Towards a Modular Precision Ecosystem for High Performance Computing

Journal Title
XX(X):1–??

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Hartwig Anzt[1,2], Goran Flegar[3], Thomas Grützmacher[1] and Enrique S. Quintana-Ortí[4]

Abstract

With the memory bandwidth of current computer architectures being significantly slower than the (floating point) arithmetic performance, many scientific computations only leverage a fraction of the computational power in today's high-performance architectures. At the same time, memory operations are the primary energy consumer of modern architectures, heavily impacting the resource cost of large-scale applications and the battery life of mobile devices. This paper tackles this mismatch between floating point arithmetic throughput and memory bandwidth by advocating a disruptive paradigm change with respect to how data is stored and processed in scientific applications. Concretely, the goal is to radically decouple the data storage format from the processing format and, ultimately, design a "modular precision ecosystem" that allows for more flexibility in terms of customized data access. For memory-bounded scientific applications, dynamically adapting the memory precision to the numerical requirements allows for attractive resource savings. In this paper, we demonstrate the potential of employing a modular precision ecosystem for the block-Jacobi preconditioner and the Page Rank algorithm — two applications that are popular in the communities and at the same characteristic representatives for the field of numerical linear algebra and data analytics, respectively.

Keywords

Modular Precision; Parallel Numerical Linear Algebra; Jacobi method; Page Rank; Conjugate Gradient; Multicore Processors and GPUs

Introduction

The digital revolution is shaping the future through breathtaking advances in all scientific fields, fueled by new scientific computing algorithms and the increasing computing power available in the complete range of digital processing devices, from supercomputers to mobile devices. The continuation of this trend, however, is challenged due to the mismatch between the arithmetic performance of processors in terms of floating point operations per second (FLOPS) on the one side, and the memory performance in terms of how fast data can be brought into the computational elements on the other side. In consequence, many applications nowadays utilize only a fraction of the available compute power as they spend a significant part of their time waiting for the required data, hitting the so-called "memory wall" Wulf and McKee (1995). With memory operations being a primary energy consumer, data access is also pivotal in the resource balance of large-scale applications and the battery life of mobile devices, facing the "power wall" Dongarra et al. (2014); Duranton et al. (2015); Lucas et al. (2014); Lavignon et al. (2013). As a result, already today, many backbone algorithms for scientific simulations, most big data applications, and many deep learning technologies are memory-bound on virtually all existing hardware architectures.

In general, the runtime and energy cost of accessing data and executing arithmetic operations depend on the precision of the operands. Typically, the quality of an application output correlates to the accuracy of the precision format used in the computations though some algorithms can accommodate the use of lower than working precision in parts of the underlying algorithm without impacting the quality of the final result. For example, mixed precision and iterative refinement, in the context of solving a system of linear equations, is a well established technique (see, e.g., the references in Higham (1996)), which has been recently re-visited due to the importance of the memory wall (see, e.g., Buttari et al. (2007); Baboulin et al. (2009); Göddeke et al. (2007) among several others). However, most of these past research efforts are based on the paradigm of the floating point format used to handle the data in memory being bonded to the floating point format employed in the arithmetic operations.

This paper advocates a disruptive paradigm change in how scientific data is stored and processed for memory-bounded scientific applications. Concretely, the goal is to radically decouple the data storage format from the processing format. In more detail, in order to leverage the heavily optimized arithmetic kernels that are natively supported by hardware as well as to benefit from high accuracy calculations, we propose to employ the IEEE standard precision formats in the arithmetic operations, but modify the policies which dictate

[1]Karlsruhe Institute of Technology, Karlsruhe, Germany
[2]University of Tennessee, Knoxville, USA
[3]Universitat Jaume I, Castellón, Spain
[4]Universitat Politècnica de València, Valencia, Spain

Corresponding author:
Hartwig Anzt, Karlsruher Institut für Technologie (KIT), Germany.
Email: hartwig.anzt@kit.edu

how the data is handled in memory operations. In this paper, we consider different strategies and aspects that arise from the disruptive decoupling of the memory format:

- We expose how an "approximate building block" of a complex algorithm can be stored in less than working precision without impacting the algorithm output. Precisely, we present a block-Jacobi preconditioner that adapts the storage format of the Jacobi blocks on an individual level to the numeric requirements of the problem.
- We propose a strategy to split the significand of data values into segments, and adapt the memory access precision to the numeric requirements of the problem. This enables fix-point methods to initially start the iteration process with reduced precision memory access, and to increment it over the algorithm execution without duplicating data in memory or invoking complex conversion routines.
- For the examples of the Jacobi method and the PageRank algorithm we demonstrate how the significand splitting improves the resource utilization without impacting the quality of the algorithm output.
- We propose to complement the storage format decoupling with a strategy that reduces the length of the values' exponents by identifying a reference point and normalizing the values of the data set. Exploiting clustering techniques or the properties of many problems which accumulate most values in a short range allows reducing the length of the exponent after normalization and compacting the format used for storing the values.
- We show that the values in an inverted diagonal block of a block-Jacobi preconditioner are in most cases of similar magnitude. We show that normalizing the exponent, in this setting, can enable resource savings beyond what is possible in an adaptive-precision block-Jacobi employing IEEE standard precision formats for storage.

We want to emphasize that this paper contributes by providing an initial step towards a disruptive paradigm change with respect to how data is stored and processed in applications ranging from scientific simulations on high-end supercomputers to mini-apps on smartphones, wearables and IoT (internet-of-things) devices. The ultimate goal of this research is to design a "modular precision ecosystem" that offers efficient storage in a variety of customized precisions while maintaining the hardware-supported IEEE standard precision formats in the arithmetic operations. Developing this ecosystem requires research and development along the complete digital food chain, including the following:

- Theoretical research on the algorithmic side to understand the impact of reduced precision data access;
- Development of new algorithms that can better cope with reading data in reduced accuracy;
- Deployment of a sustainable framework for customized precision formats;
- Development and optimization of low-level kernels for handling data in customized precision formats on a wide range of hardware architectures, from supercomputers to IoT and mobile devices;
- Research on compiler and operating system technology to support the smooth transformation of existing code into the modular precision ecosystem;
- and research on hardware technologies which natively support customized precision formats.

