners to produce particles, and detectors to analyze them. In the nuclear area, energy levels are in the range of 10keV to 20MeV and are in the gamma-ray spectrum. Germanium gamma-ray detectors have played a vital role in the discovery of new phenomena since the 1980s, and our work is part of the development of a new High Purity Germanium detector as part of the European AGATA project. This new high-resolution detector must be able to process a huge data flow that cannot reasonably be stored for later processing. It is therefore necessary to reduce this data online, in order to retain only the useful part. But this operation also requires accuracy, as the retained data must be suitable for further processing.

1 Context

1.1 Nuclear physics

Physics has made several major advances since the 19th century, with two major models now providing an understanding of the interactions of bodies and particles: general relativity and the standard model of particle physics. However, these two theories are still not unified, so there is still a great deal of research to be done to understand more precisely the interactions that govern the universe at both microscopic and macroscopic levels.

Matter is made up of molecules, crystals or ions, which are themselves made up of atoms. There are various types of atoms but they are all made up of a positively-charged nucleus and a cloud of negatively-charged electrons. Within an atomic nucleus, there are two types of nucleons: positively-charged protons and uncharged neutrons. A chemical element is determined by its number of protons (Z), but can exist in different varieties, called isotopes, which correspond to different numbers of neutrons (N). The Standard Model aims at explaining electromagnetism, weak and strong nuclear interactions and the classification of all known subatomic particles. It is based on the triptych particle, force, mediator, i.e. it distinguishes families of particles by the forces to which they are sensitive, each force being exerted by the exchange of mediators (called bosons) between the particles subjected to it (called fermions). This model is used to build new models that include hypothetical particles, extra dimensions or supersymmetries.

Among the different isotopes of an atom, only a fraction of them are stable; the others are radioactive, i.e. they spontaneously change state during a nuclear reaction. During these reactions, large quantities of energy are emitted. This is why the term "nuclear"

is familiar to the general public, because some nuclear reactions can be used to make bombs or to produce civilian energy but this is not the topic of this thesis. We focus on the research that aims at understanding the characteristics of the nuclear many-body system. This objective is often approached by studying extreme values, particularly on nuclei far from stability. New Radioactive Ion Beam facilities now make proton- and neutron-rich nuclei accessible, opening up new perspectives for physics experiments. To study the reactions taking place in these facilities, high-resolution detectors are needed. Since the 1980s, high-resolution gamma-ray spectroscopy has become a key in the study of nucleus structure. To detect increasingly rare phenomena, it is necessary to widen the focus and confront an ever-growing mass of data. This is the aim of the AGATA detector, that is based on gamma-ray energy tracking in electrically segmented high-purity germanium crystals.

1.2 Pulse-Shape Analysis of AGATA

The AGATA detector [Akkoyun *et al.*, 2012] will be composed of 180 hexagonally shaped Germanium crystals. Each crystal is electrically segmented in six sectors centred on the crystal corners and six longitudinal rings to a total of 36 segments. It relies on two new technologies: position-sensitive Germanium crystals and gamma-ray tracking technology, which reconstructs the path of a gamma-ray in the Germanium crystal. This gamma-ray tracking is only possible if the interaction points and their associated energies are known. It is the role of the Pulse-Shape Analysis (PSA) to identify the interaction points from the traces measured in each of the segments. Thus, from a set of traces sampled over 120 ms in each segment and their associated energies, PSA is able to extract the interaction point and its associated energy.

To achieve this, the PSA relies on a signal basis obtained by calibrating or simulating the crystal. The crystal is discretized into a 2 mm resolution base, to which are associated the traces measured in the hit segment and in neighbour segments, as well as in the crystal core, which collects the sum of the energies measured in the crystal. When the detector is in operation, the PSA must compare the measured values with those of the base. This is a crucial step, which must be carried out online, due to the quantity of data produced, but which must also be sufficiently accurate, as a resolution of 5mm is required to carry out the ray-tracking from the interaction points. This step is a major challenge in the AGATA data processing chain which is supposed to be continuously improved. Currently, its frequency is around 5 to 10 kHz, i.e. 5000 to 10000 elements processed per second and there are various efforts dedicated to accelerate it.

This led us to look at the numerical precision while checking that the results remain sufficiently accurate. Indeed, one way of speeding up processing would be to reduce the size of the input data.

1.3 Floating-point arithmetic

These simultaneous needs for performance and accuracy are at the heart of the development of floating-point arithmetic.

Floating-point arithmetic takes its name from the fact that, unlike fixed-point arithmetic, it uses a floating-point to represent values of different magnitudes. In the early days of computing, the use of floating-point arithmetic grew because of its practicality in

approximating real numbers and performing operations on them, while allowing numbers of very different magnitudes to be manipulated in the same program. In fact, a floating-point number is made up of three parts: a sign bit, a group of bits encoding the exponent (i.e. the magnitude), and a final group storing the significant digits of the number.

Such a definition reveals two major problems: firstly, its great flexibility, secondly, the finite number of bits available for encoding that necessarily implies an approximation for certain numbers. Concerning flexibility: while this term might sound positive, it implies a great deal of freedom for designers, whether in hardware or software, complicating the portability of a code from one machine to another. As an answer to this problem, among others, that the IEEE-754 standard [IEEE Computer Society, 1985] was developed in 1985, and has since been extended several times, setting the framework for floating-point arithmetic as we find it in today's computers. As for the rounding error accumulation, this remains a major problem as demonstrated by the various disasters it has caused.

Various approaches have therefore been developed to address the issue of rounding errors.

Firstly, it is possible to increase the storage formats. This is why the use of FP64 remains a standard for many applications and why some adopt formats beyond 64 bits such as 80-bit, 128-bit or even higher formats. This strategy is based on the fact that, whatever the format used, the number of bits lost during a rounding operation is the same. So, if you start with a high or very high margin, you have a good chance of achieving a sufficiently accurate result. However, a major limitation of this technique is the lack of hardware implementation of these formats, which requires software emulation and are therefore catastrophic in terms of performance and out of reach for large programs.

Secondly, both static and dynamic methods have been developed to ensure accuracy using a given precision format. Interval methods [Moore, 1966] for instance, is a dynamic method that replaces values by intervals that are guaranteed to contain the exact result. However, they are confronted with a rapid explosion of intervals, which reduces their interest since the result is hardly significant and usable. The static methods seek to guarantee the accuracy of a program's result without executing it. These methods have undergone considerable development following the explosion of the Ariane 5 rocket due to a conversion problem, but they are extremely time-consuming and difficult to apply to industrial codes.

Another kind of methods, called probabilistic methods have the advantage of possibly being used in a context of large-scale codes. They aim to guarantee a number of exact significant digits within a confidence interval with reasonable time and memory overhead. This is the case with the CESTAC method [Vignes & Porte, 1974; Vignes, 1978], which uses several executions of the same operation with random rounding to estimate the number of exact significant digits, thanks to a Student's t-test [Student, 1908] at 95%.

1.4 Towards a mixed precision paradigm

Accuracy issues are not only a challenge for obtaining correct results, they are also an opportunity for achieving performance gains. In the context of High Performance Com-

puting (HPC), the goal is not only to get a sufficiently accurate result, but also to identify the necessary precisions, neither too low nor too high, adapted to our problem. It has been shown that training neural networks in low precision usually does not affect the results [Yun *et al.*, 2023]. Thus, the field of artificial intelligence has massively turned to the use of reduced precision formats: half format on 16 bits (FP16 and BFLOAT16) but also more recent formats using only 8 bits that offer a potential of very significant gains in terms of memory, computation time and power consumption. Indeed, each value in FP16 or BFLOAT16 takes up half as much memory as in FP32, and for instance, some computations are up to 16 times faster in half than in FP32 on A100 GPU cards.

These gains come with a counterpart in terms of the numerical quality of the results. A computation naively performed in half will only have an accuracy of the order of 10^{-3} or 10^{-4} depending on the chosen format, compared with around 10^{-7} if it were performed in FP32. If, as we have seen, these levels of accuracy are sufficient for some applications, they are quite inadequate for others.

The attraction of performance gains has therefore motivated the development of mixed precision solutions, i.e. using two or more precisions in the same code, with the aim of maximizing the use of low precisions and reserving high precisions for critical portions. We distinguish two main classes of mixed precision solutions. On the one hand, we found solutions that aim at developing mixed precision linear algebra algorithms as summarized in [Higham & Mary, 2022]. These programming building blocks that realize matrix products, linear systems solving, eigenvalue decompositions, and other linear algebra operations, are intended to be used in the codes of various applications, as it is currently the case with the Basic Linear Algebra Subprograms (BLAS) [BLAS Technical Forum, 2001] and their updates. Their development requires a detailed analysis of the original algorithms in order to propose versions that use several precisions and whose accuracy is guaranteed. On the other hand, auto-tuning solutions are being developed to "democratize" mixed precision, i.e. to make it accessible to all, particularly for existing, large-scale codes that are difficult to handle. These solutions aim to provide tools that automatically determine the best combination of precisions in a code. Some of these tools use stochastic arithmetic to provide reference results.

2 PhD objectives and contributions

Numerical precision and its corollary, accuracy, represent a challenge, since their control conditions the validity of results. However, it is also an opportunity, as it can be used to improve computational performance. In particular, the development in recent years of mixed precision solutions made it possible to achieve gains by using reduced precision in applications requiring high accuracy. As we have seen, this has been achieved through two complementary approaches: development of mixed precision linear algebra algorithms and of auto-tuning tools.

In this thesis, we explore these two approaches. In the field of mixed precision algorithms for numerical linear algebra, we worked in an emerging subclass: adaptive precision algorithms. The idea is to take into account the data at hand to determine the precisions used. So, as we will see in Chapter 3, we have developed a Sparse Matrix-Vector Product (SpMV) algorithm for which we are able to set an accuracy target. The matrix elements are stored in a precision inversely proportional to their magnitude and

the algorithm can use various kinds of formats both existing in hardware or custom. In light of the performance achieved by this algorithm, we sought to go one step further by introducing into this adaptive SpMV, optimized accessors to take advantage of our custom precision formats, and by developing new ones using a reduced exponent. This work is detailed in Chapter 4. Finally, we have sought to take advantage of low precisions and to use mixed precision in the AGATA's Pulse-Shape Analysis, providing a theoretical and a practical analysis including a GPU implementation as detailed in Chapter 5.

3 Publications and presentations

Publication in an international journal

- S. Graillat, F. Jézéquel, T. Mary, R. Molina, Adaptive precision sparse matrix-vector product and its application to Krylov solvers, *SIAM Journal on Scientific Computing*, 46(1), pages C30-C56, 2024. [[Graillat et al., 2024a](#)]

International conference proceedings articles

- D. Chamont, R. Molina, V. Lafage, F. Jézéquel, Investigating mixed precision for AGATA pulse-shape analysis, 26th International Conference on Computing in High Energy and Nuclear Physics (CHEP 2023), Norfolk, USA, May 2023. [[Molina et al., 2023b](#)]
- R. Molina, S. Graillat, F. Jézéquel, T. Mary, D. Mukunoki, Reduced-Precision and Reduced-Exponent Formats for Accelerating Adaptive Precision SpMV, International European Conference on Parallel and Distributed Computing (Euro-Par), Madrid, Spain, August 2024. [[Graillat et al., 2024b](#)]

International conference proceedings abstracts

- R. Molina, S. Graillat, F. Jézéquel, T. Mary, Adaptive Precision Sparse Iterative Solvers, SIAM Conference on Computational Science and Engineering (CSE23), Amsterdam, Netherland, February - March 2023.
- S. Graillat, F. Jézéquel, T. Mary, R. Molina, Adaptive precision sparse matrix-product and application to Krylov solvers, International Congress on Industrial and Applied Mathematics (ICIAM 2023), Tokyo, Japan, August 2023. [[Molina et al., 2023a](#)]

Poster in an international conference

- R. Molina, S. Graillat, F. Jézéquel, T. Mary, Adaptive Precision Sparse Matrix-Vector Product and its Application to Krylov Solvers, Sparse Days conference, Saint-Girons, France, June 2022. [[Molina et al., 2022](#)]

Presentations in workshops or seminars

- R. Molina, D. Chamont, F. Jézéquel, V. Lafage, PSA cache-misses and precision analysis with CADNA, AGATA Week, Orsay, June 2022.
- R. Molina, S. Graillat, F. Jézéquel, T. Mary, Adaptive Precision Sparse Matrix-Vector Product and its Application to Krylov Solvers, RAIM, Nantes, November 2022.
- R. Molina, D. Chamont, F. Jézéquel, V. Lafage, AGATA computation control: from the crystal to the final measure, AGATA France, Orsay, November 2022.
- R. Molina, D. Chamont, F. Jézéquel, V. Lafage, AGATA computation control: from the crystal to the final measure, Journées Informatique IN2P3, Le Croisic, November 2022.
- R. Molina, S. Graillat, F. Jézéquel, T. Mary, Adaptive Precision Sparse Matrix-Vector Product and its Application to Krylov Solvers, xSDK Multiprecision Seminar, Online, December 2022.
- R. Molina, S. Graillat, F. Jézéquel, T. Mary, Adaptive Precision Sparse Matrix-Vector Product and its Application to Krylov Solvers, PREMIX Meeting, Paris, May 2023.
- R. Molina, S. Graillat, F. Jézéquel, T. Mary, Adaptive Precision Sparse Matrix-Vector Product and its Application to Krylov Solvers, MUMPS User Days, Paris, June 2023.

4 Outline

This manuscript is organized as follows. Chapter 1 introduces the notions of floating-point arithmetic, digital error and its control, and leads on to the emergence of a mixed precision paradigm based on two approaches. Chapter 2 presents the AGATA detector and the nuclear physics foundations on which it is based, and elaborates on the step of the Pulse-Shape Analysis, as a key step on the data processing of the AGATA detector. Chapter 3 sets out the error analysis and the experiments carried out around the adaptive Sparse Matrix-Vector Product and its use in the context of Krylov solvers. Chapter 4 presents the work carried out around the use of optimized accessors for reduced precision and reduced exponent formats in the adaptive SpMV context. Finally, Chapter 5 describes the work and results obtained in exploring the use of reduced precision and mixed precision in the AGATA's Pulse-Shape Analysis.

Efficient and reliable floating-point computation

Real numbers can have an infinite number of digits but computers are finite boxes that can only store a fixed number of digits. In order to deal with this apparent paradox various approaches have been developed, in particular fixed-point arithmetic and floating-point arithmetic. Fixed-point arithmetic allows to perform operations efficiently on numbers of the same order of magnitude. Indeed, as the decimal point is fixed, we always have the same number of digits after the decimal point, whether the number is large or small. However, many applications require the manipulation of numbers of extremely different magnitudes. For this reason, floating-point arithmetic, that will be at the heart of the work carried out in this thesis, has become widely accepted. Floating-point arithmetic is based on a variety of formats that allow numbers to be represented with varying degrees of precision. The level of precision is inversely proportional to energy efficiency and computing speed, and it is in this context that mixed precision algorithms offer an opportunity to combine the advantages of low precision with the seek of high accuracy provided by high precision.

1 Reliable floating-point computation

Until the 1980s, the representation of floating-point numbers was not regulated and varied from one architecture to another, posing major problems of code portability. In 1985, the IEEE754 standard [IEEE Computer Society, 1985] filled this gap by imposing a common representation and proposing 4 floating-point formats. This standard was rapidly adopted by manufacturers in the years that followed, and has since been supplemented by revisions in 2008 [IEEE Computer Society, 2008] and 2019 [IEEE Computer Society, 2019], responding to new issues and introducing additional formats. Today, most processors implement floating-point operations as prescribed by the IEEE754 standard, but there are also independent formats, as we will see below.

1.1 Floating-point arithmetic

1.1.1 First definitions

A system of floating-point numbers \mathbb{F} is a finite subset of \mathbb{R} mainly characterized by four integer parameters:

- the basis $\beta \geq 2$, that is 2 for virtually any modern computer
- t the precision, meaning the number of digits that are used to represent the value of $x \in \mathbb{F}$
- e_{\min}, e_{\max} the extrema exponents that define the range of elements representable in \mathbb{F}

An element of \mathbb{F} has the form

$$x = M \cdot \beta^{e-t+1}. \quad (1.1)$$

Where M is an integer called the significand and respects $|M| \leq \beta^t - 1$ and e is the exponent of x and respects $e_{\min} \leq e \leq e_{\max}$.

Another representation of $x \in \mathbb{F}$ is the *normal* definition

$$x = (-1)^s \cdot m \cdot \beta^e \quad (1.2)$$

In this definition,

- $s \in \{0, 1\}$ is the sign
- $m = \beta^{1-t} \cdot |M|$ is the normal significand that respects $0 \leq m < \beta$ and has 1 digit before the radix point and at most $t - 1$ after.
- e is the exponent

To fix the uniqueness of the representation, one chose the representation of x for which e is minimum while still larger than e_{\min} . Then the system is called normalized.

1.1.2 Normal and subnormal numbers

In a normalized floating-point system, there are two kinds of numbers:

- those for which $e > e_{\min}$ and then $m \geq 1$ are called normal numbers
- those for which $e = e_{\min}$ and then $0 \leq m < 1$ are called subnormal numbers

In this system we obtain the following extrema:

- the largest finite number $\beta^{e_{\max}} \cdot (\beta - \beta^{1-t})$
- the smallest positive normal number $\beta^{e_{\min}}$
- the smallest positive subnormal number $\beta^{e_{\min}-t+1}$

The use of subnormals may seem unwise, as it requires specific operations to take into account particularly small elements. Whether or not to implement them has been a major issue during the standardization process. Today, although integrated into the standard, they are often supported in software rather than in hardware. This leads to heavy penalties in terms of execution time, and it is therefore often worth excluding them to achieve performance gains. However, their existence enables to maintain interesting arithmetic properties and it makes it easier to write numerically stable programs.

1.1.3 Special representations

To build a robust system, it seems necessary to define certain specific values.

Firstly, it is important to take into account the result of incorrect arithmetic operations (e.g. $\sqrt{-2}$, $0/0$) to obtain a closed system that allows to detect them. This is the role of Not a Number (NaN), which can use flags to bring information to the error detected. These NaNs propagate within the calculations.

Furthermore, the limited exponent of floating-point numbers leads to the need to represent values that exceed this magnitude. To do this, we can choose to introduce a single unsigned infinity (∞) or two signed infinities ($+\infty$ and $-\infty$). If signed infinities are chosen, it may be interesting to have signed zeros too.

1.1.4 Rounding modes

The result of an operation on floating-point numbers is generally not representable on a floating-point number. It is therefore necessary to round it to a representable value. In itself, this rounding introduces only a small error, as we will see later, but repeated and amplified many times in the course of a calculation, it can lead to incorrect results or results that differ according to the chosen rounding mode.

If $x \in \mathbb{R}$, $fl(x)$ denotes the first element of \mathbb{F} smaller or larger than x according to the chosen rounding mode. If x is larger than the upper bound of \mathbb{F} , one say that $fl(x)$ overflows. If it is smaller than the lower bound of \mathbb{F} , one say that $fl(x)$ underflows. When subnormal numbers are available we say the underflow is gradual.

The IEEE-754 standard proposes four different rounding modes:

- round toward $+\infty$: $RU(x)$ is the smallest element of \mathbb{F} larger than x
- round toward $-\infty$: $RD(x)$ is the biggest element of \mathbb{F} smaller than x
- round toward 0: $RZ(x)$ is $RU(x)$ if $x < 0$ and $RD(x)$ if $x > 0$
- round to nearest: $RN(x)$ is the closest element of x in \mathbb{F} . It requires a tie-breaking rule in case of elements exactly in the middle between two elements in \mathbb{F} . Round to nearest even is frequently chosen.

When the exact result of a function is given with a specified rounding mode (meaning that the result is the same as the one computed in infinite precision and then rounded), the function is called correctly rounded.

Obtaining correct rounding on basic arithmetic operations ($+$, $-$, $*$, $/$, $\sqrt{}$) is relatively simple, but can be very complicated for common mathematical functions. So we add the faithful rounding property to indicate that a function always returns $RD(x)$ or $RU(x)$.

There are interesting properties that come with the four rounding modes described above. They are non-decreasing functions, meaning that if $x \leq y$, $\circ(x) \leq \circ(y)$. We also have that if x is a floating-point number, we have $\circ(x) = x$. Moreover symmetric rounding modes (RZ and RN with symmetric tie-breaking rule) respect the symmetries of correctly rounded functions and for RU and RD modes we have a symmetry property that allows to easily switch from one mode to another:

$$\begin{aligned}RU(a + b) &= -RD(-a - b) \\RD(a * b) &= -RU((-a) * b)\end{aligned}$$

<pre> 1 float harmonicSum_1(int n){ 2 float s=0; 3 for(int i=1; i<=n; i++) 4 s = s + 1./i; 5 return s; 6 }</pre>	<pre> 1 float harmonicSum_2(int n){ 2 float s=0; 3 for(int i=n; i>=1; i--) 4 s = s + 1./i; 5 return s; 6 }</pre>
---	---

Figure 1.1: Harmonic sum

1.1.5 Properties

Arithmetic on real numbers carries some well-known properties:

- commutativity: $a + b = b + a$ and $a \cdot b = b \cdot a$ for any a and b
- associativity: $a + (b + c) = (a + b) + c$ and $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ for all a, b and c
- distributivity: $a \cdot (b + c) = a \cdot b + a \cdot c$ for all a, b and c

When you switch to floating-point arithmetic, things change. Elementary operations with correct rounding in one of the four modes described above, preserve commutativity, but neither associativity nor distributivity. In some extreme case, the loss of associativity can lead to drastically different results between $(a + b) + c$ and $a + (b + c)$. For instance, in base β and precision p and round-to-nearest mode, let us have $a = \beta^{p+1}$, $b = -\beta^{p+1}$ and $c = 1$. We have $\text{RN}(a + \text{RN}(b + c)) = \text{RN}(\beta^{p+1} - \beta^{p+1}) = 0$ whereas $\text{RN}(\text{RN}(a + b) + c) = \text{RN}(0 + 1) = 1$. With \times operation, differences remain very tight since there are no occurrence of underflow or overflow. When they happen we can observe drastically different results as in the following example:

Let be $\beta = 2$, $p = 53$, $e_{\min} = -1022$ and $e_{\max} = 1023$. With $a = b = 2^{514}$ and $c = 2^{-1022}$ we obtain $\text{RN}(\text{RN}(a \cdot b) \cdot c) = +\infty$ whereas $\text{RN}(a \cdot \text{RN}(b \cdot c)) = 64$

1.1.6 Typical errors

Absorption happens when adding or subtracting two floating-point numbers of very different magnitudes. This operation causes a massive loss of accuracy and in extreme cases it leads to simply ignore the smallest number. For instance, let $x = 1$ and $y = 2^{24}$ be two FP32 numbers as defined in IEEE standard [IEEE Computer Society, 1985], we have $fl(x+y) = y$. This phenomenon may happen when accumulating small numbers and leads to a stagnation. The harmonic sum is a typical case that can be easily resolved. Figure 1.1 presents two ways to compute the harmonic sum that are mathematically but not numerically equivalent. Indeed, whereas `harmonicSum_1(10000000)` provides the result 15.403683, `harmonicSum_2(10000000)` provides 16.686031. `harmonicSum_1` adds larger numbers first and then the latest numbers are too small to be taken into account. On the contrary `harmonicSum_2` always makes additions between numbers of the same magnitude that allow to cope with the absorption problem.

Cancellation or catastrophic cancellation arises when subtracting two very close numbers. Figure 1.2 presents two ways to compute the difference of squares. The first one produces the wrong result $1.8626451492309570e - 09$ due to a cancellation error whereas the second shows how to cope with it to produce the correct result $1.8626451518330422e - 09$

```

1 int main() {
2     double x = 1+pow(2,-29);
3     double y = 1+pow(2,-30);
4     printf("x^2 - y^2 = %1.16e\n", pow(x,2)-pow(y,2));
5     printf("(x-y)(x+y)= %1.16e\n", (x-y)*(x+y));
6     return 0;
7 }

```

Figure 1.2: Catastrophic cancellation example

	Bits			
	Sign	Exponent	Signif.	Exp. bias
FP32	1	8	24	127
FP64	1	11	53	1023

Figure 1.3: Formats specified by the IEEE 754-1985 standard

1.2 IEEE-754 1985 standard

The IEEE-754 standard or IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985), released in 1985, resolved a situation in which each manufacturer adopted a different implementation of floating-point arithmetic, resulting in a lack of code portability from one architecture to another.

It therefore provided the opportunity to set the elements presented above, but also to adopt specific conventions and formats to meet the following contradictory needs.

- portability: programs shall run on different machines without requiring modifications
- accuracy: programs must provide results with a quantified correctness
- speed: basic operations have to be fast
- range: a large range of values has to be representable
- easy implementation and use: to be adopted largely, it has to be accessible

Initially written for base two, it has been generalized two years later. In its radix-2 version, it appears that the first bit of the significand is always equal to 1, so there is no need to store it. This is known as the hidden bit convention. This convention makes it possible to represent significands that are one bit larger than what is actually stored.

The first version of the IEEE-754 standard requires the existence of two basic formats, binary32 (single precision) denoted FP32 in the following and binary64 (double precision) denoted FP64 in the following, 32-bit and 64-bit respectively, as well as two associated extended formats. The latter have been designed to store intermediate calculations on variables stored in the basic formats. In practice, the binary64 extended format is actually implemented, but the binary32 extended format is not. If a binary32 precision calculation requires higher precision, the binary64 format is used. In this document we will abbreviate the formats standardized by IEEE-754 by FPXX where XX

Decimal value	Sign	Exponent	Significand
+0	0	0000 0000	0000 0000 0000 0000 0000 0000
-0	1	0000 0000	0000 0000 0000 0000 0000 0000
1	0	0111 1111	0000 0000 0000 0000 0000 0000
-1	1	0111 1111	0000 0000 0000 0000 0000 0000
10.375	0	1000 0010	0100 1100 0000 0000 0000 0000

Figure 1.4: Exemples of numbers represented in FP32 format

indicates the number of bits occupied by these formats. Thus binary32 will be called FP32.

The standard also clarifies the implementation of exponents. Biased exponents are chosen to allow both positive and negative exponents to be represented by a positive integer, while making it easier to compare floats with each other. Let w be the number of bits dedicated to store the exponent, the bias is set to $b = 2^{w-1} - 1$. Thus the stored exponent corresponds to $e - b$. The extrema allow to store special values:

- exponent 0 is dedicated to +0 and -0 according to their bit sign and with 0 on all the significant bits and denormals that have nonzeros on the significand
- exponent $2^w - 1$ is reserved for infinities (with zeros on the significand part) and NaNs with nonzeros on their significant bits that can serve as flags to signal exceptions.

The IEEE-754 standard also establishes round to nearest by default with a tie-breaking rule to even meaning that when the result is exactly between two floating-point numbers, it is the one with a zero as least significant bit that is chosen. It also specifies that any number greater than x_{\max} is rounded to $+\infty$.

Also the IEEE-754 requires for the basic arithmetic operations (+, -, *, /, $\sqrt{\quad}$) to be furnished with correct rounding.

Table 1.3 presents the formats specified by the IEEE-754-1985 standard and the sizes of their different fields.

1.3 Revisions and other formats

Although the 1985 version of IEEE-754 filled a glaring lack of standardization, it did not solve all the problems associated with floating-point arithmetic, and has since been supplemented by two revisions and the emergence of new formats driven by manufacturers.

In 2008, a major revision of the standard took place [IEEE Computer Society, 2008], merging the 1987 IEEE-854 standard, which introduced generalization to any base, into IEEE-754 and standardizing practices already in use: the existence of a quadruple precision format (FP128) and the FMA operation. The latter, whose full name is full-multiply-and-add, is an operation that performs both multiplication and addition in a single operation, saving both time and accuracy, since the result is rounded only once instead of twice. This revision also resolved ambiguities and reduced implementation choices. It also standardized a 16-bit floating-point format called binary16 or FP16. Initially developed for low-precision applications, particularly on graphics processing units

	word size	Bits			Range
		Sign	Exponent	Signif.	
FP128	128	1	15	113	$10^{\pm 4932}$
FP16	16	1	5	11	$10^{\pm 5}$
BFLOAT16	16	1	8	8	$10^{\pm 38}$
FP8-E4M3	8	1	5	3	$10^{\pm 2}$
FP8-E5M2	8	1	6	2	$10^{\pm 1}$

Figure 1.5: Formats developed after the IEEE 754-1985 standard

(GPUs), this format is now attracting growing interest thanks to its major performance gains.

In 2019, a new minor revision [IEEE Computer Society, 2019] has been added to the IEEE-754 standard, containing mainly uncontroversial corrections, clarifications and improvements compared to the 2008 version. In particular it provides a list of operations and mathematical functions that must be correctly rounded.

In parallel with this standardization effort, certain technological developments have led to the creation of new formats by manufacturers. In particular, the rise of neural networks, for which low accuracy is suitable, has encouraged the development of low or even very low precision. In particular, a new 16-bit format, BFLOAT16, was developed in 2018 by Google for its Tensor Processor Units (TPUs), specialized computing units for neural networks. This format uses an exponent size identical to that of FP32, enabling it to cover a similar range of values, but it has a smaller significand size and therefore lower precision than FP16. Even more recently, two formats on 8 bits have been developed.

1.4 Error analysis

As we have seen, there are different formats corresponding to different levels of precision, as well as different rounding modes. In any case, floating-point arithmetic faces the problem of rounding and the error it generates, which can propagate and amplify. Let us start by looking at how this error can be quantified.

1.4.1 Absolute and relative error

Let \tilde{x} the approximation of the real number x . The two most important measures of the accuracy of x are its absolute error

$$E_{abs}(\tilde{x}) = |x - \tilde{x}| \quad (1.3)$$

and its relative error

$$E_{rel}(\tilde{x}) = \frac{|x - \tilde{x}|}{|x|} \quad \text{if } x \neq 0 \quad (1.4)$$

The relative error is especially useful because it is scale independent, not affected by the magnitude of the variables and is adapted to scientific computation that can deal with very different kind of magnitudes.

If x and \tilde{x} are vectors, one can use alternatively the normwise relative error defined as

$$E_{NW}(\tilde{x}) = \frac{\|x - \tilde{x}\|}{\|x\|} \quad (1.5)$$

or the componentwise relative error

$$E_{CW}(\tilde{x}) = \max_i \frac{|x_i - \tilde{x}_i|}{|x_i|} \quad (1.6)$$

The latter puts the individual relative errors on equal footing. The choice of one or the other depends on the application.

1.4.2 Ulp function and unit roundoff

The ulp function, for unit in the last place, aims to provide an evaluation of the weight of the last bit of the mantissa or the spacing between floating-point numbers. In this document, we will use the definition provided by Goldberg [Goldberg, 1991] and extended to reals.

Definition 1 *With the same notations as before, If $x \in [\beta^e, \beta^{e+1})$ then, $\text{ulp}(x) = \beta^{\max(e, e_{\min}) - t + 1}$*

It is important to observe that this spacing does not behave the same way for normal and denormal numbers. Whereas denormal numbers are equally spaced by a value equal to the smallest positive subnormal, normal numbers are not. In fact, their spacing jumps by a factor β at each power of β . From this definition derives the definition of the unit roundoff.

Definition 2 *The unit roundoff u of a floating-point system is defined as*

$$u = \begin{cases} \frac{1}{2} \text{ulp}(1) = \frac{1}{2} \beta^{1-t} & \text{with round-to-nearest mode} \\ \text{ulp}(1) = \beta^{1-t} & \text{with directed-rounding modes} \end{cases}$$

This definition is useful in the analysis of numerical algorithms. Indeed, for an operation $\text{op} \in \{+, -, \times, /\}$, a rounding mode $\circ \in \{\text{RU}, \text{RD}, \text{RZ}, \text{RN}\}$ and two elements $a, b \in \mathbb{F}$ we have

$$fl(a \text{ op } b) = (a \text{ op } b)(1 + \delta_1) = (a \text{ op } b)/(1 + \delta_2), \quad |\delta_1|, |\delta_2| < u \quad (1.7)$$

This property eases rounding analysis.

1.4.3 Accuracy or precision?

In common language, accuracy and precision are used interchangeably but in our field it is worth making a distinction between them. Accuracy corresponds to the absolute or relative error of an approximate quantity whereas precision designates the accuracy with which the basic arithmetic operations are performed. In floating-point arithmetic precision is measured by the unit roundoff. Even if it might seem counterintuitive, accuracy is not limited by precision as we will see below.

1.4.4 Forward and Backward error

Let \tilde{y} be the approximation of $y = f(x)$ computed in floating-point arithmetic with precision u . How do we measure the quality of \tilde{y} ? The most natural way to do so is to check that $E_{rel}(\tilde{y}) \approx u$. We will refer to it as the forward error. But this is not always achievable and then another analysis has been introduced by Wilkinson [Wilkinson, 1985] that consists in asking the question "for what problem did we actually solve the problem?". More formally we want to determine the smallest Δx such that $\tilde{y} = f(x + \Delta x)$. The value Δx is referred to as the backward error of \tilde{y} and bounding this quantity is called backward error analysis. A method is called backward stable if the value Δx is small enough with definition of small that depends on the context.

This analysis carries interesting properties. It considers the rounding errors as being equivalent to perturbations in the data. Indeed data can be flawed for many reasons from previous storing truncation to measurement errors. If the backward error is not larger than these uncertainties, the computed solution seems to be good enough. It also allows to reduce the error analysis to the perturbation theory that is well understood for a wide range of problems.

1.4.5 Conditioning

The conditioning of a problem governs the relationship between forward and backward error, meaning the sensibility of the result on the perturbation on data.

With the same $y = f(x)$ as before, and considering f is twice continuously differentiable, we have

$$\tilde{y} - y = f(x + \Delta x) - f(x) = f'(x)\Delta x + \frac{f''(x + \theta\Delta x)}{2!}(\Delta x)^2, \quad \theta \in (0, 1)$$

then

$$\frac{\tilde{y} - y}{y} = \left(\frac{xf'(x)}{f(x)} \right) \frac{\Delta x}{x} + O((\Delta x)^2)$$

and we obtain

$$c(x) = \left| \frac{xf'(x)}{f(x)} \right| \tag{1.8}$$

as the condition number of f .

When all quantities are well defined we have the following rule of thumb

$$\text{forward error} \lesssim \text{condition number} \times \text{backward error}$$

Which means that the computed solution with reduced backward error of an ill-conditioned problem can have a large forward error.

1.4.6 Examples of dramatic errors

Numerical errors might appear as a too exotic problem that should be left to the specialists and that users could cope with just by using double precision arithmetic. But this statement is not true as shown in [Ogita *et al.*, 2005] with the polynomial $P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$ evaluated at $x = 77617$ and $y = 33096$. Computed in single, double and quadruple precision, the result provides none correct digit and not even the correct sign.

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

The history of computing and engineering contains a few landmark episodes that should serve as a reminder to those who think that numerical accuracy is a secondary issue. We mention two examples¹ bellow.

- **The Patriot Missile Failure:** Patriot is the american surface-to-air missile system. It is used for destroying enemy missiles for which it must accurately compute their trajectory. But on February 25, 1991 during the Gulf War, a Patriot Missile missed an incoming Iraqi Scud missile leading to the death of 28 soldiers. It appeared lately that this error was due to a miscalculation of the time in seconds from the one in tenths of seconds provided by the system's internal clock. It was made with multiplication by $1/10$ which is not exactly representable in binary basis. Amplified by the 100 hours between the ignition of the missile and its launching, it led the missile to miss the Scud by almost half a kilometer.
- **The Ariane 5 rocket explosion:** On June 4, 1996 the Ariane 5 unmanned rocket exploded. The investigation that followed evidenced that it was due to an error in the inertial reference system computation and more precisely the conversion of a 64-bit floating-point number to a 16-bit signed integer. But the number was larger than 32767, the largest integer representable in a 16-bit signed integer and then the conversion failed.

1.5 Error control

As we saw in section 1.4, the use of floating-point arithmetic exposes us to errors that accumulate and can produce disasters. This is why various approaches have been developed to control these errors and their propagation.

