Mixed Precision Strategies for Solving Sparse Linear Systems with BiCGStab

ANI ANCIAUX-SEDRAKIAN, IFPEN, France
HUGO DORFSMAN, IFPEN, France and Sorbonne Université, CNRS, LIP6, France
THOMAS GUIGNON, IFPEN, France
FABIENNE JÉZÉQUEL, Sorbonne Université, CNRS, LIP6, France and Université Paris-Panthéon-Assas, France
THEO MARY, Sorbonne Université, CNRS, LIP6, France

While numerical simulations have traditionally been carried out in double precision, the emergence of efficient hardware providing lower precisions has motivated their introduction to increase performance. In particular, there is a clear interest in using lower precision for iterative linear solvers based on Krylov subspaces, which are widely used in large-scale simulations. In this article, we investigate various methods for applying mixed precision to preconditioned BiCGStab solvers, including iterative refinement techniques, and introduce a new mixed-precision approach based on flying restart (BiCGStab-FR). We perform an in-depth, detailed performance analysis of these mixed precision BiCGStab strategies and show how to maximize the performance gains resulting from the use of lower precisions by carefully implementing the core kernels with SIMD intrinsics and suitable sparse matrix layouts. We present experimental results on a wide range of real-life matrices, including reservoir simulation and CO2 storage applications. Our performance results demonstrate significant time reductions of up to 45% with mixed (double/single) precision BiCGStab.

CCS Concepts: • **Mathematics of computing → Mathematical software performance**; **Solvers**; *Numerical analysis.*

Additional Key Words and Phrases: mixed precision, BiCGStab, numerical linear algebra, iterative solver, sparse matrix, SpMV, triangular solve, SIMD

ACM Reference Format:
Ani Anciaux-Sedrakian, Hugo Dorfsman, Thomas Guignon, Fabienne Jézéquel, and Theo Mary. 2025. Mixed Precision Strategies for Solving Sparse Linear Systems with BiCGStab. 1, 1 (October 2025), 23 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Solving linear systems of equations is a fundamental problem in many computational scientific fields. In this work, we focus on solving large sparse linear systems of equations that arise from solving partial differential equations. The obtained linear systems are in general solved with iterative Krylov subspace solvers; in this article, we focus in particular on the BiConjugate Gradient STABilized (BiCGStab) [Saad 2003; van der Vorst 1992]. The solution phase is generally the most memory

Authors' Contact Information: Ani Anciaux-Sedrakian, IFPEN, France, ani.anciaux-sedrakian@ifpen.fr; Hugo Dorfsman, IFPEN, France and Sorbonne Université, CNRS, LIP6, France, hugo.dorfsman@ifpen.fr; Thomas Guignon, IFPEN, France, thomas.guignon@ifpen.fr; Fabienne Jézéquel, Sorbonne Université, CNRS, LIP6, France and Université Paris-Panthéon-Assas, France, fabienne.jezequel@lip6.fr; Theo Mary, Sorbonne Université, CNRS, LIP6, Paris, France, theo.mary@lip6.fr.

ACM XXXX-XXXX/2025/10-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

and time consuming part of the simulation. In this context, the performance of the linear solver is therefore a key issue to tackle large-scale problems.

Recent trends in hardware development have introduced highly efficient but less precise hardware. Reduced precision operations are faster to compute and use less memory and energy. In order to leverage the available computing power to reduce the execution time of sparse iterative solvers while maintaining acceptable accuracy, we will explore mixed precision strategies that combine both high and low precision computations.