The Path towards a Modular Precision Ecosystem

The IEEE standard for floating point numbers

In digital processing, real numbers are generally represented in binary floating point format, with a certain number of bits used for storing significand, exponent, and sign of the floating point number representation. Typically, the energy and runtime cost of accessing data and performing arithmetic operations correlates with the complexity of the precision format Horowitz (2014): the more bits are involved, the higher is the cost. The Institute of Electrical and Electronics Engineers (IEEE) formulated a standard for floating point arithmetic (IEEE 754 IEEE) that defines IEEE 64-bit double precision (`fp64`), IEEE 32-bit single precision (`fp32`), and IEEE 16-bit half precision (`fp16`), among others. For scientific simulation codes and many applications in data analytics, IEEE double precision is established as the de-facto standard.

Mixed-precision IEEE floating point numbers

With the intention to reduce the resource footprint, the idea of combining different precision formats has already received some attention. This applies in particular to the area of numerical linear algebra, where the quality of the algorithms' output usually depends on the condition number of the problem and the floating point format that is employed to represent the values. Numerical effects like rounding errors result in a less accurate solution or even failure if a "lower precision format" (in terms of less significand and exponent bits) is used.

At the same time, running an iterative solver in a lower precision format may allow it to generate a solution approximation at lower resource cost: The iterations converge to the (less accurate) attainable accuracy after fewer iterations, and every iteration only accesses and operates with reduced precision. As memory operations are the primary energy consumer in digital processing Molka et al. (2010), reducing the data transfer volume directly lowers the energy footprint of the application. For applications where the performance is bound by the memory bandwidth, the execution time is also decreased.

Leveraging this property can enable resource savings even when aiming for double precision solutions. The idea here is to combine different precision formats inside a single algorithm and use double precision only if needed. A popular realization of this technique is mixed precision iterative refinement (MPIR Buttari et al. (2007); Baboulin et al. (2009); Strzodka and Göddeke (2006); Anzt et al. (2010)). The idea in MPIR is to refine a solution approximation by solving a residual equation in lower than working precision. In many situations, double precision accuracy

can be achieved Higham (1996). For example, Carson et al. Carson and Higham (2017) suggest the use of an incomplete factorization preconditioner computed in lower precision inside an iterative F-GMRES framework, and the authors even extend this approach by cascading multiple formats of decreasing precision Carson and Higham (2018).

What these approaches all share is the tight coupling between the arithmetic precision format, which is the format used for the arithmetic operations, and the storage format. While this seems to be a natural choice, it ignores the hardware trend of the computational power growing at a much faster pace than the memory bandwidth.

Decoupling IEEE storage format from IEEE arithmetic

In recognition of the growing mismatch between arithmetic and memory performance, in Anzt et al. (2019) we initially suggest to decouple the arithmetic format from the memory format. Concretely, we maintain IEEE `fp64` in all arithmetic operations, but employ a more compact lower precision IEEE format (`fp32` or `fp16`) for the memory operations. Using the block-Jacobi preconditioner Anzt et al. (2018) as a representative building block in scientific computing, we estimate the energy and runtime savings this strategy renders over using `fp64` throughout the complete application.

In some detail, block-Jacobi preconditioners require the inversion of small diagonal blocks of the system matrix Hegland and Saylor (1992); Anzt et al. (2018), and it is possible to optimize the memory format for each block individually. Naturally, the use of customized precisions has to be aligned with the algorithmic requirements, and the block-specific optimization of the precision format has to carefully consider the numerical effects against the objective that the quality of the preconditioner is not diminished. Precisely, in order to preserve the regularity of the block-Jacobi preconditioner, the floating point format must be chosen taking into account the condition number of each block (see top of Figure 1 for a distribution of the condition numbers of the distinct blocks for a set of test matrices Anzt et al. (2019)). Additionally, the floating point format has to be chosen taking into account the data range of the numerical values to protect against overflow and underflow. This can be particularly restrictive when considering only the the rigid IEEE standard formats, where the number of bits assigned to significand and exponent are pre-defined by the IEEE standard. Taking both aspects, the data range and the matrix conditioning, into account, the middle of Figure 1 visualizes which fraction of the diagonal blocks of the block-Jacobi matrix can be stored in lower than working precision, i.e. in `fp32` and `fp16`. Based on this analysis and the premise that the data transfer time linearly depends on the data volume, we can estimate the resource savings gained from using an adaptive Jacobi preconditioner instead of a standard (`fp64`) preconditioner inside a Conjugate Gradient iterative solver Saad (2003); see bottom of Figure 1 Anzt et al. (2019).

Decoupling non-IEEE storage and IEEE arithmetic

A conceptually different mixed precision strategy for improving the resource footprint is presented in Anzt et al. (2015), where an application dynamically modulates the memory precision over the execution time. Concretely, the distinct components in the solution vector of an iterative solver are handled in different precision formats which are adapted over the application execution to the component's convergence progress. For an iterative process, the underlying idea is to start iterating all components in a low precision format featuring few significand bits, and then successively increase the precision as needed for convergence to a solution with IEEE double precision accuracy. This requires close interaction between the application and the floating point format, in particular the real-time adaption of the precision to the algorithmic needs. Careful consideration of all the components' dependencies is required. Similar to the efforts in Anzt et al. (2019), the arithmetic format is decoupled from the memory format, and all arithmetic operations use `fp64`. The work in Anzt et al. (2015) does not employ the IEEE standard precision formats. Instead, as part of a more experimental research, it uses artificial precisions that arise by arbitrarily truncating the significand of the IEEE double precision format. The elegance of this approach is that the number of exponent bits remains unchanged, which virtually eliminates the danger of overflow and underflow. Once read into the processing units, the values are converted to `fp64` by filling the omitted significand bits with zeros. Considering the arithmetic intensity of widespread-used algorithms, most processors are over-provisioned for floating point operations. As a result, using `fp64` in the numerical operations rarely hurts the algorithm's overall performance. The memory precision is then successively increased over the application execution to enable `fp64` accuracy.