1.5.1 Interval arithmetic

Interval arithmetic is not dealing with scalars but intervals. Its modern development started in the 1960s, with the work of R.E. Moore [Moore, 1966]. The aim is to manipulate closed, related subsets and operations that satisfy an inclusion property. Given two intervals U and V and a mathematical operation $\diamond \in \{+, -, \times, /, \dots\}$, the resultant interval $U \diamond V$ respects

$$\forall u \in U, \forall v \in V, u \diamond v \in U \diamond V \quad (1.9)$$

When implemented on floating-point arithmetic it consists on bounding each number by an interval that contains it and can be represented in computer arithmetic. This interval can alternatively be represented with its upper and lower bounds or with its center and radius. To ensure the inclusion property, the arithmetic operations have to

¹<https://www-users.cse.umn.edu/~arnold/disasters/>

be redefined. It offers an important reliability on the results but it faces severe limitations. Indeed, the obtained interval is sometimes too large and depends on the way the expression is written that may lead to overestimate the rounding error. This overestimation is due to multiple factors

- the use of floating-point arithmetic leads to wider intervals than those that we would obtain with real arithmetic
- correlated variables can be treated as decorrelated ones
- a wrapping effect occurs when the image of a multidimensional interval is not a multidimensional interval. This effect can be reduced by certain techniques as change of variables

Algorithms used in interval arithmetic therefore differ from classic ones. S. Rump has shown [Rump, 1999] that with the center-radius representation, interval arithmetic can be fully implemented using the Basic Linear Algebra Subprograms (BLAS) [BLAS Technical Forum, 2001].

It has been implemented in various libraries for Pascal, C or C++ ([Hammer, 1995], [Kulisch *et al.*, 1992], [Hofschuster & Krämer, 2004]) and Fortran languages [Kelch, 1993] and for tools as MAPLE [Grimmer, 2003] and MATLAB [Hargreaves, 2003].

1.5.2 Static methods

Interval methods present various problems, as we have seen above, leading to overestimation of rounding errors, particularly when variables are correlated. In response to this problem, so-called static analysis methods were developed [Goubault & Putot, 2006; Védrine *et al.*, 2021]. They allow us to take into account all the values that could be assumed by the variables and provide a rigorous analysis. They are not based on one or more executions of a program, but on an analysis of the code. However, they remain unsuitable for very large programs.

1.5.3 Probabilistic methods

Probabilistic approaches were therefore developed with a view to their application to large scale codes and they will be one of the main issues of this thesis.

Probabilistic methods rely on the use of a random rounding mode to estimate the errors produced. Indeed G. Forsythe [Bauer, 1974] noted in the 1950s that rounding errors do not behave as independent random variables. Various approaches have therefore been proposed to make rounding random. On processors complying with the IEEE-754 standard, it is possible to efficiently simulate a random rounding that will adopt a rounding towards $+\infty$ or towards $-\infty$ with probability $1/2$ thanks to symmetry properties.

1.5.4 The CESTAC method

The CESTAC (Contrôle et Estimation Stochastique des Arrondis de Calcul) [Vignes & Porte, 1974; Vignes, 1978] method is based on the idea that the result of each floating-point operation actually corresponds to two equally valid floating-point values, one corresponding to the upper rounding and the other to the lower rounding. A program performing n successive operations therefore generates a combinatorial of 2^n possible results R.

The invariant part of these results corresponds to the exact significant digits of a single execution. Obviously, calculating all the elements of R is unrealistic for a large n , the aim of the CESTAC method is then to calculate a subset that allows us to evaluate the number of exact decimals.

It is a probabilistic method that uses a random rounding mode. Let r be the exact result of computation and R the corresponding one computed using a floating-point arithmetic with t mantissa bits and random rounding mode. It has been shown that R can be approximated to the first order by

$$Z = r + \sum_{i=1}^n g_i(d)2^{-t}z_i \quad (1.10)$$

where r is the exact result, $g_i(d)$ are constants that only rely on the data and the algorithm and z_i are independent random variables uniformly distributed over $[-1, 1]$. This definition has two consequences:

- the expected value of Z is the exact result r
- under certain conditions, the distribution of Z is quasi-Gaussian and then Student law can be used on this distribution

If we neglect the second-order terms, and identify R with Z , we can use a Student's t test [Student, 1908] to estimate the number of exact significant figures in R . To do this, we need a sample of size N of results R_i , which enables us to obtain a confidence interval according to a given probability for the expectation r . Thus,

$$\forall p \in [0, 1], \exists \tau_p \in \mathbb{R} \quad \text{such that} \quad Pr(|\bar{R} - r| \leq \frac{\sigma \tau_p}{\sqrt{N}}) = 1 - p \quad (1.11)$$

where

$$\bar{R} = \frac{1}{N} \sum_{i=1}^N R_i \quad (1.12)$$

and

$$\sigma^2 = \frac{1}{N-1} \sum_{i=1}^N (R_i - \bar{R})^2 \quad (1.13)$$

τ_p is the confidence threshold of the Student distribution under $N-1$ degrees of freedom and probability $1-p$. In practice, for $p = 0.05$ and $N = 2$ or $N = 3$, we obtain $\tau_p = 12.706$ and $\tau_p = 4.303$ respectively.

$$C_{\bar{R}} = \log_{10} \left(\frac{\sqrt{N}|\bar{R}|}{\sigma \tau_p} \right) \quad (1.14)$$

In practice, to apply the CESTAC method to a program providing a result R , consists in repeating its execution N times, and calculating the bound $C_{\bar{R}}$ as defined above. These results are based on the assumptions that errors behave like centered random variables and that the first order approximation of R is valid. Even if the rounding is not actually centered it has been shown that a result R biased of a few σ s does not change the value of $C_{\bar{R}}$ of more than one, so the result remains valid within one.

In terms of the validity of the first order approximation, it is more complicated. Indeed, if addition and subtraction only produce first order terms, multiplication and divisions also produce second order terms. The approximation remains valid if these second order terms are negligible with respect to first order ones.

It has been shown that if each operand of a multiplication or the denominator of a division is non significant, i.e. its error is of the same order of magnitude as the value itself, the first order approximation is not valid anymore. The CESTAC method validation therefore requires a dynamic control of multiplications and divisions during the code execution. It implies a synchronous execution, meaning that the N executions are done in parallel and to the computational zero concept.

In a program using the CESTAC method, an intermediate or final result is a computational zero, denoted @.0, if and only if one of the following conditions is satisfied

- $\forall i, R_i = 0$,
- $C_{\bar{R}} \leq 0$

When a computational zero is detected the CESTAC method warns the user that the estimation of exact significant digits does not remain valid.

1.5.5 Continuous stochastic arithmetic

The continuous stochastic arithmetic [Vignes, 1993] is a modelisation of synchronous implementation of the CESTAC method with N executions. The N results of each arithmetic operation are then considered as realisations of a gaussian random variable centered on the exact result. We therefore define a new kind of numbers $X = (m, \sigma^2)$ where m is the expected value and σ its the standard variation. We also define a new arithmetic on these numbers: stochastic arithmetic.

Let $X_1 = (m_1, \sigma_1^2)$ and $X_2 = (m_2, \sigma_2^2)$ be two stochastic numbers, the stochastic arithmetic operations are defined as

$$X_1 \text{ s+ } X_2 = (m_1 + m_2, \sigma_1^2 + \sigma_2^2) \quad (1.15)$$

$$X_1 \text{ s- } X_2 = (m_1 - m_2, \sigma_1^2 + \sigma_2^2) \quad (1.16)$$

$$X_1 \text{ s}\times X_2 = (m_1 \times m_2, m_2^2 \sigma_1^2 + m_1^2 \sigma_2^2) \quad (1.17)$$

$$X_1 \text{ s/ } X_2 = \left(m_1/m_2, \left(\frac{\sigma_1}{m_2} \right)^2 + \left(\frac{m_1 \sigma_2}{m_2^2} \right)^2 \right) \quad \text{with } m_2 \neq 0 \quad (1.18)$$

For $X = (m, \sigma^2)$ there exists λ_β only relying on β such that

$$Pr(X \in I_{\beta,X}) = 1 - \beta \quad \text{with} \quad I_{\beta,X} = [m - \lambda_\beta \sigma, m + \lambda_\beta \sigma]$$

The lower bound on the number of digits common of all elements in $I_{\beta,X}$ is given by

$$C_{\beta,X} = \log_{10} \left(\frac{|m|}{\lambda_\beta \sigma} \right) \quad (1.19)$$

Continuous stochastic arithmetic defines the stochastic zero from the CESTAC definition of computational zero.

Definition 3 A stochastic number X , is a stochastic zero, denoted $\underline{0}$ if and only if

$$C_{\beta,X} \leq 0 \quad \text{or} \quad X = (0, 0)$$

This stochastic zero definition leads to the new order relation definition as follows

Definition 4 Let $X_1 = (m_1, \sigma_1^2)$ and $X_2 = (m_2, \sigma_2^2)$ two stochastic numbers, we define

- *stochastic equality* $s=$
 $X_1 s= X_2$ if and only if $X_1 s_+ X_2 = \underline{0}$
- *stochastic inequalities* $s>$ and $s\geq$
 $X_1 s> X_2$ if and only if $m_1 > m_2$ and $X_1 s\neq X_2$
 $X_1 s\geq X_2$ if and only if $m_1 \geq m_2$ or $X_1 s= X_2$

1.5.6 Discrete stochastic arithmetic

In discrete stochastic arithmetic (DSA), a real number x becomes a N -uplet X . Each operation is then performed on each element with random rounding. The number of exact significant digits of X can be estimated thanks to equation 1.14. Previously defined computational zero leads to the following equality and order relationship

Definition 5 Let X and Y be to N -uplets provided by the CESTAC method

- *stochastic equality* $ds=$
 $X ds= Y$ if and only if $X - Y = @.0$
- *stochastic inequalities* $ds>$ and $ds\geq$
 $X ds> Y$ if and only if $\bar{X} > \bar{Y}$ and $X ds\neq Y$
 $X ds\geq Y$ if and only if $\bar{X} \geq \bar{Y}$ or $X ds= Y$

These definitions offer a response to the lack of correlation in the ordering of exact results and computed results. Indeed, let A and B be two computed results and a and b the corresponding exacts values, we have

$$a > b \not\Rightarrow A > B \quad \text{and} \quad A > B \not\Rightarrow a > b$$

Using DSA allows to take into account the numerical quality of the operands and thus to control unstable branching.

Continuous and discrete stochastic arithmetic are correlated. Indeed, when the value N is small (2 or 3) which is the case in practice, the results provided by equations 1.19 and 1.14 are very close and represent the number of significant digits unaffected by rounding errors. Continuous stochastic arithmetic results therefore remain valid in the use of DSA.

The CADNA library provides an efficient implementation of the discrete stochastic arithmetic that allows to evaluate the number of significant digits in a result with few rewriting and to detect numerical instabilities.

1.5.7 Monte-Carlo arithmetic

Monte Carlo arithmetic [Parker & Langley, 1997; Parker *et al.*, 2000] is another probabilistic method to evaluate the accuracy of a result. As for the CESTAC method, it consists in modeling the results obtained numerically by random variables. For example, given x , the exact result of an operation, we set

$$\tilde{x} = x + \beta^{e-t}\xi \quad (1.20)$$

the inexact value of x with β the base in which x is represented, e its exponent, t is the desired precision, and ξ a random variable. ξ typically follows a uniform law on $(-1/2, 1/2)$ but other distributions can be useful. t determines the level of random perturbation applied to x and can be used to determine the minimum precision required to obtain a specified accuracy. By performing the same operation n times on the same operands, we can then estimate the exact value by taking the sample mean $\hat{\mu}$, measure the error associated with one value by calculating the standard deviation $\hat{\sigma}$ and evaluate the error of the estimated value $\hat{\mu}$ via $\hat{\sigma}/\sqrt{n}$.

An implementation of the Monte-Carlo arithmetic is provided by the tools MCALib [Frechtling & Leong, 2015] and Verificarlo [Denis *et al.*, 2016].

2 Benefits of mixed precision algorithms

2.1 Floating-point arithmetic context

Double precision, which today corresponds to the FP64 format, has long been and continues to be considered as a safe solution for scientific computing. The partially-founded belief in the ability of double precision to deliver reliable results made it the default choice for most applications in this field. However, the existence of a single-precision format is not new, and dates back to the beginnings of computing with Von Neumann's work in 1947 [von Neumann & Goldstine, 1947] and the publication of the Fortran 66 standard², which allows the use of both single and double formats. The appeal of single precision is clear, since it theoretically divides the computation times by two and reduces data storage space and transfer costs by the same amount. In the counterpart, in its traditional use, single precision only provides low accuracy and can therefore only be adopted for certain applications. The IEEE-754 standard published in 1985, which formalized single and double precision, also encourages their development. Even though, the hardware implementation of single precision did not provide automatic speedup and the comparative advantage of the FP32 format over FP64 on Intel chips only dates from the late 1990s and the development of Streaming SIMD Extension (SSE).

During this period, the acceleration in execution time was driven by Moore's two empirical laws, which respectively postulate a doubling of transistor complexity every year, and a doubling of transistors per silicon chip every two years. Thus, the search for performance gains through the use of single precision or techniques combining single and double precision may not have seemed crucial at the time, unlike the situation in recent years, which has seen a gradual disconnect between Moore's laws and observed reality. This can be explained by the physical limits reached in the miniaturization of

²<https://fortranwiki.org/fortran/show/FORTRAN+66>

transistors, and by the fact that data transfer capacities have fallen behind computing capacities. Against this backdrop, interest is growing in lower-precision transistors, also because they reduce memory transfer costs.

2.2 Rise of low precision

This phenomenon meets another, the one of the development of graphics processing units (GPUs). Initially developed as specialized processors for graphics rendering, they have gradually been adopted for computation, particularly in the context of neural networks for deep learning. These processors have a highly parallel architecture, which is not suitable for handling heterogeneous tasks as CPUs are. GPUs, on the other hand, are specialized for tasks that benefit from this parallelism, such as matrix operations. Under the influence of low-precision applications, half (FP16, BFLOAT16) and even lower-precision formats are developing strongly, with dedicated hardware delivering excellent performance. On A100 GPUs, for instance, FP16 or BFLOAT16 are 16 times faster than FP32 according to NVIDIA Ampere architecture whitepaper ³.

This performance is arousing interest in fields requiring both speed and high accuracy results, such as gamma-ray detectors. To combine these two needs, mixed precision approaches are being developed. The idea is to mix different precisions to benefit from their respective advantages.

2.3 The seek of mixing precisions

The high performance achieved by low precision, driven by high investment applications such as machine learning, is drawing the attention of other applications such as scientific computing. These require higher accuracy, but would like to benefit from the technological advances of the former. To this end, one approach gained ground in recent years: mixed precision. In the literature, we can find a number of different occurrences: multiprecision, transprecision, adaptive precision, variable precision, dynamic precision... The idea is always to use several precisions in the same computation to improve performance while maintaining a certain level of accuracy, but the approaches can vary.

In this thesis, we focus on two complementary approaches to the development of mixed precision. The first is to tune existing codes from the point of view of precision. This can be done automatically, known as autotuning, or in a guided way, as in our work. Autotuning is a particularly portable approach as its object is to develop tools that can be applied to any code without special understanding of what it does. On the contrary, it also exists another approach that is based on the maximum knowledge of the code. The key idea is to identify mixed precision opportunities and to propose specific mixed precision basic algorithms that can later be used in larger codes.

But first, why does it make sense to make the precision vary? Because not all the computations are equally important. Figure 1.6 presents what happens when we perform the summation of two FP64 numbers of different magnitudes: the smaller number b has a lot of unimportant bits that are thrown away during the summation. If b has

³<https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf>

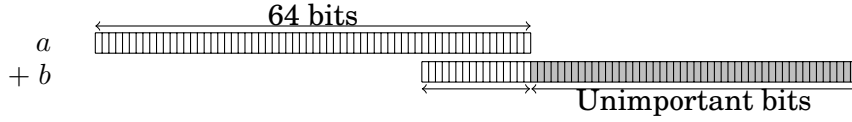


Figure 1.6: Example of different magnitude elements summation

previously been obtained by a computation, it is clear that this computation can be performed in a lower precision too. This means that reducing the precision of b does not affect the result of the sum of a and b .

2.3.1 Autotuning tools

Autotuning tools aim at automatically improve the performance (in terms of speed, memory and power consumption) by minimizing the precision of the variables used, while controlling the accuracy of the results. In order to do so, these tools modify the precision of the different variables and compare the result thus obtained with a reference result. But applied indiscriminately, this strategy requires a number of trials exponential in the number of variables. To address this problem, many of these tools use the Delta Debug algorithm, which we briefly describe below.

The Delta-Debug algorithm enables a code to be debugged automatically by considering an initial code C_\emptyset , a set of modifications $\Delta = (\Delta_1, \dots, \Delta_n)$ and the corresponding modified code C_Δ and an evaluation function `eval` that takes as input a code and that returns a boolean value. $\text{eval}(C_\emptyset) = 1$ and $\text{eval}(C_\Delta) = 0$.

The aim is to identify a minimal subset $\Delta_E \subseteq \Delta$ such that $\text{eval}(C_{\Delta_E}) = 0$.

To do so, we divide Δ into two subsets of similar size δ_1 and δ_2 and observe the behavior of the codes C_{δ_1} and C_{δ_2} obtained by their respective applications. Three types of behaviors can be observed.

- $\text{eval}(C_{\delta_1}) = 0$, so it contains a subset generating an error
- $\text{eval}(C_{\delta_2}) = 0$, so it contains a subset generating an error
- $\text{eval}(C_{\delta_1}) = 1$ and $\text{eval}(C_{\delta_2}) = 1$, so their intersection contains an error-generating subset.

In the first two cases, we simply search for the minimal subset generating an error via a recursive call. In the last case, we also recursively test the subsets that are intersections of δ_1 and δ_2 .

In the context of precision autotuning, this algorithm is used to test different combinations of accuracies for different variables, with the `eval` function applying a constraint on the accuracy of the result.

The PROMISE tool described in Figure 1.7 combines the stochastic arithmetic tool CADNA with the Delta-Debug algorithm. To do this, it first computes a reference result using CADNA, then it applies the Delta-Debug algorithm to determine a maximum subset of variables that can be passed in low precision while maintaining the requested accuracy.

Other autotuning tools can be found in the survey [Cherubin & Agosta, 2020]. Some are based on a static approach such as FPTaylor [Solovyev *et al.*, 2018] using symbolic

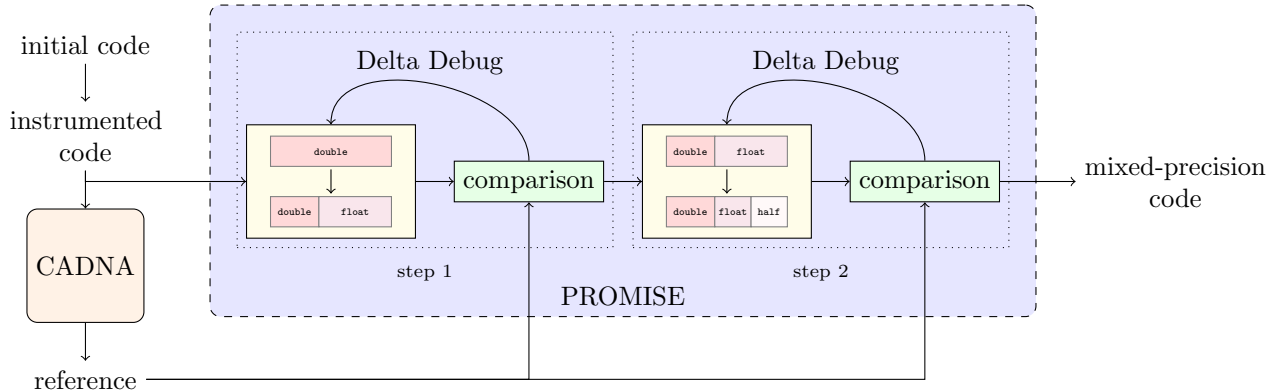


Figure 1.7: Promise dataflow

Taylor expansions, Salsa [Damouche & Martel, 2018] based on static analysis methods by abstract interpretation, TAFFO [Cattaneo *et al.*, 2022] based on LLVM technology and using programmer annotations, or POP [Ben Khalifa *et al.*, 2020] integrating a static forward and backward analysis. Others rely on a dynamic approach more suitable to process industrial codes such as CRAFT HPC [Lam *et al.*, 2013] that detects cancellations through binary instrumentation, Precimonious [Rubio-González *et al.*, 2013] that performs a search on the types of floating-point variables, HiFPTuner [Guo & Rubio-González, 2018] that exploits community structure of floating-point variables, and ADAPT [Menon *et al.*, 2018] that uses algorithmic differentiation.

There are also tools dedicated to GPUs that pay attention to cast such as AMPT-GA [Kotipalli *et al.*, 2019] that uses static analysis to help dynamic analysis, GPUMixer [Laguna *et al.*, 2019] and GRAM [Ho *et al.*, 2021] that assigns different groups of threads to different precision levels adaptively at runtime.

2.3.2 Algorithm-based mixed precision

The strength of automatic approaches is also their weakness: they cannot take advantage of the intrinsic opportunities offered by certain problems or algorithms to use different precisions. In contrast, an approach based on mixed precision linear algebra algorithms has seen significant development in recent years as witnessed by the survey [Higham & Mary, 2022]. It highlights the mixed precision approaches that have been developed to address various linear algebra problems. These range from matrix products to linear system solutions by direct methods or iterative refinement, QR factorization, least-squares problems and singular value decomposition.

In this thesis, we will focus more specifically on data-driven mixed precision algorithms, grouped together in the class of adaptive precision. The idea for these algorithms derives directly from the observation we made with the Figure 1.6 that if a and b are not of the same magnitude, they do not need to be stored with the same precision. But we can go further and assert that if b is itself the result of a computation, this computation can be carried out to a lower precision. Thus, we face the following issue: to what extent this distinction between variables and their treatment with different precisions can and should be made? Indeed, while element-by-element bucketization may seem the most appropriate way of providing fine-grained classification and assigning the most appro-

appropriate precision to each value, the extra cost generated by such processing may exceed the gain expected from the reduction in precision.

The choice of precision can be made at **matrix level** in calculations involving several matrices. For example, to calculate $C = A_1B_1 + A_2B_2$ with $|A_1| \geq |A_2|$ and $|B_1| \geq |B_2|$, if $|A_1||B_1| \gg |A_2||B_2|$, the matrix product $|A_2||B_2|$ can be computed in lower precision than $|A_1||B_1|$. This is particularly true in multiword arithmetic, where a matrix is represented by the unevaluated sum of matrices of lower precision, and the products realized by the sum of the products of these submatrices. [Fasi *et al.*, 2023] have shown that high accuracy is only required for the A_1B_1 term to avoid error accumulation.

It can also be done in the **column/row** level, storing each in a different precision. It is particularly adapted for matrices that can be decomposed as low-rank component of rapidly decreasing norm as it has been done with mixed precision truncated SVD approaches [Amestoy *et al.*, 2022; Ooi *et al.*, 2020].

In some cases it appears more interesting to adapt the precision at a **block level**. For example [Abdulah *et al.*, 2022b] and [Doucet *et al.*, 2019] propose to use a precision based on the distance of the block to the diagonal according to the fact that many sparse matrices blocks distant to the diagonal tend to have a smaller norm. Another example is the work realised to develop an adaptive precision block Jacobi preconditionner [Anzt *et al.*, 2019] and [Flegar *et al.*, 2021].

Finally, some strategies are even more aggressive and propose an **element level**. As they destroy the granularity of the computation, it shall be reserved for memory-bound applications such as sparse matrix-vector product. [Ahmad *et al.*, 2019] proposes to split A in two matrices A_d and A_s respectively stored in double and single precision. This idea is similar with the one of bucket summation [Demmel & Hida, 2004; Zhu & Hayes, 2010] however the goal is not to offer a better accuracy but to reduce the precisions used for each bucket. [Diffenderfer *et al.*, 2021] proposes a bucket algorithm for inner product that uses the IEEE prescribed precisions. We developed an algorithm for sparse matrix-vector product that can use an arbitrary number of precisions and targets a specific accuracy. This work is described in Chapter 3 and 4.

It is important to bear in mind that these precision optimizations are always matrix-dependent, and that while they may generate very significant gains in some cases, they may be less or even counter-productive in others.

3 Conclusion

As we have seen, representing and manipulating floating-point numbers involves issues of speed and accuracy, both of which being linked by the choice of precision. IEEE754 standardization, far from reducing technological possibilities, has made them possible by imposing common rules. The recent development of low-precision formats, massively used in neural networks in particular, has aroused interest in fields requiring higher accuracy, but still wishing to take advantage of these new possibilities. This is reflected in the interest in mixed precision, i.e. the mixing of different precision formats in the same code. The development of mixed precision has two facets. On the one hand, there is a sustained effort to develop algorithms specifically designed to take advantage of the use of multiple precisions. On the other hand, with the aim of “democratizing” mixed precision, tools are being developed to automatically insert different formats into existing

code while preserving the accuracy.

In this thesis, we get interest on both aspects. In the next chapter, we present an overview of nuclear physics, a field in which experimental and measurement instruments require both high accuracy and ever-increasing computational performance to detect increasingly rare phenomena.

The Advanced GAMMA Tracking Array (AGATA)

What is matter made of and how does it work? Modern physics provides essentially two answers depending on the scale observed: General Relativity in the infinitely large and quantum physics in the infinitely small. Connecting the two is a major objective. Experiments in nuclear and particle physics seek to identify particles and behaviors predicted by theory, or to disprove it by discovering contradictions. In this context, detectors play a role as important as the accelerators and colidors that produce the particles. In this thesis, we focus on gamma-ray detectors, and in particular on the AGATA detector, developed as part of an European cooperation project to develop a last generation detector, capable of extracting hitherto inaccessible information. Such an objective poses major technological challenges, including in numerical terms, since the amount of data to be accurately processed by the detector is very large.

1 Nuclear physics

1.1 Fundamental interactions

At the subatomic scale, we observe two kinds of particles, fermions and bosons whose behaviors are partially explained but which continue to be the focus of research. Their interactions are governed by four fundamental interactions.

- **Gravitation** is always attractive and depends on the mass of the interacting bodies. It was discovered at the end of the 17th century by Newton, then explained as a manifestation of the curvature of space-time by Albert Einstein in his theory of general relativity in 1915. It is the weakest of the fundamental interactions, but becomes dominant as the scale increases, and is therefore responsible for the large-scale structure of the Universe. It continues to be the focus of research aimed at linking this interaction to microscopic effects. Gravitation can be interpreted as an exchange of gravitons, although this particle has not yet been discovered experimentally.

- **Electromagnetism** can be attractive or repulsive, and depends on the electric charge of the particles. This interaction, which explains the existence of electromagnetic waves, was discovered by Maxwell in 1860. This interaction is associated with photons.
- the **weak nuclear interaction** is responsible for the β decay of subatomic particles. It was described in 1930 by Enrico Fermi and affects all known classes of fermions, and is caused by the exchange of bosons W^+ , W^- and Z^0 , which have very high masses (90 and 91 times that of the proton), which explains the very short range of this interaction.
- the **strong nuclear interaction** was discovered in the mid-twentieth century and is responsible for the cohesion of nucleons in the nucleus despite electromagnetism. It relies on a more fundamental interaction, the quantum chromodynamics. This one, discovered in the 1970s, acts on particles carrying a color charge: quarks, antiquarks and gluons, the latter themselves being carriers of the interaction. It acts in a short range by confining the quarks that compose the nucleons and allows protons and neutrons to form a nucleus.

In this thesis we are interested in nuclear physics that is on the scale of nuclei, smaller than molecules and atoms but bigger than fundamental particles quarks, our objects are of dimension around 10^{-15} m. We recall that atoms are made of a positively-charged nucleus around 10^{-14} m and a cloud of negatively-charged electrons. Nuclear physics studies the behavior of atomic nuclei and the particles with which they interact.

1.2 Nuclei behavior

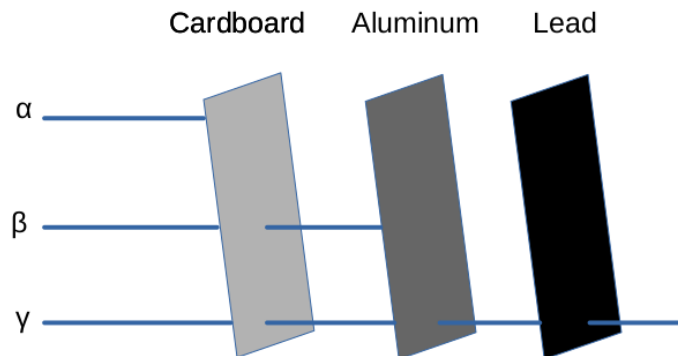
An atomic nucleus is made up of a number A of nucleons (including Z positively charged protons and N uncharged neutrons). They can interact with other nuclei, but also with elementary particles such as neutrons and electrons. Nuclear physics is concerned with nuclear structure, i.e. the proton-neutron interactions that form the nucleus, and with nuclear reactions, which describe how nuclei interact with each other and with elementary particles. In this thesis, we will be looking specifically at gamma radiation, which comes from nuclear decay or fission reactions.

1.3 Radioactive decay

A chemical element is determined by the number of protons Z of its nucleus, but it can exist in different forms called isotopes, corresponding to different numbers of neutrons N for the same number Z . Among these isotopes, only some are stable, i.e. they will not evolve without interaction with their environment. These stable isotopes correspond to nuclei with a close number of both protons and neutrons. But there are also many unstable isotopes. These isotopes are called unstable because they will eventually decay in some way. There exist seven different kinds of radioactive decays [Beiser, 2003] that are presented in Table 2.1. α particles correspond to ${}^4_2\text{He}$ nuclei while β^- and β^+ respectively correspond to electrons and positrons and γ is a gamma-ray photon. The different kind of decays correspond to various reasons of nucleus instability. In particular, gamma decay corresponds to an excess of energy that is released by the emission of gamma-ray

Table 2.1: Radioactive decays

Decay	Transformation	Reason of nucleus instability
Alpha decay	${}^A_Z X \rightarrow {}^{(A-4)}_{Z-2} Y + \alpha$	Too large
Beta ⁻ decay	${}^A_Z X \rightarrow {}^A_{Z+1} Y + \beta^- + \bar{\nu}$	Too many neutrons wrt protons
Beta ⁺ decay	${}^A_Z X \rightarrow {}^A_{Z-1} Y + \beta^+ + \nu$	Too many protons wrt neutrons
Electron capture	${}^A_Z X + e^- \rightarrow {}^A_{Z-1} Y + \nu$	Too many protons wrt neutrons
Gamma decay	${}^A_Z X^* \rightarrow {}^A_Z X + \gamma$	Excess of energy
Spontaneous fission	${}^A_Z X \rightarrow {}^A_Z Y + {}^A_Z T + \dots$	Too heavy isotope
Neutron emission	${}^A_Z X \rightarrow {}^{(A-1)}_Z X + n$	Too many neutrons wrt protons

Figure 2.1: Penetration of α , β and γ particles

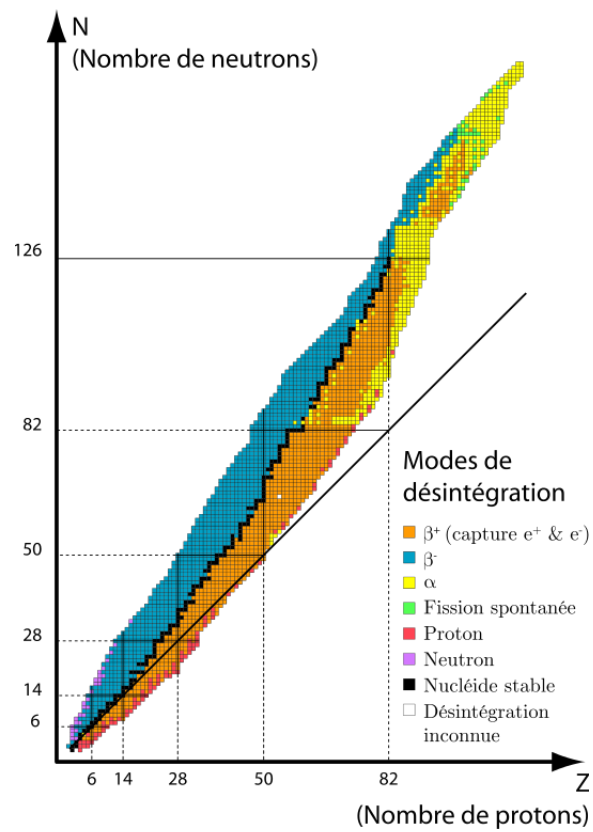


Figure 2.2: Chart of nuclides by type of decay. In black, the stable nuclides appear to form a valley, called the stability valley. ¹

photons. An excited nucleus, meaning with too much energy, is a situation that usually arise after alpha or beta decay and then gamma decay usually happens in a short time after one of the others. It can also follow nuclear reactions such as neutron capture, nuclear fission, or nuclear fusion. The distribution of stable and unstable isotopes forms a valley of stability described in Figure 2.2. We can observe that stable nuclei correspond to those close to $N = Z$ but with a slight superiority in the number of neutrons. The figure also highlights the fact that nuclides above this valley decay through β^- radioactivity (they transform their neutrons into protons), while those below decay through β^+ (they transform their protons into neutrons) and those beyond the valley reduce their neutrons number through α decay.

Radioactive decay of an atomic nucleus is a stochastic event that is unpredictable for a single nucleus. Nevertheless, for a significant number of the same nuclei it is possible to determine the time in which half of them will have decay. This measure is called half-life and is specific to each radioisotope. The shorter the half-life, the more the element is radioactive. Half-life can vary a lot from almost instantaneous to longer than the age of the universe. Extremely short lifetimes, make the elements very difficult to observe.

As presented in Figure 2.1, not all particles resulting from decay have the same penetration of matter. In particular, gamma-rays have a strong penetration of matter and they are ionizing radiation, meaning that they have sufficient energy to detach electrons from atoms and molecules. This property make them hazardous to life as they might cause DNA mutations, cancers and tumors. Gamma-rays correspond to the shortest wavelength of electromagnetic spectrum (around 1×10^{-11} m) and a range of energy from a few kiloelectronvolts (keV) to around 8 megaelectronvolts (MeV).

1.4 Gamma-ray spectroscopy

Despite their health risks, gamma-rays have industrial and medical applications, in particular to kill living organisms (for instance in the need of sterilization), to treat cancer by targetting the cancerous cells or in imaging techniques. For instance, the PET scan uses a radiolabeled sugar called fluorodeoxyglucose. This sugar emits positrons which are subsequently annihilated by electrons resulting in pairs of gamma-rays. As the cancer has a higher metabolic rate than the surrounding tissues, this allows to identify it.

Gamma-rays are also used in scientific applications as they provide information on the most energetic phenomena of the universe. In astronomy they are used to understand far and ancient events from the beginning of the Universe. They are also particularly useful to provide information on the isotopes from which they are issued. The features of incident gamma-rays are 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. As existing physics models can not be applied to every nuclei, in particular the most exotic ones with extreme proton to neutron ratios, studying these far-from-stable nuclei is a key to understanding the nuclear system.

To this end, Radioactive Ion Beam facilities have been set up all around the world. In particular FAIR (Darmstadt, Germany), HIE-ISOLDE (CERN, Geneva, Switzerland), SPIRAL2 (Caen, France), SPES (Legnaro, Italy), NSCL-MSU (East Lansing, USA),

¹<https://commons.wikimedia.org/wiki/File:DecayModeNuDat2.png>

RIKEN (Tokyo, Japan). These facilities are capable of producing a wide range of unstable nuclei, opening up new perspectives for nuclear physics experiments.

There exist different kinds of detectors. Scintillation detectors consist of a crystal coupled with a silicone oil light-couple. Scintillation is the process by which a material emits light when exposed to ionizing radiation. In practice, scintillation detectors, referred to as NaI(Tl) consist in a single crystal of sodium iodide doped by a small amount of thalium. It is coupled with a photomultiplier tube that converts the light into an electrical signal that can be treated by a computer. These detectors present certain advantages. Indeed, they are not too expensive, pretty easy to use and durable. On the other hand, they have a reduced resolution and they can not detect many small photo-peaks.

