An Error Correction Solver for Linear Systems: Evaluation of Mixed Precision Implementations

Hartwig Anzt, Björn Rocker and Vincent Heuveline

Karlsruhe Institute of Technology (KIT)
Institute for Applied and Numerical Mathematics 4
Fritz-Erler-Str. 23
76133 Karlsruhe, Germany

hartwig.anzt@kit.edu, bjoern.rocker@kit.edu,
vincent.heuveline@kit.edu

Abstract. This paper proposes an error correction method for solving linear systems of equations and the evaluation of an implementation using mixed precision techniques.
While different technologies are available, graphic processing units (GPUs) have been established as particularly powerful coprocessors in recent years. For this reason, our error correction approach is focused on a CUDA implementation executing the error correction solver on the GPU. Benchmarks are performed both for artificially created matrices with preset characteristics as well as matrices obtained from finite element discretizations of fluid flow problems.

1 Introduction

The development of modern technology is characterized by simulations, that often are no longer performed through physical experiments, but through mathematical modeling and numerical simulation. In many cases, for example in computational fluid dynamics, massive computation power is needed, in order to handle large systems of linear equations.

Often iterative solvers are chosen for the solving process, since they can exploit the sparse structure of the affiliated matrix to compute an approximation of a certain accuracy usually faster than a direct solver.

The computational complexity of this problem depends on the characteristics of the linear system, the properties of the used linear solver and the floating point format. The floating point format determines not only the execution time when performing computations, but also the occurring rounding errors. A more complex floating point format usually leads to higher accuracy and higher computational effort. Today, most hardware architectures are configured for the IEEE 754 standard containing single precision and double precision as the main floating point formats. As their names indicate, the double precision format has twice the size of the single precision format, leading to a factor of two in computational cost while offering a higher precision.

In many cases single precision floating point operations are not suitable for scientific computation. The question arises, whether the whole algorithm has to be performed in the double precision format, or whether one can gain speed by computing parts of it in single precision and other parts in double precision, and still obtain double precision accuracy for the final result. One approach is to modify the algorithm of an error correction method such that the inner error correction solver uses a lower format than the working precision in the outer loop. As the final accuracy only depends on the stopping criterion of the refinement solver, the solution approximation is not affected. Still, it can be expected that the mixed precision approach performs faster than a plain solver in high precision, since the cheaper error correction solver in the low precision format may overcompensate the additional computations and typecasts.

2 Mixed Precision Error Correction Methods

2.1 Mathematical Background

Error correction methods have been known for more than 100 years, and have finally become of interest with the rise of computer systems in the middle of the last century. The core idea is to use the residual of a computed solution as the right-hand side to solve a correction equation.

The motivation for the error correction method can be obtained from Newton's method. Newton developed a method for finding successively better approximations to the zeros of a function $f(\cdot)$ by updating the solution approximation x_i through

$$x_{i+1} = x_i - (\nabla f(x_i))^{-1} f(x_i). \tag{1}$$

We now apply Newton's method (1) to the function $f(x) = b - Ax$ with $\nabla f(x) = A$. By defining the residual $r_i := b - Ax_i$, we obtain

$$\begin{aligned} x_{i+1} &= x_i - (\nabla f(x_i))^{-1} f(x_i) \\ &= x_i + A^{-1}(b - Ax_i) \\ &= x_i + A^{-1} r_i. \end{aligned}$$

Denoting the solution update with $c_i := A^{-1} r_i$, we can design an algorithm.

1: initial guess as starting vector: x_0
2: compute initial residual: $r_0 = b - Ax_0$
3: **while** ($\| Ax_i - b \|_2 > \varepsilon \| r_0 \|$) **do**
4: $\quad r_i = b - Ax_i$
5: $\quad$ solve: $Ac_i = r_i$
6: $\quad$ update solution: $x_{i+1} = x_i + c_i$
7: **end while**

Algorithm 1: Error Correction Method

Here x_0 is an initial guess. In each iteration, the inner correction solver searches for a c_i, such that $Ac_i = r_i$ with r_i being the residual of the solution approximation x_i. Then, the approximation of the solution x_i is updated to $x_{i+1} = x_i + c_i$.