We note that there exist some previous efforts that combine a preconditioner stored in 32-bit precision in memory with a solver that operates in full 64-bit precision Tadano and Sakurai (2008); Gropp et al. (2000). In this paper, we propose a generalization of these ideas to include any reduced-precision customized format for the preconditioner, including half and single precisions as particular cases.

A Key Implementation Step: CPMS

An aspect not addressed in Anzt et al. (2015) is the question of how data is stored in different precision formats in memory. While this may seem an implementation detail, the efficient access to the values with different accuracy is performance-crucial, in particular on streaming architectures such as GPUs.

On streaming graphics processors, each memory read accesses 128 bytes of contiguous memory, and utilizing only part of the data inevitably results in low performance. The standard approach to mixed precision strategies is to duplicate the data (in different precision formats) in memory. However, this not only increases the memory footprint of the algorithm, but also makes it impossible to efficiently access different subsets of the values in different formats.

In Grützmacher and Anzt (2018) we propose the "*Customized Precision based on Mantissa Segmentation*

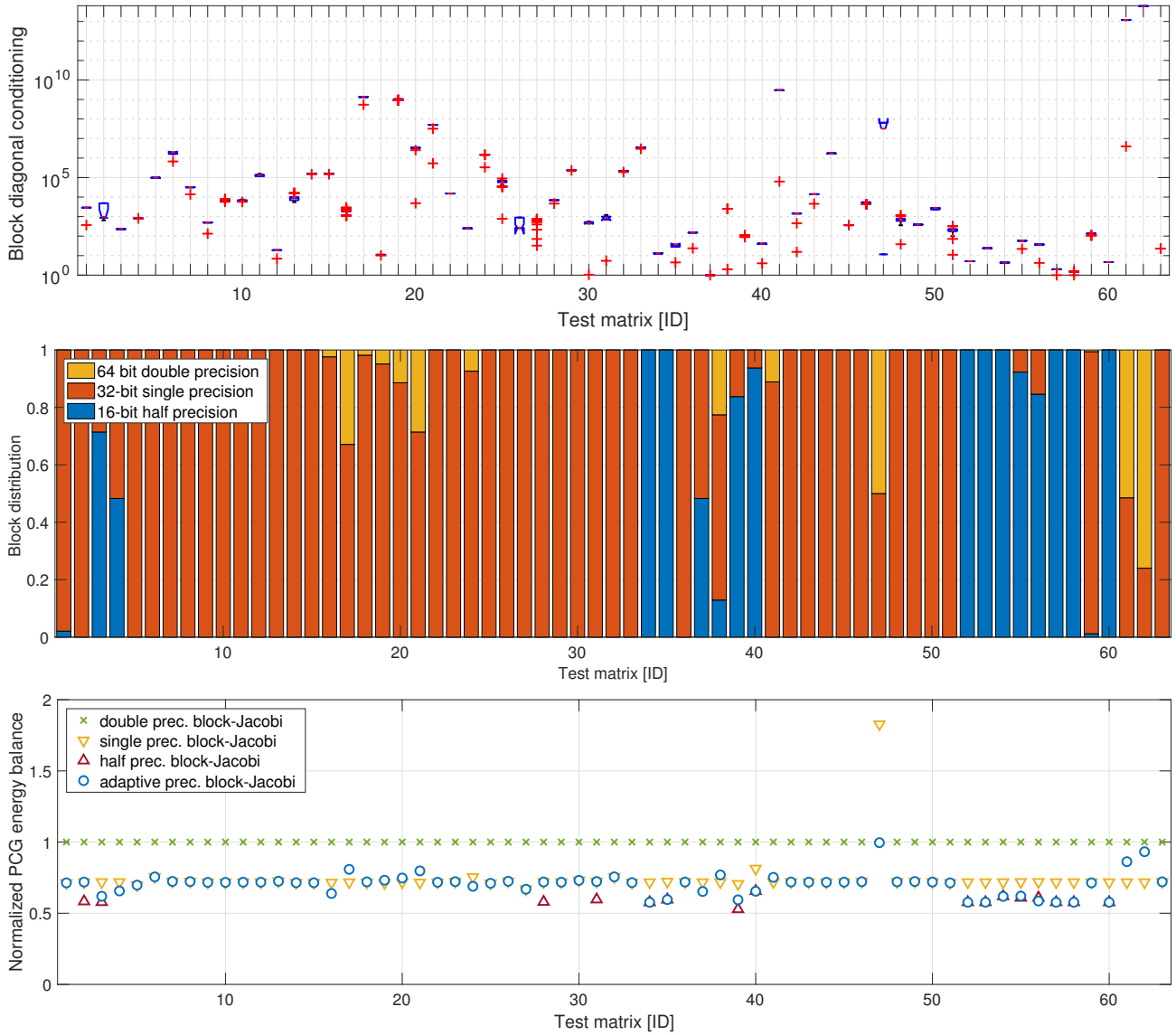

Figure 1. Top: Condition number distribution of the diagonal block-inverses in a block-Jacobi preconditioner for a set of test matrices from Suite Sparse sui (2018); Middle: Inverse blocks that can be stored in `fp64`, `fp32`, `fp16` without loosing regularity; Bottom: Energy balance of a preconditioned CG employing a `fp32`, `fp16`, or adaptive precision block-Jacobi in relation to using a `fp64` block-Jacobi.