Another kind of detectors exist, they are based on semiconductors. A semiconductor is a material in which the valence band of electrons is close to the conduction band. Thus, when hit with gamma-rays, the energy imparted by the gamma-ray allows to promote electrons to the conduction band. This change of conductivity is detected and generates a signal. In the 1980s and 1990s, semiconductor detectors using germanium, denoted Ge(Li), have become the key to the study of nuclear structure. To get a better ratio of full-energy to the total of events recorded (called Peak-to-Total), the Germanium crystals are surrounded by a dense scintillator that records gamma-rays produced by Compton scatter or that escape from the crystal. Doing so, the electronics is able to reject the partial-energy pulse in the Ge detector. This translate in a significant improvement in the Peak-to-Total ratio. This technique is called Compton suppression [Riley & Simpson, 2014].

The development of efficient 4pi spectrometers using an escape-suppression technique led to the construction of the EUROBALL [Beck, 1992; Simpson, 1997] (Europe) and GAMMA-SPHERE [Lee, 1990] (USA) detectors, which have made a major contribution to progress in nuclear structure research. 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°C that are achieved with nitrogen cryostats. However, if the escape-suppression technique increases the Peak-to-Total ratio, it also reduces the solid angle occupied by the germanium detector and hence the efficiency of the gamma-ray detection system.

In order to cope with this limitation and to go further in the nuclear structure understanding, a new kind of detectors is development. They are called High Purity Germanium (HPGe) detectors and they are developed both in the USA, with GRETA and in Europe with the AGATA collaboration.

1.5 Online and offline computing

Detectors provide a large amount of data that have to be treated by computer capabilities. This processing is historically divided into two steps in high-energy and nuclear physics experiments.

The first step, called *online*, takes place immediately on leaving the detector, and generally involves electronic processing to sort the data and eliminate noise. This steps requires high computing performance to perform an efficient and correct sort because the volume of data is important and we have to be sure not to eliminate useful data. Today, this step is usually 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.

2 The AGATA experiment

AGATA² (Advanced GAMMA-ray Tracking Array) is a European research project [Akkoyun *et al.*, 2012] with the objective of building a HPGe detector with significantly higher resolving power than existing ones. It is a mobile instrument that aims at taking advantage of the wide range of facilities available in major European laboratories. It 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. This detector will be used for experiments with intense stable and radioactive ion beams, to study the nucleus structure.

It relies on two major technical advances:

- position sensitive Ge crystals
- tracking array technology

By electronically segmenting germanium crystals and using a Pulse Shape Analysis algorithm, i.e. by comparing the signals obtained with those of a calibration database, we can precisely identify the points of interaction of a gamma-ray, together with the associated energies and timestamps. This information can then be used to reconstruct the complete path of a gamma-ray through the detector. The position sensitive Ge crystals are obtained through highly electrically segmented Ge detectors. This allows the identification of the individual points of interaction of the gamma-rays within the volume of the Ge crystals and the deposited energy with high resolution.

This technique eliminates the need for Compton-suppression shields, increasing efficiency while maintaining spectral quality. Furthermore, the direction of emission of each gamma-ray can be accurately assessed, which is necessary to perform a good Doppler energy correction and thus obtain a more precise measurement of the energy of a gamma-ray emitted by a very fast nucleus, as it is the case in most nuclear reactions.

2.1 The geometry of the detector

The geometry of the detector was the subject of advanced reflection using Monte-Carlo simulations with a simulation code based on GEANT4 [Farnea *et al.*, 2010], leading to the use of 180 hexagones and 12 pentagones. The 180 hexagones are filled by Germanium crystals grouped into 60 Agata Triple Clusters (ATC). The participation of the 12 pentagones to the general resolution is considered sufficiently low to avoid the cost of development and then their size is minimized and they are used for mechanical support,

²<https://www.agata.org>

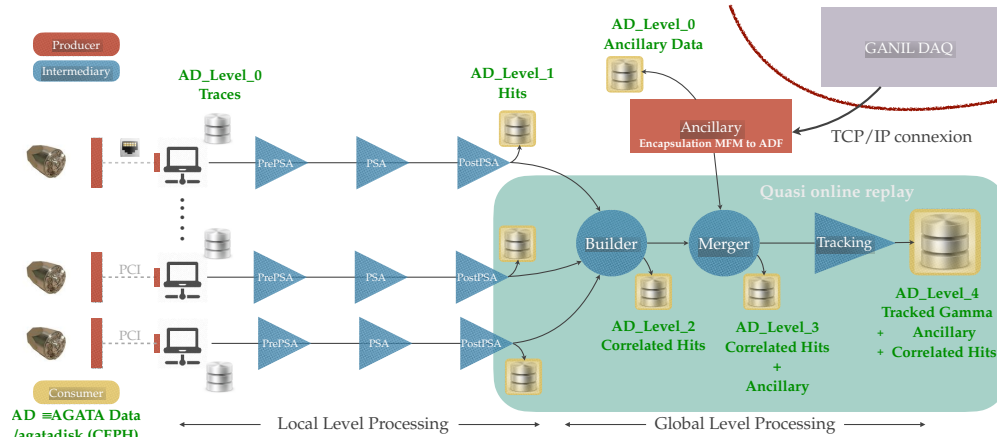


Figure 2.3: Agata data processing

insertion of complementary detectors, beam entry and exit pipes. The inner radius will be 23.5 cm.

This geometry allows to maximise the solid angle coverage to 82% and the full energy efficiency to 43% (28%) and 59% (43%) peak to total ratio for photon multiplicities of 1 and 30 respectively.

The use of hexagonal crystals leads quite naturally to azimuthal segmentation in six sectors. For longitudinal segmentation, the effective volume of the segments must be balanced, and they must be wide enough to achieve a sufficiently fine positioning resolution of 5mm. Each Germanium detector of AGATA is then segmented into 36 segments (6 slices along the axis times 6 sectors around the axis).

2.2 AGATA data processing

Figure 2.3 describes the data processing between the signals captured by the germanium and the reconstruction of the gamma-ray path. Each step is briefly explained below.

2.2.1 Detection

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. When a gamma-ray interacts with germanium, it produces an electrical signal in the hit segment, called the netcharge segment, but also in its neighbors, and on the central contact (core) meaning a total of 37 signals per crystal.

All signal channels are equipped with a cold preamplifier that operates close to the liquid nitrogen temperature of the cryostat to reduce the noise contribution. Preamplifiers have to provide good energy and timing resolution but also clean transfer functions to register unperturbed signal traces to be used to identify the interaction point in the Pulse-Shape-Analysis (PSA) step. The high electronic integration density implies some crosstalk contributions, typically in the order of 0.1%. Its effect is taken into account to correct the measured signals.

Analog signals are then converted into digital signals using an analog-to-digital converter. This conversion is performed at a frequency of 100 MHz and produces 14-bit integers [Akkoyun *et al.*, 2012]. 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 to be computationally processed.

2.2.2 Data processing

The processing of data relies on a series of actors running as individual processes on a network of computer nodes.

The first actor is the crystal producer that reads the measurements and casts them into FP32, which generates an increase in storage size, and passes it on to the preprocessing actor. This preprocessing actor defines thresholds and corrects signal noise to produce energy-calibrated and time-aligned data.

This is followed by the Pulse Shape Analysis (PSA) described in more details in Section 3. In this step the measured signals 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.

2.2.3 Gamma-ray tracking

Tracking algorithms are used to reconstruct the trajectories of incident photons to determine their energy and direction. To do this, it is necessary to separate the interaction points identified by the PSA in the different segments and detectors to reconstruct the correct interaction sequences.

There are two classes of tracking algorithms: forward tracking algorithms [Deleplanque *et al.*, 1999] and backward tracking algorithms [van der Marel & Cederwall, 1999]. All these algorithms rely on the interaction properties of photons with matter. For forward tracking, interaction points are first grouped into spatial clusters. For backward tracking, we use the fact that the last photoelectric interaction generally falls within a narrow energy band. Starting from this last point, we trace back to the first, using the physical characteristics of the interactions and choosing the most probable interaction pattern.

At the energies of interest to us, i.e. between 10keV and 20MeV, the main physical processes are Compton scattering, Rayleigh scattering, pair creation and photoelectric

interaction. Compton scattering is dominant between 150keV and 10MeV, so all tracking algorithms should be highly efficient in this domain.

3 The pulse-shape analysis of AGATA

A gamma-ray will normally interact several times with germanium. The aim of the PSA is to precisely identify the location of these interactions and their associated energies. These different interactions may take place in the same segment, but also in different segments, or even in an adjacent detector or outside the shell. The PSA needs to provide interaction localization accuracy greater than 5mm to perform the tracking process with high efficiency. Interaction points and their associated energies must be identified very quickly. Indeed, the flow of information to be processed is far greater than what can be stored, both in terms of storage capacity and recording speed. The PSA therefore enables online reduction to the parameters of interest: position, energy and timestamp. To achieve this, the measured signals are compared with a database of signals for which the location of the interaction is known.

Several algorithms have been developed, including a gridsearch [Venturelli & Bazzacco, 2005; Schlarb, 2009], genetic algorithms, a wavelet decomposition and a matrix method [Olariu, 2007; Olariu *et al.*, 2006], optimized for different event types. The AGATA partnership expects a continuous development of the PSA performance as reviewed in [Boston, A. J. *et al.*, 2023]. Currently, the gridsearch algorithm is preferred for its efficiency, simplicity and robustness. It is implemented in the Narval environment and it is to this algorithm that we study in this thesis.

To correctly compare the experimental signals with those of the base, it is necessary to perform an energy calibration, which is achieved by the front-end electronics, but also to take into account the influence of noise, pedestal, time adjustment and cross-talk effects.

3.1 AGATA Data Library

To identify points of interaction, the measured signals in the net-charge segment, its neighbours and the core are compared with reference signals previously obtained that must have a 2 mm resolution. This reference basis is named the AGATA Data Library (ADL) and can be obtained by two different ways: simulation and crystal calibration. In both cases, we seek to associate an interaction point in the crystal with signals measured or simulated in the net-charge segment, its neighbors and the core. In all cases, we obtain a grid of discretized crystal points and the associated signal values.

To obtain the signal basis experimentally, we have to use calibration tables. They are used to target a specific point in the crystal and measure the associated signals in the various segments.

But this calibration stage is extremely time-consuming, so simulation is preferred. For the simulation to be good enough to enable a correct PSA, the input elements must be precisely known: crosstalk effects, crystal impurities and the response functions and space charge distributions of the detector. Using Poisson equations, we can then calculate the signal basis.

3.2 Gridsearch algorithm

3.2.1 Full gridsearch

The full gridsearch algorithm (FGS), Algorithm 1 is based on the minimization of a Figure of Merit calculated for each grid point. The crystal has been discretized, and for each point we have the signals measured in the hit segment, in neighboring segments and in the core over a certain number of time steps. We can then calculate which point in the hit segment has the lowest value, indicating that we have found the point closest to the interaction. This Figure of Merit is as follows

$$\text{FOM}_i = \sum_{s,t} (V_m[s][t] - V_r[i][s][t])^p \quad (2.1)$$

Where V_m and V_r denote the measured and reference signals respectively, and s and t iterate over the different segments and time steps respectively. The aim is to determine the value of i for which FOM_i is minimal, and therefore the corresponding point.

Algorithm 1 Pulse-Shape Analysis - Full gridsearch algorithm

```

1: Input: Measured signal  $V_m$  and reference signals  $V_r$ 
2: Output: Interaction point
3: MINFOM = MAXFLOAT
4: INDFOM = MAXINT
5: for  $i = 1 : NPTS$  do
6:   FOM $_i$  = 0
7:   for  $s = 1 : NSEGS$  do
8:     for  $t = 1 : NTIME$  do
9:       FOM $_i$  +=  $(V_m[s][t] - V_r[i][s][t])^p$ 
10:    end for
11:  end for
12:  if FOM $_i$  < MINFOM then
13:    MINFOM = FOM $_i$ 
14:    INDFOM =  $i$ 
15:  end if
16: end for
17: RETURN INDFOM

```

3.2.2 Coarse-Fine gridsearch

The current variant of the algorithm, called coarse-fine gridsearch (CFGs), Algorithm 2, is an optimized version of the FGS. Instead of measuring the Figure of Merit for all grid points, a first search for the minimum is performed on a coarse subset of grid points with a resolution of 6 mm. Then a new search is performed around this coarse point to find the true minimum on a grid with a resolution of 2 mm. If the energy of the interaction is below a threshold, only the coarse grid is searched. Indeed, for these interactions, the resolution does not reach that of the fine grid.

Algorithm 2 Pulse-Shape Analysis - Coarse-fine gridsearch algorithm

```

1: Input: Measured signal  $V_m$  and reference signals  $V_r$ , coarse grid CGRID and fine
   grid FGRID
2: Output: Interaction point
3: MINCOARSE = MAXFLOAT
4: INDCOARSE = MAXINT
5: for  $k = 1 : NCOARSE$  do
6:    $i = CGRID[k]$ 
7:   FOM $i$  = 0
8:   for  $s = 1 : NSEGS$  do
9:     for  $t = 1 : NTIME$  do
10:      FOM $i$  +=  $(V_m[s][t] - V_r[i][s][t])^p$ 
11:     end for
12:   end for
13:   if FOM $i$  < MINFOM then
14:     MINCOARSE = FOM $i$ 
15:     INDCOARSE =  $i$ 
16:   end if
17: end for
18: MINFOM = MINCOARSE
19: INDFOM = INDCOARSE
20: for  $k = 1 : NFINE$  do
21:    $i = FGRID[INDCOARSE][k]$ 
22:   FOM $i$  = 0
23:   for  $s = 1 : NSEGS$  do
24:     for  $t = 1 : NTIME$  do
25:      FOM $i$  +=  $(V_m[s][t] - V_r[i][s][t])^p$ 
26:     end for
27:   end for
28:   if FOM $i$  < MINFOM then
29:     MINFOM = FOM $i$ 
30:     INDFOM =  $i$ 
31:   end if
32: end for
33: RETURN INDFOM

```

3.3 Metric selection

The choice of the metric used in the FOM is important to accurately identify the point of interaction. This choice depends on the distribution of the differences between the basis signals and the measured signal, and also on the weighting chosen between the net signal and the transient signal [Lewandowski *et al.*, 2019].

A Gaussian distribution is expected on the $V_m[s][t] - V_r[i][s][t]$ difference, since the basis signals generally over- or under-estimate the real signal, mainly due to noise. The amplitude of the measured signal varies around its expectation due to the statistical nature of the charge collection. If a non-Gaussian distribution is observed, we can deduce a systematic deviation between the measured signals and those of the basis.

In practice, the choice of the metric used corresponds to a variation in the value of p , which is a positive real number. The usual distance corresponds to a Euclidean metric and is obtained with a value of $p = 2$. A very high value of p implies a high weight given to strong deviations between the measured signal and the basis signals, even if these deviations are few in number. This reduces the impact of noise, but also means that less weight is given to transient signals, which are very important in identifying the point of interaction. Lower p values do not give as much weight to these important but rare deviations. They do, however, require good overall matching, including for transient signals.

It is difficult to determine an optimal p -value by analyzing the distribution of amplitude differences. This choice depends not only on the energies observed, but also on the multiplicity of interactions. So far, the choice of p has been determined empirically. Various experiments have indicated that a value of $p = 0.3$ was optimal [Recchia, 2008] and we have retained this value in our experiments.

3.4 Multiple interactions

Several interactions can occur in the detector at the same time, adding complexity to the identification of interaction points. While this is not a problem if these interactions occur in different crystals, the situation is different if they take place in the same crystal or even in the same segment. In the latter case, the gridsearch algorithm treats the multiple interactions as a single one by determining the barycenter. In the case of interactions taking place in different segments of the same crystal, two situations need to be distinguished. Either the transient signals overlap and the gridsearch does not allow correct identification or the signals do not overlap and the algorithm remains valid. [Akkoyun *et al.*, 2012]

3.5 A time consuming step

The PSA operation is both crucial in the AGATA processing chain, because it determines the ability to identify precisely the gamma-ray trajectories, and very costly in terms of execution time and memory.

The signal database contains information corresponding to each discretized point on the crystal, i.e. 47292 in our case. For each of these points, we have the coordinates of the interaction point, the associated $T0$, the energy fraction it represents, the core trigger, the energy collected in each segment and the core, and, much more expensive, the traces

measured over 120 5 ns time steps in each segment and the core, including the first 10 empty traces (corresponding to a 50 ns pretrigger). The floats are stored in FP32 and the integer in 4 bits, so we obtain 17932 bytes per point in the base, i.e. a total of more than 848 Mb for the whole base.

To identify the point of interaction, measurements are taken on the 36 segments and the two cores (high gain and low gain) with an accuracy of 10ms per time step. Each trace is stored as a 2-byte integer. At PSA level signal is reduced to 60 samples 4 bytes floats and we only use the traces measured in the neighboring segments. However, all the data remains necessary to process several events and/or perform time adjustment i.e. determine the most appropriate T_0 .

Finally, the key operation at the heart of the nested loop of points, segments and time steps 2.1 is not trivial and remains quite costly due to its use of mathematical functions.

PSA is therefore an intrinsically costly operation, which it is legitimate to seek to speed up by various means, notably by seeking to reduce the volume of data processed, in particular by reducing the precision formats used.

4 Conclusion

As we have seen, progress in physics since the 19th century have led to significant advances in our understanding of the particles that make up matter. However, there are still shadows and contradictions in the models proposed. To resolve these contradictions and further improve our knowledge, we need to develop experiments that require major technological advances. The European AGATA project is part of this approach, with the aim of building a state-of-the-art, gamma-ray detector using High Purity Germanium (HPGe) crystals. This detector will be mobile, making it possible to take advantage of the various European radioactive and stable ion beam facilities, and to achieve unprecedented precision in the interactions observed. To achieve this, the detector must be able to identify the path of each gamma-ray in the germanium, and therefore the points of interaction within it, and the energies released during these interactions. This is obtained by a step called Pulse-Shape-Analysis, which compares the measured signals with a database of signals for which the point of interaction is known. This step must be performed live, or online according to dedicated terminology, but it must also be performed with accuracy. During this thesis, our aim was therefore to speed up the computation by adopting a data size perspective, while preserving the accuracy of the results.

36 segments + 2 cores (high gain and low gain) precision is 10ms per time step. 2 bytes per bin for raw traces. At PSA level signals reduced to 60 samples 4 bytes per bin (float)

Adaptive SpMV and application to Krylov solvers

We introduce in this chapter a mixed precision algorithm for computing sparse matrix-vector products and use it to accelerate the solution of sparse linear systems by iterative methods. Our approach is based on the idea of adapting the precision of each matrix element to its magnitude: we split the elements into buckets and use progressively lower precisions for the buckets of progressively smaller elements. We carry out a rounding error analysis of this algorithm that provides us with an explicit rule to decide which element goes into which bucket and allows us to rigorously control the accuracy of the algorithm. We implement the algorithm on a multicore computer and obtain significant speedups (up to a factor $7\times$) with respect to uniform precision algorithms, without loss of accuracy, on a range of sparse matrices from real-life applications. We showcase the effectiveness of our algorithm by plugging it into various Krylov solvers for sparse linear systems and observe that the convergence of the solution is essentially unaffected by the use of adaptive precision.

1 Introduction

Motivated by the growing availability of lower precision arithmetics, mixed precision algorithms are being developed for a wide range of numerical computations [Higham & Mary, 2022]. One subclass of mixed precision algorithms that has recently and increasingly proven successful is what we call adaptive precision algorithms. These algorithms are based on the idea of adapting the precision to the data involved in the computation, by selecting a level of precision proportional to the importance of the data, where the definition of “importance” is application dependent. For example, Anzt et al. [Anzt et al., 2019], [Flegar et al., 2021] have proposed an adaptive precision block Jacobi preconditioner in which the precision of each block is chosen based on its condition number. Another example is the mixed precision low-rank compression proposed by Amestoy et

al. [Amestoy *et al.*, 2022], which partitions a low-rank matrix into several low-rank components of decreasing norm and stores each of them in a correspondingly decreasing precision. Ahmad *et al.* [Ahmad *et al.*, 2019] develop a sparse matrix–vector product algorithm in which elements in the range $[-1, 1]$ are switched to single precision while the other elements are kept in double precision. Diffenderfer *et al.* [Diffenderfer *et al.*, 2021] propose a “quantized” dot product algorithm that adapts the precision of each vector element based on its exponent. For a unified presentation of these adaptive precision algorithms, see [Higham & Mary, 2022, sect. 14].

In this chapter, we propose an adaptive precision algorithm at the element level for matrix–vector products. Specifically, our matrix–vector product algorithm partitions the elements into several buckets and uses a different precision for each bucket. We perform a rounding error analysis of this algorithm that reveals how the precisions should be chosen: we prove that it suffices to take the precisions to be proportional to the magnitude of the elements, that is, elements of large magnitude should be kept in high precision, but elements of smaller magnitude can be switched to correspondingly lower precisions. Intuitively, this discovery can be explained by the fact that the least significant bits of the smaller elements end up being lost when they are summed to the larger elements: hence, we might as well avoid computing those bits to begin with.

Based on this analysis, we develop an adaptive precision sparse matrix–vector product and evaluate experimentally its performance and accuracy on a range of real-life large sparse matrices. We show that the storage and hence the data movement costs of the product can be significantly reduced for many matrices, while preserving a user-prescribed accuracy target. We develop an implementation for CPUs that uses double and single precision arithmetic as well as dropping (discarding sufficiently small elements), and obtain speedups of up to an order of magnitude on a multicore computer. We then apply our algorithm to the solution of sparse linear systems by plugging it into various Krylov solvers with iterative refinement. Our experiments demonstrate that the convergence of the solution is essentially unaffected by the use of adaptive precision.

The rest of this chapter is organized as follows. We begin by recalling the error analysis of the standard matrix–vector product in uniform precision in section 2. Then, we propose in section 3 an adaptive precision matrix–vector product algorithm and carry out its error analysis. In section 4, we investigate experimentally both its accuracy and performance. In section 5 we apply this algorithm to the solution of linear systems with Krylov solvers. Finally, we provide our concluding remarks in section 6.

2 Uniform precision matrix–vector product

Before proposing an adaptive precision matrix–vector product, let us recall the error analysis of the uniform precision case, where the same precision is used across all operations.

Throughout this work we use the standard model of floating-point arithmetic [Higham, 2002, sec. 2.2]

$$\text{fl}(a \text{ op } b) = (a \text{ op } b)(1 + \delta), \quad |\delta| \leq u, \quad \text{op} \in \{+, -, \times, /\}, \quad (3.1)$$

where u is the unit roundoff of the precision used and fl represents the computed results in floating-point arithmetic.

Let $y_i = \sum_{j \in J_i} a_{ij}x_j$ be the inner product between the i th row of A and x , where J_i is the set of the column indices of the nonzero elements in row i of A . In uniform precision u , the computed \hat{y}_i satisfies

$$|\hat{y}_i - y_i| \leq \#J_i u \sum_{j \in J_i} |a_{ij}x_j|, \quad (3.2)$$

where $\#J_i$ denotes the cardinality of J_i . Note that here, and throughout this chapter, we have used the analysis of inner products of Jeannerod and Rump [Jeannerod & Rump, 2013] to obtain more refined bounds, where constants of the form $\gamma_n = nu/(1 - nu)$ can be replaced simply by nu . The analysis of [Jeannerod & Rump, 2013] assumes the use of rounding to nearest, but it was later shown in [Lange & Rump, 2017, Corollary 3.3] that these refined bounds also hold for directed roundings by replacing u with $2u$. We also note that constants n could be further reduced to \sqrt{n} to obtain probabilistic bounds that hold with high probability [Higham & Mary, 2019, 2020; Connolly *et al.*, 2021]. In this work the size of the constants is not the main focus (as they are typically small for sparse matrices), and so we use the more general worst-case error bounds.

Algorithm 3 Uniform precision matrix–vector product.

- 1: **Input:** $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$. J_i is the set of column indices of the nonzeros in row i of A .
 - 2: **Output:** $y = Ax$
 - 3: **for** $i = 1 : m$ **do**
 - 4: $y_i = 0$
 - 5: **for** $j \in J_i$ **do**
 - 6: $y_i \leftarrow y_i + a_{ij}x_j$
 - 7: **end for**
 - 8: **end for**
-

As a consequence of the Oettli–Prager [Higham, 2002, Thm. 7.3], [Oettli & Prager, 1964] and Rigal–Gaches [Higham, 2002, Thm. 7.1], [Rigal & Gaches, 1967] theorems, we have the following formulas for the componentwise backward error

$$\varepsilon_{\text{cw}} = \min \{ \varepsilon : \hat{y} = (A + \Delta A)x, |\Delta A| \leq \varepsilon |A| \} = \max_i \left[\frac{|\hat{y}_i - y_i|}{\sum_{j \in J_i} |a_{ij}x_j|} \right] \quad (3.3)$$

and for the normwise backward error

$$\varepsilon_{\text{nw}} = \min \{ \varepsilon : \hat{y} = (A + \Delta A)x, \|\Delta A\| \leq \varepsilon \|A\| \} = \frac{\|\hat{y} - y\|}{\|A\| \|x\|}, \quad (3.4)$$

respectively. Throughout this chapter, the unsubscripted norm $\|\cdot\|$ denotes the infinity norm

$$\|A\|_{\infty} = \max_i \sum_j |a_{ij}|.$$

Note that the componentwise error is always larger than the normwise one, since we have

$$\varepsilon_{\text{nw}} = \frac{\|\hat{y} - y\|}{\|A\| \|x\|} \leq \frac{\|\hat{y} - y\|}{\|A\| \|x\|} = \frac{\max_i |\hat{y}_i - y_i|}{\max_i (|A\| \|x\|)_i} \leq \max_i \frac{|\hat{y}_i - y_i|}{(|A\| \|x\|)_i} = \varepsilon_{\text{cw}}. \quad (3.5)$$

Moreover, using (3.2), we obtain the bound

$$\varepsilon_{\text{nw}} \leq \varepsilon_{\text{cw}} \leq pu, \quad (3.6)$$

where $p = \max_i \#J_i$ is the maximum number of nonzero elements per row of A .

3 Adaptive precision matrix–vector product: error analysis

In this section we propose an adaptive precision matrix–vector product algorithm. To do so we perform the error analysis of a general mixed precision matrix–vector product that partitions the nonzero elements of the matrix into buckets and computes the partial inner products associated with each bucket in a different precision. Our analysis shows how to build these buckets so as to minimize the precisions used while achieving a prescribed backward error.

We analyze Algorithm 4, which computes a mixed precision matrix–vector product $y = Ax$ using q precisions $u_1 < u_2 < \dots < u_q$. Each row i of A is partitioned into q buckets $B_{ik} \subset \llbracket 1, n \rrbracket$, $k = 1: q$, and the inner product $y_i^{(k)} = \sum_{j \in B_{ik}} a_{ij}x_j$ associated with bucket B_{ik} is computed in precision u_k . All the partial inner products are then summed in precision u_1 .

For Algorithm 4 to be well defined, we require that the B_{ik} form a partition of J_i (the nonzero elements in row i of A), that is, that they are disjoint and that their union is equal to J_i .

Algorithm 4 Adaptive precision matrix–vector product in q precisions $u_1 < \dots < u_q$.

```

1: Input:  $A \in \mathbb{R}^{m \times n}$ ,  $x \in \mathbb{R}^n$ , a partitioning of  $A$  into buckets  $B_{ik}$ 
2: Output:  $y = Ax$ 
3: for  $i = 1: m$  do
4:   for  $k = 1: q$  do
5:      $y_i^{(k)} = 0$ 
6:     for  $j \in B_{ik}$  do
7:        $y_i^{(k)} \leftarrow y_i^{(k)} + a_{ij}x_j$  in precision  $u_k$ 
8:     end for
9:   end for
10:   $y_i = \sum_{k=1}^q y_i^{(k)}$  in precision  $u_1$ 
11: end for

```

According to (3.2) the computed partial inner product $\widehat{y}_i^{(k)}$ satisfies

$$|\widehat{y}_i^{(k)} - y_i^{(k)}| \leq p_{ik}u_k(1 + u_k)^2 \sum_{j \in B_{ik}} |a_{ij}x_j|, \quad (3.7)$$

where $p_{ik} = \#B_{ik}$ and where the $(1 + u_k)^2$ term accounts for the need to first convert both a_{ij} and x_j to precision u_k . Then, defining $\bar{y}_i = \sum_{k=1}^q \widehat{y}_i^{(k)}$ as the exact sum of the $\widehat{y}_i^{(k)}$, and as $y_i = \sum_{k=1}^q y_i^{(k)}$, we have

$$|\bar{y}_i - y_i| \leq \sum_{k=1}^q \left[p_{ik}u_k(1 + u_k)^2 \sum_{j \in B_{ik}} |a_{ij}x_j| \right], \quad (3.8)$$

and the computed \widehat{y}_i satisfies

$$|\widehat{y}_i - \bar{y}_i| \leq (q-1)u_1 \sum_{k=1}^q |\widehat{y}_i^{(k)}| \quad (3.9)$$

$$\leq (q-1)u_1 \sum_{k=1}^q \left[(1 + p_{ik}u_k(1+u_k)^2) \sum_{j \in B_{ik}} |a_{ij}x_j| \right], \quad (3.10)$$

where the conversion of $\widehat{y}_i^{(k)}$ back to precision u_1 does not introduce any error since $u_1 \leq u_k$ for all k . Using the fact that the B_{ik} form a partition of J_i , we have that

$$\sum_{k=1}^q \sum_{j \in B_{ik}} |a_{ij}x_j| = \sum_{j \in J_i} |a_{ij}x_j|$$

and we therefore obtain

$$|\widehat{y}_i - y_i| \leq |\widehat{y}_i - \bar{y}_i| + |\bar{y}_i - y_i| \quad (3.11)$$

$$\leq (q-1)u_1 \sum_{j \in J_i} |a_{ij}x_j| + (1 + (q-1)u_1) \sum_{k=1}^q \left[p_{ik}u_k(1+u_k)^2 \sum_{j \in B_{ik}} |a_{ij}x_j| \right]. \quad (3.12)$$

Dividing both sides by $\sum_{j \in J_i} |a_{ij}x_j|$, we obtain the componentwise backward error bound

$$\varepsilon_{\text{cw}} \leq (q-1)u_1 + (1 + (q-1)u_1) \max_i \left[\sum_{k=1}^q p_{ik}u_k(1+u_k)^2 \alpha_{ik} \right], \quad (3.13)$$

which shows that the ratios

$$\alpha_{ik} = \frac{\sum_{j \in B_{ik}} |a_{ij}x_j|}{\sum_{j \in J_i} |a_{ij}x_j|} \quad (3.14)$$

play a fundamental role in controlling the size of the backward error.

Now we want to determine how to build the buckets B_{ik} such that the backward error is at most in $O(\epsilon)$, where $\epsilon \geq u_1$ is a user-prescribed target accuracy. The analysis above shows that to do so, we need to control the ratios α_{ik} , which are essentially a measure of how large the elements in bucket B_{ik} are with respect to all the elements in J_i . Thus, the analysis tells us that elements smaller in magnitude can be placed in lower precision buckets. Specifically, writing a_i the i th row of A so that $\sum_{j \in J_i} |a_{ij}x_j| = |a_i|^T|x|$, let us define the intervals

$$P_{ik} = \begin{cases} (\epsilon|a_i|^T|x|/u_2, +\infty) & \text{for } k = 1, \\ (\epsilon|a_i|^T|x|/u_{k+1}, \epsilon|a_i|^T|x|/u_k] & \text{for } k = 2: q-1, \\ [0, \epsilon|a_i|^T|x|/u_q] & \text{for } k = q, \end{cases} \quad (3.15)$$

which form a partition of $[0, +\infty)$, and let us define the buckets B_{ik} as the column indices of the nonzero elements of A such that $|a_{ij}x_j|$ belongs to the corresponding interval P_{ik} :

$$B_{ik} = \{j \in J_i : |a_{ij}x_j| \in P_{ik}\}. \quad (3.16)$$

The definition of the P_{ik} intervals is illustrated with four precisions in Figure 3.1. This construction yields $\alpha_{ik} \leq p_{ik}\epsilon/u_k$; note that this holds for $k = 1$ since $\epsilon \geq u_1$. Therefore, by (3.13),

$$\varepsilon_{\text{cw}} \leq (q-1)u_1 + c\epsilon = O(\epsilon), \quad (3.17)$$

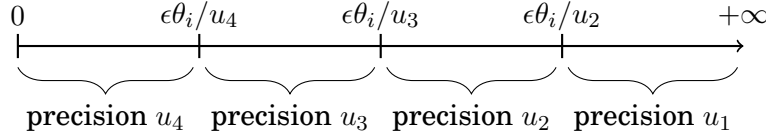


Figure 3.1: Illustration of the bucket construction with four precisions $u_1 < u_2 < u_3 < u_4$. The real line $[0, +\infty)$ is partitioned into intervals P_{ik} defined by (3.15) (componentwise criteria, $\theta_i = |a_i|^T |x|$) or (3.23) (normwise criteria, $\theta_i = \|A\|$).

with

$$c = (1 + (q - 1)u_1) \max_i \sum_{k=1}^q p_{ik}^2 (1 + u_k)^2. \quad (3.18)$$

We note that we have not taken into account any rounding error occurring in the computation of the intervals P_{ik} , which we assume to be evaluated in sufficiently high precision to be considered exact. Indeed, as a sum of positive values, the problem is well conditioned.

Since, by (3.5), $\varepsilon_{\text{nw}} \leq \varepsilon_{\text{cw}}$, this bucket construction also yields a normwise backward error in $O(\epsilon)$. However, if we only need to bound the normwise backward error, and can afford a potentially large componentwise error, we can improve the use of low precisions by modifying the buckets as follows. Taking norms in (3.12) shows that

$$\varepsilon_{\text{nw}} \leq (q - 1)u_1 + (1 + (q - 1)u_1) \max_i \left[\sum_{k=1}^q p_{ik} u_k (1 + u_k)^2 \beta_{ik} \right], \quad (3.19)$$

where it is now the ratios

$$\beta_{ik} = \frac{\sum_{j \in B_{ik}} |a_{ij} x_j|}{\|A\| \|x\|} \quad (3.20)$$

that play a role in controlling the size of the normwise backward error. Importantly, unlike the ratios α_{ik} in (3.14), the ratios β_{ik} can be bounded above independently of x :

$$\beta_{ik} \leq \frac{\sum_{j \in B_{ik}} |a_{ij}|}{\|A\|}. \quad (3.21)$$

As a result, we can redefine the buckets as

$$B_{ik} = \{j \in J_i : |a_{ij}| \in P_{ik}\}. \quad (3.22)$$

with the intervals P_{ik} as in (3.15) with $|a_i|^T |x|$ replaced with $\|A\|$:

$$P_{ik} = \begin{cases} (\epsilon \|A\|/u_2, +\infty) & \text{for } k = 1, \\ (\epsilon \|A\|/u_{k+1}, \epsilon \|A\|/u_k) & \text{for } k = 2: q - 1, \\ [0, \epsilon \|A\|/u_q] & \text{for } k = q. \end{cases} \quad (3.23)$$

This is sufficient to ensure that $\beta_{ik} \leq p_{ik} \epsilon / u_k$ and thus that $\varepsilon_{\text{nw}} = O(\epsilon)$. However, in this case we can no longer guarantee a small ε_{cw} , since the ratios $\alpha_{ik}/\beta_{ik} = \|A\| \|x\| / |a_i|^T |x|$ can be arbitrarily large for some rows i .

We summarize the main conclusions of our analysis in the next theorem.

Theorem 1 *Let $A \in \mathbb{R}^{m \times n}$ and $x \in \mathbb{R}^n$ and let $y = Ax$ be computed with Algorithm 4. If the bucket partitioning is defined by (3.15)–(3.16), then we have*

$$\varepsilon_{\text{nw}} \leq \varepsilon_{\text{cw}} \leq (q-1)u_1 + c\varepsilon,$$

where the expression of c is given by (3.18). If instead it is defined by (3.22)–(3.23), then we only have

$$\varepsilon_{\text{nw}} \leq (q-1)u_1 + c\varepsilon.$$

Remark 3.1 *For sparse matrices, since the performance of SpMV is memory bound, in principle we could only store the elements of A in lower precisions and keep the floating-point operations in precision u_1 in order to avoid error accumulation. The error analysis above can be easily adapted to this scenario by replacing (3.7) with*

$$|\widehat{y}_i^{(k)} - y_i^{(k)}| \leq (p_{ik}u_1(1+u_k) + u_k) \sum_{j \in B_{ik}} |a_{ij}x_j|, \quad (3.24)$$

which roughly reduces the p_{ik}^2 term in (3.18) to p_{ik} .