Several approaches to exploit mixed precision arithmetic in iterative solvers have been investigated; see Higham and Mary [2022] and Abdelfattah et al. [2020] for an overview. One common approach consists in reducing the cost of the preconditioner either by storing, computing, and/or applying it in low precision. Arioli and Duff [2008] use a low precision LU factorization to precondition flexible GMRES (FGMRES); Anzt et al. [2018] store a block Jacobi preconditioner in adaptive precision to reduce communication costs while keeping the computation in high precision. Buttari et al. [2025] analyze several mixed precision preconditioned GMRES strategies. Another approach that has been explored under many variations is to use an inner–outer or iterative refinement scheme, where the inner loop in low precision performs most of the computations and the outer loop in higher precision refines the accuracy of the solution. Several algorithms for iterative refinement have been studied in mixed precision, see Carson and Higham [2018] for a general analysis and further references; in the context of iterative solvers, the inner–outer scheme often takes the form of a mixed precision restarted GMRES, where the inner cycles are performed in low precision [Lindquist et al. 2022; Loe et al. 2021; Zhao et al. 2022]. Both approaches (low precision preconditioner and inner–outer scheme) can naturally be combined: for example, Carson and Higham [2018] and Amestoy et al. [2023, 2024] use a low precision LU factorization combined with a GMRES-based iterative refinement to solve ill-conditioned systems.

As highlighted, most of the existing literature has concentrated on GMRES and its variants when designing mixed precision iterative solvers for general unsymmetric systems. In contrast, there has been relatively few studies on BiCGStab, despite its lower time and storage per-iteration costs that it achieves by not storing the Krylov basis [Saad 2003]. To the best of our knowledge, only one previous work has explored the use of mixed precision in BiCGStab, namely Zhao et al. [2023]. This motivates a dedicated study of how mixed precision can accelerate BiCGStab, along with a thorough performance analysis.

We evaluate three different strategies. First, we show that the preconditioner can be not only computed but also applied in low precision without any extra cost; this is because the standard formulation for preconditioned BiCGStab is inherently flexible [Vogel 2007], that is, can tolerate variations coming from low precision rounding errors. This is in contrast to GMRES, where the flexible variant requires to double the size of the Krylov basis. This first strategy only reduces the cost of the preconditioning operations; the next two strategies aim at also reducing the cost of the remaining operations.

The second strategy investigates the use of BiCGStab-based iterative refinement (hereinafter BiCGStab-IR), that is, iterative refinement with a low precision BiCGStab as inner solver. This strategy is much less common than GMRES-based iterative refinement, primarily because restarting GMRES is helpful to contain the size of the Krylov basis, whereas restarting BiCGStab (in a uniform precision context) is generally unhelpful and can on the contrary delay convergence. Here, we show that in a mixed precision context restarting BiCGStab becomes relevant since it allows for attaining high accuracy despite performing most of the operations in low precision. The downside of this approach is that restarting can slow down the convergence; hence there is a tradeoff between the cost of the iterations and their number. We note that the same idea has been recently and

independently investigated by Zhao et al. [2023], who also observed the same issue with delayed convergence.

The third strategy that we investigate aims at alleviating the delayed convergence of BiCGStab-IR. It is based on a variant of BiCGStab called "flying restart" (hereinafter BiCGStab-FR), which was proposed in 1996 by Sleijpen and van der Vorst [1996]. The original motivation for this variant was to improve the attainable accuracy of BiCGStab in finite (uniform) precision. Similarly to BiCGStab-IR, BiCGStab-FR regularly restarts the solver by replacing the right-hand side with the explicit residual of the current solution; however, BiCGStab-FR does not reset the internal state of the solver and thus can decrease the risk of delaying the convergence. We propose a novel mixed precision variant of BiCGStab-FR which also attains high accuracy and can converge faster than BiCGStab-IR. Moreover, the convergence of this mixed precision BiCGStab-FR method is less sensitive to the restart parameter than BiCGStab-IR, which makes its numerical behavior more consistent.

We carry out an in-depth performance analysis and comparison of these strategies, using single (fp32) and double (fp64) precision arithmetics. Our benchmarks uses real-life matrices, notably arising in reservoir simulation applications; we also use matrices from the publicly available SuiteSparse collection [Davis and Hu 2011] for reproducibility purposes. Our results demonstrate the potential of these strategies to accelerate BiCGStab, with reductions of the solution time by up to 45%.