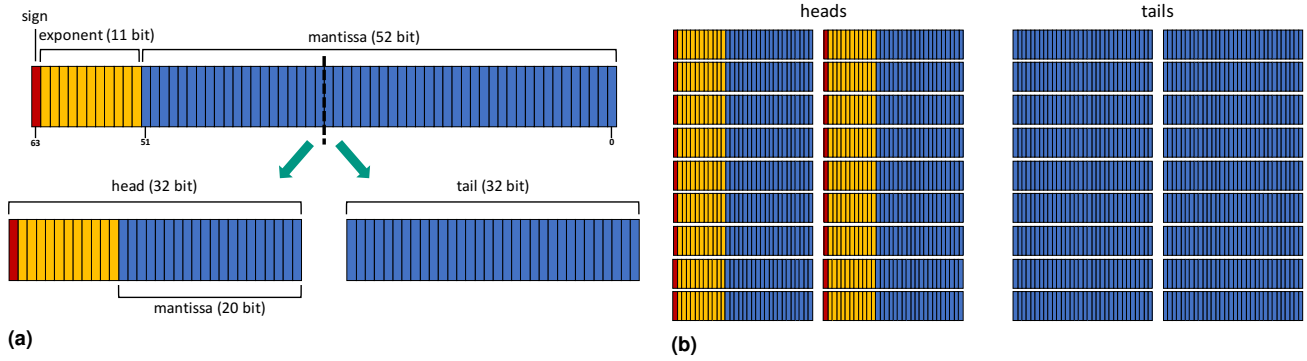

Figure 2. Splitting an IEEE double precision number into two segments we call “head” and “tail” (a). Storing the segmented data in interleaved fashion (b).

(CPMS)” to address the challenge of efficiently storing and accessing values in different precision formats. The strategy is based on breaking up the hardware-supported IEEE formats, and storing the segments separately, interleaved with segments of other values. In Figure 2 we visualize this strategy for a 2-segment splitting of the IEEE double

precision format Grützmacher and Anzt (2018). For this specific decomposition we refer to the two 32-bit segments as the "head" and the "tail" of the customized precision format. If an algorithm can work with the accuracy provided by the head segment, no access to the second memory block is necessary. Higher accuracy becomes available from accessing additional significand segments.

As the CPMS strategy preserves the length of the exponent, the first 32 bits include less significand bits than the 32 bits of IEEE single precision IEEE. Hence, the head of the 2-segment CPMS carries less accuracy than the IEEE single precision format. The advantages of this strategy are that 1) in specialized data access routines, no exponent conversion is necessary; 2) preserving the length of the exponent avoids overflow/underflow; 3) the data does not have to be duplicated in memory, as reading additional segments of the value will increase the values' accuracy; and 4) CPMS with one 64-bit segment is identical to IEEE `fp64`. Preserving the exponent bits of the IEEE standard precision format, the segmentation cannot turn a valid number into "NaN" or "infinity," as both are defined by all exponent bits being filled with "1s" IEEE. Aside from tracking the access information ("head-only" or "head-and-tail"), the total memory footprint of CPMS remains identical to that of the IEEE `fp64` standard precision format.

Obviously, CPMS can be realized independently of the format decoupling, but arithmetic operations in CPMS are not natively supported by hardware. At the same time, converting back to `fp64` allows leveraging the vendor-tuned kernels and, simultaneously, reduces the arithmetic rounding effects. As a result, complementing CPMS with format decoupling is usually the best choice.

As proof-of-concept, in Grützmacher and Anzt (2018) we re-engineered the adaptive precision Jacobi Anzt et al. (2015) using CPMS. For a finite difference discretization of the Laplace problem using a mesh of 100×100 elements, in Figure 3 we show how the memory format of the distinct components changes over the application execution: Each row corresponds to one unknown; in the white area, components updates access the head only; in the blue area, additional significand segments are needed to enable double-precision convergence. Depending on the benefits and the overhead introduced by using the adaptive precision Jacobi in the CPMS framework, the solution approximation can be generated faster than a standard Jacobi method in `fp64`, see Figure 4 Grützmacher and Anzt (2018).

In Grutzmacher et al. (2018) we use the CPMS prototype implementation to evaluate the potential of customized precision in big data analytics. The PageRank algorithm is re-engineered to start the power iteration with low accuracy data access, and successively increase the access precision to generate solutions with `fp64` accuracy Grutzmacher et al. (2018). The faster iterations using only a subset of the significand segments reduce the overall execution time compared to the reference implementation based on `fp64`, see Figure 5 Grutzmacher et al. (2018).

While already this preliminary research based on a prototype implementation of CPMS with segment sizes fixed to 16 and 32 reveals attractive performance improvements, we are convinced that the CPMS strategy holds an enormous potential for a broad range of applications.

We acknowledge that this section provides a general description of the CPMS technique along with a few implementation details, but does not cover all implementation aspects, such as, e.g., what is the cost of transforming the data between reduced and full precision formats; how are the caches affected by this approach; or who is responsible for splitting the numbers and what is the optimal number of segments. We believe that, though relevant, these aspects can be addressed with different strategies, and thus refer to future work realizing a production-ready implementation of the modular precision ecosystem.

A Complementary Step in the Path: CPEN

The idea of decoupling the memory format from the arithmetic format only brings benefits if the algorithm and the data involved allow for reducing the complexity of the format used in the memory operations without impacting the quality of the algorithm output. The strategies we considered so far are 1) using CPMS to gradually increase the precision in fixed-point methods like Jacobi iterations or the PageRank algorithm; and 2) storing the data of "approximate building blocks" like the block-Jacobi preconditioner in more compact IEEE standard precision formats. In the case of the adaptive precision block-Jacobi (see Figure 1), we were able to choose the precision format for the distinct blocks individually. However, this required analyzing the properties of the data such as value range and conditioning. For each Jacobi block, the limits to relaxing the memory precision format are 1) the regularity of the block must be preserved; and 2) the data range of the values must be covered by the precision format. We notice that the first requirement has implications on the length of the significand and the exponent, while the second requirement primarily has implications on the exponent, only. Using the IEEE standard precision formats, the number of exponent bits is fixed by the specification, and splitting the arithmetic precision format into segments like in CPMS and considering the first (significand-truncated) segment only results in a fixed number of exponent bits independent of the splitting. At the same time, adapting the number of exponent bits to the requirements can result in additional resource savings. Significant benefits are in particular available if the numeric values in the data are all of similar magnitude, and it is possible to identify a reference point the numbers are normalized to. In this sense, a memory format which adapts significand and exponent lengths to the requirements is not generic, but data-dependent as it involves identifying a suitable reference exponent and value normalization. Depending on the range of the values involved in the dataset, the "Customized Precision format Normalizing the Exponents to a reference exponent (CPEN)" can offer attractive memory savings. An appealing side-effect is that analyzing the values' magnitude and adapting the exponent length efficiently eliminates the danger of over- and underflow.