2.2 Mixed Precision Approach

The underlying idea of mixed precision error correction methods is to use different precision formats within the algorithm of the error correction method, updating the solution approximation in high precision, but computing the error correction term in lower precision. This approach was also suggested by [1],[2], [3], and [4].

Hence, one regards the inner correction solver as a black box, computing a solution update in lower precision. The term high precision refers to the precision format that is necessary to display the accuracy of the final solution and we can obtain the following algorithm where $.^{high}$ denotes the high precision value and $.^{low}$ denotes the value in low precision. The conversion between the formats will be left abstract throughout this paper. Because especially the conversion of the matrix A is expensive, it should be stored in both precision formats, high and low precision. In the case of using hybrid hardware, A should be stored in the local memory of the hardware devices in the respectively used format.

Using the mixed precision approach to the error correction method, we have to be aware of the fact that the residual error bound of the error correction solver may not exceed the accuracy of the lower precision format. Furthermore, each error correction produced by the inner solver in lower precision cannot exceed the data range of the lower precision format. This means that the smallest possible error correction is the smallest number ϵ_{low}, that can be represented in the lower precision. Thus, we can not guarantee an accuracy of the final solution exceeding ϵ_{low} either. This can become a problem when working with very small numbers, because then the solution correction terms can not be denoted in low precision. However, in most cases, the problem can be avoided by converting the original values to a lower order of magnitude.

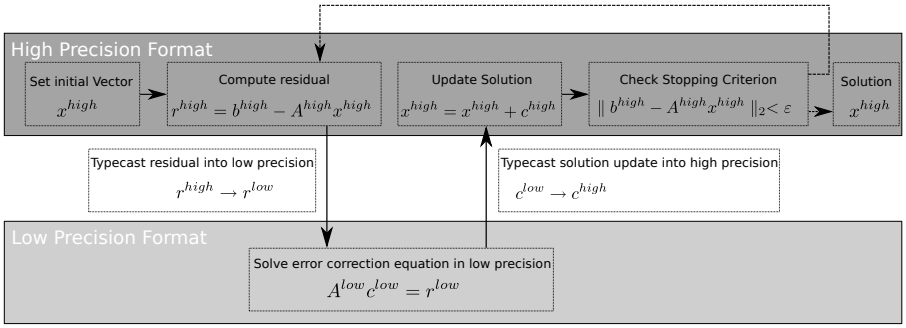


Fig. 1: Visualizing the mixed precision approach to an error correction solver

Using the displayed algorithm we obtain a mixed precision solver. If the final accuracy does not exceed the smallest number that can be represented in the lower precision, it gives exactly the same solution approximation as if the solver was performed in the high precision format. Theoretically, any precision can be chosen, but in most cases it is comfortable to use the IEEE 754 standard formats.

The computation of the correction loop $A^{low}c^{low} = r^{low}$ can be performed with a direct solver, or again with an iterative method. This implies that it is even possible to cascade a number of error correction solvers using decreasing precision.

In the case of an iterative solver as error correction solver, especially the iterative approaches to the Krylov subspace methods are of interest, since these provide an approximation of the residual error iteratively in every computation loop. Hence, one is able to set a certain relative residual stopping criterion for the iterative error correction solver. Possible Krylov subspace solvers include the CG Algorithm, GMRES, CGSTAB etc.(see e.g. [5]). The mixed precision error correction method based on a certain error correction solver poses the same demands to the linear problem, as the within used Krylov subspace solver.