To summarize, the main contributions of this paper are:

- Demonstration of BiCGStab's flexibility for low-precision preconditioning: We demonstrate that the intrinsic flexibility of the BiCGStab algorithm enables the effective use of preconditioners computed in lower precision, thereby reducing computational cost without significantly compromising convergence.
- Integration of BiCGStab with iterative refinement: We propose a hybrid strategy that combines BiCGStab with iterative refinement to enhance both numerical accuracy and overall performance in mixed-precision environments.
- Introduction of the mixed-precision flying restart BiCGStab algorithm: We develop a new variant, BiCGStab-FR, specifically designed to mitigate delayed convergence phenomena and improve robustness and consistency when operating in mixed precision.
- Extensive numerical validation: We perform extensive numerical experiments to validate the proposed approaches and to assess their convergence behavior across a variety of test systems and matrix types.
- Detailed performance analysis: We provide an in-depth analysis of the performance improvements achieved through mixed (double/single) precision computations, and we identify strategies to maximize these gains using optimized SIMD intrinsics and sparse matrix layouts.

The rest of this article is organized as follows. We begin, in section 2, by providing background on sparse linear solvers and mixed precision algorithms and describing related work. Then, in section 3, we present several mixed precision strategies specific to BiCGStab. We experimentally evaluate their numerical behavior and performance in section 4. Finally, we provide concluding remarks in section 5.

2 Background and related work

2.1 BiCGStab

The BiCGStab method, introduced by van der Vorst [1992], is an iterative Krylov subspace technique designed to solve general nonsingular linear systems $Ax = b$. While both the BiConjugate gradient

(BiCG) method and BiCGStab aim to solve linear systems, BiCGStab modifies the approach to address the often erratic convergence patterns seen in BiCG by incorporating a stabilization step.

BiCGStab shares similarities with the Generalized Minimal RESidual (GMRES) method, as both are Krylov subspace solvers that iteratively compute solutions to unsymmetric linear systems. Due to its residual minimization property, GMRES converges faster than BiCGStab. However, BiCGStab offers advantages in terms of memory usage and computational complexity compared with GMRES, as it avoids the need to store and orthogonalize a growing Krylov basis, making it more efficient for large-scale problems.

In this work, we focus on the right-preconditioned version of BiCGStab described in Algorithm 1. At each iteration, the main computational bottleneck consists of two matrix–vector products, two applications of the preconditioner, and several vector operations (dot products and axpy). We stop the solver whenever the norm of the residual $\|r_i\|$ falls below a prescribed threshold ϵ, or after the maximum number of iterations $i_{\max}$ has been performed.

Algorithm 1: Right-preconditioned BiCGStab.

Input: A, b, $i_{\max}$, ϵ
Output: x_i
Initialize $x_0, \bar{r_0}$ arbitrarily.
Construct preconditioner M
$p_0 = r_0 \leftarrow b - Ax_0$
for ($i = 1$ **to** $i_{\max}$)
  Solve: $M\hat{p} = p_{i-1}$
  $\alpha = (\bar{r_0}, r_{i-1})/(\bar{r_0}, A\hat{p})$
  $s = r_{i-1} - \alpha A\hat{p}$
  Solve: $M\hat{s} = s$
  $\omega = (A\hat{s}, s)/(A\hat{s}, A\hat{s})$
  $x_i = x_{i-1} + \alpha\hat{p} + \omega\hat{s}$
  $r_i = s - \omega A\hat{s}$
  if ($\|r_i\| \leq \epsilon$) **then** exit loop.
  $\beta = \alpha(\bar{r_0}, r_i)/\omega(\bar{r_0}, r_{i-1})$
  $p_i = r_i + \beta(p_{i-1} - \omega A\hat{p})$

A variant of the BiCGStab algorithm called "flying restart," was first introduced in 1996 by Sleijpen and van der Vorst [1996]. It is based on a periodic recomputation of the residual with the aim to improve the attainable accuracy of the solver. Indeed, by grouping the updates to the residual, this variant aims to minimize cumulative numerical errors, thereby enhancing the accuracy of the solution. The original flying restart variant was developed in a uniform precision setting; in section 3.3, we will propose a mixed precision implementation of this variant.