In general, the efficient use of CPEN requires an initial clustering step that splits the original dataset into subsets (clusters) containing values of similar magnitude. The

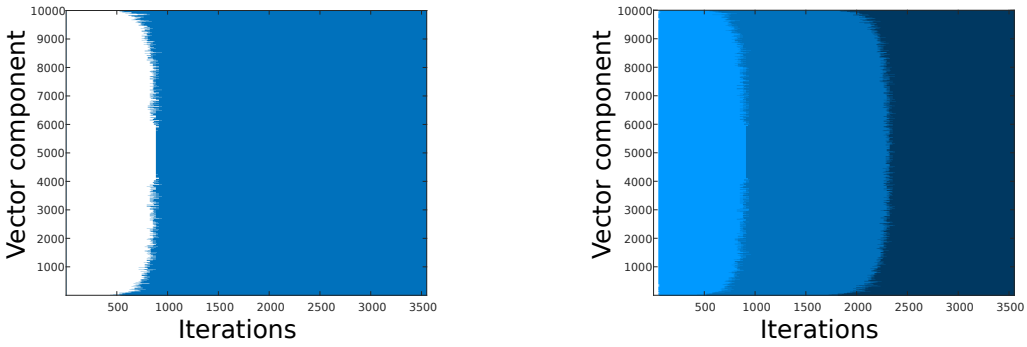

Figure 3. Accuracy needs in adaptive Jacobi in a 2-segment (left) and a 4-segment (right) CPMS realization. The white-colored area indicates only the head is accessed, the blue areas indicate additional significand segment reads.

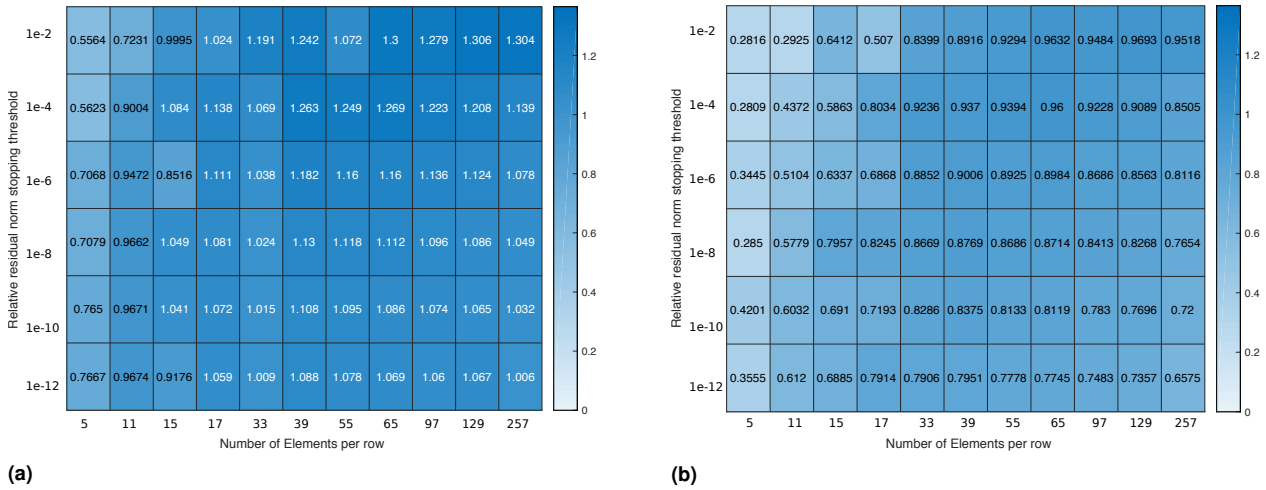

Figure 4. Speed-up factors of the adaptive precision Jacobi in a 2-segment CPMS realization (left) and a 4-segment CPMS realization (right), respectively.

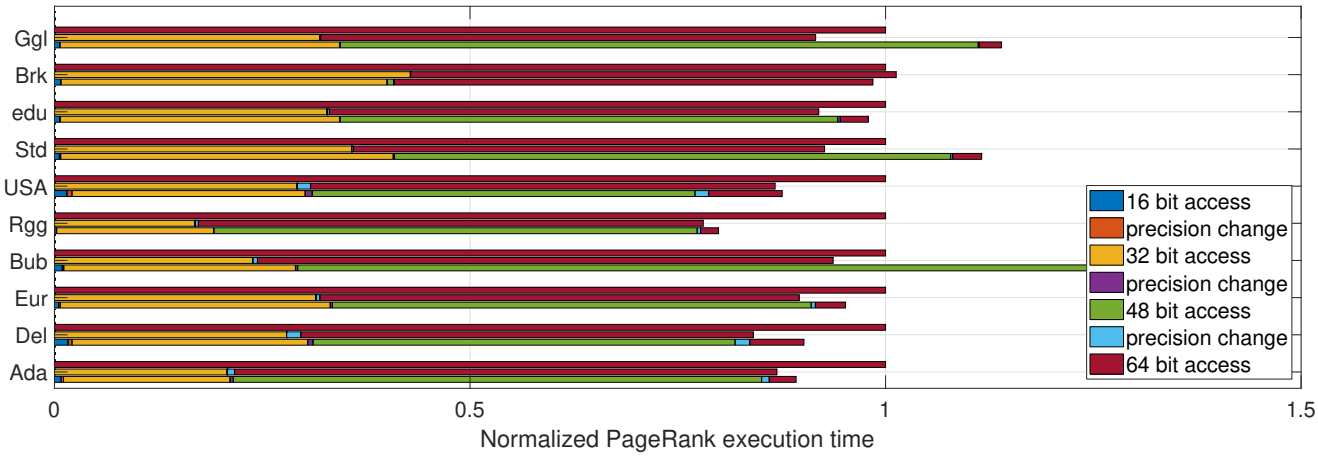

Figure 5. Breakdown of the (normalized) PageRank execution time into the iteration sequences with different CPMS access and in-place format conversion. Different network problems from Suite Sparse sui (2018) are addressed. The top bar is the runtime of the reference PageRank in `fp64`, the middle bar uses 2-segment CPMS, the bottom bar uses 4-segment CPMS.

values in each cluster are then normalized to a subset-specific reference exponent. Depending on the data structure, splitting the original dataset into subsets can introduce some overhead in terms of secondary structure information, which makes the clustering step a data-dependent optimization.