In the case of a direct error correction solver, the solution update usually has a quality depending on the condition number of the system and the lower precision format. Hence, the solution improvement normally depends on the system, but is generally high. Despite the fact that direct methods are computationally expensive, they are of interest as error correction solver, since some of them own pleasant characteristics:

Using for example the LU solver as error correction solver, the LU decomposition has to be computed only in the first error correction loop. In the following loops, the stored decomposition can be used to perform the forward and backward substitution. Since these substitutions imply only low computational effort, they can, depending on the hardware structure, even be performed in the high precision format. This leads to accuracy advantages and economizes the algorithm by omitting the computationally expensive typecasts of the residual and the solution update.

It should be mentioned, that the solution update of the error correction solver is usually not optimal for the outer system, since the discretization of the problem in the lower precision format contains rounding errors, and it therefore solves a perturbed problem. When comparing the algorithm of an error correction solver to a plain solver, we realize, that the error correction method has more computations to execute. Each outer loop consists of the computation of the residual error term, a typecast, an initialization of a vector, the scaling process, the inner solver for the correction term, the reconversion of the data and the solution update. The computation of the residual error itself consists of a matrix-vector multiplication, a vector addition and a scalar product. Using a hybrid architecture, the converted data additionally has to be transmitted between the devices.

The mixed precision refinement approach to a certain solver is superior to the

plain solver in high precision, if the additional computations and typecasts are overcompensated by the cheaper inner correction solver using a lower precision format.

3 Numerical Experiments

3.1 Test Configurations

To be able to compare the performance of different implementation of the GMRES-(10) solver, we perform tests with different linear systems. In this work 10 denotes the restart parameter for the GMRES.

All solvers use the relative residual stopping criterion $\varepsilon = 10^{-10} \parallel r_0 \parallel_2$. Due to the iterative residual computation in the case of the plain GMRES-(10) solvers, the mixed GMRES-(10) solvers based on the mixed precision error correction method usually iterate to a better approximation since they compute the residual error explicitly, but as the difference is generally small, the solvers are comparable. In case of the mixed precision GMRES-(10) on the TESLA-System, the error correction solver is performed on one of the four available GPUs, while the solution update is led to the CPU of the same system. This is done to be able to handle larger problems since the amount of memory on the GPU is limited to 4 GB. Our hardware platform is therefore similar to a system equipped with one TESLA C1060, but in the following we denote the results with S1070.

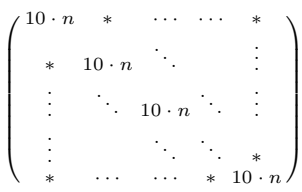
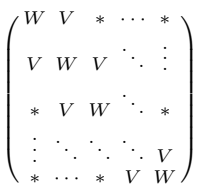
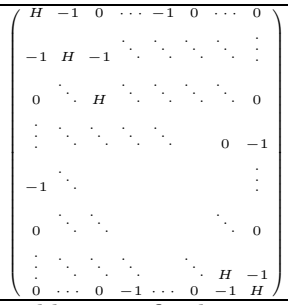
On the one hand, we use matrices with a preset condition number, preset sparsities, and increase the dimension. Depending on the sparsity, the matrices are stored in the matrix array storage format (MAS) or the compressed row storage format (CRS).

M1 The first test matrix, is a dense matrix that is generated with the DLATMR-routine out of the lapack-library [6]. As parameter set, we choose all entries smaller 1, and a condition number of 3. One drawback is that we cannot set positive definiteness in the routine itself. To ensure this property, we set the diagonal entries to be $10 \cdot n$, where n is the dimension of the matrix. By doing so, we lose the control of the condition number, but it is bounded by the former choice.

M2 The second test case is a sparse matrix similar to a 5-point stencil. The difference is the the term $H = 4 + 10^{-3}$ instead of $H = 4$ on the diagonal. Furthermore, the second and fifth upper and lower diagonal is filled with -1. The term 10^{-3} on the diagonal is used to control the condition number.