Remark 3.2 *Our analysis allows for the case where some elements of A are simply dropped. Indeed, this can be modeled as using a “precision” $u_q = 1$, since replacing an element by zero introduces a relative perturbation equal to 1. Thus, taking $u_q = 1$ in (3.15) or (3.23) shows that elements of magnitude smaller than $\varepsilon|a_i|^T|x|$ or $\varepsilon\|A\|$ can be dropped while preserving a componentwise or normwise backward error of order ε , respectively.*

Remark 3.3 *Our analysis can be trivially specialized to adaptive precision inner products, for which A is a row vector, and to adaptive precision summation, for which $A = e = [1, \dots, 1]$. In the latter case, we obtain*

$$\varepsilon_{\text{bwd}} \leq (q-1)u_q + (1 + (q-1)u_q) \sum_{k=1}^q (p_k - 1)u_k\alpha_k, \quad (3.25)$$

where the ratios $\alpha_k = \frac{\sum_{|x_j| \in P_k} |x_j|}{\|x\|}$ for $k \in \{1, \dots, q\}$ determine which terms in 3.25 dominate. Taking the P_k as above we have $\varepsilon_{\text{bwd}} = O(\varepsilon)$.

Remark 3.4 *Our analysis considers that the bucket construction is made in maximal precision u_q and that the loss of accuracy induced can be dropped.*

3.1 A more practical componentwise bucket criteria

The approach presented above presents a practical limitation: to guarantee *componentwise* backward stability, the adaptive precision representation of matrix A must depend on the vector x we want to multiply it with, as shown by (3.15)–(3.16). Unfortunately, taking the values of x into account is unrealistic, since it would require to change the representation of A every time we want to compute its product with a different vector. A more practical scenario is to compute an adaptive precision representation of A independent of x and use it to accelerate many SpMVs with different vectors. The

bucket construction defined by (3.22)–(3.23) satisfies this practical constraint, but can only guarantee normwise stability.

This motivates us to propose a bucket construction

$$B_{ik} = \{j \in J_i : |a_{ij}| \in P_{ik}\} \quad (3.26)$$

with the definition of the intervals P_{ik} modified as follows:

$$P_{ik} = \begin{cases} (\epsilon|a_i|^T e/u_2, +\infty) & \text{for } k = 1, \\ (\epsilon|a_i|^T e/u_{k+1}, \epsilon|a_i|^T e/u_k] & \text{for } k = 2: q-1, \\ [0, \epsilon|a_i|^T e/u_q] & \text{for } k = q, \end{cases} \quad (3.27)$$

where $e = [1, \dots, 1]^T$, so that $|a_i|^T e = \sum_{j \in J_i} |a_{ij}|$. This modified definition essentially amounts to drop x in the componentwise bucket construction (3.15)–(3.16). With this bucket construction, we can bound the ratios α_{ik} (3.14)

$$\alpha_{ik} \leq \frac{p_{ik}\epsilon}{u_k} \frac{|a_i|^T e}{|a_i|^T |x|} \|x\|, \quad (3.28)$$

whereas with the normwise bucket construction (3.22)–(3.23), the best bound on α_{ik} we can get is

$$\alpha_{ik} \leq \frac{p_{ik}\epsilon}{u_k} \frac{\|A\|}{|a_i|^T |x|} \|x\|. \quad (3.29)$$

Clearly, the right-hand side of (3.29) can be larger than that of (3.28), especially for badly scaled matrices with rows such that $\|a_i\| \ll \|A\|$. Therefore, we can expect that at least in some cases, construction (3.26)–(3.27) can lead to much smaller ε_{cw} than construction (3.22)–(3.23). It is important to note that, unfortunately, construction (3.26)–(3.27) cannot always guarantee a small ε_{cw} , since the ratio $|a_i|^T e/|a_i|^T |x|$ can be arbitrarily large for an unlucky choice of vector x .

4 Adaptive precision SpMV: numerical experiments

We now evaluate the performance of our adaptive precision matrix–vector product, Algorithm 4, by applying it to a range of real-life large sparse matrices.

4.1 Implementation

We have developed a Fortran code that implements Algorithm 4 and made it publicly available¹. Our code uses up to seven different precisions: the IEEE binary64 and binary32 formats (hereinafter denoted as FP64 and FP32), the bfloat16 format, and four custom formats using 56, 48, 40, and 24 bits, which we will refer to as “FP x ”, with x the number of bits. The FP56, FP48, and FP40 formats use 11 bits for the exponent and thus have unit roundoffs 2^{-45} , 2^{-37} , and 2^{-29} , whereas the FP24 format uses 8 bits for the exponent, which corresponds to a unit roundoff 2^{-16} . This choice of formats aims at spanning as uniformly as possible the range of precisions used. In principle, we could

¹<https://gitlab.com/romeomolina/adaptive-spmv>

have used many more precision formats by adapting the precision bit by bit, but focusing on formats that use multiples of 8 bits simplifies the implementation of the cast operations. We also do not experiment with formats using a reduced number of bits for the exponent, such as IEEE binary16. In addition to these seven precision formats, we also drop the matrix elements that are sufficiently small, as explained in Remark 3.2. The list of precision formats is summarized in Table 3.1.

Table 3.1: List of precision formats used in our experiments.

	Sign	Numbers of bits		Range	Unit roundoff
		Exponent	Significand		
bfloat16	1	8	7	$10^{\pm 38}$	$2^{-8} \approx 4 \times 10^{-3}$
FP24	1	8	15	$10^{\pm 38}$	$2^{-16} \approx 2 \times 10^{-5}$
FP32	1	8	23	$10^{\pm 38}$	$2^{-24} \approx 6 \times 10^{-8}$
FP40	1	11	28	$10^{\pm 308}$	$2^{-29} \approx 2 \times 10^{-9}$
FP48	1	11	36	$10^{\pm 308}$	$2^{-37} \approx 7 \times 10^{-12}$
FP56	1	11	44	$10^{\pm 308}$	$2^{-45} \approx 3 \times 10^{-14}$
FP64	1	11	52	$10^{\pm 308}$	$2^{-53} \approx 1 \times 10^{-16}$

For the cast from FP64 to FP32 we use the Fortran `REAL` function, whereas for casting to the other custom formats (including bfloat16, which our hardware does not support), we use our own cast implementation, which uses the `MVBITS` subroutine of the GNU Fortran compiler. To be specific, for each coefficient we chop the desired bits by moving the bits that are to be kept in a variable of smaller size; for example, to cast an FP32 variable to FP24 format, we move the leading 24-bit to a 3-byte variable. Our environment only supports floating-point operations in FP64 or FP32. As a result, after casting the matrix elements to these custom precision formats, we must cast them back during the computation, either to FP32 (in the case of bfloat16 and FP24) or to FP64 (in the case of FP40, FP48, and FP56). As mentioned in Remark 3.1, performing the computations in a higher precision than the storage format only affects the constants in the error bounds. The “cast back” operation also relies on `MVBITS`: we simply move all the bits into an FP32 or FP64 variable and add as many zeros as needed. For example, to cast an FP24 variable back to FP32, we must add one byte of zeros.

We must mention that this cast implementation is far from optimized, and leads to a heavy performance overhead. We aim to use it only to validate the numerical behavior of our approach, rather than to provide acceleration with custom precision formats. However, it is important to note that achieving performance gains from the use of custom precisions is certainly possible, by relying on more efficient, lightweight cast implementations. For example, such implementations are described by Mukunoki and Imamura [Mukunoki & Imamura, 2016], or more recently by Grützmacher et al. [Grützmacher *et al.*, 2021]. This suggests that the three- and seven-precision versions could meet their potential with a more optimized implementation. Moreover lower precision formats such as bfloat16 are increasingly supported in hardware. The implementation of the adaptive precision SpMV on top of such an efficient accessor is therefore one of the main research perspectives of this work.

Our SpMV implementation uses the CSR format for all matrices and is multithreaded

by parallelizing the loop on the row indices with OpenMP. We recall that the CSR format consists of a row index array of size $n + 1$, a column index array of size nnz , and a value array of size nnz . As a result, in the uniform precision case, the total storage for the matrix is equal to

$$(nnz + n + 1)s_{\text{int}} + nnz s_{\text{FP}}, \quad (3.30)$$

where s_{int} is the size of the integer type and s_{FP} is the size of the floating-point type. For all our matrices, 4-byte integers suffice. For the adaptive precision SpMV, we use a different CSR matrix for each precision. Since each nonzero element belongs to a unique CSR matrix, the column index and value arrays of size nnz are splitted among the different CSR matrices, and so do not require any extra storage. However, the row index array of size $n + 1$ must be duplicated. This represents a storage increase of approximately $qn s_{\text{int}}$, where q is the number of precisions. In most cases this increase is compensated by the storage reduction of the floating-point values, but for matrices with low potential for low precisions and a small number of nonzeros per row (small nnz/n ratio), this may lead to a noticeable overhead cost. In our experiments we take into account the cost of reading the indices in addition to the one of reading the floating-point values when measuring the storage cost of the SpMV. In particular, the index access cost explains why the use of dropping may have a huge impact on the performance: storing an element in any precision does not change the need to store its column index, whereas dropping it allows for dropping its index too. We will further analyze this effect in section 4.4.

4.2 Experimental setting

All the experiments were performed on one node of the Olympe supercomputer, which is equipped with two 18-core Intel Skylake 6140 processors (for a total of 36 cores). We use 18 threads throughout the experiments, as this seems to be the optimal setting as we will observe in section 4.5. For the time measurements, we perform one hundred products and report the average timings. We do not include the time for reading the matrix from a file and putting it into CSR format. We also do not include the time for preprocessing the matrix into its adaptive precision representation (that is, for computing the bucket partitioning and creating the corresponding data structures). This preprocessing requires at most two passes over the nonzero elements of the matrix: one to compute the intervals P_{ik} (which is optional for the normwise criteria if $\|A\|$ is already known or can be cheaply estimated) and another to place the nonzeros into the corresponding bucket (CSR matrix). Therefore the cost of the preprocessing is negligible as long as we require several SpMVs (say, at least a dozen) with the same matrix, which is typically the case in iterative solvers.

Most of the matrices used in these experiments come from the SuiteSparse collection [Davis & Hu, 2011]. The others come from our industrial partners (see Table 3.2) and are described below. The thmgaz matrix corresponds to a coupled thermal, hydrological, and mechanical problem. The series of matrices Aghora_DGO{2,3,4} arise from the resolution of adjoint RANS equations in the context of high-fidelity simulations of turbulent compressible flows in aerodynamics. The spatial discretization of these equations relies on a high-order discontinuous Galerkin (DG) method with third, fourth, and fifth order accurate schemes. The test case corresponds to a subsonic laminar flow

over a NACA0012 airfoil. Jacobian matrices have been built with the ONERA Aghora DG solver [Renac *et al.*, 2015] and are real, nonsymmetric, not positive definite, with a blockwise structure and a symmetric pattern.

Clearly, by its very design, the potential of the adaptive precision algorithm completely depends on the matrix values: there must be sufficient variations in their magnitudes. For example, SuiteSparse has several binary matrices (with only zeros and ones) that present no potential at all. In our experiments, we have selected a range of matrices that present at least some potential, listed in Table 3.2. As for the vector x , we set it to $e = [1, \dots, 1]^T$ throughout the experiments. We emphasize that our adaptive precision SpMV is guaranteed to deliver the requested accuracy ϵ , and so must “work” for any matrix. The worst possible behavior is obtained for a matrix that presents no potential for mixed precision, which will lead the adaptive precision algorithm to use the highest precision for all elements, becoming equivalent to the uniform precision algorithm.

4.3 Main results

We begin in Figure 3.2 by evaluating the accuracy of our adaptive precision algorithm to confirm that we are able to control the backward error, which, according to Theorem 1, should remain of order ϵ . We check this both for the normwise and componentwise backward errors by plotting, in Figure 3.2a, the normwise backward error for the algorithm with the normwise bucket criteria (3.22)–(3.23), and, in Figure 3.2b, the componentwise backward error for the algorithm with the componentwise bucket criteria (3.15)–(3.16). We use three different target accuracies, that is, two values of ϵ , 2^{-53} and 2^{-24} , which correspond to the unit roundoffs of FP64 and FP32 respectively and an additional intermediate accuracy $\epsilon = 2^{-37}$, and compare its backward error to the one obtained by the uniform precision algorithm in the corresponding precision ($\epsilon = 2^{-24}$, $\epsilon = 2^{-37}$, $\epsilon = 2^{-53}$). Moreover, we also investigate how the backward error is affected if, instead of using all 7 precision formats, we only use 2 (FP64 and FP32) or 3 (FP64, FP32, and bfloat16). As expected, the measured errors remain close to the target accuracy, for all targets ϵ , and for any configuration of precision formats. Using more precision formats slightly increases the error, which is explained by the analysis, since the constant c in (3.18) increases with q .

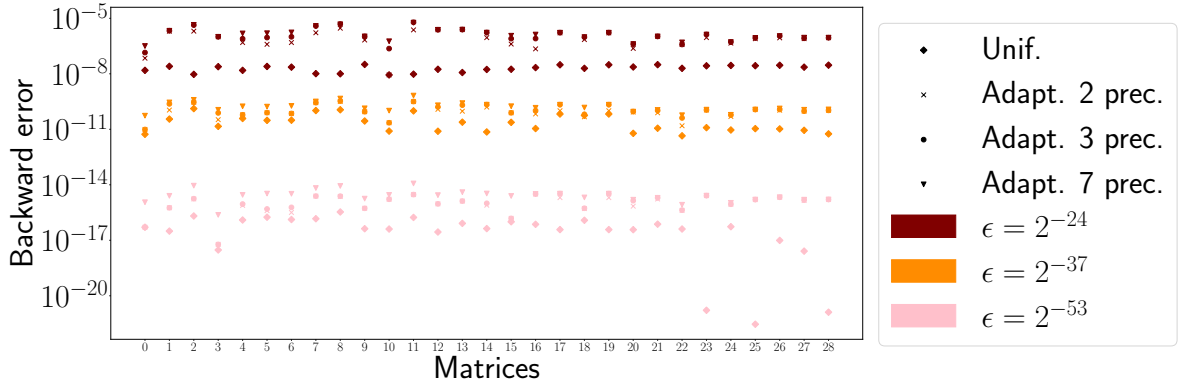
Next, we evaluate the performance gains achieved by the adaptive precision algorithm. We first measure the storage gains, that is, the number of bytes necessary to store the matrix in adaptive precision. The storage cost is a relevant metric because it drives the data movement costs of the SpMV, which is a memory-bound algorithm.

Figure 3.3 plots the storage cost of the adaptive precision algorithm as a percentage of the uniform precision FP64 algorithm. As for Figure 3.2, several configurations of the adaptive precision algorithm are tested, depending on the accuracy target ($\epsilon = 2^{-24}$, $\epsilon = 2^{-37}$, $\epsilon = 2^{-53}$), the number of precisions used (2, 3, or 7, with dropping being used in all cases), and on whether the buckets are built with the componentwise criteria (3.15)–(3.16) or the normwise one (3.22)–(3.23). Clearly, the more precision formats are used, the larger are the gains, since we can better adapt the choice of precisions to each element. In some cases, the use of more than two precisions appears to be critical: for example, the storage cost for matrices 14 or 20 with an FP32 accuracy target (Figure 3.3c) is nearly divided by two when adding bfloat16 (3 precisions instead of 2).

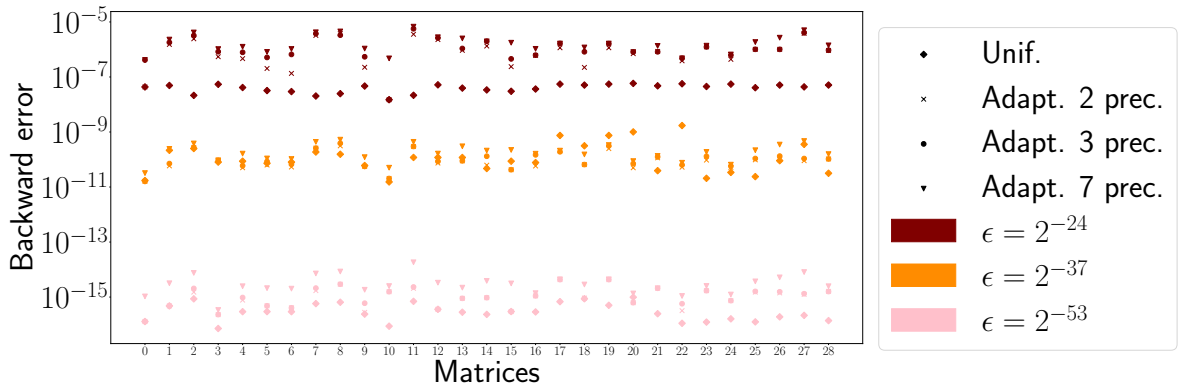
Table 3.2: List of matrices used in our experiments.

Number	Matrix	n	nnz
0	Transport	$1.6e + 06$	$2.4e + 07$
1	Serena	$1.4e + 06$	$3.3e + 07$
2	A_DGO3.mtx	$1.5e + 05$	$1.8e + 07$
3	vas_stokes_4M	$4.4e + 06$	$1.3e + 08$
4	A_DGO4.mtx	$2.6e + 05$	$5.1e + 07$
5	Emilia_923	$9.2e + 05$	$2.1e + 07$
6	A_DGO5.mtx	$3.8e + 05$	$1.1e + 08$
7	Hook_1498	$1.5e + 06$	$3.1e + 07$
8	ML_Geer	$1.5e + 06$	$1.1e + 08$
9	ML_Laplace	$3.8e + 05$	$2.8e + 07$
10	vas_stokes_1M	$1.1e + 06$	$3.5e + 07$
11	stokes	$1.1e + 07$	$3.5e + 08$
12	Geo_1438	$1.4e + 06$	$3.2e + 07$
13	ss	$1.7e + 06$	$3.5e + 07$
14	vas_stokes_2M	$2.1e + 06$	$6.5e + 07$
15	Fault_639	$6.4e + 05$	$1.5e + 07$
16	Queen_4147	$4.1e + 06$	$1.7e + 08$
17	PFlow_742	$7.4e + 05$	$1.9e + 07$
18	Flan_1565	$1.6e + 06$	$5.9e + 07$
19	Cube_Coup_dt0	$2.2e + 06$	$6.5e + 07$
20	Long_Coup_dt6	$1.5e + 06$	$4.4e + 07$
21	CoupCons3D	$4.2e + 05$	$2.2e + 07$
22	Long_Coup_dt0	$1.5e + 06$	$4.4e + 07$
23	StocF-1465	$1.5e + 06$	$1.1e + 07$
24	nv2	$1.5e + 06$	$5.3e + 07$
25	power9	$1.6e + 05$	$2.5e + 06$
26	test1	$3.9e + 05$	$1.3e + 07$
27	imagesensor	$1.2e + 05$	$1.9e + 06$
28	mosfet2	$4.7e + 04$	$1.5e + 06$
29	dgreen	$1.2e + 06$	$3.8e + 07$
30	radiation	$2.2e + 05$	$7.6e + 06$
31	nv1	$7.5e + 04$	$2.4e + 06$

Moreover, as expected, the storage gains are always larger with the normwise criteria (blue bars), which offers more room to the use of lower precisions than the component-wise one (green bars). Finally, it is also worth noting that the relative storage gains also become larger as the accuracy target is lowered, even when compared with the uniform precision algorithm in the corresponding precision. That is, while lowering the accuracy target from FP64 (Figure 3.3a) to FP32 (Figure 3.3c) reduces the storage cost of the uniform precision algorithm by a factor two, it can reduce the cost of the adaptive precision algorithm by a much larger factor. This is for example the case for matrix 16, for which the adaptive precision algorithm (with 7 precisions and a normwise criteria) achieves



(a) Normwise backward error (3.4) (the adaptive precision algorithm uses the normwise bucket criteria (3.22)–(3.23)).



(b) Componentwise backward error (3.3) (the adaptive precision algorithm uses the componentwise bucket criteria (3.15)–(3.16)).

Figure 3.2: Backward error for the adaptive precision Algorithm 4 with different target accuracies ϵ and different number of precisions used, compared with the uniform precision Algorithm 3 in the corresponding precision ($\epsilon = 2^{-24}$, $\epsilon = 2^{-37}$, $\epsilon = 2^{-53}$).

a cost of about 60% of the uniform FP64 cost for an FP64 target, to be compared with only about 5% of the uniform FP64 cost (and hence 10% of the uniform FP32 cost) for an FP32 target.

In any case, the storage gains are overall significant in all configurations and for several matrices, with reductions of up to a factor $36\times$ in the best case.

Finally, we measure the execution time of the algorithms. Since SpMV is memory bound, in principle we can hope the time gains to roughly follow the storage gains, even though the execution time depends on several other factors such as the overhead cost of the cast operations and the latency costs. In our experiments, we have found the time cost of the adaptive precision SpMV to roughly match its storage cost in the case where we only use precision formats that are natively supported in our environment, that is, the FP64 and FP32 formats (which corresponds to the two-precision version plus dropping). Unfortunately, as mentioned in section 4.1, our cast implementation is not optimized and is only designed to validate the numerical behavior of the adaptive preci-

sion algorithm. As a result, we have found the use of other custom precision formats to lead to slowdowns due to a heavy performance penalty associated with our cast implementation, and restrict our time performance analysis to the two-precision version plus dropping.

Figure 3.4 reports the execution time of the adaptive precision SpMV for $\epsilon = 2^{-24}$ and $\epsilon = 2^{-53}$ target accuracies, as a percentage of the execution time of the uniform precision SpMV in the corresponding precision (FP64 or FP32). The time cost of the algorithm follows a trend similar to the storage cost, with the gains being in general smaller but still significant, with speedups of up to $7\times$ in the best case.

Interestingly, for some matrices, the time reduction is larger than the storage one, and this effect is not explained by measurement noise and can be consistently reproduced across several runs. A possible explanation is that the smaller storage cost of the matrix reduces the number of cache misses and hence benefits from the doubled effect of a lower volume of data movement and higher bandwidth. Alternatively, it could also be explained by the dropping of sufficiently small elements, which not only reduces the bandwidth costs but also the latency ones, since the dropped elements are not read at all.

Finally, we also report the execution time in the case of an $\epsilon = 2^{-37}$ accuracy target in Figure 3.5. The figure also plots the time for the $\epsilon = 2^{-24}$ and $\epsilon = 2^{-53}$ targets (already presented in Figure 3.4) as a point of comparison. Figure 3.5 illustrates a valuable feature of our adaptive precision algorithm: it is able to achieve a flexible level of accuracy that does not necessarily correspond to any natively supported precision format, while only using such supported formats (here FP64 and FP32). This is because the accuracy of the adaptive precision algorithm is determined by ϵ , rather than directly by the unit roundoffs of the precision formats that are used.

4.4 Effect of dropping

The performance gains achieved by the adaptive precision SpMV are obtained thanks to the use of lower precisions but also the use of dropping. As noted in Remark 3.2, our error analysis fully accounts for the use of dropping, which effectively behaves as a precision format with unit roundoff $u_q = 1$. Nevertheless, the effect of dropping on the performance of the SpMV is quite different from the effect of lower precisions. This is because dropping increases the sparsity of the matrix and therefore allows for reducing the storage for indices too. For example, for 4-byte indices, using the two precision formats FP64 and FP32 but not dropping, the adaptive precision storage can be no less than 66% of the uniform precision one, since we must still store about $8nnz$ bytes ($4nnz$ for the indices, and $4nnz$ for the values, in the best case where all can be switched to FP32). In contrast, dropping the elements allows for dropping the associated indices, and therefore much larger gains can be obtained. The goal of this section is to analyze this effect more precisely by evaluating the impact of both dropping and low precisions separately.

We plot in Figures 3.6a, 3.6b and 3.6c the accuracy, storage, and time, respectively, of four SpMV variants: uniform FP64 (“Unif. fp64”), adaptive with two precisions (FP64 and FP32) but no dropping (“Adapt. dropleless”), adaptive with only one precision (FP64) and dropping (“Adapt. drop only”), and adaptive with two precisions and dropping at the same time (“Adapt.”). All three adaptive variants use an accuracy target $\epsilon = 2^{-53}$.

Figure 3.6a shows that both approximation tools used by the adaptive method (dropping and precision reduction) each slightly increase the error, but all variants remain of the order of the requested accuracy ϵ . As expected, Figures 3.6b and 3.6c show that the adaptive SpMV benefits both from the use of multiple precisions and of dropping, separately or combined. In some cases, dropping has a massive impact and is the main contributor to the performance gains, but in other cases, dropping has almost no effect and it is the use of multiple precisions that is responsible for most of the gains. All in all, this confirms the relevance of using an adaptive SpMV that combines both techniques.

4.5 Parallel scaling analysis

We conclude by analyzing the scalability of our SpMV implementation. For this analysis we use matrix `Cube_Coup_dt0`, which is one of the largest in our set; we have observed similar trends on other matrices. Figure 3.7a compares the uniform and adaptive precision methods with a number of threads increasing from 1 to 36 (the total number of cores on the shared-memory node). The figure shows that both methods scale well up to 18 threads, and suffer a slowdown going from 18 to 19 threads. This is due to the NUMA architecture of the node, which consists of two 18-core sockets. This is particularly visible on Figure 3.7b, which plots the parallel efficiency of the methods and shows a major loss of efficiency between 18 and 19 threads. These observations have led us to choose a number of threads equal to 18 for all experiments, in order to maximize the data locality and performance of the methods.

5 Application to Krylov solvers

We now apply our adaptive precision SpMV (Algorithm 4) to the solution of linear systems $Ax = b$ by Krylov methods. Iterative solvers are indeed a natural application for our algorithm: since the matrix A remains fixed throughout the computation, we can partition it into adaptive precision form only once before using it throughout the iterations in potentially many matrix–vector products, as long as we rely on either the normwise bucket criteria (3.22)–(3.23) or the relaxed componentwise one (3.26)–(3.27).

5.1 Adaptive precision Krylov solvers

We will focus our discussion and experiments on three choices of Krylov solvers [Saad, 2003]: GMRES, CG, and BiCGStab, respectively outlined in Algorithms 5, 6, and 7. CG is specifically designed for symmetric positive-definite matrices. BiCGStab is designed to handle general matrices by building two Krylov subspaces (thus requiring two SpMVs per iteration); it incorporates a stabilization step compared with the original BiCG algorithm. Finally, GMRES is the most robust Krylov method; it relies on the construction of an orthonormal basis for the Krylov subspace, whose size grows at each iteration. This requires computationally expensive orthogonalization operations, whose cost can be limited by restarting the method. In the case of CG and BiCGStab, the SpMV is usually the computational bottleneck; for the GMRES algorithm, the orthogonalization of the Krylov basis is also expensive, but nevertheless the SpMV still represents a significant fraction of the total. Therefore by accelerating the SpMV in these Krylov methods we

can expect significant speedups on the whole solution, provided that the convergence is preserved.

Algorithm 5 GMRES.

Input: a matrix $A \in \mathbb{R}^{n \times n}$, a right-hand side $b \in \mathbb{R}^n$, and an initial solution $x_0 \in \mathbb{R}^n$.

Output: a solution $x_k \in \mathbb{R}^n$.

```

1:  $r = b - Ax_0$ 
2:  $\beta = \|r\|_2$ 
3:  $q_1 = r/\beta$ 
4: for  $k = 1, 2, \dots$  do
5:    $y = Aq_k$ 
6:   for  $j = 1 : k$  do
7:      $h_{jk} = q_j^T y$ 
8:      $y = y - h_{jk}q_j$ 
9:   end for
10:   $h_{k+1,k} = \|y\|_2$ 
11:   $q_{k+1} = y/h_{k+1,k}$ 
12:  Solve the least squares problem  $\min_{c_k} \|Hc_k - \beta e_1\|_2$ .
13:   $x_k = x_0 + Q_k c_k$ 
14: end for

```

Algorithm 6 CG.

Input: a matrix $A \in \mathbb{R}^{n \times n}$, a right-hand side $b \in \mathbb{R}^n$, and an initial solution $x_0 \in \mathbb{R}^n$.

Output: a solution $x_k \in \mathbb{R}^n$.

```

1:  $r = b - Ax_0$ 
2:  $z = M^{-1}r$ 
3:  $p = z$ 
4:  $k = 0$ 
5: for  $k = 1, 2, \dots$  do
6:    $\alpha_k = \frac{r_k^T z_k}{p_k^T A p_k}$ 
7:    $x_{k+1} = x_k + \alpha_k p_k$ 
8:    $r_{k+1} = r_k - \alpha_k A p_k$ 
9:    $z_{k+1} = M^{-1}r_{k+1}$ 
10:   $\beta_k = \frac{r_{k+1}^T (z_{k+1} - z_k)}{r_k^T z_k}$ 
11:   $p_{k+1} = z_{k+1} + \beta_k p_k$ 
12: end for

```

First, from a theoretical point of view, we can state that using an adaptive precision SpMV within a normwise backward stable GMRES solver, such as MGS-GMRES [Paige *et al.*, 2006], will not endanger the normwise backward stability of the solution. Intuitively, this is not surprising since the adaptive precision SpMV is also backward stable, as we have shown in section 3. More formally, we can prove this by relying on the recent analysis of Amestoy *et al.* [Amestoy *et al.*, 2021]. Indeed, [Amestoy *et al.*, 2021, Thm. 3.1] proves, under mild assumptions, that if the products $y = Aq$ within MGS-GMRES are

Algorithm 7 BiCGStab.

Input: a matrix $A \in \mathbb{R}^{n \times n}$, a right-hand side $b \in \mathbb{R}^n$, and an initial solution $x_0 \in \mathbb{R}^n$.

Output: a solution $x_k \in \mathbb{R}^n$.

```

1:  $r = b - Ax_0$ 
2:  $rho_0 = \alpha = \omega_0 = 1$ 
3:  $v_0 = p_0 = 0$ 
4: for  $k = 1, 2, \dots$  do
5:    $\rho_i = \hat{r}_0^T r_{i-1}$ 
6:    $\beta = (\rho_i / \rho_{i-1})(\alpha / \omega_{i-1})$ 
7:    $p_i = r_{i-1} + \beta(p_{i-1} - \omega_{i-1}v_{i-1})$ 
8:    $v_i = Ap_i$ 
9:    $\alpha = \rho_i / (\hat{r}_0^T v_i)$ 
10:   $h = x_{i-1} + \alpha p_i$ 
11:   $s = r_{i-1} - \alpha v_i$ 
12:   $t = As$ 
13:   $\omega_i = (t^T s) / (t^T t)$ 
14:   $x_i = h + \omega_i s$ 
15:   $r_i = s - \omega_i t$ 
16: end for

```

performed such that the computed \hat{y} satisfies

$$\hat{y} = Aq + f, \quad \|f\| \leq \epsilon \|A\| \|q\|, \quad (3.31)$$

then the computed solution \hat{x} to $Ax = b$ satisfies a backward error in $O(\epsilon)$. We can therefore conclude from our Theorem 1 that setting the SpMV accuracy target to ϵ will also provide a backward error in $O(\epsilon)$ for the solution of $Ax = b$. Note that this theoretical discussion is limited to normwise stability, since GMRES is not known to be componentwise backward stable. Moreover neither CG nor BiCGStab are backward stable. Nevertheless, we will test GMRES with both the normwise and componentwise criteria for SpMV, because we have experimentally observed that using a componentwise stable SpMV can in some cases improve the convergence behavior of GMRES compared with using an only normwise stable SpMV. We will experiment with both criteria for the CG and BICGStab algorithms too.

5.2 Iterative refinement

In section 4, we have shown that the speedups achieved by the adaptive precision SpMV tend to be larger for lower accuracy targets. We now explain why, as a result of this property, the adaptive precision SpMV is particularly attractive in the context of iterative refinement based on Krylov solvers, such as GMRES-IR [Higham & Mary, 2022, sect. 8], [Carson & Higham, 2017, 2018; Amestoy *et al.*, 2021; Lindquist *et al.*, 2020; Loe *et al.*, 2021]. Iterative refinement, described in Algorithm 8, takes the form of an inner-outer scheme, in which the solution x_i is iteratively refined (the outer loop) by solving a correction system $Ad_i = r_i$ using a Krylov method (Algorithms 5, 6, or 7, the inner loop). Note that for GMRES, Algorithm 8 is equivalent to restarted GMRES when the inner GMRES on line 3 is initialized with $d_0 = 0$.

Algorithm 8 Krylov-based iterative refinement.

Input: a matrix $A \in \mathbb{R}^{n \times n}$, a right-hand side $b \in \mathbb{R}^n$, and an initial solution $x_0 \in \mathbb{R}^n$.

Output: a solution $x_i \in \mathbb{R}^n$.

- 1: **for** $i = 1, 2, \dots$ **do**
- 2: $r_i = b - Ax_{i-1}$
- 3: Solve $\tilde{A}d_i = r_i$ by a Krylov method (Algorithms 5, 6, or 7) using SpMVs with a lower precision matrix \tilde{A} .
- 4: $x_i = x_{i-1} + d_i$
- 5: **end for**

Importantly, it is known that Algorithm 8 can converge to a high accuracy even when the inner Krylov method is performed entirely in low precision [Higham & Mary, 2022, sect. 8], [Amestoy *et al.*, 2021; Carson & Higham, 2018]. In our adaptive precision context, we can therefore leave the outer loop SpMV (line 2 of Algorithm 8) in high (uniform) precision, and perform the inner loop SpMV of Algorithms 5, 6, and 7 with an approximate matrix \tilde{A} that exploits adaptive precision with a low accuracy target ϵ . Since the inner loop SpMV is called many more times than the outer loop one, we can expect the cost of the overall iterative refinement solution to be determined by the cost of the low accuracy inner loop SpMV.

In the following, we will assess experimentally the impact of using an adaptive precision SpMV in the inner loop on the convergence and performance of the solution. We incorporate a row scaling by solving $D^{-1}Ax = D^{-1}b$, with D a diagonal matrix whose coefficients are defined as $d_{ii} = \max_j |a_{ij}|$. This scaling also serves as a very simple Jacobi preconditioner; we leave the use of more complex preconditioners for future work.

Finally, we note that using adaptive precision for the SpMV is not the only possible strategy to exploit mixed precision in Krylov solvers; many other approaches have been proposed in the literature. In addition to approaches belong to the iterative refinement class mentioned above [Carson & Higham, 2017, 2018; Amestoy *et al.*, 2021, 2023], other possible e the use of low precision for the Krylov basis [Aliaga *et al.*, 2022], or adaptively decreasing the precision as the iterations go based on inexact Krylov theory [Simoncini & Szyld, 2003]. We emphasize that our adaptive precision SpMV algorithm is complementary to these strategies, and could be combined with them.

5.3 Adaptive GMRES-IR convergence analysis

We begin by analyzing how the use of adaptive precision SpMV affects the convergence of GMRES-IR. The goal of this section is to compare the use of uniform and adaptive precision SpMV and to analyze the effect of different parameters, mainly the accuracy target ϵ and the choice between componentwise (“CW” hereinafter) or normwise (“NW”) criteria. We illustrate different aspects of the behavior of adaptive precision GMRES-IR by using three examples, matrices ML_Laplace, CoupCons3D, and Geo_1438.

Figure 3.8 plots the convergence of GMRES-IR for matrix ML_Laplace using either uniform or adaptive precision SpMV. Our reference is the FP32 uniform precision variant, which converges to nearly FP64 accuracy after 4000 iterations. We also test a bfloat16 uniform precision variant, whose convergence is much slower, achieving only a residual of about 10^{-6} after the same number of iterations. Finally, we test the adaptive

precision SpMV variant with several values of ϵ and with CW criteria; for this matrix, the use of NW criteria significantly degrades the convergence (not shown). In the legend, we indicate the adaptive precision SpMV cost as a percentage of the FP32 uniform precision one. The figure shows that ϵ has an effect on both the SpMV cost (and therefore, the cost per iteration of GMRES-IR) and the convergence speed (and therefore, the total number of iterations). For example, with $\epsilon = 2^{-24}$, we expect the adaptive precision SpMV to be about as accurate as the FP32 uniform precision one, and indeed, the adaptive precision GMRES-IR converges at roughly the same speed with only 77% of the SpMV cost. For $\epsilon = 2^{-16}$, the SpMV cost is only 46% of the FP32 uniform precision one, but GMRES-IR converges much slower. The optimal choice of ϵ lies in between these two values; for this matrix, $\epsilon = 2^{-20}$ for example is a good choice.