Interestingly, many problems inherently consist of clusters that accumulate values of similar magnitude. An example of such a situation is the block-Jacobi preconditioner we addressed in Figure 1: For each Jacobi block in any of the test matrices considered, we analyze the exponents of the values, and visualize the ranges of the distinct blocks in Figure 6.

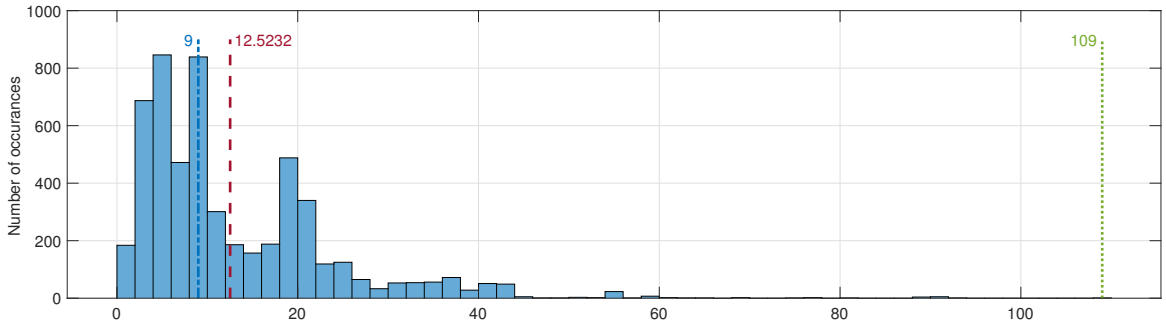

Figure 6. Histogram of the exponent ranges of the distinct Jacobi blocks in any of the matrices considered in adaptive precision block-Jacobi. The median exponent range is colored in red, the average exponent range in blue, and the maximum exponent range in green.

The histogram reveals that, on average, the magnitude of the values in a block varies by less than 12 orders of magnitude, with a median of only 9 orders of magnitude. This is much smaller than what the exponent range of IEEE `fp64` allows for.

When the values in a block are of similar magnitude, the clustering step becomes obsolete, and the exponent normalization strategy of CPEN can be readily applied to each diagonal block. Employing CPEN as memory format in the worst case preserves the complexity of the IEEE standard formats used in Figure 1, but will likely reduce the formats complexity as a block-adapted exponent length allows to fit more significand bits into a 16-bit or 32-bit block. In consequence, more Jacobi blocks can use 16-bit or 32-bit storage without impacting the blocks' regularity. In Figure 7 we illustrate this effect by visualizing the fraction of blocks stored in 16-bit, 32-bit, and 64-bit CPEN values, respectively. The advantages over employing the IEEE standard precision formats can be derived by comparing with Figure 1. We note that similar to the CPMS research, we allow in this analysis only for 16-bit, 32-bit and 64-bit formats as access functions to those are natively supported by hardware. Non-standard hardware may support data access in other data cardinalities, allowing for more fine-granular optimization.

Storing more blocks in a slimmer format suggests larger resource savings in the block-Jacobi preconditioner application. We assess these benefits by visualizing in Figure 8 the memory savings of the CPEN-based block-Jacobi over the adaptive block-Jacobi preconditioner using the IEEE standard precision formats for all memory operations. The benefits are problem-dependent: for some matrices, resizing the exponent to the requirements of the blocks allows compressing the floating point numbers, and CPEN succeeds in reducing the memory footprint of the adaptive precision block-Jacobi beyond what the IEEE formats allow for. We note that CPEN can also increase the memory footprint: For cases where the exponent reduction fails to reduce the size of the floating point format storing the values, the reference exponent for each Jacobi block introduces a moderate storage overhead. However, for the data considered, this overhead is small compared with the benefits CPEN renders for other problems. CPEN providing at least the significand information of the IEEE formats ensures that the top-level iterative solver convergence generally remains unaffected (except for rounding variation). Given the memory-bound nature of the block-Jacobi preconditioner (and sparse linear algebra in general), the memory reductions directly translate to runtime and energy savings.

The attractive benefits CPEN renders to algorithms storing part of the data in a less complex format suggest that complementing the significand-segmentation with a normalized exponent will allow for additional resource savings when employing CPMS in fix-point based algorithms. Furthermore, we are convinced that strategies clustering values of similar magnitude will allow to exploit CPEN for large data sets and, consequently, improve the memory footprint, runtime, and energy consumption data analytics.

Summary and Outlook

In this paper, we have proposed a disruptive paradigm change with respect to how data is stored and processed in scientific computing and community applications. As an initial step, we have demonstrated how adapting the format to store the blocks of a Jacobi preconditioner can improve the resource optimization; we have shown how splitting the significand into segments can allow fix-point methods like Jacobi iterations or the PageRank algorithm to adapt the data access accuracy to the numeric requirements; and we have revealed how adapting the length of the exponents to the data range can help in reducing the memory footprint and therewith reduce the resource consumption of memory-bounded algorithms. As part of future work, we plan to develop a "modular precision ecosystem" that offers efficient storage in a variety of customized precisions while maintaining the hardware-supported IEEE standard precision formats in the arithmetic operations. This ecosystem will allow a wide range of algorithms from numerical linear algebra over data analytics and machine learning to benefit from storage format decoupling and precision optimization.