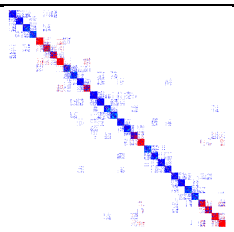
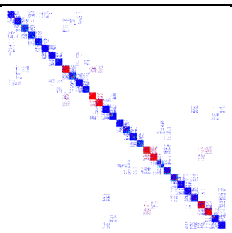
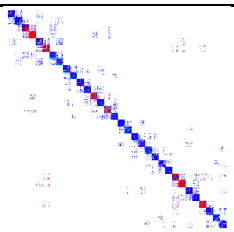
M3 We choose the third artificial test matrix to be an ill-conditioned dense matrix. As it is not easy to control the condition number of a dense matrix, we choose the term $W = 2 \cdot 10^3 \cdot n + n$ on the diagonal. The term on the upper and lower second diagonal is $V = 10^3 \cdot n$, and the rest of the matrix is filled with random double precision numbers between 0 and 1. These entries are the only entries we cannot control, and in one row, they can at most sum up

to $(n-3)\cdot(1-\varepsilon)$, but they can also sum up to $(n-3)\cdot\varepsilon$, with $\varepsilon > 0$. Since the random numbers are for large dimension evenly distributed, we assume that they sum up to $0.5\cdot(n-3)$.

M1	**M2**	**M3**
		
problem: artificial matrix problem size: variable sparsity: $nnz = n^2$ cond. number: $\kappa < 3$ storage format: MAS	problem: artificial matrix problem size: variable sparsity: $nnz = n^2$ cond. number: $\kappa \approx 8\cdot 10^3$ storage format: MAS	problem: artificial matrix problem size: variable sparsity: $nnz = 5n$ cond. number: $\kappa \approx 8\cdot 10^3$ storage format: CRS

Tab. 1: Structure plots and properties of the artificial test-matrices

On the other hand, we additionally use linear problems that were obtained out of a discretization of CFD. The three systems of linear equations CFD1, CFD2 and CFD3 are affiliated with the 2D modeling of a Venturi Nozzle in different discretization fineness. The distinct number of supporting points leads to different matrix characteristics concerning the dimension, the number of non-zeros, and the condition number.

CFD1	**CFD2**	**CFD3**
		
problem: 2D fluid flow problem size: $n = 395009$ sparsity: $nnz = 3544321$ cond. number: unknown storage format: CRS	problem: 2D fluid flow problem size: $n = 634453$ sparsity: $nnz = 5700633$ cond. number: unknown storage format: CRS	problem: 2D fluid flow problem size: $n = 1019967$ sparsity: $nnz = 9182401$ cond. number: unknown storage format: CRS

Tab. 1: Sparsity plots and properties of the CFD test-matrices

3.2 Numerical Results

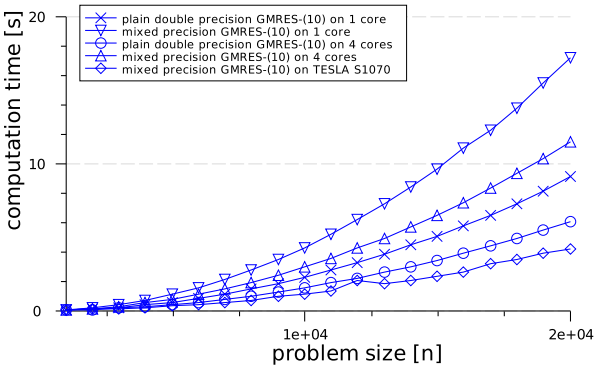


Fig. 2: Test case: M1, relative residual stopping criterion $\varepsilon = 10^{-10}$;

