A Modular Precision Format for decoupling Arithmetic Format and Storage Format

Thomas Grützmacher[1] and Hartwig Anzt[1,2]✉

[1] Karlsruhe Institute of Technology, Germany. `thogru.kit@gmx.de`
[2] University of Tennessee, USA. `hartwig.anzt@kit.edu`

Abstract. In this work, we propose to decouple the arithmetic format from the storage format in numerical algorithms. We complement this idea with a modular precision storage layout that allows runtime precision adaptation such that a value can be accessed faster if lower accuracy is acceptable. Combined with precision-aware numerical algorithms that use full precision in all arithmetic computations, this strategy can result in runtime savings without impacting the memory footprint or the accuracy of the final result. In an experimental analysis using the adaptive precision Jacobi method we assess the benefits of the modular precision format on a recent high-end GPU architecture.

Keywords: Mixed precision numerics, Modular precision ecosystem, Customized precision, GPUs, Adaptive precision Jacobi

1 Introduction

Over the last decades, the scientific computing community witnessed a widening gap between the computational performance in terms of the number of floating-point operations per second (FLOPS) on the one side, and the memory throughput in terms of how fast data can be brought into the computational elements (bandwidth) on the other side. As a result, more and more algorithms are hitting the "memory wall," which means the performance being limited by the memory bandwidth, and the algorithms executing only at a fraction of the theoretical peak performance. Already today, sparse linear algebra powering a large fraction of the scientific simulations are memory bound on virtually all existing hardware architectures. To continue the success story of simulation-based research, it is therefore essential to develop novel strategies that allow to transfer the growing computational power into algorithm performance.

In this work, we introduce a disruptive paradigm change with respect to how data is stored and processed in numerical linear algebra. To reflect the imbalance between computational power and memory bandwidth, we propose to radically decouple the storage format from the arithmetic format. We complement this idea with the introduction of a "modular precision ecosystem" with demand-fitted memory access routines. The idea behind is to decompose the IEEE standard precision formats into segments, and to store those in a fashion that enables efficient access to the values at variable accuracy. This allows to maintain standard working precision in all arithmetic floating-point operations, but radically reduces the cost of accessing the data if lower accuracy is acceptable.

We structure the rest of the paper as follows. In Section 2 we review some existing work on mixed precision numerics before we introduce the idea of the modular precision format in Section 3. We start the experimental section with a review of the adaptive precision Jacobi that we employ to assess the efficiency of the modular precision format and the developed memory access routines. The experimental results we report in Section 4 are obtained from addressing a set of artificial test problems on a high-end NVIDIA GPU. We conclude in Section 5 with an outlook on future work.

2 Related work on mixed precision numerics

To illustrate the approach we take and its uniqueness, we address the iterative solution of linear systems, which is a common task in scientific computing. The quality of an iteratively generated solution depends on the condition number of the linear system and the floating-point format that is employed to represent the numbers. Generally, numerical errors due to rounding result in a less accurate solution if a lower precision format is used. For scientific simulation codes, IEEE double precision has become the de-facto standard. The numerical values are stored in a binary format where a certain number of bits is used for storing mantissa, exponent, and sign of the floating-point number representation [10].

While running an iterative solver in lower than double precision typically results in a solution approximation of inferior quality, this solution approximation can usually be generated much faster: The approximation accuracy stagnates after fewer iterations, and every iteration only reads and writes data in reduced precision, which, for memory bound algorithms, directly corresponds to runtime savings. Leveraging this property in a smart fashion can enable savings also when generating double precision solutions. The idea here is to combine different precision formats inside a single algorithm, and use double precision only if needed.

Among the most popular mixed precision strategies is the mixed precision iterative refinement technique [5,8,12]. There, the idea 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 [9]. Other recent work suggests the use of an incomplete factorization preconditioner computed in lower precision inside an iterative F-GMRES framework [6], and even extends this approach by cascading multiple formats of decreasing precision [7]. What all these approaches share is the tight coupling between working precision format and 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.

In [2], a preconditioner stored in low precision is employed inside a high precision iterative solver. The numerical properties of the preconditioner are analyzed and, if the characteristics allow for it, stored in lower than working precision. This can be seen as a step towards decoupling storage format from arithmetic format, but as only IEEE standard formats are considered, the values have to be converted between the formats with careful protection against under- and overflow.