Acknowledgments

This work was supported by the "Impuls und Vernetzungsfond" of the Helmholtz Association under grant VH-NG-1241. G. Flegar and E. S. Quintana-Ortí were supported by project TIN2017-82972-R of the MINECO and FEDER and the H2020 EU FETHPC Project 732631 "OPRECOMP".

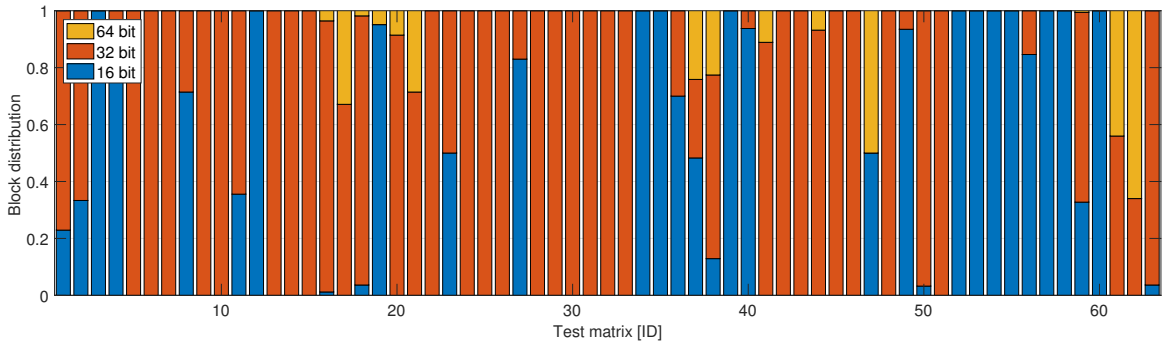

Figure 7. Fraction of the diagonal blocks stored in 16 bits, 32 bits or 64 bits CPEN. For each block, a suitable reference exponent is identified and the numeric values are stored relative to this exponent. The significand bits are reduced if appropriate to fit into a more compact format.

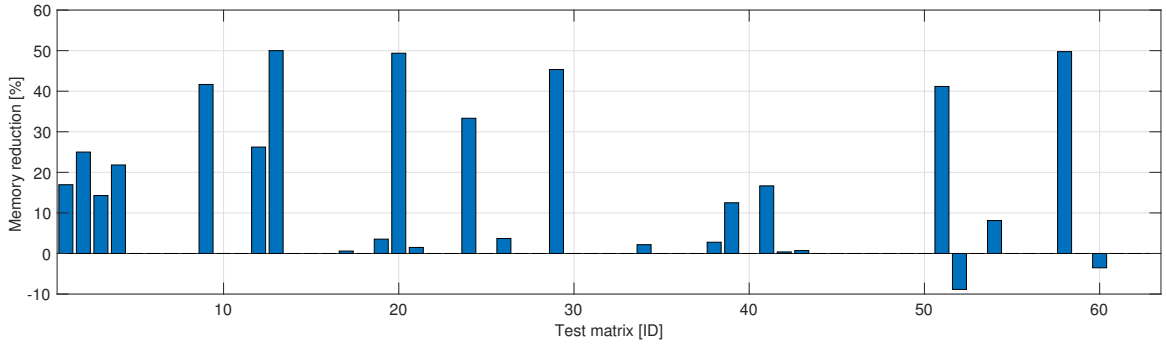

Figure 8. Memory savings employing CPEN instead of the IEEE standard precision formats inside an adaptive precision block-Jacobi preconditioner.

References

(2018) Suitesparse matrix collection. https://sparse.tamu.edu. Accessed in April 2018.

Anzt H, Dongarra J, Flegar G, Higham NJ and Quintana-Ortí ES (2019) Adaptive precision in block-Jacobi preconditioning for iterative sparse linear system solvers. *Concurrency and Computation: Practice and Experience* 31(6). DOI:10.1002/cpe.4460. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4460.

Anzt H, Dongarra J, Flegar G and Quintana-Ortí ES (2018) Variable-size batched Gauss–Jordan elimination for block-Jacobi preconditioning on graphics processors. *Parallel Computing* DOI:https://doi.org/10.1016/j.parco.2017.12.006. URL http://www.sciencedirect.com/science/article/pii/S0167819117302107.

Anzt H, Dongarra J and Quintana-Ortí ES (2015) Adaptive precision solvers for sparse linear systems. In: *Proceedings of the 3rd International Workshop on Energy Efficient Supercomputing*, E2SC '15. New York, NY, USA: ACM. ISBN 978-1-4503-3994-0, pp. 2:1–2:10. DOI:10.1145/2834800.2834802. URL http://doi.acm.org/10.1145/2834800.2834802.

Anzt H, Heuveline V and Rocker B (2010) Mixed precision error correction methods for linear systems Convergence analysis based on Krylov subspace methods. In: Jonasson K (ed.) *PARA 2010, Part II, LNCS 7134*. Springer, Heidelberg, pp. 237–248. DOI:10.1007/978-3-642-28145-7$_$24.

Baboulin M, Buttari A, Dongarra JJ, Langou J, Langou J, Luszczek P, Kurzak J and Tomov S (2009) Accelerating Scientific Computations with Mixed Precision Algorithms. *Computer Physics Communications* 180(12): 2526–2533.

Buttari A, Dongarra JJ, Langou J, Langou J, Luszczek P and Kurzak J (2007) Mixed precision iterative refinement techniques for the solution of dense linear systems. *Int. J. of High Perf. Comp. & Appl.* 21(4): 457–486.

Carson E and Higham NJ (2017) A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems. *SIAM J. Scientific Computing* 39(6): A2834–A2856. DOI:10.1137/17M1122918.

Carson E and Higham NJ (2018) Accelerating the solution of linear systems by iterative refinement in three precisions. *SIAM J. Scientific Computing* 40(2): A817–A847. DOI:10.1137/17M1140819.

Dongarra J et al. (2014) Applied mathematics research for exascale computing. Technical report, U.S. Dept. of Energy, Office of Science, Advanced Scientific Computing Research Program. https://science.energy.gov/~/media/ascr/pdf/research/am/docs/EMWGreport.pdf.