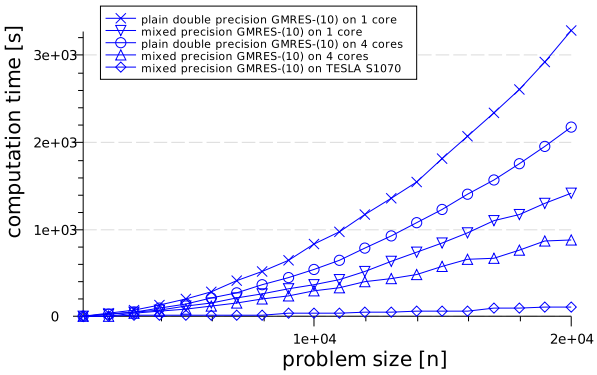


Fig. 3: Test case: M2, relative residual stopping criterion $\varepsilon = 10^{-10}$;

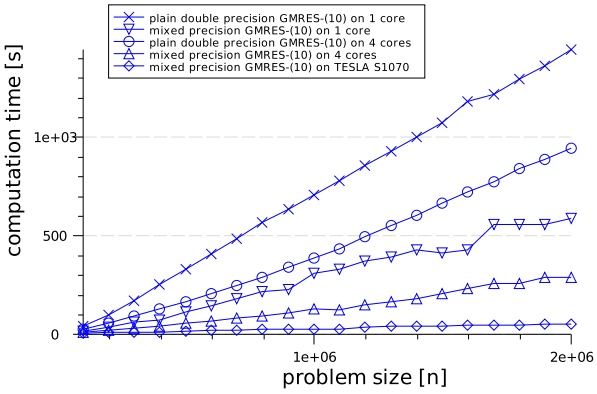


Fig. 4: Test case: M3, relative residual stopping criterion $\varepsilon = 10^{-10}$;

CFD1

solver type	computation time [s]
sequential double GMRES-(10) on IC1	3146.61
sequential mixed precision GMRES-(10) on IC1	1378.56
parallel double GMRES-(10) on IC1 using 4 cores	1656.53
parallel mixed precision GMRES-(10) on IC1 using 4 cores	712.83
mixed precision GMRES-(10) on NVIDIA TESLA S1070	438.13

CFD2

solver type	computation time [s]
sequential double GMRES-(10) on IC1	13204.70
sequential mixed precision GMRES-(10) on IC1	5924.32
parallel double GMRES-(10) on IC1 using 4 cores	6843.66
parallel mixed precision GMRES-(10) on IC1 using 4 cores	3495.09
mixed precision GMRES-(10) on NVIDIA TESLA S1070	2092.84

CFD3

solver type	computation time [s]
sequential double GMRES-(10) on IC1	60214.50
sequential mixed precision GMRES-(10) on IC1	41927.40
parallel double GMRES-(10) on IC1 using 4 cores	32875.10
parallel mixed precision GMRES-(10) on IC1 using 4 cores	19317.00
mixed precision GMRES-(10) on NVIDIA TESLA S1070	10316.70

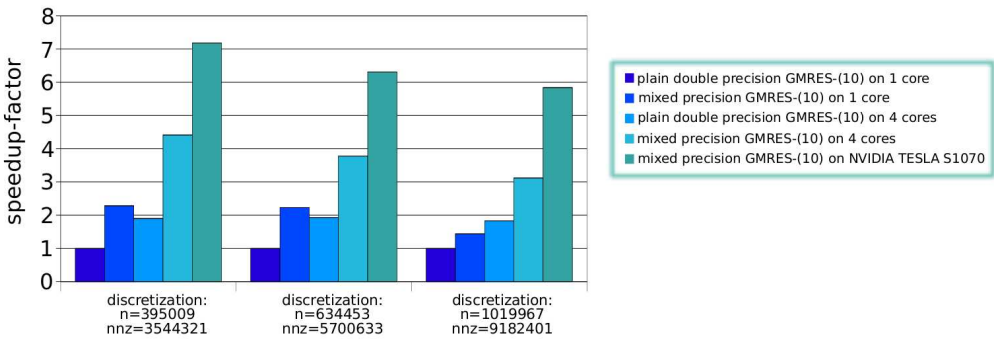


Fig. 5: Speedup of the different solvers for the CFD simulation of the Venturi Nozzle in different discretization fineness; $\varepsilon = 10^{-10}$; $\varepsilon_{\text{inner}} = 0.1$;