A different mixed precision strategy was presented in [3], where the distinct components in the solution vector are handled in different precision formats, each adapted to the component's convergence progress. The underlying idea is to truncate the double precision format by chopping off mantissa bits. The iteration process is started with few mantissa bits, and the mantissa length is then successively increased individually for each component as needed for convergence to a solution of double precision accuracy. This way, and in contrast to the previously-mentioned mixed precision strategies, the work in [3] does not refer to the IEEE standard precision formats, but, as part of a more experimental research, employs artificial precisions that arise by arbitrarily truncating the mantissa 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 over- and underflow. Once read into the processing units, the values are converted to double precision by filling the truncated mantissa bits with zeros. The floating-point operations themselves all use double precision accuracy.

What [3] fails to address is a concept that handles the artificial precision format in memory. While this seems to be an implementation detail, the question of how data is accessed is performance-crucial, in particular on streaming architectures such as GPUs. There, each memory read accesses 128 bytes of contiguous memory, and utilizing only part of the data inevitably results in low performance [11]. Usually, mixed precision numerics duplicate the data (in different precision formats) in memory. However, this not only increases the memory footprint of the algorithm, but also makes it difficult to efficiently access different subsets of the values in different formats.

3 Modular precision format

The two key ideas of the modular precision format are (1) to completely decouple the storage format from the operating format, and (2) to abandon the IEEE-supported standard precision formats to store the data, but split the arithmetic format into segments, and store the segments of the values in the dataset in interleaved fashion such that the same segments of all values are consecutive in memory.

These two ideas can be addressed independently, however, they work efficiently in particular when used in combination. Decoupling the operational format from the storage format is motivated by the performance of many linear algebra routines being bound by memory bandwidth: If the algorithm can accept reading values with less accuracy, the data can be accessed much faster in a lower precision format. The arithmetic operations can still use high precision without impacting the performance as long as the algorithm remains memory bound. Decoupling the storage format from the operational format in an environment supporting IEEE standard precision requires to duplicate the data in memory if it is used in different precision formats over the algorithm execution. Also, as the IEEE standard formats differ in the exponent length (and therewith in the range or representable values), the conversion between the formats has to meticulously protect against under- and overflow.

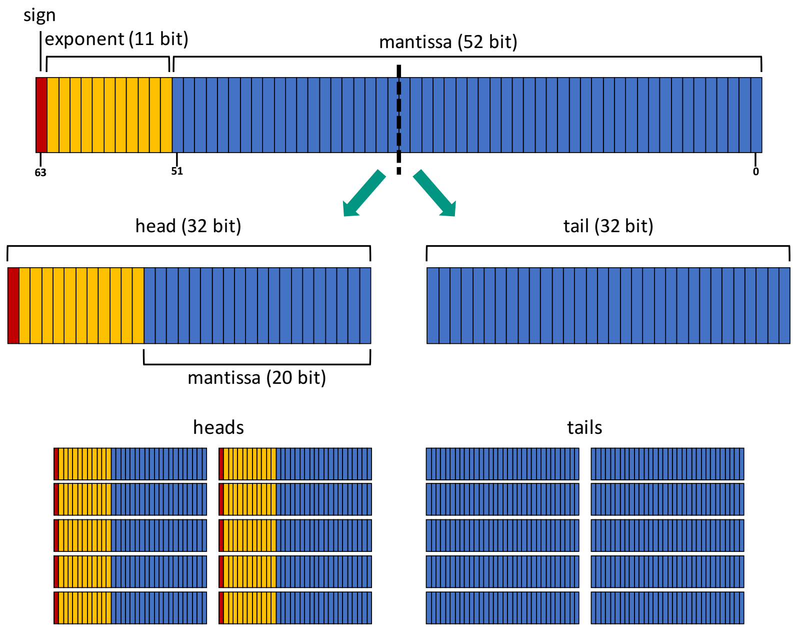


Fig. 1: Splitting an IEEE double precision number into "head" and "tail" (top) and storing head and tail of the data in the customized precision format in separate blocks (bottom).