Figure 3.9 plots the convergence of GMRES-IR for matrix CoupCons3D. Here, the adaptive precision variants can converge both for CW and NW criteria, and the figure illustrates the different tradeoffs that each option offers: for a fixed value of ϵ , NW variants achieve a lower cost but a slower convergence than CW ones. Therefore, the best choice of ϵ can be different for the NW and CW variants. In this example, $\epsilon = 2^{-24}$ leads to the best NW variant, which converges in 1040 iterations with an SpMV cost of 36% of the FP32 uniform one, whereas $\epsilon = 2^{-20}$ leads to the best CW variant, which converges in 320 iterations with a corresponding SpMV cost of 56%. Here, the CW variant therefore outperforms the NW one, but the figure illustrates that both options should be considered.

Finally, Figure 3.10 plots the convergence of GMRES-IR for matrix Geo_1438, which we use to illustrate a surprising behavior. As the figure shows, the NW adaptive precision variant can converge much faster than all the other variants, including the FP32 uniform precision one. Thus, the NW variant is much more efficient for this matrix since it requires both less iterations and a lower cost per iteration. This behavior can be consistently reproduced and occurs for several other matrices in our set. We do not have a completely satisfactory explanation; one possibility is that by aggressively dropping small coefficients from the matrix, the NW variant leads to a “nicer” matrix for which GMRES can converge quickly.

5.4 Performance comparison for different Krylov solvers

To conclude these experiments, we present execution time results in Tables 3.3 and 3.4. We compare four different Krylov solvers: CG, BiCGstab, and GMRES with two different restart sizes (40 or 80). Table 3.3 presents results on the matrices for which GMRES and BiCGStab both converge; for some of these matrices, CG converges too. Table 3.4 presents results on the matrices for which only GMRES converges. For each solver, the tables report the time and the backward error after convergence for different matrices and different SpMV variants: uniform or adaptive precision, with either the CW or NW criteria; we have tested three accuracy targets $\epsilon = 2^{-24}$, 2^{-20} , and 2^{-16} , and report the best for each variant and matrix. We report the total time, as well as the time spent in the SpMV calls between parentheses.

We first note that the total time of BiCGStab (which requires two SpMVs per iteration), and to a lesser extent that of CG (which requires only one), is dominated by the SpMV time. In contrast, the SpMV time represents a smaller, but still significant, fraction of the total time of GMRES; unsurprisingly, this fraction is larger for a smaller

restart size. It is also worth noting that for a given matrix, the best solver is not always the same depending on the SpMV variant that is used: in particular, the use of adaptive precision with NW criteria often prevents BiCGstab from converging, whereas the more robust GMRES solver can converge, and can sometimes do so faster than using the more expensive CW criteria.

Overall, this range of experiments shows that significant time reductions can be obtained by using an adaptive precision SpMV. In some cases, the speedup with respect to the uniform precision variant is huge because of the unexpected behavior observed in Figure 3.10, in which the adaptive precision NW variant actually converges in much less iterations than the uniform precision one (in addition to Geo_1438, this also happens, for example, for Emilia_923). Not counting these special cases, we still obtain significant speedups for many of the other matrices, especially those in Table 3.4.

Finally, we mention that the use of adaptive precision SpMV will lead to even larger speedups when the SpMV cost relative to the total increases. This is the case when the cost of the orthonormalization is reduced. Various strategies have recently been proposed in this direction, such as using low precision [Aliaga *et al.*, 2022] or using faster orthonormalization algorithms, for example based on randomized methods [Balabanov & Grigori, 2022]. techniques, such as using low precision or faster orthonormalization algorithms. Conversely, the relative cost of the SpMV may increase when part of a preconditioner, for example in the case of polynomial preconditioners (which require multiple SpMVs per iteration) or SPAI preconditioners (which require SpMVs with a matrix M that approximates A^{-1}).

Table 3.3: Results with GMRES-IR, BiCGStab-IR and CG-IR for various matrices and SpMV variants.

			GMRES(80)	GMRES(40)	BiCGstab	CG
CoupCons3D	Time	Uniform	2.09 (0.71)	1.35 (0.62)	1.26 (0.88)	5.31 (2.84)
		Adaptive CW	2.04 (0.71)	1.32 (0.62)	1.47 (1.04)	5.12 (2.75)
		Adaptive NW	4.86 (1.74)	1.86 (0.89)	4.13 (2.96)	5.12 (2.75)
	Error	Uniform	3e-15	4e-13	6e-13	1e-12
		Adaptive CW	4e-15	3e-13	3e-14	1e-12
		Adaptive NW	3e-13	4e-13	3e-13	2e-12
Geo_1438	Time	Uniform	38.76 (11.69)	29.11 (11.82)	38.52 (24.04)	—
		Adaptive CW	38.24 (11.49)	28.54 (11.68)	38.12 (23.71)	—
		Adaptive NW	5.02 (1.38)	1.97 (0.70)	—	—
	Error	Uniform	4e-10	5e-08	9e-07	—
		Adaptive CW	4e-10	3e-08	4e-05	—
		Adaptive NW	7e-14	3e-13	—	—
ML_Laplace	Time	Uniform	11.97 (5.04)	9.36 (5.11)	13.66 (10.23)	—
		Adaptive CW	10.63 (3.69)	7.96 (3.75)	10.79 (7.39)	—
		Adaptive NW	10.51 (3.59)	7.90 (3.68)	—	—
	Error	Uniform	6e-10	2e-08	7e-08	—
		Adaptive CW	4e-09	3e-08	6e-03	—
		Adaptive NW	9e-04	3e-02	—	—

Serena	Time	Uniform	32.25 (10.72)	29.03 (12.94)	39.77 (26.23)	26.70 (13.34)
		Adaptive CW	29.66 (9.80)	29.15 (12.91)	39.74 (25.97)	26.67 (13.29)
		Adaptive NW	8.11 (2.56)	23.17 (10.84)	—	—
	Error	Uniform	1e-13	5e-12	2e-12	2e-05
		Adaptive CW	4e-13	7e-12	2e-12	8e-04
		Adaptive NW	4e-14	9e-08	—	—
ss1	Time	Uniform	0.04 (0.00)	0.03 (0.00)	0.24 (0.18)	0.10 (0.01)
		Adaptive CW	0.03 (0.00)	0.03 (0.00)	0.22 (0.15)	0.16 (0.02)
		Adaptive NW	0.03 (0.00)	0.03 (0.01)	0.23 (0.17)	0.17 (0.04)
	Error	Uniform	6e-13	6e-13	3e-16	7e-09
		Adaptive CW	6e-13	6e-13	3e-15	2e-12
		Adaptive NW	2e-13	6e-13	3e-14	2e-11

Table 3.4: Results with GMRES-IR for various matrices and SpMV variants.

		GMRES(80)	GMRES(40)	GMRES(80)	GMRES(40)
		Time (s)		Backward error	
Cube_Coup_dt0	Uniform	65.69 (23.43)	49.75 (23.59)	4e-10	5e-10
	Adaptive CW	59.78 (17.64)	44.74 (18.15)	7e-09	8e-09
	Adaptive NW	56.28 (14.03)	41.10 (14.15)	4e-09	4e-09
Emilia_923	Uniform	24.74 (7.53)	18.50 (7.68)	7e-07	8e-07
	Adaptive CW	24.64 (7.69)	18.44 (7.79)	7e-07	8e-07
	Adaptive NW	8.24 (1.90)	3.03 (0.99)	4e-13	5e-13
Fault_639	Uniform	17.38 (5.40)	12.85 (5.46)	3e-07	4e-07
	Adaptive CW	17.25 (5.24)	12.55 (5.27)	5e-07	5e-07
	Adaptive NW	13.99 (2.24)	9.65 (2.30)	2e-06	1e-06
Flan_1565	Uniform	52.34 (22.64)	41.74 (23.02)	5e-07	6e-07
	Adaptive CW	47.91 (18.12)	37.25 (18.46)	7e-07	6e-07
	Adaptive NW	48.02 (18.11)	37.07 (18.15)	6e-07	1e-06
Hook_1498	Uniform	40.38 (11.96)	29.98 (12.15)	1e-06	2e-06
	Adaptive CW	39.96 (11.61)	29.84 (11.78)	2e-06	2e-06
	Adaptive NW	40.40 (11.85)	29.84 (11.99)	2e-06	2e-06
Long_Coup_dt0	Uniform	44.21 (16.27)	33.62 (16.52)	5e-12	5e-12
	Adaptive CW	39.27 (11.81)	29.50 (12.02)	2e-11	8e-12
	Adaptive NW	29.66 (1.53)	19.39 (1.86)	8e-12	2e-11
Long_Coup_dt6	Uniform	44.06 (16.21)	34.07 (16.48)	8e-11	3e-10
	Adaptive CW	40.15 (12.31)	30.45 (12.71)	2e-10	3e-09
	Adaptive NW	29.82 (1.60)	22.91 (5.43)	4e-09	5e-12
ML_Geer	Uniform	48.97 (20.11)	38.72 (20.46)	2e-07	9e-07
	Adaptive CW	45.24 (16.75)	35.23 (17.05)	5e-07	1e-06
	Adaptive NW	44.71 (15.99)	34.46 (16.20)	9e-04	1e-03
PFlow_742	Uniform	20.93 (7.20)	15.83 (7.43)	2e-10	2e-10

	Adaptive CW	19.38 (5.64)	14.21 (5.76)	3e-10	2e-10
	Adaptive NW	15.68 (1.75)	10.27 (1.85)	5e-05	7e-05
Queen_4147	Uniform	164.19 (63.27)	126.04 (64.33)	3e-07	8e-07
	Adaptive CW	159.58 (62.23)	124.42 (63.28)	7e-07	8e-07
	Adaptive NW	110.97 (11.46)	72.69 (12.24)	1e-05	1e-05
StocF-1465	Uniform	31.71 (4.74)	21.88 (4.82)	8e-09	8e-09
	Adaptive CW	30.39 (3.50)	20.60 (3.57)	8e-09	9e-09
	Adaptive NW	28.55 (0.72)	18.06 (0.83)	2e-08	5e-09
Transport	Uniform	35.49 (4.99)	23.92 (4.97)	2e-07	1e-05
	Adaptive CW	33.12 (2.98)	22.03 (3.11)	1e-06	2e-06
	Adaptive NW	33.64 (2.99)	22.06 (3.08)	2e-06	4e-06
dgreen	Uniform	1.78 (0.50)	0.74 (0.26)	3e-15	5e-15
	Adaptive CW	1.47 (0.19)	0.59 (0.11)	5e-16	1e-15
	Adaptive NW	1.42 (0.14)	0.56 (0.06)	1e-15	1e-15
imagesensor	Uniform	0.20 (0.05)	0.07 (0.03)	7e-16	5e-16
	Adaptive CW	0.13 (0.01)	0.05 (0.01)	6e-16	2e-15
	Adaptive NW	0.13 (0.00)	0.05 (0.00)	6e-16	7e-16
mosfet2	Uniform	0.10 (0.04)	0.04 (0.02)	8e-14	6e-14
	Adaptive CW	0.05 (0.01)	0.02 (0.01)	4e-13	1e-13
	Adaptive NW	0.05 (0.01)	0.02 (0.01)	1e-13	3e-14
nv1	Uniform	0.15 (0.06)	0.06 (0.03)	1e-18	4e-18
	Adaptive CW	0.10 (0.01)	0.02 (0.01)	1e-18	2e-18
	Adaptive NW	0.09 (0.01)	0.02 (0.00)	1e-18	1e-17
nv2	Uniform	2.74 (1.17)	1.20 (0.60)	5e-17	7e-17
	Adaptive CW	1.81 (0.25)	0.73 (0.14)	4e-17	5e-17
	Adaptive NW	1.70 (0.10)	0.68 (0.06)	6e-17	2e-17
power9	Uniform	0.21 (0.04)	0.07 (0.01)	5e-19	6e-19
	Adaptive CW	0.17 (0.01)	0.06 (0.01)	3e-19	9e-20
	Adaptive NW	0.17 (0.01)	0.06 (0.00)	9e-20	6e-19
radiation	Uniform	0.40 (0.15)	0.16 (0.08)	2e-13	2e-13
	Adaptive CW	0.27 (0.03)	0.11 (0.02)	1e-13	6e-13
	Adaptive NW	0.27 (0.02)	0.10 (0.02)	1e-13	2e-13
ss	Uniform	42.98 (11.52)	31.58 (11.65)	2e-10	2e-03
	Adaptive CW	40.84 (9.44)	29.42 (9.62)	2e-10	2e-03
	Adaptive NW	41.35 (9.33)	29.29 (9.39)	9e-11	2e-03
stokes	Uniform	569.80 (169.68)	435.12 (172.14)	2e-05	3e-05
	Adaptive CW	536.18 (133.60)	399.76 (136.11)	3e-05	4e-05
	Adaptive NW	527.09 (122.99)	390.46 (126.10)	3e-05	3e-05
test1	Uniform	0.58 (0.15)	0.23 (0.07)	1e-14	1e-14
	Adaptive CW	0.47 (0.05)	0.18 (0.03)	3e-14	8e-14
	Adaptive NW	0.47 (0.05)	0.18 (0.03)	9e-15	7e-14

vas_stokes_1M	Uniform	36.02 (16.08)	29.03 (16.36)	5e-04	6e-04
	Adaptive CW	33.86 (13.96)	26.74 (14.13)	5e-04	6e-04
	Adaptive NW	33.34 (13.62)	26.32 (13.83)	6e-04	5e-04
vas_stokes_2M	Uniform	70.88 (28.22)	55.27 (28.64)	1e-04	9e-05
	Adaptive CW	63.33 (21.67)	48.64 (22.14)	1e-04	9e-05
	Adaptive NW	61.75 (20.48)	46.93 (20.64)	9e-05	9e-05
vas_stokes_4M	Uniform	159.89 (51.20)	118.03 (51.91)	9e-05	9e-05
	Adaptive CW	152.22 (44.27)	111.38 (45.08)	9e-05	9e-05
	Adaptive NW	151.02 (41.94)	108.61 (42.49)	8e-05	9e-05
Aghora_DGO3	Uniform	6.64 (3.72)	5.56 (3.76)	2e-03	2e-03
	Adaptive CW	6.43 (3.58)	5.34 (3.63)	3e-03	2e-03
	Adaptive NW	5.81 (3.14)	4.76 (3.15)	2e-03	2e-03
Aghora_DGO4	Uniform	15.39 (10.30)	13.40 (10.46)	1e-03	7e-04
	Adaptive CW	14.13 (9.30)	12.33 (9.45)	1e-03	7e-04
	Adaptive NW	12.95 (8.35)	11.26 (8.46)	1e-03	6e-04
Aghora_DGO5	Uniform	30.61 (23.27)	28.12 (23.58)	3e-03	2e-03
	Adaptive CW	27.46 (20.06)	24.89 (20.41)	3e-03	2e-03
	Adaptive NW	18.29 (11.08)	22.09 (17.68)	1e-04	2e-03

6 Conclusion

We have presented a mixed precision algorithm to compute SpMV and we have used it to accelerate the solution of sparse linear systems by iterative methods. Our algorithm is based on the idea of adapting the precision of each matrix element according to its magnitude: the elements are split into buckets that are summed in progressively lower precisions as their magnitudes decrease. We carried out a rounding error analysis of this algorithm, summarized in Theorem 1, which provides us with an explicit rule to build the buckets and to control its accuracy via a user-prescribed parameter ϵ .

Our experiments on a wide range of sparse matrices from real-life applications have demonstrated the significant potential of the method. The adaptive precision algorithm achieves storage reductions of up to a factor $36\times$ compared with the uniform precision algorithm, and these reductions translate to large time speedups on a multicore computer, up to a factor $7\times$; these gains are achieved while maintaining an accuracy comparable to that of the uniform precision algorithm. We have then investigated the use of our adaptive precision SpMV within Krylov solvers for the solution of sparse linear systems. We have shown that the convergence speed of the solvers is essentially unaffected by the use of adaptive precision SpMV with conservative choices for the value of ϵ , such as $\epsilon = 2^{-24}$, which yields an equivalent accuracy to using a uniform FP32 precision SpMV. Moreover, we have shown that using larger values of ϵ may often be beneficial by reducing the SpMV cost at the expense of a possibly slower convergence. Since ϵ does not need to correspond to the unit roundoff of a floating-point arithmetic, our adaptive precision solver is not constrained by the available precisions on the hardware and can achieve a flexible compromise between cost per iteration and total number of iterations.

While we have focused here on Krylov solvers with a simple diagonal preconditioner,

our adaptive precision framework is general and we expect it to be usable in other contexts. For example, we expect it to behave similarly with other iterative methods such as flexible GMRES. In future work we wish to extend the adaptive precision framework to cover other crucial steps of the solver, such as the construction of the Krylov basis or the preconditioner.

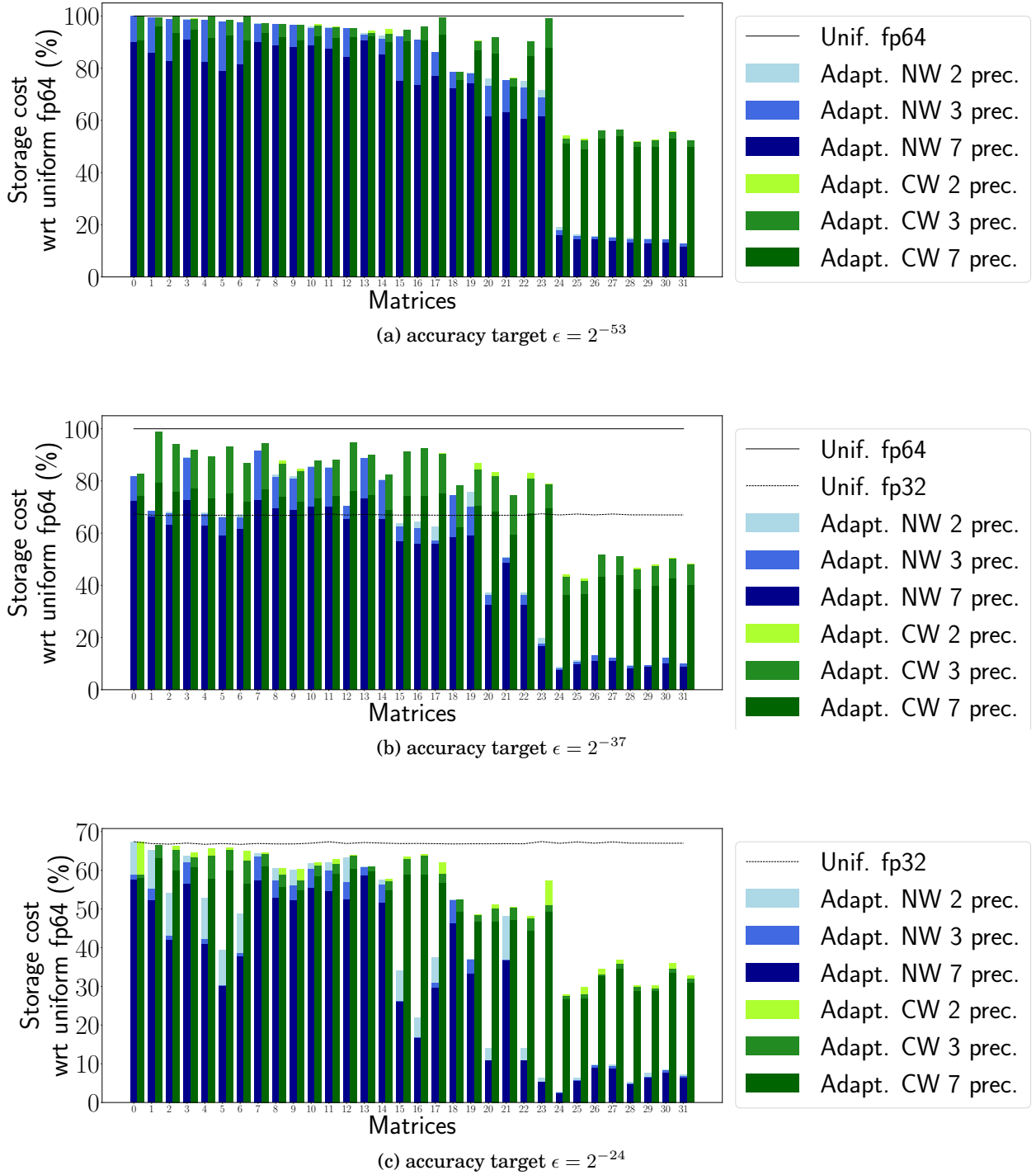


Figure 3.3: Storage cost of the adaptive precision SpMV, as a percentage of the storage cost of the uniform precision FP64 SpMV, for three different accuracy targets. For each plot, we report the storage gains depending on which of the componentwise (“CW”) or normwise (“NW”) criteria is considered and on how many precision formats are used.

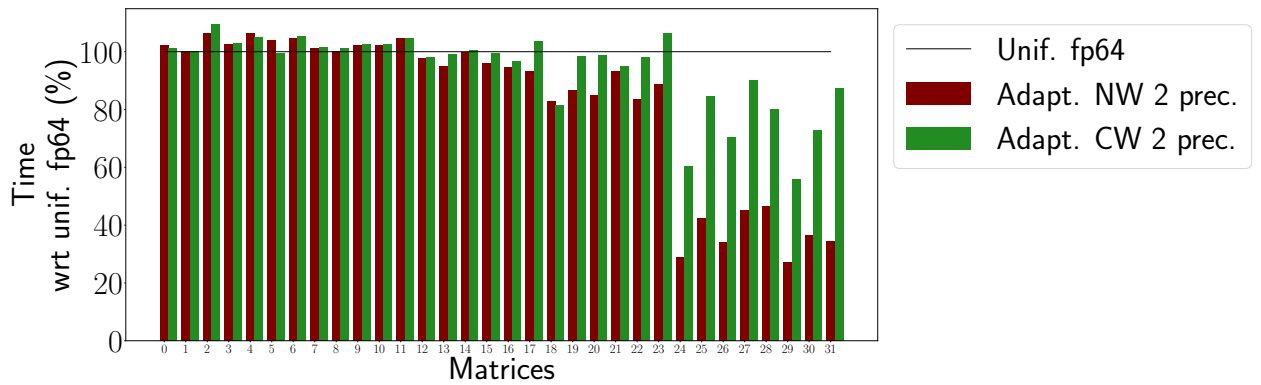
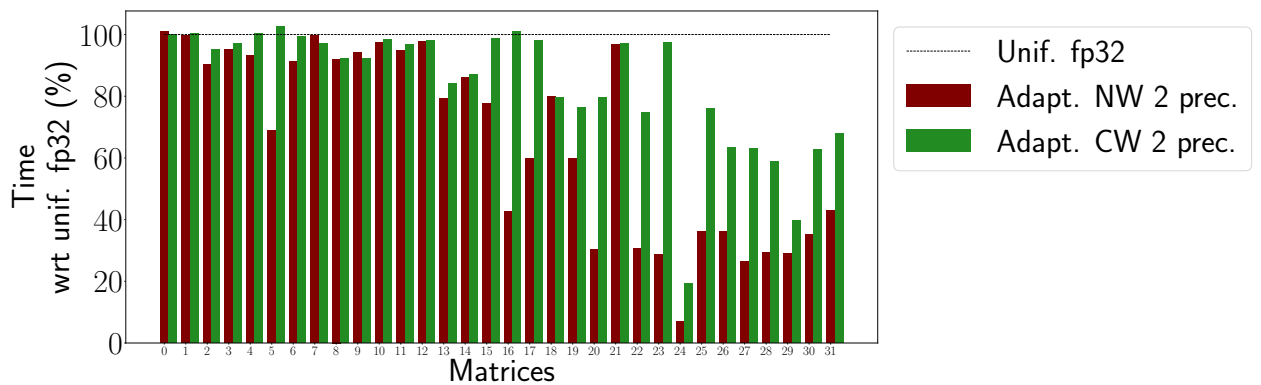
(a) accuracy target $\epsilon = 2^{-53}$ (b) accuracy target $\epsilon = 2^{-24}$

Figure 3.4: Execution time of the adaptive precision SpMV for $\epsilon = 2^{-24}$ and $\epsilon = 2^{-53}$ target accuracies, as a percentage of the execution time of the uniform precision SpMV in the corresponding precision. Both the normwise (“NW”) and componentwise (“CW”) criteria are reported.

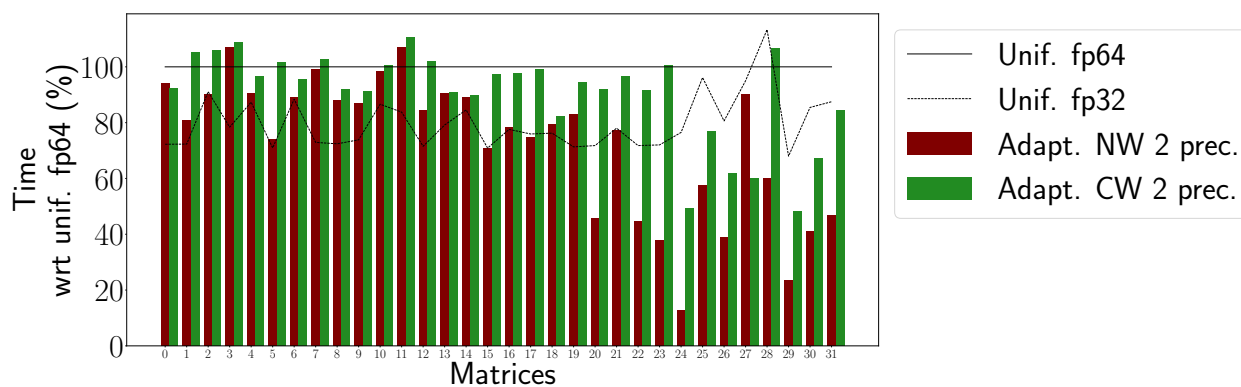
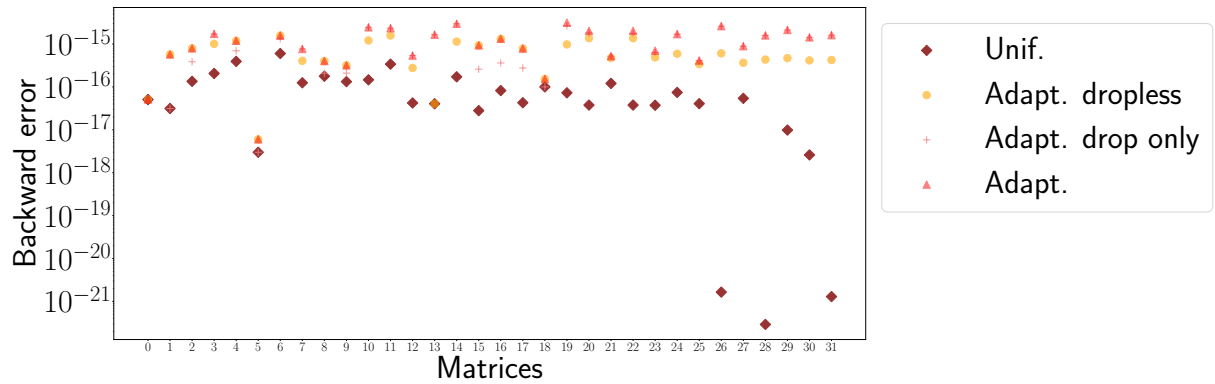
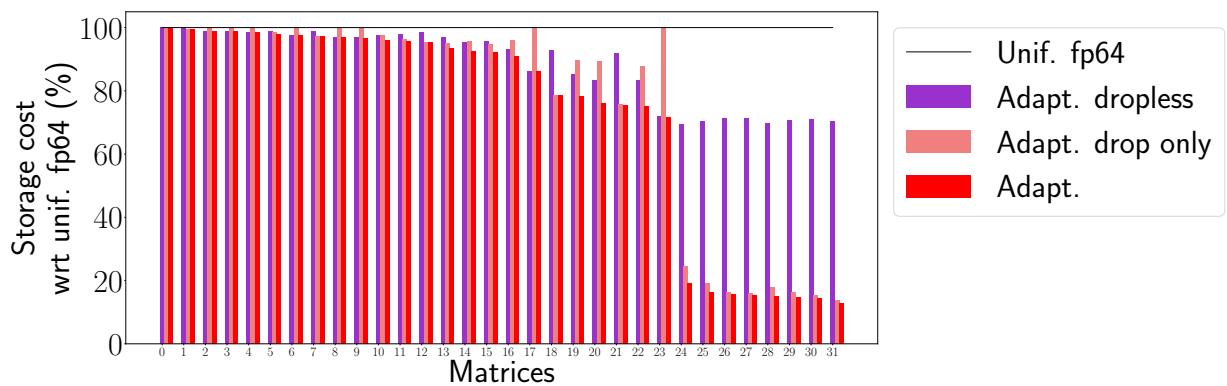


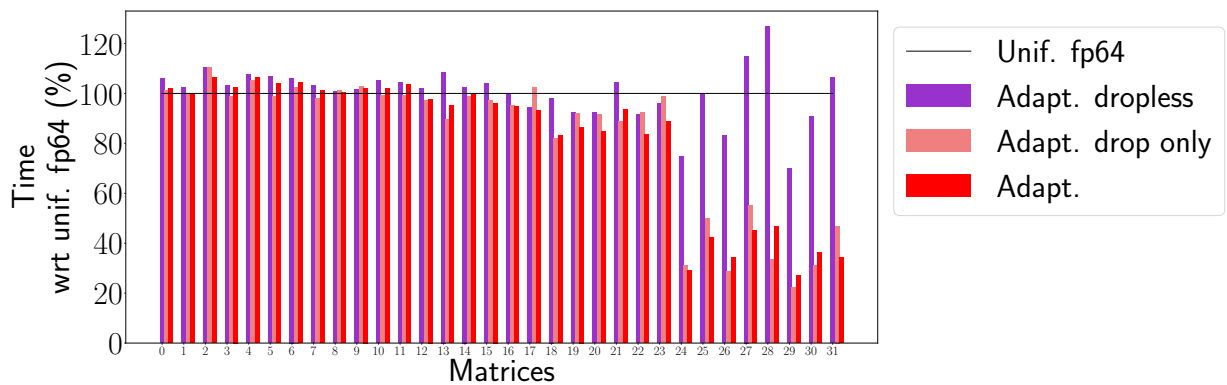
Figure 3.5: Execution time of the adaptive precision SpMV for an $\epsilon = 2^{-37}$ target accuracy, as a percentage of the execution time of the uniform precision FP64 SpMV. Both the normwise (“NW”) or componentwise (“CW”) criteria are reported.



(a) Backward error.

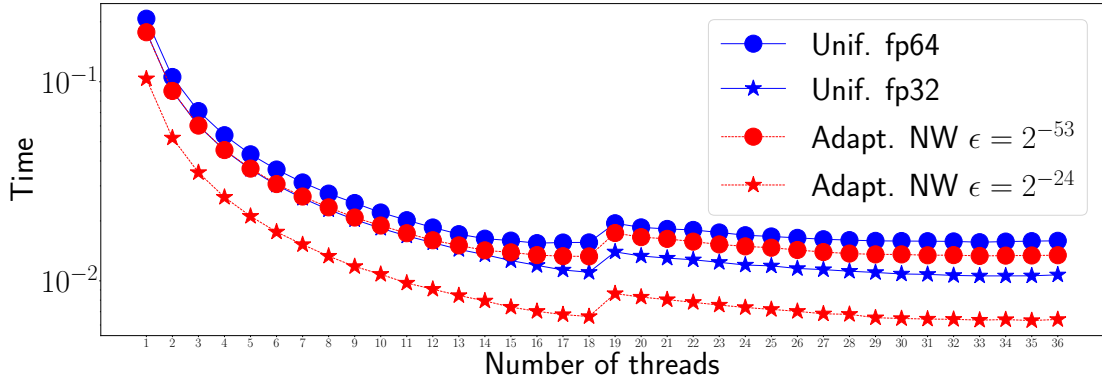


(b) Storage cost.

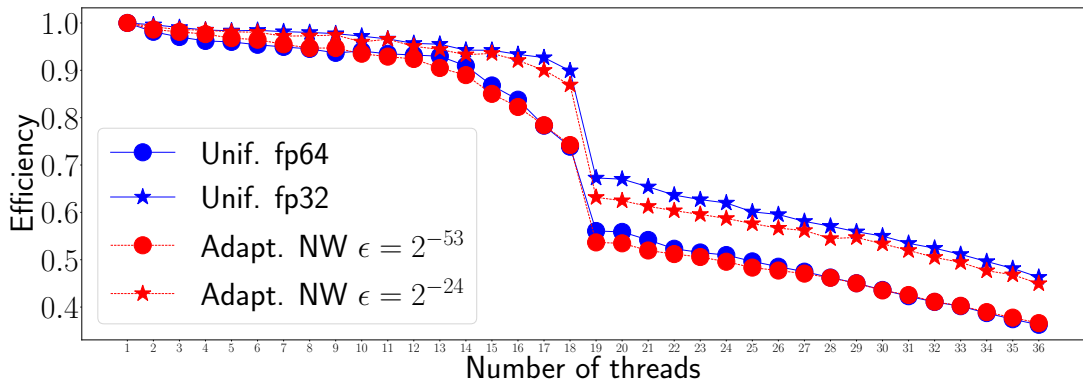


(c) Time cost.

Figure 3.6: Backward error, storage cost, and time cost of four SpMV variants: FP64 uniform precision (“Unif. fp64”), adaptive precision with two precisions but no dropping (“Adapt. dropless”), adaptive precision with only one precision and dropping (“Adapt. drop only”), and adaptive precision with both two precisions and dropping (“Adapt.”). All three adaptive variants use $\epsilon = 2^{-53}$ as target accuracy.



(a) Scaling on 1 to 36 threads



(b) Parallel efficiency on 1 to 36 threads

Figure 3.7: Parallel scaling experiments on Cube_Coup_dt0.

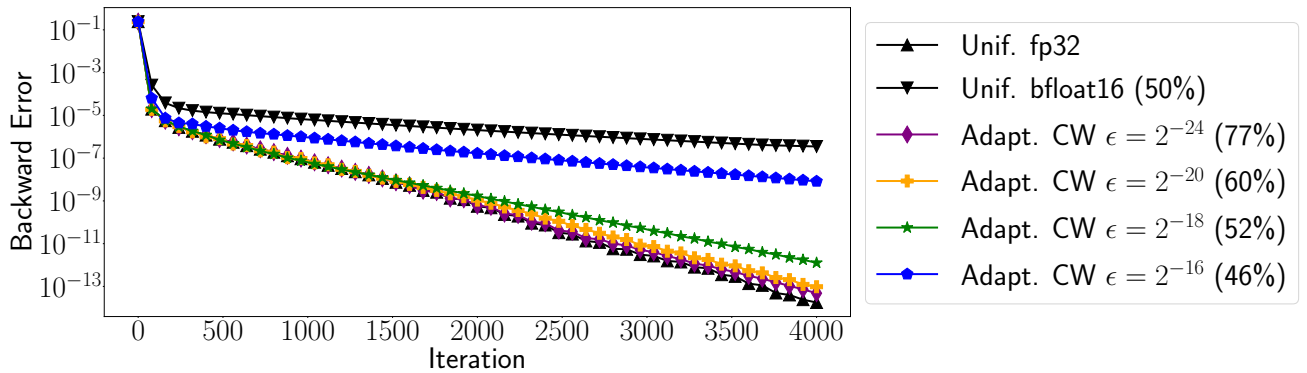


Figure 3.8: Convergence of GMRES-IR for matrix ML_Laplace: illustration of the effect of the ϵ parameter.