2.2 Low/mixed precision preconditioning

Low and/or mixed precision arithmetic has been particularly successful to accelerate the preconditioning cost of iterative solvers. Indeed, since preconditioners are inherently approximate, exploiting low precision for preconditioning is a natural strategy. We can distinguish various strategies depending on which specific step is carried out in lower precision.

A first approach is to both compute and apply the preconditioner in lower precision. This reduces both the cost of constructing the preconditioner, and the cost of applying it at each iteration.

However, the rounding errors incurred when applying the preconditioner in low precision depend on the vector it is applied to, and therefore vary from one iteration to the other. This essentially makes the preconditioner non-constant and therefore requires a flexible solver. For example, Arioli and Duff [2008] show that FGMRES implemented in double precision and preconditioned with an LU factorization computed in single precision can achieve backward stability at double precision, even for ill-conditioned systems. More recently, Carson and Daužickaitė [2024] and Buttari et al. [2025] have analyzed various mixed precision preconditioned GMRES strategies, and shown that FGMRES achieves better bounds on the attainable accuracy when the preconditioner is applied in lower precision.

Alternatively, the preconditioner can be computed and stored in low precision but applied in high precision. This removes the need for a flexible solver, but the cost of applying the preconditioner becomes quite preconditioner- and architecture-dependent. Indeed, for memory bound computations, applying in high precision a preconditioner stored in low precision may be almost as fast as applying it in low precision, provided that an efficient conversion from low to high precision is available. This has motivated the development of so-called memory accessor approaches that aim at accelerating memory-bound computations by reducing the cost of memory accesses. Such approaches have for example been proposed for accelerating memory-bound BLAS operations [Grützmacher et al. 2023], sparse matrix–vector products [Graillat et al. 2024; Mukunoki et al. 2023], hierarchical matrix–vector products and LU factorizations [Kriemann 2023, 2024], and triangular solves with dense or block low-rank LU factors [Amestoy et al. 2025].

Finally, a last strategy is to compute and/or apply the preconditioner in mixed precision itself. This is meaningful when the preconditioner can be split into several parts each stored in a different precision; such approaches as referred to as adaptive precision [Higham and Mary 2022, sect. 14]. For example, Anzt et al. [2018] propose an adaptive precision block-Jacobi preconditioner where the precision of each block is determined by its condition number. Amestoy et al. [2022] describe an adaptive precision block low-rank factorization where each low-rank block is partitioned into low-rank components of different precisions.

2.3 Iterative refinement

Given an approximate solution $\widehat{x}$ to $Ax = b$, iterative refinement is a long-standing technique for improving the accuracy of $\widehat{x}$. As described in Algorithm 2, it consists in evaluating the residual $r = b - A\widehat{x}$ and using this residual to compute a correction term d, an (approximate) solution of the system $Ad = r$. By iteratively updating $\widehat{x} \leftarrow \widehat{x} + d$, the algorithm progressively improves the solution's accuracy. We stop the refinement whenever the norm of the residual $\|r\|$ falls below a prescribed threshold ϵ, or after the maximum number of iterations $i_{\max}$ has been performed.

This method has been analyzed in various contexts, including both uniform and mixed precision scenarios; see Higham and Mary [2022, sect. 6] for an extensive review.

Algorithm 2: Iterative refinement.

Input: A, b, $\widehat{x}$, $i_{\max}$, ϵ
Output: $\widehat{x}$

1 **for** ($i = 1$ **to** $i_{\max}$)
2 $\quad r \leftarrow b - A\widehat{x}$ (Prec u_{high})
3 $\quad$ **if** ($\|r\| \leq \epsilon$) **then** exit loop.
4 $\quad$ Solve $Ad = r$. (Prec u_{low})
5 $\quad \widehat{x} \leftarrow \widehat{x} + d$ (Prec u_{high})

In uniform precision, iterative refinement is interesting not only for improving the attainable accuracy of an already stable linear solver, but also for stabilizing an unstable solver. This use was analyzed by Jankowski and Woźniakowski [1977], who proved that any linear solver that is not too unstable can be made normwise backward stable. Skeel [1980] refined the analysis for linear solvers based on LU factorization with partial pivoting, proving a small componentwise backward error can be obtained. The analysis was later generalized by Higham [1997, 1991]. Uniform precision iterative refinement has been particularly used to improve the accuracy of sparse direct solvers based on less robust pivoting strategies than standard partial pivoting, such as static pivoting: see, for example, Amestoy et al. [2023]; Arioli et al. [2007]; Li and Demmel [1998].