The customized precision format based on mantissa segmentation ("`CPMS`") does not convert between IEEE standard formats, but instead splits the high precision number into segments. In Figure 1 (top) we visualize this strategy for a 2-segment splitting of the IEEE double precision format. For this specific decomposition we refer to the two 32-bit segments as "head" and "tail" of the customized precision format. Other splittings are possible. As the `CPMS` strategy preserves the length of the exponent, the first 32 bits include less mantissa bits than the 32 bits of IEEE single precision [10]. 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) for specialized data access routines, no format conversion is necessary; (2) preserving the length of the exponent avoids overflow/underflow; and (3) the data does not have to be duplicated in memory, but reading additional segments of the value will increase the value's accuracy.

We point out that by preserving the exponent bits of the IEEE standard precision format, the segmentation can not turn a valid number into "NaN" or infinity, as both are defined by all exponent bits being filled with "1 bits" [10].

To enable efficient access to the values in low precision, e.g. only the first segment of each value, it is important to separate the head from the tail in memory, and store the head of all values consecutively in memory, see bottom of Figure 1. As long as considering all values under the accuracy of the head is acceptable, no access to the second part of the memory is necessary. We emphasize that the memory footprint for storing the values only is identical to storing the data in IEEE standard double precision, if the data is accessed in different precisions, an additional array is needed for storing the segment information for each value.

Obviously, the customized precision format could be realized independent of the format decoupling, but not only are the arithmetic operations in this non-standard format not natively supported by hardware, but also would this introduce additional rounding errors in the numerical operations. Combining `CPMS` with the idea of decoupling the arithmetic format eliminates the need of customized routines for a format that is not natively supported by hardware, and incurs no performance penalty as long as the algorithm remains memory-bound.

4 Experimental evaluation

Problem description and algorithm details. The problem we consider is the iterative solution of a sparse linear system via the adaptive precision Jacobi method proposed in [3]. The algorithm is based on the numeric property of the Jacobi relaxation method typically having a constant convergence rate, and the possibility to detect stagnation in the iteration vector on a component level. Concretely, this property establishes that, for any component of the approximate solution vectors at relaxation step k and $k-1$, there exists a $\theta_i < 1$:

$$\left|x_i^{\{k\}} - x_i^{\{k-1\}}\right| \leq \theta_i \left|x_i^{\{k-1\}} - x_i^{\{k-2\}}\right| \leq \theta_i^2 \left|x_i^{\{k-2\}} - x_i^{\{k-3\}}\right| \dots \tag{1}$$

Furthermore, due to the linear convergence rate of the Jacobi iteration, the ratios

$$c_i^{\{k\}} := \frac{z_i^{\{k-1\}}}{z_i^{\{k\}}} = \frac{\left|x_i^{\{k-1\}} - x_i^{\{k-2\}}\right|}{\left|x_i^{\{k\}} - x_i^{\{k-1\}}\right|}, \quad k \geq 2, \tag{2}$$

are, in general, different for the distinct components, but they remain constant up to convergence; i.e., $c_i^{\{2\}} = c_i^{\{3\}} = c_i^{\{4\}} = \ldots = c_i$, where we note that $c_i > 1$ is necessary for convergence [3]. The adaptive precision Jacobi presented in [3] utilizes this property by monitoring $z_i^{\{k\}}$ at component level and some periodicity ϕ, and use a stagnation test with some threshold $\tilde{\delta}$

$$\left| \frac{z_i^{\{k-\phi\}}}{z_i^{\{k\}}} - c_i^{\phi} \right| > \tilde{\delta} \tag{3}$$

that detects the necessity of mantissa extension [3].

While the test periodicity ϕ and the stagnation test threshold $\tilde{\delta}$ can be optimized for each problem individually, we use the default setting of $\tilde{\delta} = 0.9 \cdot \left(c_i^{\phi} - 1\right)$ and $\phi = 10$.

Experiment environment and test matrices. The experimental analysis was conducted on a single node of the Piz Daint supercomputer[3] featuring an NVIDIA P100 GPU. The complete algorithm was implemented in the CUDA language [11] and compiled and executed with CUDA in version 8.0.

The test matrices we consider are all of size 1,000,000 × 1,000,000. They differ in the number of nonzeros they carry in each row, the bandwidth, and the condition number. The matrices are generated as band matrices with the aggregated number of nonzeros in a row on the main diagonal, and the values adjacent to the main diagonal set to -1.