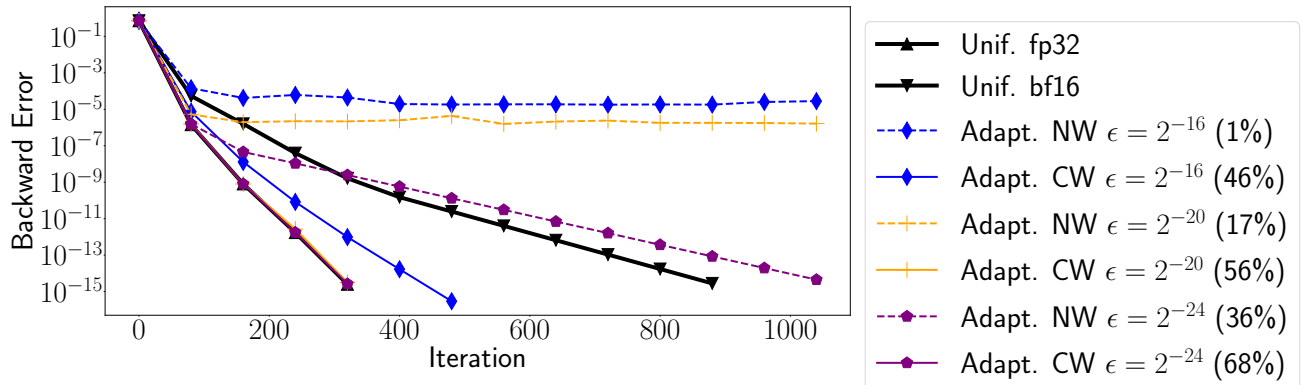


Figure 3.9: Convergence of GMRES-IR for matrix CoupCons3D: illustration of the difference between CW and NW criteria.

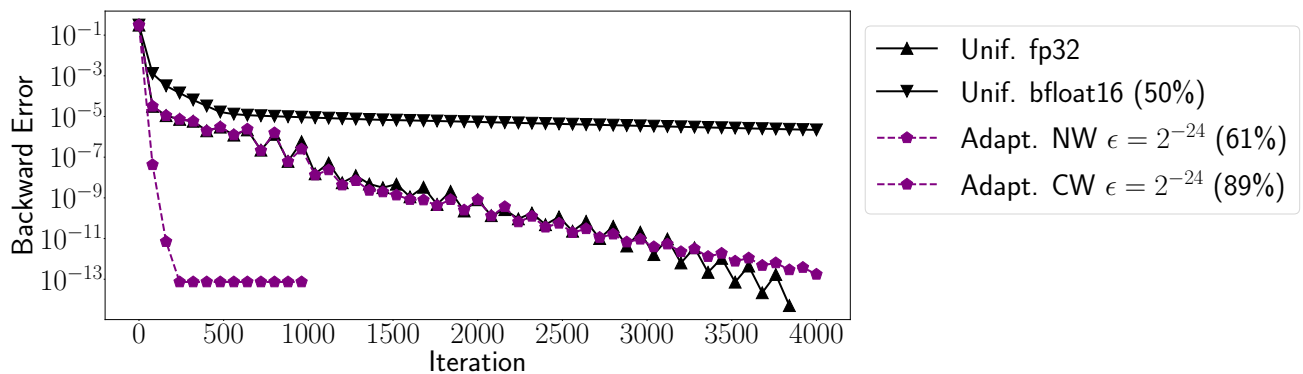


Figure 3.10: Convergence of GMRES-IR for matrix Geo_1438: illustration of a surprising behavior of NW variants.

Reduced-precision and reduced-exponent formats for accelerating adaptive SpMV

Mixed precision algorithms aim at taking advantage of the performance of low precisions while maintaining the accuracy of high precision. In particular adaptive precision algorithms dynamically adapt at runtime the precisions used for different variables or operations. For instance, the adaptive precision sparse matrix-vector product presented in chapter 3, stores the matrix elements in a precision inversely proportional to their magnitude. In theory, this algorithm can therefore make use of a large number of different precisions, but the practical results previously obtained only achieved high performance using natively supported double and single precisions. In this chapter we combine this algorithm with an efficient memory accessor for custom reduced precision formats (Mukunoki et al. 2016). This allows us to experiment with a large set of different precision formats with fine variations of the number of bits dedicated to the significand. Moreover we also explore the possibility to reduce the number of bits dedicated to the exponent using the fact that the elements that share the same precision format are of similar magnitude. We experimentally evaluate the performance of using four or seven different custom formats using reduced precision and possibly reduced exponent, and demonstrate their effectiveness compared with the existing version only using double and single precisions.

1 Introduction

The use of low precision floating-point formats in scientific computations is becoming more and more common due to the storage and performance gains that they offer. To maintain a rigorous control over the accuracy of the result, mixed precision algorithms combine various precision formats to find the desired tradeoff between performance and accuracy [Higham & Mary, 2022]. Adaptive precision algorithms [Higham & Mary, 2022, sect. 14], a subclass of mixed precision algorithms, have recently attracted interest due

to their ability to dynamically detect at runtime opportunities for reducing the precision based on the data at hand. Adaptive precision algorithms have for example been developed for low-rank approximations [Amestoy *et al.*, 2022], block Jacobi preconditioners [Anzt *et al.*, 2019; Flegar *et al.*, 2021], block/tile low-rank factorizations [Amestoy *et al.*, 2022; Abdulah *et al.*, 2022a], and sparse matrix-vector product (SpMV) [Graillat *et al.*, 2024a; Ahmad *et al.*, 2019].

In this Chapter, we are particularly interested in the adaptive precision SpMV presented in Chapter 3. SpMV is a key computational kernel in many applications, such as the solution of sparse linear systems with Krylov methods. Accelerating the SpMV while preserving control over its accuracy is therefore an important goal with a wide range of potential applications.

The approach of the adaptive sparse matrix-vector product is to accelerate SpMV by storing some of the nonzero elements of the matrix in reduced precision. Indeed, a theoretical error analysis demonstrates that the accuracy can be rigorously controlled by switching the elements to a precision with a unit roundoff inversely proportional to their magnitude: that is, smaller elements can be stored in lower precisions. This leads to storage reduction which can translate to a corresponding time reduction because SpMV is a memory-bound operation.

In addition to the error analysis, we have carried out a practical implementation of the algorithm that achieved significant performance gains compared with SpMV in uniform precision, at a comparable accuracy. However, these performance results are only satisfactory when using precisions with native hardware support, that is, in our case, the standard IEEE FP64 (double) and FP32 (single) precisions.

In this chapter, we are motivated by the fact that the adaptive precision SpMV can in principle make use of any number of precision levels and, in fact, achieves larger storage reductions when more precisions are available, since this allows for a fine tuning of the precision of each element. We are interested in the potential of emerging technologies for reduced-precision memory accessors [Mukunoki & Imamura, 2016; Grütz-macher *et al.*, 2021; Mukunoki *et al.*, 2023], which allow for efficiently accessing data stored in reduced precision formats. We focus in particular on the work of Mukunoki and Imamura [Mukunoki & Imamura, 2016], who propose a *custom* reduced precision accessor, thus allowing for many different precision formats. We develop an adaptive precision SpMV algorithm that relies on this memory accessor, and that can use up to seven different precision formats with fine variations of the number of bits dedicated to the significand. Moreover we also explore the possibility to reduce the number of bits dedicated to the exponent using the fact that the elements that share the same precision format are of similar magnitude. We provide numerical experiments on a multicore CPU architecture and with a range of real-life matrices. We evaluate the performance of adaptive precision SpMV with varying numbers of precision formats, using reduced precision and possibly reduced exponent, and demonstrate the effectiveness of the custom memory accessor compared with the existing version that only uses natively supported double and single precisions.

2 Methods

2.1 Adaptive precision SpMV

The adaptive precision SpMV presented in Chapter 3 decomposes the input matrix A into several matrices A_k :

$$A = \sum_{k=1}^q A_k,$$

where each A_k is stored in a different precision format with unit roundoff u_k , and the sparsity patterns of the A_k are all mutually disjoint (that is, each nonzero element of A is assigned to exactly one matrix A_k). The q precision levels satisfy $u_1 < u_2 < \dots < u_q$.

The SpMV $y = Ax$ is then computed as the sum of the partial SpMVs:

$$y = Ax = \sum_{k=1}^q A_k x.$$

The accuracy of the computed vector \hat{y} can be controlled by suitably building the decomposition. Specifically, the error analysis carried out in Chapter 3 and [Grailat *et al.*, 2024a] proves that, given a prescribed accuracy $\epsilon \geq u_1$, the computed \hat{y} satisfies the normwise backward error bound

$$\hat{y} = (A + \Delta A)x, \quad \|\Delta A\| \leq c\epsilon\|A\|,$$

where c is a modest constant, under the condition that all the nonzero elements a_{ij} of A_k satisfy the criterion

$$a_{ij} \in A_k \Leftrightarrow |a_{ij}| \in \left(\frac{\epsilon}{u_{k+1}}\|A\|, \frac{\epsilon}{u_k}\|A\| \right]. \quad (4.1)$$

This criterion shows that the precision u_k used to store each nonzero element should be chosen to be inversely proportional to the magnitude of the element. One special case is when $|a_{ij}| \leq \epsilon\|A\|$: in this case, the element can be *dropped*, that is, replaced by zero (this can be interpreted as using a “unit roundoff” $u_q = 1$, as mentioned in 3). Moreover, we recall that Chapter 3 also proposes an alternative criterion which bounds the componentwise backward error, instead of the normwise one.

The error analysis also accounts for the possibility of performing the partial SpMVs $A_k x$ in precision u_k , but since the SpMV is a memory-bound operation, this does not bring any significant performance improvement. We will therefore only use the reduced precision formats for storage, while keeping the arithmetic operations in double precision (which corresponds to the highest precision u_1).

2.2 Custom reduced-precision formats

Mukunoki and Imamura [Mukunoki & Imamura, 2016] have proposed a reduced-precision memory accessor, called RFPF, that allows for representing custom floating-point numbers with a reduced significand. This is achieved by truncating the IEEE FP64 (double) or FP32 (single) formats, as shown in Table 4.1: the RP56, RP48, and RP40 formats are truncated versions of FP64, whereas the RP24 and RP16 formats are truncated versions of FP32. Note that RP16 is equivalent to the bfloat16 format, although in our case we do not have native support for bfloat16 operations on our target hardware.

Table 4.1: List of IEEE formats and RFPF’s reduced-precision formats

Format	Numbers of bits			Unit roundoff
	Sign	Exponent	Significand	
FP64	1	11	52+1	$2^{-53} \approx 1 \times 10^{-16}$
RP56	1	11	44+1	$2^{-45} \approx 3 \times 10^{-14}$
RP48	1	11	36+1	$2^{-37} \approx 7 \times 10^{-12}$
RP40	1	11	28+1	$2^{-29} \approx 2 \times 10^{-9}$
FP32	1	8	23+1	$2^{-24} \approx 6 \times 10^{-8}$
RP24	1	8	15+1	$2^{-16} \approx 2 \times 10^{-5}$
RP16	1	8	7+1	$2^{-8} \approx 4 \times 10^{-3}$

This RFPF accessor is implemented in the C/C++ language and relies internally on a structure with multiple words composed of one, two, or four bytes: for example RP40 is represented using a 32-bit integer and an 8-bit one. When dealing with an array of RFPF numbers, each integer composing the RFPF format is allocated separately from the other integers (so-called structure-of-arrays layout). The decoding of an RP40 number to an FP64 one is illustrated in Figure 4.1. It is a relatively lightweight operation that consists in copying the 8-bit and 32-bit integers into 64-bit integers, suitably realigning them with a bit shift, and combining them with a binary `OR` operation.

The RFPF accessor has been shown to accelerate various types of memory-bound operations, such as dot products, dense matrix–vector products, and SpMVs. In particular, the recent work [Mukunoki *et al.*, 2023] focuses on the SpMV and shows that the performance is often proportional to the storage and thus to the number of bits. Naturally, in a uniform precision setting where the same RFPF format is used for all the elements, the accuracy is also proportional to the number of bits. In the following, we investigate the use of this kind of accessor in an adaptive precision setting which preserves a controlled accuracy.

2.3 Reduced-precision formats for adaptive precision SpMV

The adaptive precision SpMV is particularly amenable to the use of custom precision formats, for two reasons. First, the reduced precisions are only used as storage formats, and hence do not require native support for arithmetic operations. Second, because the precisions should be set to be inversely proportional to the magnitude of the elements, we can in principle exploit a continuous level of precisions: the finer we can tune the precision level, the higher the storage reduction.

The implementation presented in Chapter 3 achieves significant performance gains, but unfortunately only when using the natively supported FP64 and FP32 precisions available in the target architecture. Experiments with custom precision formats are presented, but these lead to a heavy performance penalty due to an unoptimized memory accessor implemented in Fortran.

The goal of this work is therefore to implement the adaptive precision SpMV algorithm with the much more efficient RFPF accessor from [Mukunoki & Imamura, 2016], and to evaluate to what extent custom precision formats can improve the performance. To do so, we ported the Fortran implementation of the adaptive precision SpMV pre-

```

1 union union64 {
2     uint64_t i;
3     double f;
4 };
5 __inline__ double
6 RpArrayToFp (rpfp64in40barray sa, size_t i){
7     union union64 u64;
8     uint64_t H, L;
9     H = (uint64_t)sa.i32[i];
10    H = i64h << 32;
11    L = (uint64_t)sa.i8[i];
12    L = i64l << 24;
13    u64.i = H | L;
14    return u64.f;
15 }

```

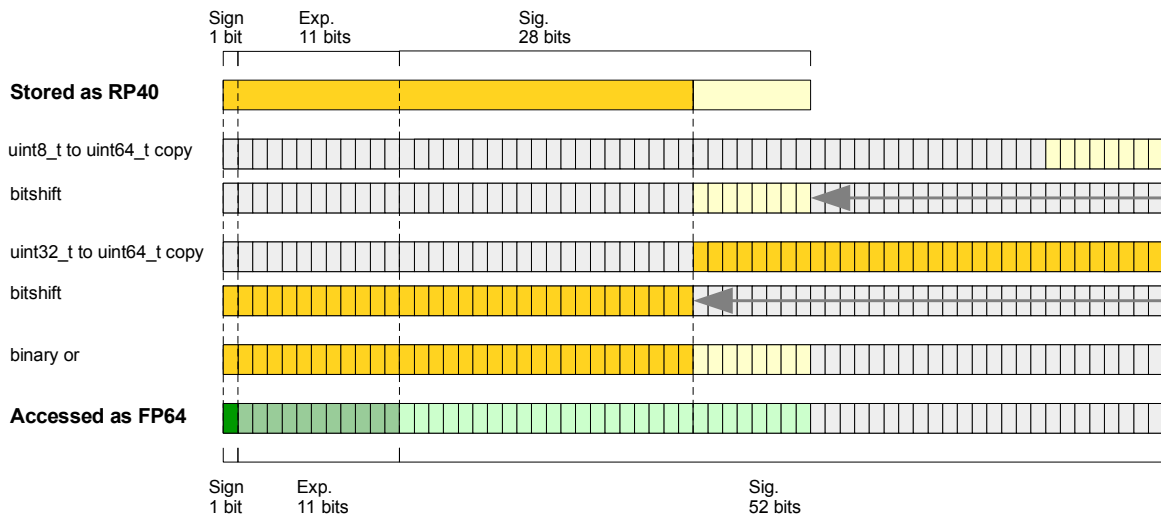


Figure 4.1: Conversion from RP40 to FP64

sented in Chapter 3 to the C language and combined it with the RFPF implementation for CPUs from [Mukunoki & Imamura, 2016].

To represent the sparse matrices A and A_k we use the compressed sparse row (CSR) format with 32-bit row and column indices. For each matrix A_k , this leads to $n + nnz_k$ 32-bit values for the indices, and nnz_k RFPF values for the nonzero elements, where n is the number of rows of A_k and nnz_k its number of nonzero elements.

The SpMV is parallelized with OpenMP using a static schedule: the rows of the matrix are distributed among the available threads.

Figure 4.2 presents an excerpt of the adaptive precision SpMV code with seven precision levels.

2.4 Reduced-exponent formats for adaptive precision SpMV

While the RFPF accessor proposed in [Mukunoki & Imamura, 2016] only reduces the significand, it may also be beneficial to reduce the number of bits allocated to the exponent field in order to further reduce the storage.

```

1 void ap_csrnv (int n, rpMultiCSR A, double* x, double* y) {
2     #pragma omp parallel for
3     for (int i = 0; i < n; i++) {
4         double tmp = 0.;
5         for (int k = A.ia16[i]; k < A.ia16[i+1]; k++) { // RP16
6             float aij_r = RpArrayToFp(A.a16, k);
7             tmp += aij_r * x[A.ja16[k]];
8         }
9         for (int k = A.ia24[i]; k < A.ia24[i+1]; k++) { // RP24
10            float aij_r = RpArrayToFp(A.a24, k);
11            tmp += aij_r * x[A.ja24[k]];
12        }
13        ...
14        for (int k = A.ia56[i]; k < A.ia56[i+1]; k++) { // RP56
15            double aij = RpArrayToFp(A.a56, k);
16            tmp += aij * x[A.ja56[k]];
17        }
18        for (int k = A.ia64[i]; k < A.ia64[i+1]; k++) { // FP64
19            double aij = A.a64[k];
20            tmp += aij * x[A.ja64[k]];
21        }
22        y[i] = tmp;
23    }
24 }

```

Figure 4.2: Adaptive precision SpMV with seven precision levels (excerpt). `RpArrayToFp` converts a reduced-precision format to the IEEE FP64 format.

This idea is particularly promising for the adaptive precision SpMV because the matrix elements stored in the same format are by design of similar magnitude. As shown in equation (4.1), all elements of A_k stored in a precision with unit roundoff u_k are in the interval $(\epsilon \|A\|/u_{k+1}, \epsilon \|A\|/u_k]$. Therefore, the dynamic range of the elements of A_k is u_{k+1}/u_k , which means that $\lceil \log_2(u_{k+1}/u_k) \rceil$ exponent values are sufficient to represent elements, and so

$$\left\lceil \log_2 \left\lceil \log_2 \frac{u_{k+1}}{u_k} \right\rceil \right\rceil$$

bits are sufficient for the exponent field. In particular, if we use all seven precision formats from Table 4.1, we have $u_{k+1}/u_k \leq 2^8$ and so we can reduce the number of bits dedicated to the exponent to $\log_2 \log_2 2^8 = 3$ bits. This represents a large storage reduction compared with the 11 or 8 bits used by the formats in 4.1.

Concretely, we represent the elements a_{ij} of A_k as

$$a_{ij} = \frac{\epsilon \|A\|}{u_{k+1}} \cdot \alpha_{ij}$$

where $\alpha_{ij} \in [1, 2^8)$ is represented with reduced precision *and* reduced exponent (RPRE). Note that we must change the interval in equation (4.1) to be closed on the left and open on the right, in order to exclude the right endpoint $\alpha_{ij} = 2^8$ which would require four bits. This leads us to define the RPRE formats listed in Table 4.2. The RPRE32, RPRE40, and RPRE48 formats have respectively the same unit roundoff as RP40, RP48, RP56 but three bits of exponent instead of eleven. The RPRE24, RPRE16, and RPRE8 formats are similar to the FP32, RP24 and RP16 formats, with only three bits of exponent instead of eight and slightly different unit roundoffs.

Table 4.2: List of RPRE formats used for each interval of values.

Interval ($\epsilon' = \epsilon \ A\ $)	Format	Numbers of bits			Unit roundoff
		Sign	Exponent	Significand	
$[\epsilon'2^{45}, \ A\]$	FP64	1	11	52+1	$2^{-53} \approx 1 \times 10^{-16}$
$[\epsilon'2^{37}, \epsilon'2^{45})$	RPRE48	1	3	44+1	$2^{-45} \approx 3 \times 10^{-14}$
$[\epsilon'2^{29}, \epsilon'2^{37})$	RPRE40	1	3	36+1	$2^{-37} \approx 7 \times 10^{-12}$
$[\epsilon'2^{21}, \epsilon'2^{29})$	RPRE32	1	3	28+1	$2^{-29} \approx 2 \times 10^{-9}$
$[\epsilon'2^{13}, \epsilon'2^{21})$	RPRE24	1	3	20+1	$2^{-21} \approx 5 \times 10^{-7}$
$[\epsilon'2^5, \epsilon'2^{13})$	RPRE16	1	3	12+1	$2^{-13} \approx 1 \times 10^{-4}$
$[\epsilon', \epsilon'2^5)$	RPRE8	1	3	4+1	$2^{-5} \approx 3 \times 10^{-2}$
$[0, \epsilon')$	dropping	0	0	0	$2^0 = 1$

In the same spirit, we can further reduce the storage by splitting the elements of each A_k into A_k^+ and A_k^- according to their sign. This allows us to drop the sign bit in the floating-point representation, which can instead be used for the significand. This leads to unsigned RPRE formats (RPREU), listed in Table 4.3, which can be applied to a slightly better interval of values (with a smaller unit roundoff). For example, RPREU8 can be applied to values up to $\epsilon'2^6$, instead of $\epsilon'2^5$ for RPRE8. The tradeoff is that we double the number of matrices A_k , which doubles the number of row index arrays of size n (the total size of the column index arrays remains equal to nnz the number of nonzero elements of A). Therefore the RPREU formats can be beneficial only when the matrix is not too sparse (sufficiently large nnz/n ratio).

Table 4.3: List of RPREU formats used for each interval of values.

Interval ($\epsilon' = \epsilon \ A\ $)	Format	Numbers of bits			Unit roundoff
		Sign	Exponent	Significand	
$[\epsilon'2^{46}, \ A\]$	FP64	1	11	52+1	$2^{-53} \approx 1 \times 10^{-16}$
$[\epsilon'2^{38}, \epsilon'2^{46})$	RPREU48	0	3	45+1	$2^{-46} \approx 1 \times 10^{-14}$
$[\epsilon'2^{30}, \epsilon'2^{38})$	RPREU40	0	3	37+1	$2^{-38} \approx 4 \times 10^{-12}$
$[\epsilon'2^{22}, \epsilon'2^{30})$	RPREU32	0	3	29+1	$2^{-30} \approx 9 \times 10^{-10}$
$[\epsilon'2^{14}, \epsilon'2^{22})$	RPREU24	0	3	21+1	$2^{-22} \approx 2 \times 10^{-7}$
$[\epsilon'2^6, \epsilon'2^{14})$	RPREU16	0	3	13+1	$2^{-14} \approx 6 \times 10^{-5}$
$[\epsilon', \epsilon'2^6)$	RPREU8	0	3	5+1	$2^{-6} \approx 2 \times 10^{-2}$
$[0, \epsilon')$	dropping	0	0	0	$2^0 = 1$

In practice, the decoding of RPRE and RPREU formats is a little heavier than that of RP formats. To decode an RPRE number, as shown in Figure 4.3, we need to separate the bit of sign from the exponent, which requires extra binary and or operations. We also need to reset the exponent to its actual value but this operation can be realised only once per bucket. Decoding an RPREU number does not require the sign bit separation as there is none.

```

1  __inline__ double
2  RPREArrayToFp (rpre40barray sa, size_t i){
3      union union64 u64;
4      uint64_t H, M, L;
5      H = (uint64_t) ((sa.i8[i] & 0x80) | 0x40);
6      H = i64h << 56 ;
7      M = (uint64_t) (sa.i8[i] & 0x7F);
8      M = i64m << (32+16);
9      L = (uint64_t) sa.i32[i];
10     L = i64l << 16;
11     u64.i = H | M | L;
12     return u64.f;
13 }

```

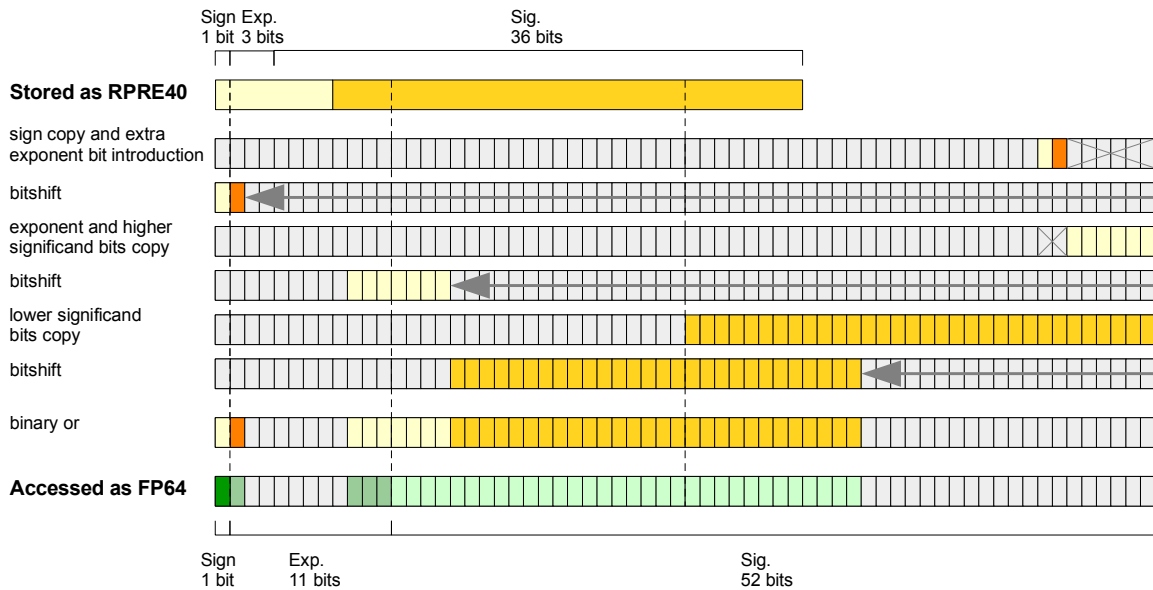


Figure 4.3: Conversion from RPRE40 to FP64

3 Evaluation

We perform the performance evaluation of our methods on one node of the Jean Zay supercomputer, equipped with two Intel Cascade Lake 6248 processors with 20 cores running at 2.5 GHz each, for a total of 40 cores. The experimental code was compiled using GCC 8.5.0 with `-O3 -march=native -fopenmp -lgomp (1 thread/core)`. It was executed with `numactl -interleave=all`.

We collected eight matrices from the SuiteSparse Matrix Collection [Davis & Hu, 2011], listed in Table 4.4 (each matrix has size of $n \times n$ with nnz nonzero elements). The matrices are ordered by nnz in descending order. We do not exploit the potential symmetry of the matrices: symmetric matrices are expanded to unsymmetric ones before the execution. The vector x is set to $e = [1, \dots, 1]^T$.

We have chosen to store the input and output vectors x and y in FP64 for all our experiments. We observed that this does not negatively affect the performance compared

Table 4.4: Test matrices (Sorted by nnz).

#	Matrix	n	nnz
0	vas_stokes_4M	4,382,246	131,577,616
1	Cube_Coup_dt0	2,164,760	127,206,144
2	Flan_1565	1,564,794	117,406,044
3	Long_Coup_dt6	1,470,152	87,088,992
4	bone010	986,703	71,666,325
5	vas_stokes_2M	2,146,677	65,129,037
6	Hook_1498	1,498,023	60,917,445
7	RM07R	381,689	37,464,962

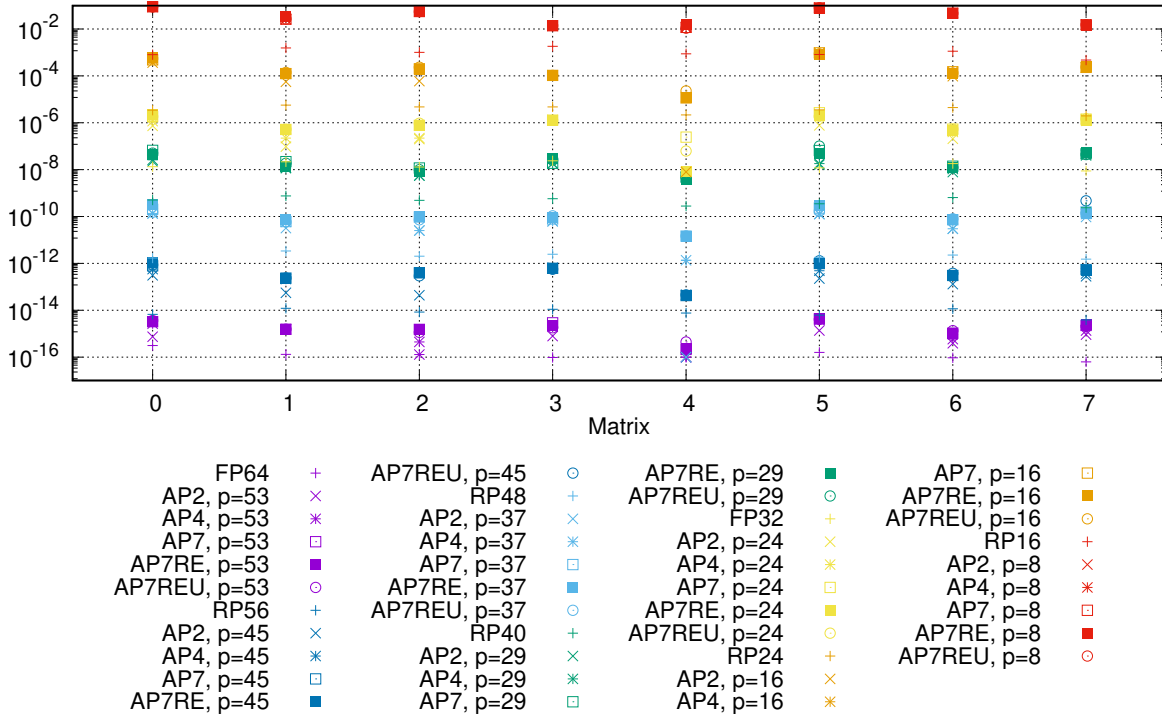


Figure 4.4: Normwise backward error computed from the FP128 uniform precision SpMV

with storing them in FP32 and can even improve it in some cases due to the appearance of denormalized numbers. To further minimize the risk of incurring underflow, overflow, and subnormality, we also scale the matrix by its norm so that all elements are bounded by 1.

For each result, we report the shortest execution time out of 15 executions (the SpMV is executed five times in a single program, and the program is executed three times.).

To evaluate the potential offered by the introduction of RFPF formats in the adaptive precision SpMV, we compare the following configurations:

- **FPxx**: Uniform precision SpMV with FPxx (xx=32 or 64).

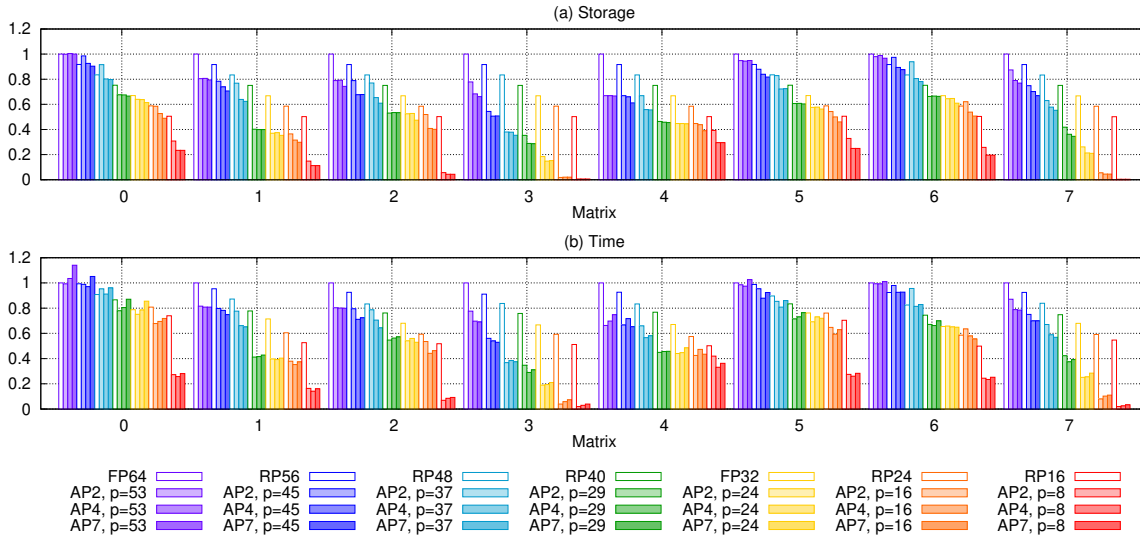


Figure 4.5: Storage and time gains achieved by adaptive precision variants over uniform precision ones (normalized by the FP64 cost)

- **RPxx**: Uniform precision SpMV with RPxx (xx=16, 24, 40, 48, or 56).
- **AP2**: Adaptive precision SpMV with two precision levels: the natively supported IEEE FP64 and FP32, as well as dropping.
- **AP4**: Adaptive precision SpMV with four precision levels: FP64, RP48, FP32, and RP16, as well as dropping.
- **AP7**: Adaptive precision SpMV with all seven precision levels: FP64, RP56, RP48, RP40, FP32, RP24, and RP16, as well as dropping.

In addition, to evaluate the potential offered by the reduced-exponent formats, we also compare the above variants with the following configurations.

- **AP7RE**: Adaptive precision SpMV with seven precision levels: FP64, RPRE48, RPRE40, RPRE32, FP32, RPRE16, and RPRE8, as well as dropping.
- **AP7REU**: Adaptive precision SpMV with seven precision levels: FP64, RPREU48, RPREU40, RPREU32, FP32, RPREU16, and RPREU8, as well as dropping.

Moreover, we test the adaptive precision variants with various accuracy targets $\epsilon = 2^{-p}$, and compare them with the uniform precision variant of corresponding accuracy (for example, for $p = 29$ we compare with the RP40 variant).

We begin by checking the correctness of our code. Figure 4.4 presents the backward errors achieved by each configuration. The figure confirms that all adaptive precision variants achieve the prescribed accuracy of order ϵ .

3.1 Performance of adaptive precision SpMV with RFPF

Figure 4.5 presents the storage and time costs of each variant, normalized by the FP64 cost. The time cost closely follows the storage cost as expected. The first observation

is that the adaptive precision variants are usually faster than the uniform precision variant at comparable accuracy, with very large speedups in some cases (up to 96%) that are explained by huge storage reductions (up to 99%) thanks to the use of dropping. The AP2 variant, which only uses natively supported precisions, performs well but we can see that the AP4 and AP7 variants can in many cases further improve the performance by exploiting custom precision formats. For example, the maximum storage gain of AP4 over AP2 is 24% and the maximum speedup is 21%. As for the difference between AP4 and AP7, it is less significant, and AP4 can be more efficient in some cases because of the increased weight of the indices storage. Still, using seven rather than just four precisions can bring an improvement up to 11% in the storage and a maximum speedup of 9%.

3.2 Performance of adaptive precision SpMV with RPRE and RPREU

In order to provide performance evaluations of the RPRE and RPREU variants, we have chosen not to use the RPRE24 format but to use FP32 instead, which we have observed to be more efficient since the latter is a natively supported format. We have also experimented using both FP32 and RPRE24 and obtained similar, although slightly less efficient results than when only using FP32 (which can be explained by the short length of the intervals associated with these two formats).

Figure 4.6 compares the storage and time costs of the AP7RE and AP7REU variants with those of the previously analyzed AP7 and uniform precision variants. The figure shows that the use of reduced-exponent formats can improve the performance in some cases and on the contrary degrade it in other cases. Nevertheless, these variants can achieve up to 16% storage reduction and a maximum speedup of 13%. The heavier decoding of these reduced-exponent formats presumably explains why their use does not always improve the performance.

Finally, we plot in Figure 4.7 the distribution of the precision formats used for each nonzero element of the matrix. The figure illustrates that having a greater number of reduced-precision formats (AP7 instead of AP4 or AP2) allows for a finer tuning of the precisions assigned to each element. Moreover, the figure also shows that using reduced-exponent formats (AP7RE or AP7REU) allows for reusing the exponent (and possibly sign) bits for the significand, which increases the proportion of elements stored in reduced precisions.

4 Conclusion

We have demonstrated the potential of using custom floating-point formats for accelerating the adaptive precision sparse matrix–vector product algorithm presented in Chapter 3. We have shown that using up to seven different reduced-precision formats from [Mukunoki & Imamura, 2016] can lead to speedups of up to 96%. Moreover, we have developed new reduced-exponent formats that can improve performance even more with further speedups of up to 13%. Since the adaptive precision algorithm allows for rigorously controlling the loss of accuracy, all these performance gains are achieved at an accuracy comparable with that of obtained with uniform precision algorithms.