Iterative refinement has been equally as successful in mixed precision settings. The most general analysis is that of Carson and Higham [2018], which covers the use of three different precisions and an arbitrary solver for the inner system $Ad = r$. In this work we are particularly interested in the variants of iterative refinement that use lower precision to solve this inner system. Indeed, the analysis of iterative refinement shows that even if the inner system is solved in low precision, the outer refinement process can achieve high accuracy, provided that the matrix A is not too ill-conditioned. The precise condition for convergence to be guaranteed depends on the specific solver that is used. Direct, iterative, or a combination of both types of solvers have been proposed.

In particular, a low precision LU factorization $A \approx \widehat{L}\widehat{U}$ can be used to solve the inner system by substitution, $d \approx \widehat{U}^{-1}\widehat{L}^{-1}r$, also using low precision. This approach was first proposed by Langou et al. [2006] with a single precision factorization and then extensively investigated in several other works, including based on half precision factorizations [Amestoy et al. 2023; Baboulin et al. 2009; Haidar et al. 2020, 2018].

Alternatively, the inner system can instead be solved with a (low precision) iterative solver; in this case, we obtain an inner–outer scheme. This was first proposed by Turner and Walker [1992], who describe a mixed precision restarted GMRES solver with the inner loop in single precision and the outer loop in double precision; this is equivalent to iterative refinement with GMRES as solver of the inner system $Ad = r$, called GMRES-based iterative refinement (GMRES-IR). This type of approach has also been extensively studied in subsequent works [Lindquist et al. 2022; Loe et al. 2021; Zhao et al. 2022]. Buttari et al. [2008] propose a slightly different approach using FGMRES as the outer solver in double precision (instead of iterative refinement) and GMRES as the preconditioner in single precision. This approach also takes the form of a mixed precision inner–outer scheme, albeit not based on iterative refinement. It was also studied by Baboulin et al. [2009].

The two previous approaches can be combined by computing low precision LU factors and solving the inner system $Ad = r$ by an iterative method preconditioned by these LU factors. This approach is then equivalent to mixed precision preconditioned restarted GMRES. It is particularly useful to handle ill-conditioned matrices. Indeed, as mentioned above, if the inner system is solved in a too low precision (either by direct substitution or by a low precision iterative solver), convergence will no longer be guaranteed. Instead, convergence can be maintained even with low precision LU factors if these are only used to precondition an iterative solver. This was first analyzed by Carson and Higham [2017, 2018] with two and then three precisions, and their analysis was further generalized by Amestoy et al. [2024] to allow for up to five precisions. These analyses allow for proving convergence even for ill-conditioned matrices. The practical potential of this LU-preconditioned GMRES-IR has been demonstrated in various studies, including those of Haidar et al. [2020] for dense systems and Amestoy et al. [2023] for sparse systems. GMRES-IR has also been used with preconditioners other than direct LU factorizations, such as incomplete LU and

Jacobi [Lindquist et al. 2022], block low-rank LU [Amestoy et al. 2023], or sparse approximate inverse (SPAI) [Carson and Khan 2023].

Finally, another recent development has combined mixed precision GMRES with deflation or augmentation techniques; see the papers by Oktay and Carson [2023] and Jang et al. [2025].

3 Mixed precision strategies for BiCGStab

In this section, we investigate various strategies to exploit mixed precision in BiCGStab. In section 3.1, we show how the preconditioning operations can be accelerated by performing them in low precision. Then, in section 3.2, we investigate a mixed precision BiCGStab-based iterative refinement (BiCGStab-IR) approach. Finally, in section 3.3, we propose a mixed precision version of the flying restart (BiCGStab-FR) approach. We compare BiCGStab-IR and BiCGStab-FR and discuss how to choose their restart parameter in section 3.4.