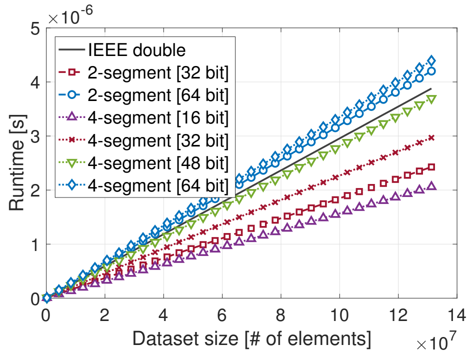


Fig. 2: Runtime for reading and writing data in double precision or customized precision with the accuracy of the data accesses indicated in the brackets.

Experimental results. In a first experiment we assess the cost of reading and writing data not stored in IEEE-supported formats but in the 2-segment and the 4-segment `CPMS`, respectively. The access routines for `CPMS` are not natively supported by hardware, and the hardware-specific implementations we developed include the access to the segment information array, the element-individual decision of the segment access, some instruction logic to access the distinct segments in memory, the type cast to the double precision operating format, and the reassembling of the double precision format from head and mantissa segments.

[3] `https://www.cscs.ch/computers/piz-daint/`

The results in Figure 2 reveal that reading 64-bit accuracy is 8% slower when using 2-segment CPMS and 13% slower when using 4-segment CPMS. The advantage of the customized precision format lies in the fact that the data access is much faster if reading the values with a shorter mantissa is acceptable. Reading 32-bit heads only is 1.6× faster than reading the data in double precision; Reading 16-bit heads is about 1.9× times faster.

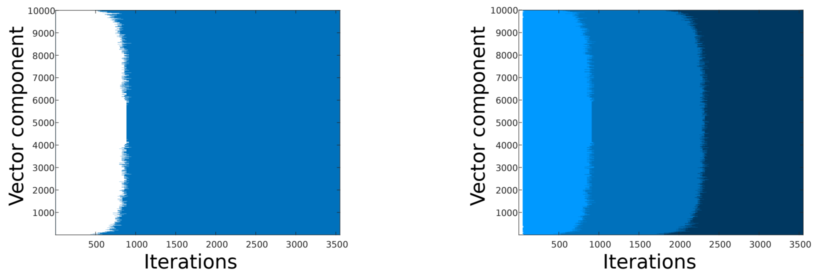
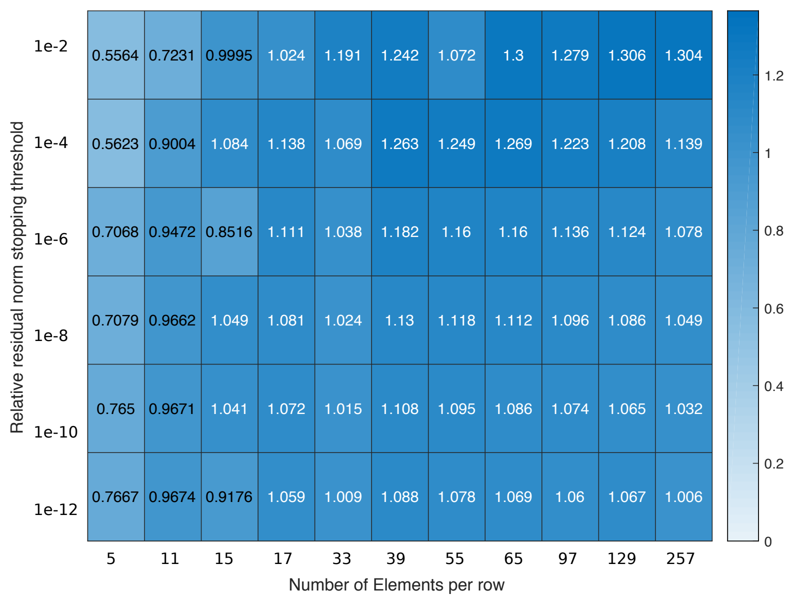


Fig. 3: Accuracy needs in adaptive Jacobi in a 2-segment (left) and a 4-segment (right) CPMS realization. The white-colored area indicates only the head is accessed, the blue areas indicate additional mantissa segment reads.