3.3 Result Interpretation

In the first test (Fig. 2), the low condition number leads to a good convergence rate of GMRES-(10) and there are only few iterations necessary to obtain a solution approximation fulfilling the stopping criterion. The additional computational cost of the mixed precision iterative refinement approach is large com-

pared to the computational cost of the pure double solver. Therefore is the mixed precision GMRES-(10) neither for the sequential, nor for the parallel case able to compete with the plain double precision GMRES-(10). The TESLA S1070-implementation of the mixed GMRES-(10) outperforms the solvers on the IC1 due to the larger number of cores and the excellent single precision performance of the GPU, that can be exploited by the inner error correction solver. It should be mentioned, that the factors between the computation time of the different solver types are independent of the dimension n of the linear system that is solved.

The difference of the second test (Fig. 3) to the first test case is the fairly high condition number of $\kappa \approx 8 \cdot 10^3$ of the linear system. Due to the high number of iterations the linear solvers have to perform, the overhead for the mixed precision method is considerably small. Therefore, also on the IC1, the additional costs can be overcompensated by the speedup gained by performing the inner solver in a lower precision format. Both, for the parallel and the sequential case we gain a factor of about two using the mixed precision iterative refinement approach instead of the plain double precision GMRES-(10). The lower precision format leads to a shorter execution time when performing elementary computations on the one hand, and to a more efficient use of the memory bandwidth on the other hand. The memory space needed to store one single precision floating point number is half the size that is needed for one double precision floating point number. The memory bandwidth is usually the limiting factor of the computational power of a system. Using a lower precision format, the processors have shorter waiting time for the data, and the system gains a higher efficiency. Since this argument applies to all memory levels, the speedup using single precision for the GMRES-(10) can even exceed the factor 2 that characterizes the speedup of a general purpose CPU when switching from double to single precision computations.

The speedup factor gained by performing the mixed GMRES-(10) on the TESLA S1070 is almost 15 with respect to the sequential plain double GMRES-(10) on the IC1. Again we can observe, that the speedup factors between the different solvers on the different Hardware platforms remain constant, independent of the problem size.

For the third test case (Fig. 4), again an artificial test matrix is used with a condition number of $\kappa \approx 8 \cdot 10^3$. The difference to the former test cases is, that we now apply the solvers to sparse linear systems where the matrices are stored in the CRS format. The low number of nonzero entries leads no longer to a computational cost that is quadratically increasing with the dimension, but linearly. Furthermore is the total computational effort lower compared to the tests with matrix structure M2. Despite some perturbations, that can be explained by rounding effects and the use of different cache levels, we can still observe that the quotients between the solver types remain the same, independently of the dimension of the linear system. Again, both for the sequential and the parallel case, the mixed precision GMRES-(10) on the IC1 outperform the plain double implementations due to the fact, that the additional computational cost of the iterative refinement scheme is overcompensated by the speedup gained through

the execution of the inner solver in single precision. The implementation on the TESLA S1070 can additionally exploit the excellent single precision performance of the highly parallelized GPU. Furthermore approximate the speedups gained by using the mixed precision GMRES-(10) the speedups of test case M2.

The tests with the matrices CFD1, CFD2 and CFD3 show that the mixed precision iterative refinement approach is also beneficial when applying solvers to real world problems (Fig. 5). The mixed GMRES-(10) solvers outperform the plain double GMRES-(10) implementations for all test problems, both in the sequential and the parallel case. The reason is again the fact, that the additional computational cost of the iterative refinement approach is overcompensated by the cheaper inner solver using a lower precision format.

Using hybrid hardware, the mixed GMRES-(10) on the TESLA S1070 even generates speedups up to 7 with respect to the plain double implementation on the IC1. It can be observed, that this factor decreases for increasing dimension. The reason is, that for large data amounts, the connection between the host CPU and the GPU slows the mixed GMRES-(10) down.