This section illustrates the behavior of the proposed strategies through experiment on the sherman4 matrix (see Table 1). We use this matrix as a representative test case to illustrate some interesting behaviors and trends; we will perform a much more extensive set of experiments on a wider ranger of matrices in section 4.

3.1 Low precision preconditioning

As explained in section 2.2, a natural strategy to accelerate preconditioned iterative solvers is to perform the preconditioning operations in low precision. To maximize the performance, it is desirable to use low precision not only to construct the preconditioner, but also to apply it at each iteration. This, however, requires an iterative solver that can handle non-constant preconditioners, since the rounding errors incurred in the application of the preconditioner introduce variations from one iteration to the other. In the case of Krylov subspace solvers, this means that we must use a flexible formulation of the solver, such as FGMRES [Saad 1993], FCG [Notay 2000], etc.

Importantly, the standard formulation of right-preconditioned BiCGStab (as outlined in Algorithm 1) is already flexible. This formulation was proposed by Vogel [2007]; the motivation there was to use another iterative solver as preconditioner, but it can also be used to exploit low precision preconditioners. We are not aware of any previous work making this observation.

The algorithm resulting from this first strategy is described in Algorithm 3. It is a mixed precision variant of Algorithm 1 where we construct and apply the preconditioner M in precision u_{low}, and keep the rest of the operations in precision u_{high}, with $u_{\text{high}} < u_{\text{low}}$. We call this variant Single Precision Preconditioner (SPP) in the rest of the document. Crucially, this mixed precision variant does not require to change the rest of the algorithm and does not incur any extra cost. This is notably in contrast to GMRES, where the flexible variant FGMRES requires to double the size of the Krylov basis.

We illustrate the flexibility of BiCGStab and the reliability of Algorithm 3 with an experiment on matrix sherman4 in Figure 1. The methodology for generating the figure is detailed in section 4.1. We plot the explicit relative residual $\|Ax - b\|/\|b\|$ computed in fp64 at each iteration. The solver is stopped whenever the norm of the computed residual falls below $\epsilon = 10^{-13}$.

Figure 1 compares three different variants: the first two are uniform precision implementations of BiCGStab (Algorithm 1), run entirely in either double precision (fp64) or single precision (fp32). The explicit residual of the uniform fp32 variant stagnates around 10^{-4} after 30 iterations. This observed behavior highlights the limitations of the variant's attainable accuracy, which is constrained by a combination of factors. Among these, the matrix's properties play a significant role, with 10^{-7} representing the lower bound imposed by the inherent constraints of single precision floating-point numbers. The third variant is a mixed precision implementation corresponding to Algorithm 3 with fp64 as u_{high} and fp32 as u_{low}: that is, we compute and apply the preconditioner in single precision,

Algorithm 3: Right-preconditioned BiCGStab in mixed precision (SPP $u_{\text{low}}/u_{\text{high}}$).

Input: A, b, $i_{\max}$, ϵ
Output: x_i

1 Initialize $\bar{r_0}$ arbitrarily.
2 Set $x_0 = 0$ and $p_0 = r_0 = b$
3 Construct preconditioner M (Prec u_{low})
4 **for** ($i = 1$ **to** $i_{\max}$)
5 *Solve:* $M\hat{p} = p_{i-1}$ (Prec u_{low})
6 $\alpha = (\bar{r_0}, r_{i-1})/(\bar{r_0}, A\hat{p})$ (Prec u_{high})
7 $s = r_{i-1} - \alpha A\hat{p}$
8 *Solve:* $M\hat{s} = s$ (Prec u_{low})
9 $\omega = (A\hat{s}, s)/(A\hat{s}, A\hat{s})$
10 $x_i = x_{i-1} + \alpha\hat{p} + \omega\hat{s}$
11 $r_i = s - \omega A\hat{s}$ (Prec u_{high})
12 **if** ($\|r_i\| < \epsilon$) **then** exit loop
13 $\beta = \alpha(\bar{r_0}, r_i)/\omega(\bar{r_0}, r_{i-1})$
14 $p_i = r_i + \beta(p_{i-1} - \omega A\hat{p})$