Next, we realize the adaptive precision Jacobi in the modular precision format. In Figure 3 we visualize for a small example problem with 129 nonzeros per row how the adaptive precision Jacobi method accesses the modular precision formats over the algorithm execution. Initially, the iteration process only reads the heads. As the execution progresses, mantissa segments are accessed on a component-individual basis once the stagnation test indicates the need for higher accuracy. The 16 bit head in the 4-segment modular precision format quickly becomes insufficient. We notice that the wavefront indicating the need for higher accuracy than 32 bits (which is reflected in the switch to 64 bits in the 2-segment modular precision and the switch to 48 bits in the 3-segment modular precision) is in both cases detected at the same iteration.

The experimental results presented in [3] reveal that the adaptive Jacobi can exhibit some convergence delay compared to a plain Jacobi as the mantissa extension detector may, depending on the test periodicity ϕ, not immediately identify stagnating components, and the threshold $\tilde{\delta}$ has to accept some rounding effects [3]. The question is whether this convergence delay, the overhead of the modular precision access routines, and the overhead of the stagnation detector is compensated by the faster access to reduced precision values in some relaxation steps. For this we compare the time-to-solution of the adaptive precision Jacobi with a reference implementation of plain Jacobi in IEEE double precision, both returning a solution approximation of the same accuracy. We consider different relative residual stopping thresholds as Jacobi relaxations are often employed as



	5	11	15	17	33	39	55	65	97	129	257
1e-2	0.5564	0.7231	0.9995	1.024	1.191	1.242	1.072	1.3	1.279	1.306	1.304
1e-4	0.5623	0.9004	1.084	1.138	1.069	1.263	1.249	1.269	1.223	1.208	1.139
1e-6	0.7068	0.9472	0.8516	1.111	1.038	1.182	1.16	1.16	1.136	1.124	1.078
1e-8	0.7079	0.9662	1.049	1.081	1.024	1.13	1.118	1.112	1.096	1.086	1.049
1e-10	0.765	0.9671	1.041	1.072	1.015	1.108	1.095	1.086	1.074	1.065	1.032
1e-12	0.7667	0.9674	0.9176	1.059	1.009	1.088	1.078	1.069	1.06	1.067	1.006

Fig. 4: Speedup factors of the adaptive precision Jacobi in a 2-segment modular precision realization.

smoother in multigrid methods or for providing rough solution approximations, e.g. in approximate sparse triangular solves [1,4].

Taking the plain Jacobi as reference, we report in Figure 4 the speedup factors of the adaptive precision Jacobi in a 2-segment modular precision realization for the distinct matrix/threshold combinations. The experimental results reveal that the adaptive precision Jacobi is attractive (about 30% faster) in particular for settings where a significant amount of matrix data has to be accessed in every iteration (many nonzero elements in every matrix row), and a large residual norm is acceptable (few component iterations requiring the data with 64 bit accuracy). The faster access to the matrix values fails to compensate the overhead of the stagnation detection for problems with only few nonzeros in every row.

In Figure 5 we report the same data for adaptive precision Jacobi in a 4-segment modular precision realization. Here, the reference Jacobi is always faster. This indicates that the modular precision format with finer segmentation is suitable only if high iteration counts allow to reduce the frequency of the stagnation test.

5 Concluding Remarks

We have presented the idea of radically decoupling the arithmetic format used in the floating-point operations from the format to store the data. We have pro-

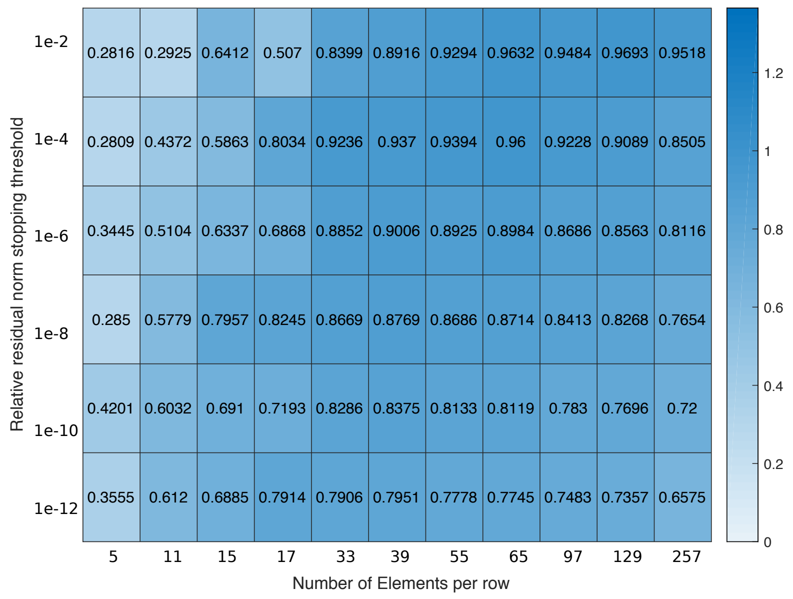


Fig. 5: Speedup factors of the adaptive precision Jacobi in a 4-segment modular precision realization.