Duranton M et al. (2015) HiPEAC vision 2015. https://www.hipeac.org/publications/vision/.

Göddeke D, Strzodka R and Turek S (2007) Performance and accuracy of hardware–oriented native–, emulated– and mixed–precision solvers in FEM simulations. *Int. J. of Parallel, Emergent and Distributed Systems* 22(4): 221–256.

Gropp WD, Kaushik DK, Keyes DE and Smith BF (2000) Latency, bandwidth, and concurrent issue limitations in high-performance cfd. DOI:10.1016/B978-008043944-0/50783-6.

Grützmacher T and Anzt H (2018) A modular precision format for decoupling arithmetic format and storage format. In: *Lecture Notes in Computer Science, 16th Int. Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous*

Platforms – HeteroPar'18. Springer. To appear.

Grutzmacher T, Anzt H, Quintana-Orti ES and Scheidegger F (2018) High-performance GPU implementation of PageRank with reduced precision based on mantissa segmentation. In: *Proceedings of the 8th Workshop on Irregular Applications: Architecture and Algorithms (IA3)*. pp. 61–68.

Hegland M and Saylor PE (1992) Block-Jacobi preconditioning of the Conjugate Gradient method on a vector processor. *International Journal of Computer Mathematics* 44(1-4): 71–89. DOI:10.1080/00207169208804096. URL https://doi.org/10.1080/00207169208804096.

Higham NJ (1996) *Accuracy and Stability of Numerical Algorithms*. ISBN 0-89871-355-2.

Horowitz M (2014) Computing's energy problem (and what we can do about it). In: *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. pp. 10–14. DOI:10.1109/ISSCC.2014.6757323.

IEEE (2008) IEEE standard for floating-point arithmetic. *IEEE Std 754-2008* : 1–70DOI:10.1109/IEEESTD.2008.4610935.

Lavignon JF et al. (2013) ETP4HPC strategic research agenda achieving HPC leadership in Europe. http://www.etp4hpc.eu/.

Lucas R et al. (2014) Top ten Exascale research challenges. http://science.energy.gov/~/media/ascr/ascac/pdf/meetings/20140210/Top10reportFEB14.pdf.

Molka D, Hackenberg D, Schöne R and Müller MS (2010) Characterizing the energy consumption of data transfers and arithmetic operations on x86-64 processors. In: *International Green Computing Conference 2010, Chicago, IL, USA, 15-18 August 2010*. pp. 123–133. DOI:10.1109/GREENCOMP.2010.5598316. URL https://doi.org/10.1109/GREENCOMP.2010.5598316.

Saad Y (2003) *Iterative Methods for Sparse Linear Systems*. 2nd edition. SIAM. ISBN 0898715342.

Strzodka R and Göddeke D (2006) Pipelined mixed precision algorithms on FPGAs for fast and accurate PDE solvers from low precision components. In: *IEEE Proceedings on Field–Programmable Custom Computing Machines (FCCM 2006)*. IEEE Computer Society Press.

Tadano H and Sakurai T (2008) On single precision preconditioners for krylov subspace iterative methods. In: Lirkov I, Margenov S and Waśniewski J (eds.) *Large-Scale Scientific Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-78827-0, pp. 721–728.

Wulf WA and McKee SA (1995) Hitting the memory wall: Implications of the obvious. *SIGARCH Comput. Archit. News* 23(1): 20–24. DOI:10.1145/216585.216588. URL http://doi.acm.org/10.1145/216585.216588.

Author biographies

Hartwig Anzt is a Helmholtz-Young-Investigator Group leader at the Steinbuch Centre for Computing at the Karlsruhe Institute of Technology. He obtained his PhD in Mathematics at the Karlsruhe Institute of Technology, and afterwards joined Jack Dongarra's Innovative Computing Lab at the University of Tennessee in 2013. Since 2015 he also holds a Senior Research Scientist position at the University of Tennessee. Hartwig Anzt has a strong background in numerical mathematics, specializes in iterative methods and preconditioning techniques for the next generation hardware architectures. Hartwig Anzt has a long track record of high-quality software development. He is author of the MAGMA-sparse open source software package managing lead and developer of the Ginkgo numerical linear algebra library, and part of the US Exascale computing project delivering production-ready numerical linear algebra libraries.

Goran Flegar is a PhD candidate in the High Performance Computing & Architectures group at the Jaume I University. His research interests include sparse linear algebra, accelerator computing and software design. He holds a bachelor's degree in mathematics and a master's degree in computer science and mathematics from the University of Zagreb. He is also one of the main developers of Ginkgo, a modern C++ library for the iterative solution of sparse linear systems, via Krylov subspace methods, on multicore architectures.

Thomas Grützmacher is a PhD researcher at the Karlsruhe Institute of Technology. He received a bachelor's degree from the KIT in 2015 with emphasis on software development for mobile computing and IoT. In 2018 he completed his Master's studies with Emphasis on High Performance Computing and unconventional precision formats. Thomas Grützmacher's research focus is designing a modular precision ecosystem. He also is among the core developer team of the Ginkgo open source software.

Enrique S. Quintana-Ortí received the bachelor and Ph.D. degrees in computer sciences from the Universidad Politecnica de Valencia, Spain, in 1992 and 1996, respectively. Currently, he is a Professor in Computer Architecture in the Universitat Politecnica de Valencia, Spain. He has published more than 300 papers in international conferences and journals, and has contributed to software libraries like PLiC/SLICOT, MAGMA, FLARE, BLIS and libflame for control theory and parallel linear algebra. He has also been member of the program committee for around 100 international conferences. In 2008 Enrique received an NVIDIA professor partnership award for his contributions to the acceleration of dense linear algebra kernels on graphics processors, and he also received two technical awards from NASA for his contributions to fault-tolerant dense linear algebra libraries for space vehicles. Recently, he has participated/participates in EU projects on parallel programming, such as TEXT, INTERTWinE, and energy efficiency such as EXA2GREEN and OPRECOMP. His current research interests include parallel programming, linear algebra, energy consumption, transprecision computing and bioinformatics as well as advanced architectures and hardware accelerators.