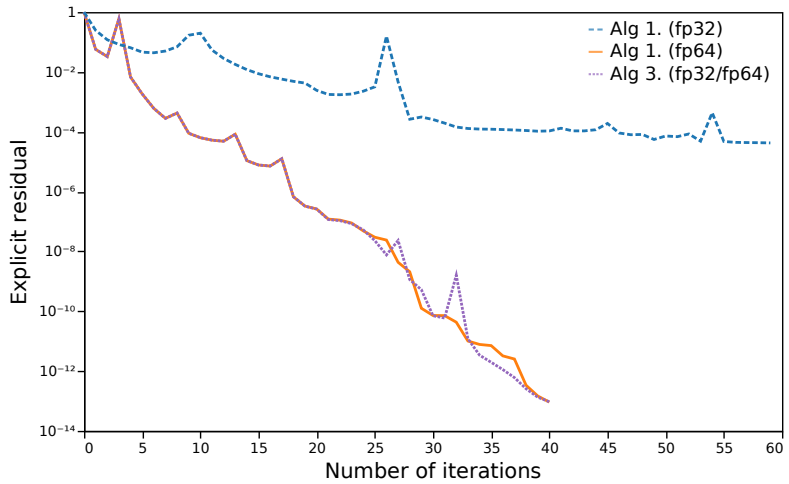


Fig. 1. Illustration of the first strategy (low precision preconditioning) on matrix sherman4, with a block Jacobi preconditioner.

and perform the rest of the operations in double precision. As expected from a flexible solver, we confirm that this variant successfully converges to the same accuracy as the uniform fp64 one.

We conclude that the first mixed precision strategy is a reliable method to accelerate the preconditioning operations. However, with this strategy, the rest of the operations, particularly the costly sparse matrix-vector products, remain in fp64 arithmetic. The next two strategies explore possible techniques to compensate the loss of accuracy incurred when also switching these operations to lower precision arithmetic.

3.2 Mixed precision BiCGStab-based iterative refinement (BiCGStab-IR)

The iterative refinement approach described in section 2.3 (see Algorithm 2) can use any arbitrary linear solver for the inner system $Ad = r$. In this section, we investigate a mixed precision BiCGStab-based iterative refinement (BiCGStab-IR) strategy that uses a lower precision BiCGStab as solver.

The resulting algorithm is described in Algorithm 4. The inner loop (lines 9–18) corresponds to a standard right-preconditioned BiCGStab in precision u_{low}, whereas the refinement outer loop is performed in precision u_{high}. As previously explained, as long as the matrix is not too ill conditioned, this algorithm should achieve an accuracy of level u_{high} despite using precision u_{low} for most of its operations.

We use two separate stopping criterions for the inner and outer loops. For the inner loop (BiCGStab), we stop the solver either after the norm of the inner residual $\|r_i\|$ becomes smaller than a prescribed threshold ϵ_{in} or after the maximum number of inner iterations $i_{\max}$ has been performed. For the outer loop (refinement), we use a similar criterion but with a threshold $\epsilon < \epsilon_{\text{in}}$ on the norm of the outer residual $\|R\|$ and a maximum number of outer iterations $j_{\max}$. We will discuss how to choose these parameters in section 3.4.

Algorithm 4: BiCGStab-IR in mixed precision.