A promising perspective for further improvements is to use different sparse matrix

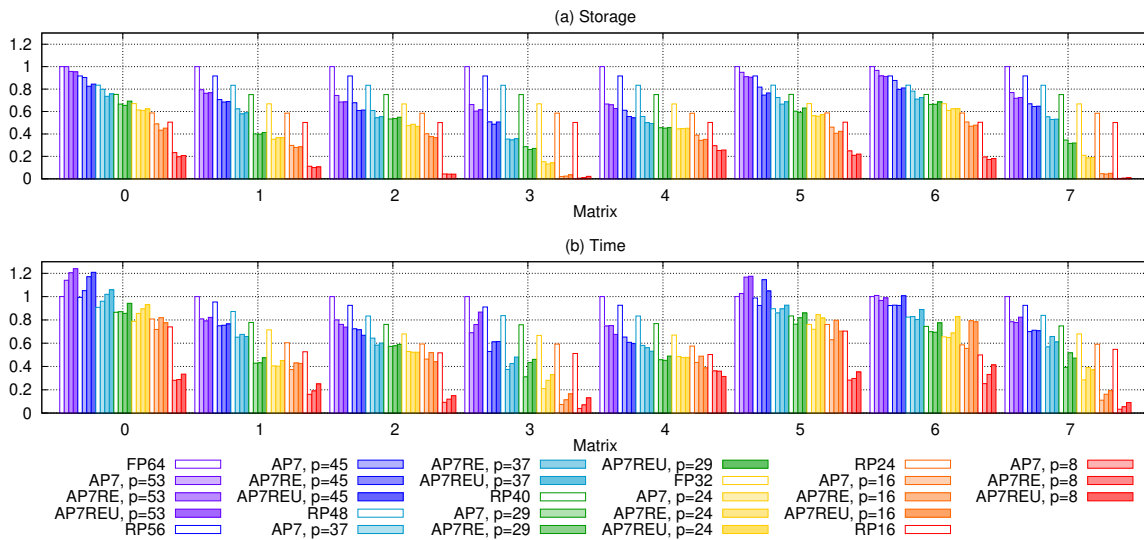


Figure 4.6: Storage and time gains achieved by AP7RE and AP7REU variants over the uniform precision and AP7 ones (normalized by the FP64 cost)

formats with a reduced relative weight of the indices, which would significantly increase the potential of adaptive precision. This is in particular the case of diagonal or block sparse formats.

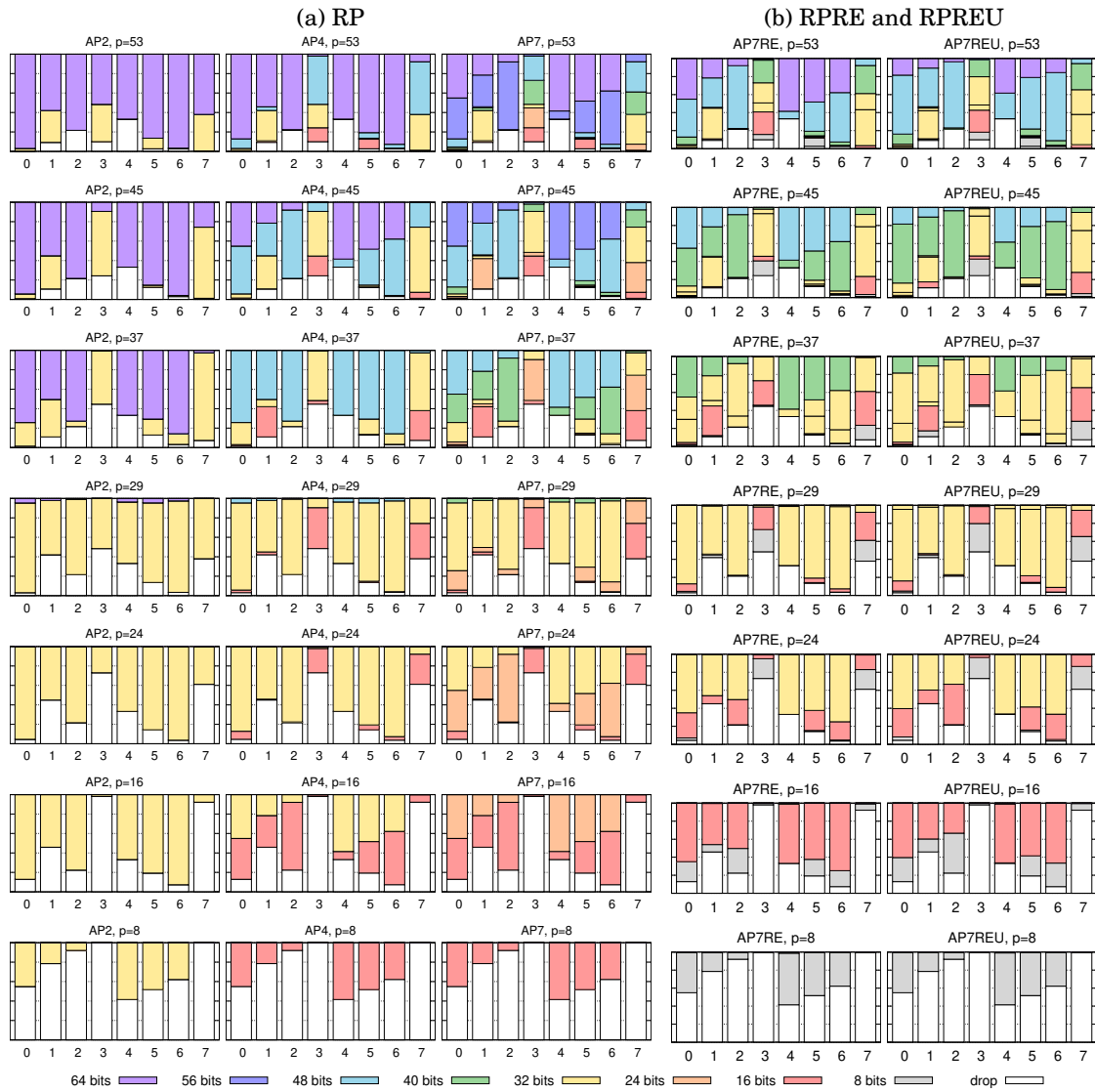


Figure 4.7: Distribution of the precision formats used for each nonzero element. Each bar on the x-axis corresponds to a different matrix and the y-axis indicates the percentage of nonzero elements stored in each format.

Mixed precision for AGATA Pulse-Shape Analysis

The AGATA project aims to build a High Purity Germanium (HPGe) gamma-ray spectrometer consisting of 180 crystals divided into 36 segments. Each gamma-ray interaction with the germanium crystal produces a net-charge signal within the hit segment and transient signals in neighbour segments. During the Pulse-Shape-Analysis (PSA), these signals are compared with a database of signals to locate the interaction point. This step has to be both accurate because the resolution of the detection relies on it and fast because the input data flow is beyond the capabilities of recording it. In the execution chain leading to the PSA, we can observe successive data conversions: the original 14-bit integers given by the electronics are finally converted to 32-bit FP32. This observation makes us wonder about the actual numerical accuracy of the results, and we investigated the use of shorter floats, with the hope of speeding up the computation, and reducing the number of cache accesses. In this chapter, we present the numerical validation of the PSA code, realized with the CADNA library. We also present how we have refactored the code to mix different numerical formats, using high precision only when necessary. Finally we explain how we benefited from GPUs speedup and native FP16 through a CUDA rewriting of the code.

1 Introduction

AGATA¹ (Advanced GAMMA-ray Tracking Array) is a European research project with the objective of building a 4π detector with significantly higher resolution than existing ones. The detector is still under development, but is already in use in its partial configuration. It is used for experiments with intense stable and radioactive ion beams, to study the nucleus structure. In the present work we deal with two connected problems: the uncertainty of the results in numerical computation due to the floating-point arithmetic that we will refer to as accuracy and the need to accelerate the processing of data

¹<https://www.agata.org>

coming from the detector that is too important to be reasonably stored. We will explore how these aspects are interconnected through the precision formats used, upon which both the performance and the accuracy of the computations depend.

Today's computers are usually using floating-point operations to simulate computations 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 lower precision formats, FP16 and BFLOAT16 (both called half precision) are more and more available in hardware in the last years, in particular on GPUs.

Applications that require results in high accuracy and that deal with data of various magnitudes usually use the FP64 and FP32 formats, whose performances have been optimized in hardware and embedded in development practices. Those formats enable results with respectively 15 and 7 significant decimal digits, and allow to deal with a large range of values. In the counterpart, they are expensive in terms of execution time, energy and memory. Under the influence of low-accuracy applications as machine-learning, half precision (FP16, BFLOAT16) and even lower precision formats are undergoing a rapid development, with dedicated hardware delivering excellent performance. On NVIDIA A100 GPUs, for instance, 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 particle and nuclear physics. To combine these two needs, mixed precision approaches are being developed leveraging the strategic use of various precision formats within the same code to benefit from their respective advantages.

However, because floating-point arithmetic relies on approximations that accumulate as computations proceed, uncontrolled computations can yield completely erroneous results. We therefore need to control the accuracy of our computations, especially when we intend to mix several precisions. In the present work we are using a tool named CADNA which allows us to monitor the evolution of the number of exact significant digits in our code and detect numerical instabilities.

In this chapter we will present our results on the stability of the code of the AGATA Pulse-Shape Analysis by comparing the results obtained with various precision formats, including stochastic types from CADNA. We will also present the results obtained using different variants of the PSA algorithm in both CPU and GPU. All experiments were carried out using the code and data available on the PSA Performance Tests repository ².

2 Profiling of AGATA computations

2.1 Gridsearch configuration

PSA is a crucial step in the processing of AGATA data and is therefore the subject of sustained development, with different algorithms being developed and different versions of these algorithms being used. In our work, we focused on the gridsearch algorithm, which compares the measured signal with the signals in the database. As we saw in Chapter 2, there are several versions of gridsearch: a full gridsearch (FGS) version 1, which searches the entire grid of the interaction segment, and a coarse-fine gridsearch (CFGS)

²https://gitlab.in2p3.fr/IPNL_GAMMA/narval_emulator/-/wikis/PSA-Performance-Tests

version 2, which performs a first minimum search on a coarse grid and then a second search on a fine grid in the neighbourhood of the coarse minimum previously found. This latest version is currently in production. In addition, the gridsearch implementation, in both full and coarse-fine versions, includes a look-up table (LUT) to avoid calculating the power p of the Figure of Merit. Meaning that in Algorithm 1 and Algorithm 2 we replace

```
1: FOMi += (Vm[s][t] - Vr[i][s][t])p
```

by

```
1: float D = Vm[s][t] - Vr[i][s][t]
2: int I = int(D)
3: FOMi += metric[I]
```

where `metric` is an array pre-calculated according to the chosen metric, i.e. the value of p . This LUT has a size of 2MB which is negligible compared to the storage of the signal base (~ 800 MB). Note that the cast operation from float to int is extremely brutal in terms of loss of precision. Furthermore, this loss of accuracy is invisible to CADNA, which considers the `metric[idiff]` value to be exact, whereas it is the result of a computation that is not, and an addressing that neither is. In our experiments, we therefore carried out a run with and a run without the LUT whenever possible. As we will see later, the use of the LUT has a positive effect on performance, but a negative effect on accuracy. In the remainder of this chapter, we will systematically specify whether the algorithm is performed with or without the LUT in the results we present.

2.2 Performance analysis

The code that processes the data of AGATA 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, performed on the CFGS in FP32 (CFGS-FP32) using the LUT, 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 of these factors make this function, and the PSA that includes it, a prime target for accelerating the code of AGATA. The number of memory accesses, due to the fact that we are processing a very large database, led us to seek ways to reduce this amount of data.

In addition, we observed successive data conversions in the execution chain leading to the PSA. Data coming from the detector is 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 references made by the PSA by reducing the volume of data to be accessed and take advantage of

³<https://perf.wiki.kernel.org>

the arithmetics of modern architectures, while controlling the numerical quality of our results.

2.3 Accuracy control

As presented in Chapter 1, standardized representation of a normalized floating-point number 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 $1 \leq m < \beta$. 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 work, 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 significand and the exponent, as described in Table 1.3.

In this context, each operation implies a rounding that can accumulate and lead to a completely false result as we have seen in Chapter 1. Stochastic arithmetic [Vignes, 1993] is a probabilistic method that allows to control the accuracy of numerical results. It relies on performing each arithmetic operation several times with a random rounding mode: each result being randomly rounded up or down with the same probability. The number of exact significant digits can therefore be estimated using a Student's t-test [Student, 1908] with a confidence level of 95 %.

CADNA⁴ [Chesneaux, 1990; Jézéquel & Chesneaux, 2008; Eberhart *et al.*, 2015] is a library that implements stochastic arithmetic by performing synchronously each computation three times to estimate the number of exact significant digits of a floating-point number resulting from numerical computation. This library provides new numerical types, called *stochastic types* on which numerical validation is automatically performed. CADNA is available in, C, C++ and Fortran and is a relatively lightweight tool with a memory overhead of 4 and an execution time overhead of around 10. As it also provides the overloading of all arithmetic operations and mathematical functions, its use generally requires minimal code rewriting. Furthermore, the fact that operations are performed synchronously enables the detection of numerical instabilities, usually due to operands with no exact significant digits.

In our work, the use of CADNA is particularly relevant. Indeed, CADNA supports MPI, OpenMP, CUDA and vectorized codes, but also half precision [Jézéquel *et al.*, 2021], whether native or simulated that is particularly attracting in an High Performance Computing (HPC) context. As previously mentioned, we had to eliminate the use of the LUT for those evaluations.

The FGS algorithm consists in comparing signals from a database with a measured signal to determine which is the most similar and thus find the point of interaction. We compare traces sampled over 40 time steps on an average of 1314 points per segment, which exposes us to a significant risk of accumulating catastrophic cancellations. Using CADNA, we were able to test the accuracy of the PSA in its FGS version in FP32, as well as different variants and combinations of precisions. To do so, we have instrumented the PSA code so that it is easy to use different precision settings and also stochastic types.

We have analyzed the version of the code implementing the FGS and we observed that the code was slightly sensitive to perturbations. Indeed, using whether stochastic

⁴<http://www.lip6.fr/cadna>

FP32, stochastic FP64, or classic FP64, instead of the original classic FP32 in the FGS, results in 0.02 % difference in the points found. This first analysis validates the original results and allows us to consider further reduction of the precision formats. In the following, we consider the full gridsearch version running in FP32, denoted FGS-FP32 and the results derived from it as a reference result.

3 Reduced precision formats

Having confirmed the numerical quality of the results obtained by the PSA, we sought to use reduced-precision formats within it. Throughout this section, we present the results obtained through different configurations but always without using the LUT.

The FP16 format, has been introduced in the 2008 revision of the IEEE754 standard [IEEE Computer Society, 2008]. It enables one to perform floating-point operations on 16 bits only. These 16 bits are distributed as follows: 1 sign bit, 5 exponent bits, and 10 + 1 significand bits. The low number of exponent bits implies a reduced range of representable data: ± 65504 . Similarly, while the number of significand bits remains reasonable and enables numbers to be represented with an accuracy of the order of 10^{-5} , this is still well below that achieved by the FP32, of the order of 10^{-7} . These two constraints make FP16 unsuitable for certain contexts.

This format is becoming increasingly widely available, in particular on recent GPUs. It is driven by the dynamism and massive investment in neural network computation which allows for low accuracy. Therefore, it is poised to continue evolving, making it a central element in performance gains for years to come. Indeed, it does not only reduce storage and energy consumption, it also delivers significant speed-up. For instance, a computation performed in FP16 instead of FP32 can be accelerated by up to $\times 16$ using A100 GPUs.

We therefore sought to take advantage of this format despite the need for high accuracy in the context of nuclear physics, and in this case the execution of PSA. In the case of the PSA, this need for high accuracy translates into the need to identify the points of interaction of gamma-rays with Germanium crystal with a resolution of 5 mm, in order to be able to carry out the gamma-ray tracking step afterwards. Table 5.1 summarizes the results in terms of point similarity between the different versions, using the FGS-FP32 as a reference, of the experiments detailed below.

Table 5.1: Signals identified identically to FGS-FP32 result without LUT (%)

	FP32	FP16	MIXED
FGS	100	100	–
CFGS	91.78	85.96	91.45

3.1 Half-precision computation

To experiment the possibility of using a reduced precision format, we used a library⁵ that emulates FP16 half precision. As the FP16 are emulated, this experiment does not

⁵<https://half.sourceforge.net>

intend into providing execution time results but only to compare the final accuracy of the results.

Thanks to our initial instrumentation and the library mentioned above, we have been able to run the FGS algorithm in emulated FP16 (FGS-FP16). This yields to 100% similarity with the results obtained by the reference. We also tested the use of FP16 on the coarse-fine gridsearch version (CFGS-FP16), which showed similar results to the FGS-FP32 for 85.96 % of the experimental signals. Table 5.1 summarizes the rates of similar points obtained by the different configurations with respect to the FGS-FP32 results.

The results obtained using the FGS-FP16 are highly satisfying, as it indicates that, in the context of the fullgrid search algorithm, FP16 can be used in place of FP32 without any loss of quality. This result is promising and prompted us to develop a GPU implementation of PSA to take advantage of the performance gains of native FP16, as we will see in 4.4.

The results obtained by the CFGS-FP16 show how difficult it is for the CFGS algorithm to maintain high accuracy when carried out in reduced precision. However they require a more detailed analysis, since we need to assess the distance between the points found and those identified by FGS-FP32, as well as the energy levels at which these errors occur. We develop this analysis in 3.3.

3.2 Mixed precision computation

Mixed precision computation is an emerging paradigm that aims at mixing several precisions in a same code with the goal of getting the performance benefits from 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 instance that in an addition between two numbers of different magnitudes, most of the significant digits of the smaller one may be eliminated. The main challenge for this programming mode is to decide in which precision each operation shall be performed. This work takes place in the area of precision tuning that aims at using a tool, here CADNA, 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 gridsearch. 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 CFGS in FP32 (CFGS-FP32) identifies 91.78% similarly to FGS-FP32 while the mixed precision version (CFGS-MIXED) obtains 91.45 %. These similar results indicate that the two methods can be adopted alternatively with equal confidence. Of course, we could also perform both steps in low precision (CFGS-FP16), but under the same conditions, this results in only 85.96 % of the signals identified identically to FGS-FP32 as already mentioned in section 3.1. In the current implementation of the CFGS, the coarse step represents around 20 % of the computation time. This is a limit to the time savings that can be expected without change of the grid size to increase the weight of the coarse part. However, any modification of the grid may lead to a reduction in accuracy and should therefore be carried out carefully. These preliminary results confirm the interest of mixed precision for applications to improve the performance while maintaining the accuracy and reinforcing our commitment to continue in

this direction.

3.3 Error acceptability

In this work, we encountered various methods for analyzing computation errors. Initially, we focused on numerical accuracy, assessed in terms 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. Since our criterion for evaluating the accuracy of results is the number of points identified similarly and the energies involved between a method and our reference, using an autotuning tool like PROMISE, which takes as its metric only the number of exact significant digits, is not suitable.

In this section we try to measure how different the points found are and in which energy class they lie. To this end, we produced Figures 5.1 and 5.2. In Figure 5.1 we can observe that most of the points found differently from the full gridsearch FP32 method are less than 3 mm far from the original points and that the number of points found differently from the full gridsearch FP32 points decreases with distance. Moreover, as we can observe on Figure 5.2, 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 gridsearch 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 gridsearch.

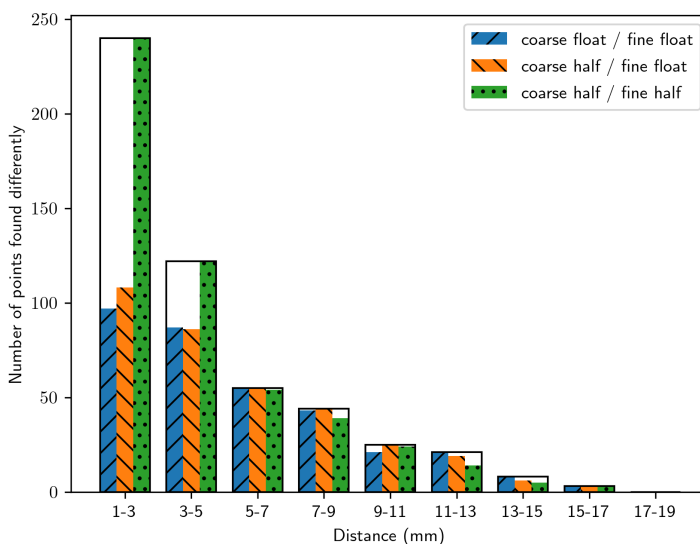


Figure 5.1: Distances between points found by the full gridsearch FP32 algorithm and alternative methods

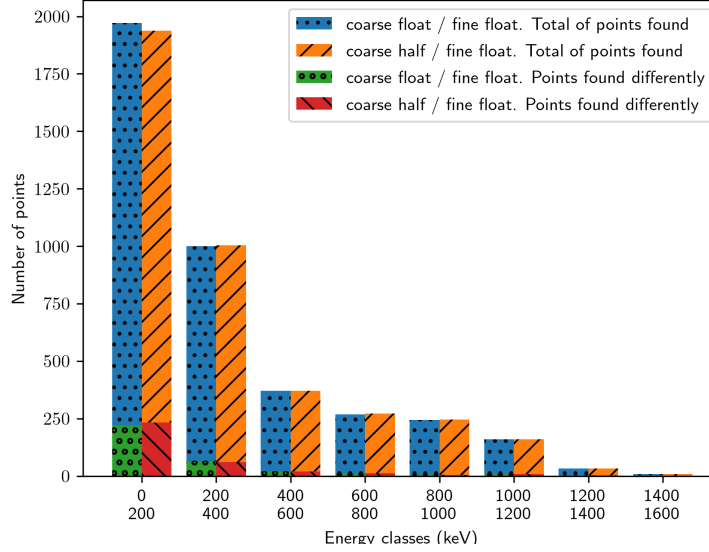


Figure 5.2: Energy classes in which the points found by various methods belong to

4 Adapting the code to modern hardware

4.1 The PSA test environment

To go further and obtain execution time evaluations, we decided to work on adapting the gridsearch to modern hardware. To be able to easily experiment the impact of the PSA and its parameters, we extracted it from the complete AGATA execution chain and turned it into a stand-alone application⁶. We then adopted modern C++ conventions before building several variants of this application, using the FGS 1 and CFGS 2 algorithms, with different precisions (FP32 or FP16), using or not the LUT, and running on CPU or GPU.

As a reminder, the gridsearch is based on the minimization of the following Figure of Merit.

$$\text{FOM}_{i+} = \sum_{s,t} (V_m[s][t] - V_r[i][s][t])^p \quad (5.1)$$

Where i is the grid point, s designates the segment number and t the time sample. V_m is the measured signal and V_r the basis of reference signals. p is a positive real number that corresponds to the chosen metric.

This FOM allows us to take into account signals measured in different segments: hit segment and core signals, as well as signals in segments adjacent to the interaction segment. To make the PSA test environment as simple as possible, we chose to consider only the signals from two channels: the hit segment and the core. This corresponds to the situation observed when transient signal values are not available. We also set the value of p to 0.3, as currently implemented. Our experimental code makes it easy to add neighbors or modify the value of p . We performed our experiments on a sample of 5342

⁶<https://gitlab.com/romeomolina/psa-test-env.git>

events with energies ranging from 15keV to 5MeV.

We decided to express the performances of the different versions in terms of the cost of processing a single event. The time scale is therefore very small, and we chose to measure it in ticks. The CPU code was compiled by the G++ 9.4.0 compiler using `-O3 -std=c++17` and the GPU code by the NVCC 12.4.0 compiler using `-O3 -std=c++17`. We carried out the CPU experiments on an Intel® Core™ i9-11950H Processor with 8 cores at 2.6GHz with 24MB cache memory and the GPU experiments on a NVIDIA RTX A2000 with 3328 CUDA cores and 4GB memory. Comparing CPU and GPU performance provides limited information, as the results obviously depend on the hardware chosen. However it allows us to offer an first assessment of GPU performance and validate their use in the PSA framework. Our reference result is that achieved by the FGS without the LUT on CPU giving an average execution time per event of 624 ticks. For the other experiments, we measure their execution time and the number of points found identically to this reference. We also measure the number of points found at a distance below 5 mm, since this corresponds to the AGATA specification.

Results on both GPU and CPU with the different configurations are summarized in Tables 5.2 and 5.3 and discussed below.

Table 5.2: Execution time for the different configurations on CPU and GPU (ticks)

	CPU-FP32	GPU-FP32	GPU-FP16
FGS-NOLUT	624	55	52
FGS-LUT	97	51	–
CFGS-NOLUT	102	–	–
CFGS-LUT	17	–	–

Table 5.3: Points identified within 5mm of those found by FGS-FP32 without the LUT executed in CPU (%)

	CPU-FP32	GPU-FP32	GPU-FP16
FGS-NOLUT	100	100	94
FGS-LUT	90	90	–
CFGS-NOLUT	72	–	–
CFGS-LUT	68	–	–

4.2 Experiments on CPU

The FGS algorithm using the LUT takes only 97 ticks on average, but identifies only 76% of the points identically. If we accept points at a distance of less than 5 mm, as this is the resolution required for the ray-tracking, we obtain 90% of points found similarly. If we consider the distance between the points identified with the LUT and those without it, we observe that the number of points found decreases in inverse proportion to this distance. Thus, less than 2% of points are found more than 20 mm apart.

The CFGS executed on CPU delivers significant performance gains. For example, the CFGS using no LUT achieves an average runtime of 102 ticks for 68% of points found

identically to the reference and 72% within 5mm. However, this strategy results in 8% of points identified more than 20 mm from the reference result.

It is possible to combine the CFGS with the use of the LUT to reduce the average execution time to 17 ticks, but this leads to a further deterioration in the quality of the results, with only 54% of points found identically to the reference, 68% within 5 mm and still 8% more than 20 mm away.

4.3 GPU deployment

In recent years, Graphics Processing Units (GPUs) have become an essential aspect of HPC. These massively parallel computing units, initially developed for graphics display management, have now made their mark in fast-developing sectors such as neural networks. Their performance, which saves energy and time compared with CPU-based operations, is encouraging developers to turn to this solution.

However, this developing technology is less standardized than the CPU environment, and highly dependent on the manufacturer chosen. In our case, we decided to use CUDA, the language developed by NVIDIA to program its GPUs that are the dominant GPU technology today. This language is based on C++ and is therefore suited to our context. Furthermore, CADNA is able to deal with CUDA programs.

As the massively parallel GPU architecture is completely different from that of a CPU, a great deal of rewriting was required, both on the data structures and on the computation, in order to take advantage of the GPU performance. In particular, the interest of the coarse-fine gridsearch disappears with the use of GPU. This is because in this case, unlike in CPU execution, FOMs are calculated simultaneously for all points on the grid, before being reduced to determine the smallest. Thus, using the coarse-fine algorithm would imply an intermediate reduction step followed by an additional broadcast to all the cores in use, which would be particularly costly in the massively parallel context. The computational strategy therefore consists in calculating all the FOMs for each grid point in parallel, then performing a single reduction to identify the minimum.

This involves GPU memory transfers that can be crucial to the final execution time. First of all, the entire reference signal base must be copied into GPU memory. This base is stable over time, and we do not want to transfer it block by block for each event. We therefore need sufficient memory to store it entirely, i.e. 848 MB that is available on our NVIDIA RTX A2000 with 4 GB of memory.

Executing the algorithm as modified to run on the GPU (FGS-GPU) without the LUT, we obtained exactly the same results as our CPU reference, i.e. 100% of points identified identically to the FGS without the LUT, when reducing the execution time to 55 ticks. While this execution time is still longer than that obtained with the coarse-fine version and the LUT, it should be set against the accuracy of the results obtained, compared with only 54% of points identified identically in the previous case.

In addition, it should be pointed out that there are still optimization possibilities. As it stands, FOM is calculated per base point (700-2000 according to the segment) on each available GPU core. However, A6000 GPU cards for instance have 10752 CUDA cores, making it possible to simultaneously process from 5 to 15 events and effectively using all the cores. This could theoretically reduce as much the execution time. The use of GPUs to carry out the PSA therefore appears to be an important opportunity to improve its

performance while maintaining the quality of the results obtained.

4.4 Use of native FP16

GPUs are used in a context in which not all computations require high accuracy. Indeed, it has been shown that low-precision computations do not usually break the validity of neural network learning [Yun *et al.*, 2023]. The development of GPUs has therefore been accompanied by significant progress in the support of reduced precisions, including both integer and floating-point formats. In particular, there are currently two relatively widespread floating-point formats on GPUs: the FP16 format, derived from the 2008 revision of the IEEE 754 standard, and the BFLOAT16 format developed by Google for its TensorCore. As we saw above, using FP16 keeps the results close to those obtained with FP32.

On current NVIDIA GPUs, FP16 precision can be used with two types available in CUDA: `half` and `half2`. These two types work in a complementary way. The `half` type enables FP16 values to be stored contiguously, while the `half2` type enables this data to be accessed for pairwise processing, as in the case of vectorized execution.

The use of half precision on GPU (FGS-FP16) without the LUT enables us to obtain 87% of points similar to our reference and 94% within 5 mm of the reference, for only 1% of points identified more than 20 mm from the reference. It therefore appears that this solution delivers more accurate results than the FGS versions in FP32 using the LUT. We obtained an average execution time of 52 ticks, which is better than the GPU result obtained with FP32, but we could have expected a greater speedup. This difference in performance can probably be explained by the costs of pre- and post-processing required for half computation. Indeed, the potential of the half execution, that essentially corresponds to a vectorized execution is affected by the need of obtaining a single value. Similarly, the use of the LUT in this context is not appropriate, as it would break any possibility of vector execution.

5 Conclusion and perspectives

In the course of this work, we conducted investigations into the quality of the results produced by the PSA in its various existing and new configurations. We instrumented the PSA code to modify the precision formats used. This enabled us to use the CADNA tool, which, with the help of stochastic arithmetic, enabled us to evaluate the number of exact significant digits obtained which numerically validated the results provided by the PSA.

On the one hand, we performed an experimental evaluation using emulated FP16, which demonstrated that a mixed precision version of the coarse-fine gridsearch (CFGS) – using FP16 in the coarse step and FP32 in the fine step – was the most suitable strategy. This strategy enabled us to obtain very similar results to those provided by the CFGS in uniform precision FP32. We were also able to verify that points found differently from the full gridsearch (FGS) FP32 algorithm were at a small distance and very similar to those found by the coarse-fine FP32 gridsearch. Finally, we examined the energy ranges in which the different points were found and observed the same similarities and a distribution primarily in the low energies that require less accuracy.

On the other hand, we conducted a GPU evaluation of the PSA optimization potential. To do this, we extracted the PSA code and compared results and execution times. First, we evaluated the existing versions on CPU using FP32: the four possible configurations between full and coarse-fine gridsearch and between using or not using the look-up table (LUT), taking the FGS without the LUT as a reference. We then developed a CUDA code to perform the PSA on GPU in both FP32 and FP16 precisions. As things stand, the fastest version remains the CPU coarse-fine version using the LUT at 17 ticks. However it only identifies points similarly to the reference in 54% of cases, compared with 100% for the FP32 GPU (55 ticks) and 87% for the FP16 GPU (52 ticks), both not using the LUT. Furthermore, there are still performance-enhancing opportunities to take greater advantage of GPUs by increasing their utilization by a factor of up to $15\times$.

In this work, we faced a difficulty, since the accuracy result we were seeking was not a floating-point number, but a series of discrete points that we wanted to be as close as possible to the actual points of interaction. This observation leads us to propose, as a future direction, the development of numerical validation and autotuning tools based on accuracy criteria chosen by the user. In our case, this would be a criterion considering the number of points found similar to the reference, with a weighting corresponding to the energies involved. Another interesting line of research about the PSA enhancement would consist in refining the hierarchy of coarse and fine grids in the coarse-fine gridsearch algorithm, 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. Such a change could, however, negatively affect the accuracy, and therefore requires special attention. Additionally, since the use of the LUT seems to be particularly beneficial, it would be interesting to try and improve its accuracy.

Conclusion & Perspectives

The work presented in this manuscript includes both singular and general aspects.

General first, because the aim is to propose mixed precision solutions, i.e. using several numerical formats in the same code, while controlling the accuracy of the results. This is a general objective, as the number of precision formats supported by hardware is increasing, while the validity of the results still needs to be guaranteed. General, then, in the multiplicity of approaches we have used: development of an adaptive precision numerical linear algebra algorithm based on error analysis, but also tuning of an existing code based on results validated by a probabilistic approach; experiments on CPUs, but also explorations of massively parallel GPU architectures; development of custom formats, but also exploitation of the performance of the FP16 format on GPUs.

Singular then, because it has focused on a specific algorithm, the Sparse Matrix-Vector Product, and on the code of a specific experiment, AGATA, and especially the Pulse-Shape Analysis.

We will first briefly review the main results we have obtained, before proposing some research perspectives for the continuation of our work.

6 Conclusion

First of all, as presented in Chapter 3, we have developed a Sparse Matrix-Vector Product (SpMV) algorithm in adaptive precision. Adaptive precision is a subclass of mixed precision algorithms that aims to adapt the precision to the data at hand. In our case, it involves processing the matrix and sorting the elements into different buckets according to their magnitude. Based on a rigorous error analysis, we were able to determine buckets construction criteria that guarantee a normwise or componentwise error on the results, while minimizing the precision formats used. The resulting algorithm is capable of targeting any accuracy, including non-standard ones, but also to utilize a wide variety of precisions, whether available in hardware or not. The relevance of this algorithm has been validated experimentally, showing not only that it does indeed target the desired accuracies, but also that it delivers significant memory gains (up to a factor of $36\times$). In addition, we integrated the adaptive SpMV into various Krylov solvers and found that it did not affect their convergence. On the contrary, the intermediate accuracies that can

be targeted open up new opportunities for iterative refinement versions of these solvers. Finally, we have seen that memory gains translate into significant speedups (up to a factor of $7\times$) using native FP32 and FP64 precisions, although they are hampered by the lack of hardware implementation of custom formats.

This observation led us to integrate accessors optimized for custom formats into the adaptive SpMV, and we turned to the accessors proposed in [Mukunoki & Imamura, 2016]. This work, recalled in Chapter 4, enabled us to validate the use of these optimized accessors in the context of adaptive SpMV with a maximum storage gain of 24% and a maximum speedup of 21% when using custom formats with optimized accessors instead of using only FP32 and FP64. It also gave us the opportunity to develop new custom formats using a reduced exponent. Indeed, we realized that the magnitude ordering imposed by the adaptive SpMV implies a lower order of magnitude variation within each bucket. More precisely, if the order of magnitude varies by less than 2^8 within each bucket, only three exponent bits are required, rather than 11 as in FP64 or 8 in FP32. To reduce the magnitude variation in a bucket to this level, it is necessary to use at least 7 buckets. We also used the fact that we could also increase the significand of these formats by one bit, by separating positive and negative elements to propose another variant. Further experimentation then enabled us to demonstrate maximum gains in storage of 16% and execution time gains of 13% compared with executions using 7 precisions without exponent reduction.

At the same time, we also sought to specifically improve the performance of the Pulse-Shape Analysis (PSA), a key element in the data processing of the AGATA Germanium gamma-ray detector as described in Chapter 5. This step consists in identifying an interaction point within a crystal exploiting the traces measured in different segments of the crystal. Our work focused on the gridsearch algorithm currently used to perform the PSA. This involves minimizing a Figure of Merit (FOM) calculated from the measured signal and a base of signals previously obtained by calibration or simulation. The minimum FOM then corresponds to the point of interaction of the gamma-ray with the Germanium crystal. This operation requires high accuracy, as the interaction points must be identified with a resolution of 5mm to enable the next step, gamma-ray tracking, which consists of reconstructing the full path of a gamma-ray in the Germanium crystal. It also has to be carried out online, as the amount of data produced during the experiment is too large to be stored.