4 Hardware Platform and Implementation Issues

The utilized TESLA-System is equipped with one NVIDIA TESLA S1070[1]. The two host nodes, each connected via a PCIe 2.0 x16 to the S1070, are each equipped with two Intel Xeon 5450 CPUs. The Intel MKL in version 10.1.1.019 and the Intel compiler in version 11.0.074 was used.

The InstitutsCluster[2] (IC1) is located at the Karlsruhe Institute of Technology (KIT) and consists of 200 computing nodes each equipped with two Intel quad-core EM64T Xeon 5355 processors, owning 16 GB of main memory. Peak performance of one node is about 85,3 GFlops. For a detailed performance evaluation see [7]. The software platform is the Intel CMKL in version 10.1.2.024 and the Intel compiler in version 10.1.022.

5 Conclusions and Future Work

The numerical tests in this paper have shown the high potential of using different precision formats within the proposed error correction solver.

The obtained algorithm is flexible in terms of choosing the inner correction solver, and robust in terms of numerical stability. The possibility of performing the error correction solver on a coprocessor increases the potential of mixed precision methods, as they can be implemented efficiently on hybrid systems. Performing the error correction solver of an error correction method in a lower format leads to an overall increase in performance for a large number of problems.

On a CPU, performing the error correction method in mixed precision, one often achieves a speedup factor of two compared to the plain solver in double precision.

[1] http://www.nvidia.com
[2] http://www.scc.kit.edu

When using hybrid hardware, consisting of coprocessors specialized on low precision performance, even higher speedup factors can be expected. In the numerical experiments for the FEM discretizations of the Venturi Nozzle we achieved speedups of more than seven for our CUDA implementation.

Still, a very ill-conditioned problem can lead to a high number of additional outer iterations necessary to correct the rounding errors, that arise from the use of a lower precision format in the error correction solver. For the worst case, the inner solver will not converge. Due to the fact that we are usually not able to determine a priori whether the mixed precision method is superior for a specific problem, an optimized implementation of the solver would execute the first solution update of the mixed precision error correction method and determine, depending on the improvement of the solution approximation, whether it should continue in the mixed precision mode or whether it should use the plain solver in high precision. The next step beyond this strategy of changing between single and double precision is to use techniques around adaptive precision, where the precision is adjusted according to the convergence in the inner solver. FPGAs and related technologies may provide the capabilities for such algorithms.

For an efficient implementation of the mixed precision error correction techniques in a solver suite, some additional work is necessary, especially concerning the use of preconditioners. This may not only increase the stability of the solver, but also its performance. In such an environment, the mixed precision error correction methods form powerful solvers for FEM simulations and beyond.

References

1. D. Göddeke, R. Strzodka, and S. Turek. Performance and accuracy of hardware–oriented native–, emulated– and mixed–precision solvers in FEM simulations. 2007.
2. D. Göddeke and R. Strzodka. Performance and accuracy of hardware–oriented native–, emulated– and mixed–precision solvers in FEM simulations (part 2: Double precision gpus). Technical report, Fakultät für Mathematik, TU Dortmund, 2008.
3. Marc Baboulin, Alfredo Buttari, Jack J. Dongarra, Julie Langou, Julien Langou, Piotr Luszcek, Jakub Kurzak, and Stanimire Tomov. Accelerating scientific computations with mixed precision algorithms. 2008.
4. Alfredo Buttari, Jack J. Dongarra, Julie Langou, Julien Langou, Piotr Luszcek, and Jakub Kurzak. Mixed precision iterative refinement techniques for the solution of dense linear systems. 2007.
5. Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.
6. Fujitsu. *BLAS LAPACK User's Guide*, January 2003.
7. V. Heuveline, B. Rocker, and S. Ronnas. Numerical simulation on the sicortex supercomputer platform: a preliminary evaluation. In *EMCL Preprint Series*, 2009.