posed a customized precision format that allows to access values much faster in memory if reduced accuracy is acceptable. Experimental results on high-end GPUs revealed that realizing mixed precision algorithms in the customized precision format can render resource savings without impacting the memory footprint or the accuracy of the final result.

We are convinced that the application field of customized precisions is much wider than what is presented in this work. We envision the customized precision realization of selection and sorting algorithms, as well as memory-bound algorithms like PageRank that are central for Big Data analytics.

Acknowledgements This work was supported by the "Impuls und Vernetzungsfond" of the Helmholtz Association under grant VH-NG-1241. The authors want to acknowledge the access to the PizDaint supercomputer at the Swiss National Supercomputing Centre granted under the project #d65. The authors would like to thank Goran Flegar and Enrique Quintana-Ortí for commenting on an earlier version of the paper.

References

1. Anzt, H., Chow, E., Dongarra, J.: Iterative sparse triangular solves for preconditioning. In: Träff, J.L., Hunold, S., Versaci, F. (eds.) Euro-Par 2015: Parallel Processing, Lecture Notes in Computer Science, vol. 9233, pp. 650–661. Springer Berlin Heidelberg (2015). https://doi.org/10.1007/978-3-662-48096-0_50, `http://dx.doi.org/10.1007/978-3-662-48096-0$_$50`
2. Anzt, H., Dongarra, J., Flegar, G., Higham, N.J., QuintanaOrtí, E.S.: Adaptive precision in blockjacobi preconditioning for iterative sparse linear system solvers. Concurrency and Computation: Practice and Experience **0**(0), e4460. https://doi.org/10.1002/cpe.4460, `https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4460`, e4460 cpe.4460
3. Anzt, H., Dongarra, J., Quintana-Ortí, E.S.: Adaptive Precision Solvers for Sparse Linear Systems. In: Proceedings of the 3rd International Workshop on Energy Efficient Supercomputing. pp. 2:1–2:10. ACM, New York, NY, USA (2015). https://doi.org/10.1145/2834800.2834802, `http://doi.acm.org/10.1145/2834800.2834802`
4. Anzt, H., Huckle, T.K., Bräckle, J., Dongarra, J.: Incomplete sparse approximate inverses for parallel preconditioning. Parallel Computing **71**(Supplement C), 1 – 22 (2018). https://doi.org/https://doi.org/10.1016/j.parco.2017.10.003, `http://www.sciencedirect.com/science/article/pii/S016781911730176X`
5. Buttari, A., Dongarra, J.J., Langou, J., Langou, J., Luszczek, P., Kurzak, J.: Mixed Precision Iterative Refinement Techniques for the Solution of Dense Linear Systems. Int. J. of High Perf. Comp. & Appl. **21**(4), 457–486 (2007)
6. Carson, E., Higham, N.J.: 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 (2017). https://doi.org/10.1137/17M1122918
7. Carson, E., Higham, N.J.: Accelerating the solution of linear systems by iterative refinement in three precisions. SIAM J. Scientific Computing **40**(2), A817–A847 (2018). https://doi.org/10.1137/17M1140819
8. Göddeke, D., Strzodka, R., Turek, S.: 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 (2007)
9. Higham, N.J.: Accuracy and Stability of Numerical Algorithms. Second edn. (2002)
10. IEEE Computer Society: 754-2008 - IEEE Standard for Floating-Point Arithmetic
11. NVIDIA Corp.: CUDA C Programming Guide. http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html, 9.0 edn.
12. Prikopa, K.E., Gansterer, W.N.: On mixed precision iterative refinement for eigenvalue problems. Procedia Computer Science **18**, 2647 – 2650 (2013). https://doi.org/https://doi.org/10.1016/j.procs.2013.06.002, `http://www.sciencedirect.com/science/article/pii/S1877050913006108`, 2013 International Conference on Computational Science