We therefore organized our work in three steps. First, an evaluation of the existing code highlighted the numerical quality of the results obtained by the full gridsearch algorithm in FP32, but also validated the significant weight of the PSA in the AGATA execution chain, and in particular the importance of cache accesses. Secondly, we carried out an evaluation based on the different algorithmic configurations and precision formats used. Thus, we compared the results obtained by the coarse-fine version, which first performs a coarse search for the minimum before refining in the neighborhood of the point found, with those of the full version previously validated using stochastic arithmetic. These experiments also enabled us to evaluate the results obtained by the different variants in FP16 through emulation of this format, and led us to propose a mixed precision solution. This solution is based on the coarse-fine version, and consists of running the coarse search in FP16 and the fine search in FP32. The results are almost identical to a coarse-fine run fully executed in FP32, as opposed to a coarse-fine run in FP16. Finally, in the light of these positive results, we decided to carry out experiments

on GPUs, both to take advantage of the massively parallel architecture and to use native FP16 which is not available on our CPUs. This allowed us to confirm the relevance of this architecture for this application. Indeed, for results identical to those obtained with our CPU reference, the computation time is reduced by a factor of $11\times$.

Our work has therefore confirmed the benefits of using low precision on both CPU and GPU, and with both native and custom formats. It has also enabled us to validate the mixed precision approach, which enables these low precisions to be used even in contexts where high accuracy is required. Finally, it has demonstrated the complementarity of mixed precision strategies by tuning existing code and by developing algorithms specially designed to maximize the efficiency of mixed precision while controlling accuracy. Having reached the end of this manuscript, we certainly do not claim to have covered all the problems and opportunities of the future of floating-point arithmetic, but to be part of the development of a new paradigm, for which we offer a few perspectives below.

7 Perspectives

With regard to the sparse matrix-vector product in adaptive precision, the work carried out and the results obtained demonstrated the benefits of adapting the precision to the data at hand, and encourage us to continue along this path. They also show the benefits of using mixed precision not only for computation but also for storage. This means to use different custom precision formats for storage and thus speeding up data loading, which is then conventionally carried out in hardware-available formats, generally FP32 and FP64. The results obtained encourage following this mixed precision approach in memory-bound applications. The results also highlight the relative importance of index loading in the adaptive sparse matrix-vector product, which increases with the number of precisions available and can have a negative impact on results. To address this, we need to develop matrix formats that reduce the weight of indices. A first approach would be to adapt the adaptive SpMV for experimentation on diagonal or block-diagonal matrices whose representation requires almost no index storage. Finally, the work we have carried out shows that the use of mixed precision for the matrix-vector product does not affect the convergence of the Krylov solvers in which it is used. We believe it is important to continue along this path, aiming to develop solvers that use mixed precision solutions in their various steps. This could involve storing the Krylov basis in adaptive precision or using mixed precision preconditioners. In addition, the ability of the adaptive SpMV to target intermediate accuracy could be exploited within the framework of a relaxed GMRES, i.e., a GMRES that allows an increasing error on the matrix-vector product.

Regarding the Pulse-Shape Analysis, a major result of our work is that, under certain conditions, using FP16 instead of FP32 does not affect the results. The challenge lies in these certain conditions, but we must build on this initial postulate: it is possible to reduce the precision without affecting the accuracy of the results. It is also worth noting that the gridsearch code adapts well to GPU architecture, which offers very significant performance gains for identical accuracy, even with the use of native FP16. To continue this work, we need to reorganize the computations to process several events simultaneously, which would increase the occupancy of the GPU cores, suggesting a possible acceleration up to a factor $\times 15$. Furthermore, the use of GPUs needs to be considered throughout the entire execution chain, to minimize costly memory exchanges between

CPU and GPU. A complementary approach would consist in improving the accuracy provided by the look-up-table, which currently provides a fast but not very accurate solution. Finally, the work carried out on Pulse-Shape Analysis could serve as a basis for studying the entire AGATA processing chain from the point of view of accuracy and the precision formats used. Indeed, the multiple conversions, from the 14-bit signal sampling to the execution of the PSA in FP32, which turns out to be just as accurate as that performed in FP16, make it possible to consider intermediate steps with reduced precision.

From a more general perspective, we would also like to emphasize the following point. The development of low-precision and mixed precision solutions is based on three arguments: increased computational speed, reduced energy consumption and memory gain. While the gain in computing speed can be measured directly and is generally effective on CPUs since the introduction of SIMD and on GPUs, the reduction in energy consumption is far more questionable. Indeed, the assertion that the use of low precision reduces energy consumption is highly imprecise and conceals a more complex reality. It is easy to measure this reduction on a specific operation but the overall impact is far less certain. Indeed, one might naturally assume that halving energy consumption per operation leads to doubling the number of operations, or even to increase by a greater factor. Current knowledge of the environmental footprint of computing and High Performance Computing is poorly understood [Roussilhe *et al.*, 2023], particularly with regard to its indirect effects, and deserves to be seriously explored.

Bibliography

ABDULAH S., CAO Q., PEI Y., BOSILCA G., DONGARRA J., GENTON M. G., KEYES D. E., LTAIEF H. & SUN Y. (2022a). Accelerating geostatistical modeling and prediction with mixed-precision computations: A high-productivity approach with PaRSEC. *IEEE Trans. Parallel Distrib. Syst.*, **33**(4), 964–976.

ABDULAH S., LI Y., CAO J., LTAIEF H., KEYES D. E., GENTON M. G. & SUN Y. (2022b). Large-scale environmental data science with exageostat.

AHMAD K., SUNDAR H. & HALL M. (2019). Data-driven mixed precision sparse matrix vector multiplication for GPUs. *ACM Trans. Archit. Code Optim.*, **16**(4).

AKKOYUN S., ALGORA A., ALIKHANI B., AMEIL F., DE ANGELIS G., ARNOLD L., ASTIER A., ATAÇ A., AUBERT Y., AUFRANC C., AUSTIN A., AYDIN S., AZAIEZ F., BADOER S., BALABANSKI D., BARRIENTOS D., BAULIEU G., BAUMANN R., BAZZACCO D., BECK F., BECK T., BEDNARCZYK P., BELLATO M., BENTLEY M., BENZONI G., BERTHIER R., BERTI L., BEUNARD R., LO BIANCO G., BIRKENBACH B., BIZZETI P., BIZZETI-SONA A., LE BLANC F., BLASCO J., BLASI N., BLOOR D., BOIANO C., BORSATO M., BORTOLATO D., BOSTON A., BOSTON H., BOURGAULT P., BOUTACHKOV P., BOUTY A., BRACCO A., BRAMBILLA S., BRAWN I., BRONDI A., BROUSSARD S., BRUYNEEL B., BUCURESCU D., BURROWS I., BÜRGER A., CABARET S., CAHAN B., CALORE E., CAMERA F., CAPSONI A., CARRIÓ F., CASATI G., CASTOLDI M., CEDERWALL B., CERCUS J.-L., CHAMBERT V., EL CHAMBIT M., CHAPMAN R., CHARLES L., CHAVAS J., CLÉMENT E., COCCONI P., COELLI S., COLEMAN-SMITH P., COLOMBO A., COLOSIMO S., COMMEAUX C., CONVENTI D., COOPER R., CORSI A., CORTESI A., COSTA L., CRESPI F., CRESSWELL J., CULLEN D., CURIEN D., CZERMAK A., DELBOURG D., DEPALO R., DESCOMBES T., DÉSESQUELLES P., DETISTOV P., DIARRA C., DIDIERJEAN F., DIMMOCK M., DOAN Q., DOMINGO-PARDO C., DONCEL M., DORANGEVILLE F., DOSME N., DROUEN Y., DUCHÈNE G., DULNY B., EBERTH J., EDELBRUCK P., EGEE J., ENGERT T., ERDURAN M., ERTÜRK S., FANIN C., FANTINEL S., FARNEA E., FAUL T., FILLIGER M., FILMER F., FINCK C., DE FRANCE G., GADEA A., GAST W., GERACI A., GERL J., GERNHÄUSER R., GIANNATIEMPO A., GIAZ A., GIBELIN L., GIVECHEV A., GOEL N., GONZÁLEZ V., GOTTARDO A., GRAVE X., GREBOSZ J., GRIFFITHS R., GRINT A., GROS P., GUEVARA L., GULMINI M., GÖRGEN A., HA H., HABERMANN T., HARKNESS L., HARROCH H., HAUSCHILD K., HE C., HERNÁNDEZ-PRIETO A., HERVIEU B., HESS H., HÜYÜK T., INCE E., ISOCRATE R., JAWORSKI G., JOHNSON A., JOLIE J., JONES P., JONSON B., JOSHI P., JUDSON D., JUNGCLAUS A., KACI M., KARKOUR N., KAROLAK M., KAŞKAŞ A., KEBBIRI M., KEMPLEY R., KHAPLANOV A., KLUPP S., KOGIMTZIS M., KOJOUHAROV I., KORICHI A., KORTEN W., KRÖLL T., KRÜCKEN R., KURZ N., KY B., LABICHE M., LAFAY X., LAVERGNE L., LAZARUS I., LEBOUTELIER S., LEFEBVRE F., LEGAY E., LEGEARD L., LELLI F., LENZI S., LEONI S.,

LERMITAGE A., LERSCH D., LESKE J., LETTS S., LHENORET S., LIEDER R., LINGET D., LJUNGVALL J., LOPEZ-MARTENS A., LOTODÉ A., LUNARDI S., MAJ A., VAN DER MAREL J., MARIETTE Y., MARGINEAN N., MARGINEAN R., MARON G., MATHER A., ME, CZYŃSKI W., MENDÉZ V., MEDINA P., MELON B., MENEGAZZO R., MENGONI D., MERCHAN E., MI-HAILESCU L., MICHELAGNOLI C., MIERZEJEWSKI J., MILECHINA L., MILLION B., MITEV K., MOLINI P., MONTANARI D., MOON S., MORBIDUCCI F., MORO R., MORRALL P., MÖLLER O., NANNINI A., NAPOLI D., NELSON L., NESPOLO M., NGO V., NICOLETTO M., NICOLINI R., LE NOA Y., NOLAN P., NORMAN M., NYBERG J., OBERTELLI A., OLARIU A., ORLANDI R., OXLEY D., ÖZBEN C., OZILLE M., OZIOL C., PACHOUD E., PALACZ M., PALIN J., PANCIN J., PARISEL C., PARISSET P., PASCOVICI G., PEGHIN R., PELLEGGRI L., PEREGO A., PERRIER S., PETCU M., PETKOV P., PETRACHE C., PIERRE E., PIETRALLA N., PIETRI S., PIGNANELLI M., PIQUERAS I., PODOLYAK Z., LE POUHALEC P., POUTHAS J., PUGNÈRE D., PUCKNELL V., PULLIA A., QUINTANA B., RAINE R., RAINOVSKI G., RAMINA L., RAMPAZZO G., LA RANA G., REBESCHINI M., RECCHIA F., REDON N., REESE M., REITER P., REGAN P., RIBOLDI S., RICHER M., RIGATO M., RIGBY S., RIPAMONTI G., ROBINSON A., ROBIN J., ROCCA Z., ROPERT J.-A., ROSSÉ B., ROSSI ALVAREZ C., ROSSO D., RUBIO B., RUDOLPH D., SAILLANT F., ŞAHIN E., SALOMON F., SALSAC M.-D., SALT J., SALVATO G., SAMPSON J., SANCHIS E., SANTOS C., SCHAFFNER H., SCHLARB M., SCRAGGS D., SEDDON D., ŞENYİĞİT M., SIGWARD M.-H., SIMPSON G., SIMPSON J., SLEE M., SMITH J., SONA P., SOWICKI B., SPOLAORE P., STAHL C., STANIOS T., STEFANOVA E., STÉZOWSKI O., STRACHAN J., SULIMAN G., SÖDERSTRÖM P.-A., TAIN J., TANGUY S., TASHENOV S., THEISEN C., THORNHILL J., TOMASI F., TONIOLO N., TOUZERY R., TRAVERS B., TRIOSI A., TRIPON M., TUN-LANOË K., TURCATO M., UNSWORTH C., UR C., VALIENTE-DOBON J., VANDONE V., VARDACI E., VENTURELLI R., VERONESE F., VEYSSIERE C., VISCIONE E., WADSWORTH R., WALKER P., WARR N., WEBER C., WEISSHAAR D., WELLS D., WIELAND O., WIENS A., WITWER G., WOLLERSHEIM H., ZOCCA F., ZAMFIR N., ZIEBLIŃSKI M. & ZUCCHIATTI A. (2012). AGATA—Advanced GAMMA Tracking Array. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, **668**, 26–58.

ALIAGA J. I., ANZT H., GRÜTZMACHER T., QUINTANA-ORTÍ E. S. & TOMÁS A. E. (2022). Compressed basis GMRES on high-performance graphics processing units. *Int. J. High Perform. Comput. Appl.*, p. 10943420221115140.

AMESTOY P., BOITEAU O., BUTTARI A., GEREST M., JÉZÉQUEL F., L'EXCELLENT J.-Y. & MARY T. (2022). Mixed precision low-rank approximations and their application to block low-rank LU factorization. *IMA Journal of Numerical Analysis*, **43**(4), 2198–2227.

AMESTOY P., BUTTARI A., HIGHAM N. J., L'EXCELLENT J.-Y., MARY T. & VIEUBLÉ B. (2021). *Five-Precision GMRES-based Iterative Refinement*. MIMS EPrint 2021.5, Manchester Institute for Mathematical Sciences, The University of Manchester, UK. Revised April 2022.

AMESTOY P., BUTTARI A., HIGHAM N. J., L'EXCELLENT J.-Y., MARY T. & VIEUBLÉ B. (2023). Combining sparse approximate factorizations with mixed-precision iterative refinement. *ACM Trans. Math. Softw.*, **49**(1).

ANZT H., DONGARRA J., FLEGAR G., HIGHAM N. J. & QUINTANA-ORTÍ E. S. (2019). Adaptive precision in block-Jacobi preconditioning for iterative sparse linear system solvers. *Concurrency Computat. Pract. Exper.*, **31**(6), e4460.

BALABANOV O. & GRIGORI L. (2022). Randomized Gram–Schmidt Process with Application to GMRES. *SIAM J. Sci. Comput.*, **44**(3), A1450–A1474.

BAUER F. L. (1974). Computational graphs and rounding error. *SIAM Journal on Numerical Analysis*, **11**(1), 87–96.

BECK F. (1992). EUROBALL: Large gamma ray spectrometers through european collaborations. *Progress in Particle and Nuclear Physics*, **28**, 443–461.

- BEISER A. (2003). *Concepts of Modern Physics*. McGraw-Hill, 6 edition.
- BEN KHALIFA D., MARTEL M. & ADJÉ A. (2020). POP: A tuning assistant for mixed-precision floating-point computations. In O. HASAN & F. MALLET, editors, *Formal Techniques for Safety-Critical Systems*, p. 77–94: Springer International Publishing.
- BLAS TECHNICAL FORUM (2001). Basic Linear Algebra Subprograms Technical Forum Standard. *The International Journal of High Performance Computing Applications*.
- BOSTON, A. J., CRESPI, F. C. L., DUCHÊNE, G., DÉSESQUELLES, P., GERL, J., HOLLOWAY, F., JUDSON, D. S., KORICHI, A., HARKNESS-BRENNAN, L., LJUNGVALL, J., QUINTANA-ARNÉS, B., REITER, P. & STEZOWSKI, O. (2023). AGATA characterisation and pulse shape analysis. *Eur. Phys. J. A*, **59**(9), 213.
- CARSON E. & HIGHAM N. J. (2017). A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems. *SIAM J. Sci. Comput.*, **39**(6), A2834–A2856.
- CARSON E. & HIGHAM N. J. (2018). Accelerating the solution of linear systems by iterative refinement in three precisions. *SIAM J. Sci. Comput.*, **40**(2), A817–A847.
- CATTANEO D., CHIARI M., AGOSTA G. & CHERUBIN S. (2022). TAFFO: The compiler-based precision tuner. *SoftwareX*, **20**, 101238.
- CHERUBIN S. & AGOSTA G. (2020). Tools for reduced precision computation: A survey. *ACM Comput. Surv.*, **53**(2).
- CHESNEAUX J.-M. (1990). Study of the computing accuracy by using probabilistic approach. In C. ULLRICH, editor, *Contribution to Computer Arithmetic and Self-Validating Numerical Methods*, p. 19–30, IMACS, New Brunswick, New Jersey, USA.
- CONNOLLY M. P., HIGHAM N. J. & MARY T. (2021). Stochastic rounding and its probabilistic backward error analysis. *SIAM J. Sci. Comput.*, **43**(1), A566–A585.
- DAMOUCHE N. & MARTEL M. (2018). Salsa: An automatic tool to improve the numerical accuracy of programs. In *AFM@NFM*.
- DAVIS T. A. & HU Y. (2011). The University of Florida Sparse Matrix Collection. *ACM Trans. Math. Softw.*, **38**(1).
- DELEPLANQUE M., LEE I., VETTER K., SCHMID G., STEPHENS F., CLARK R., DIAMOND R., FALLON P. & MACCHIAVELLI A. (1999). GRETA: utilizing new concepts in γ -ray detection. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, **430**(2), 292–310.
- DEMME J. & HIDA Y. (2004). Accurate and efficient floating point summation. *SIAM Journal on Scientific Computing*, **25**(4), 1214–1248.
- DENIS C., DE OLIVEIRA CASTRO P. & PETIT E. (2016). Verificarlo: checking floating point accuracy through Monte Carlo Arithmetic. In IEEE, editor, *2016 IEEE 23rd Symposium on Computer Arithmetic (ARITH)*, Santa Clara, United States.
- DIFFENDERFER J., OSEI-KUFFUOR D. & MENON H. (2021). QDOT: Quantized dot product kernel for approximate high-performance computing.
- DOUCET G., LEE W. & FRANGOU S. (2019). Evaluation of the spatial variability in the major resting-state networks across human brain functional atlases. *Human Brain Mapping*, **40**.
- EBERHART P., BRAJARD J., FORTIN P. & JEZEQUEL F. (2015). High performance numerical validation using stochastic arithmetic. *Reliable Computing*, **21**.

- FARNEA E., RECCHIA F., BAZZACCO D., KRÖLL T., PODOLYÁK Z., QUINTANA B. & GADEA A. (2010). Conceptual design and Monte Carlo simulations of the AGATA array. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, **621**(1), 331–343.
- FASI M., HIGHAM N. J., LOPEZ F., MARY T. & MIKAITIS M. (2023). Matrix multiplication in multiword arithmetic: Error analysis and application to GPU tensor cores. *SIAM Journal on Scientific Computing*, **45**(1), C1–C19.
- FLEGAR G., ANZT H., COJEAN T. & QUINTANA-ORTÍ E. S. (2021). Adaptive precision block-jacobi for high performance preconditioning in the ginkgo linear algebra software. *ACM Trans. Math. Softw.*, **47**(2).
- FRECHTLING M. & LEONG P. H. W. (2015). MCALIB: Measuring sensitivity to rounding error with Monte Carlo programming. *ACM Trans. Program. Lang. Syst.*, **37**(2).
- GOLDBERG D. (1991). What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, **23**(1), 5–48.
- GOUBAULT E. & PUTOT S. (2006). Static analysis of numerical algorithms. *International Static Analysis Symposium*, p. 18–34.
- GRAILLAT S., JÉZÉQUEL F., MARY T. & MOLINA R. (2024a). Adaptive precision sparse matrix-vector product and its application to Krylov solvers. *SIAM Journal on Scientific Computing*, **46**(1), C30–C56.
- GRAILLAT S., JÉZÉQUEL F., MARY T., MOLINA R. & MUKUNOKI D. (2024b). Reduced-Precision and Reduced-Exponent Formats for Accelerating Adaptive Precision Sparse Matrix-Vector Product. working paper or preprint.
- GRIMMER M. (2003). Interval arithmetic in maple with intpakx. *PAMM*, **2**, 442 – 443.
- GRÜTZMACHER T., ANZT H. & QUINTANA-ORTÍ E. S. (2021). Using Ginkgo’s memory accessor for improving the accuracy of memory-bound low precision blas. *Software: Practice and Experience*.
- GUO H. & RUBIO-GONZÁLEZ C. (2018). Exploiting community structure for floating-point precision tuning. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2018, p. 333–343, New York, NY, USA: Association for Computing Machinery.
- HAMMER R. (1995). *C++ Toolbox for Verified Scientific Computing - Theory, Algorithms and Programs: Basic Numerical Problems*. Berlin, Heidelberg: Springer-Verlag.
- HARGREAVES G. (2003). Interval analysis in matlab. *Manchester Institute for Mathematical Sciences School of Mathematics*.
- HIGHAM N. J. (2002). *Accuracy and Stability of Numerical Algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, second edition.
- HIGHAM N. J. & MARY T. (2019). A new preconditioner that exploits low-rank approximations to factorization error. *SIAM J. Sci. Comput.*, **41**(1), A59–A82.
- HIGHAM N. J. & MARY T. (2020). Sharper probabilistic backward error analysis for basic linear algebra kernels with random data. *SIAM J. Sci. Comput.*, **42**(5), A3427–A3446.
- HIGHAM N. J. & MARY T. (2022). Mixed precision algorithms in numerical linear algebra. *Acta Numerica*, **31**, 347–414.
- HO M., SILVA H. & WONG W.-F. (2021). GRAM: A framework for dynamically mixing precisions in GPU applications. *ACM Transactions on Architecture and Code Optimization*, **18**, 1–24.

- HOFSCHESTER W. & KRÄMER W. (2004). C-xsc 2.0 – a c++ library for extended scientific computing. In R. ALT, A. FROMMER, R. B. KEARFOTT & W. LUTHER, editors, *Numerical Software with Result Verification*, p. 15–35, Berlin, Heidelberg: Springer Berlin Heidelberg.
- IEEE COMPUTER SOCIETY (1985). IEEE Standard for Binary Floating-Point Arithmetic. *ANSI/IEEE Std 754-1985*, p. 1–20.
- IEEE COMPUTER SOCIETY (2008). IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008*, p. 1–70.
- IEEE COMPUTER SOCIETY (2019). Ieee standard for floating-point arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, p. 1–84.
- JEANNEROD C.-P. & RUMP S. M. (2013). Improved error bounds for inner products in floating-point arithmetic. *SIAM J.Matrix Anal. Appl.*, **34**(16).
- JÉZÉQUEL F. & CHESNEAUX J.-M. (2008). CADNA: a library for estimating round-off error propagation. *Computer Physics Communications*, **178**(12), 933–955.
- JÉZÉQUEL F., SADAT HOSEININASAB S. & HILAIRE T. (2021). 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.
- KELCH R. (1993). Numerical quadrature by extrapolation with automatic result verification. In E. ADAMS & U. KULISCH, editors, *Scientific Computing with Automatic Result Verification*, volume 189 of *Mathematics in Science and Engineering*, p. 143–185. Elsevier.
- KOTIPALLI P. V., SINGH R., WOOD P., LAGUNA I. & BAGCHI S. (2019). AMPT-GA: automatic mixed precision floating point tuning for GPU applications. In *Proceedings of the ACM International Conference on Supercomputing*, ICS '19, p. 160–170, New York, NY, USA: Association for Computing Machinery.
- KULISCH U., KLATTE R., RATZ D., NEAGA M. & ULLRICH C. (1992). *PASCAL-XSC*. Springer Berlin, Heidelberg, number1 edition.
- LAGUNA I., WOOD P. C., SINGH R. & BAGCHI S. (2019). GPUMixer: Performance-driven floating-point tuning for GPU scientific applications. In M. WEILAND, G. JUCKELAND, C. TRINITIS & P. SADAYAPPAN, editors, *High Performance Computing*, p. 227–246: Springer International Publishing.
- LAM M. O., HOLLINGSWORTH J. K. & STEWART G. (2013). Dynamic floating-point cancellation detection. *Parallel Computing*, **39**(3), 146–155. High-performance Infrastructure for Scalable Tools.
- LANGE M. & RUMP S. M. (2017). Error estimates for the summation of real numbers with application to floating-point summation. *BIT Numerical Mathematics*, **57**(3), 927–941.
- LEE I.-Y. (1990). The gammasphere. *Nuclear Physics A*, **520**, c641–c655. Nuclear Structure in the Nineties.
- LEWANDOWSKI L., REITER P., BIRKENBACH B., BRUYNEEL B., CLEMENT E. & ET AL (2019). Pulse-shape analysis and position resolution in highly segmented HPGe AGATA detectors. *Eur.Phys.J.A*, **55**, 81–93.
- LINDQUIST N., LUSZCZEK P. & DONGARRA J. (2020). Improving the performance of the GMRES method using mixed-precision techniques. In J. NICHOLS, B. VERASTEGUI, A. B. MACCABE, O. HERNANDEZ, S. PARETE-KOON & T. AHEARN, editors, *Communications in Computer and Information Science*, p. 51–66. Springer, Cham, Switzerland.

- LOE J. A., GLUSA C. A., YAMAZAKI I., BOMAN E. G. & RAJAMANICKAM S. (2021). Experimental evaluation of multiprecision strategies for GMRES on GPUs. In *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, p. 469–478.
- MENON H., LAM M. O., OSEI-KUFFUOR D., SCHORDAN M., LLOYD S., MOHROR K. & HITTINGER J. (2018). Adapt: Algorithmic differentiation applied to floating-point precision tuning. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, p. 614–626.
- MOLINA R., GRAILLAT S., JÉZÉQUEL F. & MARY T. (2022). Adaptive Precision Sparse Matrix-Vector Product and its Application to Krylov Solvers. Sparse Days Meeting 2022. Poster.
- MOLINA R., GRAILLAT S., JÉZÉQUEL F. & MARY T. (2023a). Adaptive Precision Sparse Matrix-Vector Product and its Application to Krylov Solvers. In *International Congress on Industrial and Applied Mathematics (ICIAM 2023)*, Tokyo (Japan), Japan.
- MOLINA R., LAFAGE V., CHAMONT D. & JÉZÉQUEL F. (2023b). Investigating mixed-precision for AGATA pulse-shape analysis. In *26th International Conference on Computing in High Energy and Nuclear Physics (CHEP 2023)*, volume 295, p. 03020, Norfolk, VA, United States.
- MOORE R. E. (1966). *Interval analysis*. Prentice-Hall.
- MUKUNOKI D. & IMAMURA T. (2016). Reduced-Precision Floating-Point Formats on GPUs for High Performance and Energy Efficient Computation. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, p. 144–145.
- MUKUNOKI D., KAWAI M. & IMAMURA T. (2023). Sparse Matrix-Vector Multiplication with Reduced-Precision Memory Accessor. In *2023 IEEE 16th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoc)*, p. 608–615.
- OETTLI W. & PRAGER W. (1964). Compatibility of approximate solution of linear equations with given error bounds for coefficients and right-hand sides. *j-NUM-MATH*, **6**, 405–409.
- OGITA T., RUMP S. & OISHI S. (2005). Accurate sum and dot product. *SIAM J. Scientific Computing*, **26**, 1955–1988.
- OLARIU A. (2007). *Pulse shape analysis for the gamma-ray tracking detector AGATA*. Phd thesis, Université de Paris-Sud, Paris, France. 2007PA112349.
- OLARIU A., DESESQUELLES P., DIARRA C., MEDINA P., PARISEL C. & COLLABORATION C. (2006). Pulse shape analysis for the location of the gamma-interactions in agata. *IEEE Transactions on Nuclear Science*, **53**(3), 1028–1031.
- OOI R., IWASHITA T., FUKAYA T., IDA A. & YOKOTA R. (2020). Effect of mixed precision computing on h-matrix vector multiplication in bem analysis. In *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region, HPCAsia2020*: ACM.
- PAIGE C. C., ROZLOŽNÍK M. & STRAKOŠ Z. (2006). Modified Gram-Schmidt (MGS), least squares, and backward stability of MGS-GMRES. *SIAM J. Matrix Anal. Appl.*, **28**(1), 264–284.
- PARKER D., PIERCE B. & EGGERT P. (2000). Monte Carlo arithmetic: how to gamble with floating point and win. *Computing in Science & Engineering*, **2**(4), 58–68.
- PARKER D. S. & LANGLEY D. (1997). Monte Carlo Arithmetic: exploiting randomness in floating-point arithmetic.
- RECCHIA F. (2008). *In-beam test and imaging capabilities of the AGATA prototype detector*. PhD thesis, Università degli Studi di Padova.
- RENAC F., DE LA LLAVE PLATA M., MARTIN E., CHAPELIER J.-B. & COUAILLIER V. (2015). Aghora: A High-Order DG Solver for Turbulent Flow Simulations, In *IDIHOM: Industrialization of High-Order Methods - A Top-Down Approach*, p. 315–335. Springer International Publishing: Cham.

- RIGAL J. & GACHES J. (1967). On the compatibility of a given solution with the data of a linear system. *Journal of the ACM*, **14**, 526–543.
- RILEY M. & SIMPSON J. (2014). *Nuclear γ -Spectroscopy and the γ -Spheres*, In *Encyclopedia of Applied Physics*, p. 247–270. John Wiley & Sons, Ltd.
- ROUSSILHE G., LIGOZAT A.-L. & QUINTON S. (2023). A long road ahead: a review of the state of knowledge of the environmental effects of digitization. *Current Opinion in Environmental Sustainability*, **62**, 101296.
- RUBIO-GONZÁLEZ C., NGUYEN C., NGUYEN H. D., DEMMEL J., KAHAN W., SEN K., BAILEY D. H., IANCU C. & HOUGH D. (2013). Precimonious: Tuning assistant for floating-point precision. In *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, p. 1–12.
- RUMP S. M. (1999). Fast and parallel interval arithmetic. *BIT*, **39**(3), 534–554.
- SAAD Y. (2003). *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, second edition.
- SCHLARB M. C. (2009). *Simulation and Real-Time Analysis of Pulse Shapes from segmented HPGe-Detectors*. Phd thesis, Universität München, Munich, Germany.
- SIMONCINI V. & SZYLD D. B. (2003). Theory of inexact Krylov subspace methods and applications to scientific computing. *j-SISC*, **25**(2), 454–477.
- SIMPSON J. (1997). The EUROBALL spectrometer. *The European Physical Journal*, **358**, 139–143.
- SOLOVYEV A., BARANOWSKI M. S., BRIGGS I., JACOBSEN C., RAKAMARIĆ Z. & GOPALAKRISHNAN G. (2018). Rigorous estimation of floating-point round-off errors with symbolic taylor expansions. *ACM Trans. Program. Lang. Syst.*, **41**(1).
- STUDENT (1908). The probable error of a mean. *Biometrika*, **6**(1), 1–25.
- VAN DER MAREL J. & CEDERWALL B. (1999). Backtracking as a way to reconstruct compton scattered γ -rays. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, **437**(2), 538–551.
- VÉDRINE F., JACQUEMIN M., KOSMATOV N. & SIGNOLES J. (2021). Runtime abstract interpretation for numerical accuracy and robustness. In *International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*.
- VENTURELLI R. & BAZZACCO D. (2005). *LNL Annual Report 2004*. Internal report, INFN-LNL, Legnaro, Italy.
- VIGNES J. (1978). New methods for evaluating the validity of the results of mathematical computations. *Mathematics and Computers in Simulation*, **20**(4), 227–249.
- VIGNES J. (1993). A stochastic arithmetic for reliable scientific computation. *Mathematics and Computers in Simulation*, **35**(3), 233–261.
- VIGNES J. & PORTE M. L. (1974). Error analysis in computing. In J. L. ROSENFELD, editor, *Information Processing, Proceedings of the 6th IFIP Congress 1974, Stockholm, Sweden, August 5-10, 1974*, p. 610–614: North-Holland.
- VON NEUMANN J. & GOLDSTINE H. H. (1947). Numerical inverting of matrices of high order. *Bull. Amer. Math. Soc.*, **53**.
- WILKINSON J. H. (1985). The state of the art in error analysis. *NAG Newsletter*, p. 2/85:5–28. Invited lecture for the NAG 1984 Annual General Meeting.

YUN J., KANG B., RAMEAU F. & FU Z. (2023). Comparative study: Standalone ieee 16-bit floating-point for image classification.

ZHU Y.-K. & HAYES W. B. (2010). Algorithm 908: Online exact summation of floating-point streams. *ACM Trans. Math. Softw.*, **37**(3).

List of Figures

1.1	Harmonic sum	12
1.2	Catastrophic cancellation example	13
1.3	Formats specified by the IEEE 754-1985 standard	13
1.4	Exemples of numbers represented in FP32 format	14
1.5	Formats developed after the IEEE 754-1985 standard	15
1.6	Example of different magnitude elements summation	25
1.7	Promise dataflow	26
2.1	Penetration of α , β and γ particles	31
2.3	Agata data processing	36
3.1	Illustration of the bucket construction with four precisions $u_1 < u_2 < u_3 < u_4$. The real line $[0, +\infty)$ is partitioned into intervals P_{ik} defined by (3.15) (componentwise criteria, $\theta_i = a_i ^T x $) or (3.23) (normwise criteria, $\theta_i = \ A\ $).	48
3.2	Backward error for the adaptive precision Algorithm 4 with different target accuracies ϵ and different number of precisions used, compared with the uniform precision Algorithm 3 in the corresponding precision ($\epsilon = 2^{-24}$, $\epsilon = 2^{-37}$, $\epsilon = 2^{-53}$).	55
3.3	Storage cost of the adaptive precision SpMV, as a percentage of the storage cost of the uniform precision FP64 SpMV, for three different accuracy targets. For each plot, we report the storage gains depending on which of the componentwise (“CW”) or normwise (“NW”) criteria is considered and on how many precision formats are used.	67
3.4	Execution time of the adaptive precision SpMV for $\epsilon = 2^{-24}$ and $\epsilon = 2^{-53}$ target accuracies, as a percentage of the execution time of the uniform precision SpMV in the corresponding precision. Both the normwise (“NW”) and componentwise (“CW”) criteria are reported.	68
3.5	Execution time of the adaptive precision SpMV for an $\epsilon = 2^{-37}$ target accuracy, as a percentage of the execution time of the uniform precision FP64 SpMV. Both the normwise (“NW”) or componentwise (“CW”) criteria are reported.	69

3.6	Backward error, storage cost, and time cost of four SpMV variants: FP64 uniform precision (“Unif. fp64”), adaptive precision with two precisions but no dropping (“Adapt. dropless”), adaptive precision with only one precision and dropping (“Adapt. drop only”), and adaptive precision with both two precisions and dropping (“Adapt.”). All three adaptive variants use $\epsilon = 2^{-53}$ as target accuracy.	70
3.7	Parallel scaling experiments on Cube_Coup_dt0.	71
3.8	Convergence of GMRES-IR for matrix ML_Laplace: illustration of the effect of the ϵ parameter.	71
3.9	Convergence of GMRES-IR for matrix CoupCons3D: illustration of the difference between CW and NW criteria.	72
3.10	Convergence of GMRES-IR for matrix Geo_1438: illustration of a surprising behavior of NW variants.	72
4.1	Conversion from RP40 to FP64	77
4.2	Adaptive precision SpMV with seven precision levels (excerpt). RpArrayToFp converts a reduced-precision format to the IEEE FP64 format.	78
4.3	Conversion from RPRE40 to FP64	80
4.4	Normwise backward error computed from the FP128 uniform precision SpMV	81
4.5	Storage and time gains achieved by adaptive precision variants over uniform precision ones (normalized by the FP64 cost)	82
4.6	Storage and time gains achieved by AP7RE and AP7REU variants over the uniform precision and AP7 ones (normalized by the FP64 cost)	84
4.7	Distribution of the precision formats used for each nonzero element. Each bar on the x-axis corresponds to a different matrix and the y-axis indicates the percentage of nonzero elements stored in each format.	85
5.1	Distances between points found by the full gridsearch FP32 algorithm and alternative methods	93
5.2	Energy classes in which the points found by various methods belong to . .	94

List of Tables

2.1	Radioactive decays	31
3.1	List of precision formats used in our experiments.	51
3.2	List of matrices used in our experiments.	54
3.3	Results with GMRES-IR, BiCGStab-IR and CG-IR for various matrices and SpMV variants.	62
3.4	Results with GMRES-IR for various matrices and SpMV variants.	63
4.1	List of IEEE formats and RFPF's reduced-precision formats	76
4.2	List of RPRE formats used for each interval of values.	79
4.3	List of RPREU formats used for each interval of values.	79
4.4	Test matrices (Sorted by <i>nnz</i>).	81
5.1	Signals identified identically to FGS-FP32 result without LUT (%)	91
5.2	Execution time for the different configurations on CPU and GPU (ticks)	95
5.3	Points identified within 5mm of those found by FGS-FP32 without the LUT executed in CPU (%)	95