Input: A, b, $i_{\max}$, $j_{\max}$, ϵ_{in}, ϵ
Output: y_j

1 Initialize y_0 and $\bar{r}_0$ arbitrarily.
2 Construct preconditioner M (Prec u_{low})
3 **for** ($j = 1$ **to** $j_{\max}$)
4 $R \leftarrow b - Ay_{j-1}$
5 **if** ($\|R\| \leq \epsilon$) **then** exit loop. (Prec u_{high})
6 $p_0 = r_0 = R$
7 $x_0 = 0$
8 **for** ($i = 1$ **to** $i_{\max}$)
9 *Solve:* $M\hat{p} = p_{i-1}$
10 $\alpha = (\bar{r}_0, r_{i-1})/(\bar{r}_0, A\hat{p})$
11 $s = r_{i-1} - \alpha A\hat{p}$
12 *Solve:* $M\hat{s} = s$
13 $\omega = (A\hat{s}, s)/(A\hat{s}, A\hat{s})$ (Prec u_{low})
14 $x_i = x_{i-1} + \alpha\hat{p} + \omega\hat{s}$
15 $r_i = s - \omega A\hat{s}$
16 **if** ($\|r_i\| \leq \epsilon_{\text{in}}$) **then** exit loop.
17 $\beta = \alpha(\bar{r}_0, r_i)/\omega(\bar{r}_0, r_{i-1})$
18 $p_i = r_i + \beta(p_{i-1} - \omega A\hat{p})$
19 $y_j \leftarrow y_{j-1} + x_i$ (Prec u_{high})

We illustrate the convergence behavior of BiCGStab-IR (Algorithm 4) in Figure 2 for matrix sherman4. We plot the relative explicit residual obtained at each iteration for two variants of BiCGStab in uniform precision (either fp64 or fp32). We also plot the relative residual obtained after each restart (at each outer iteration) for BiCGStab-IR. We have used $\epsilon = 10^{-13}$ for the BiCGStab variants and for the outer stopping criterion of BiCGStab-IR, and $\epsilon_{\text{in}} = 10^{-5}$ for the inner stopping criterion of BiCGStab-IR.

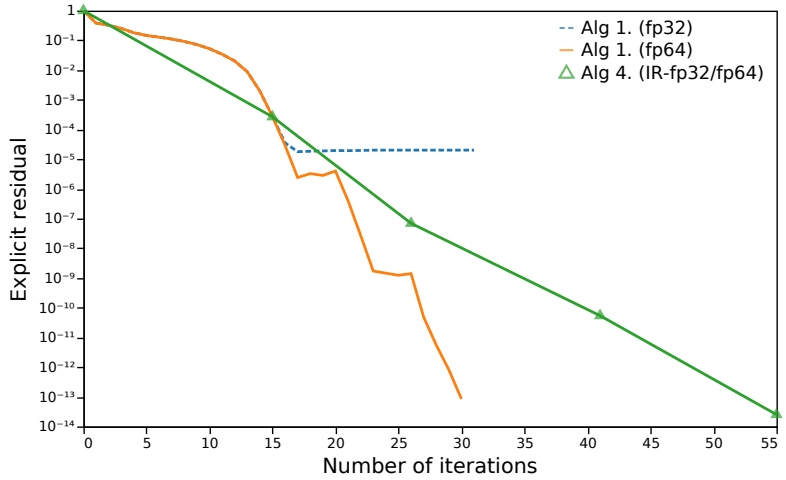


Fig. 2. Illustration of the second strategy (BiCGStab-IR) for matrix sherman4, with preconditioner ILU0.

Figure 2 confirms that mixed precision BiCGStab-IR successfully converges to the same accuracy as BiCGStab in double precision, despite performing most of its operations in single precision. This not only includes the preconditioning operations but also the sparse matrix–vector products and the other vector operations. Therefore, we can expect the per-iteration cost of BiCGStab-IR to be significantly reduced. However, as illustrated on this matrix, the convergence of BiCGStab-IR can be slower than that of BiCGStab in double precision. Here, BiCGStab-IR requires nearly twice as many iterations as fp64 BiCGStab to reach 10^{-13} accuracy; note that the increase in the number of iterations depends on the target accuracy: if only 10^{-10} is requested, the number of iterations increasesy only by about 50%.

The slower convergence of BiCGStab-IR is explained by the nonlinear convergence behavior of BiCGStab. Figure 2 also illustrates the possible "plateau" effects of BiCGStab: the convergence is relatively slow for the first 40 iterations and then speeds up. With BiCGStab-IR, this plateau must be overcome each time the inner solver is restarted, that is, at each outer iteration. It is worth noting that, for the same reason, in the case where BiCGStab exhibits a faster convergence for the early iterations and then slows down, BiCGStab-IR can actually converge faster than BiCGStab; we illustrate this remark in section 4.2 (see Figure 5). Unfortunately, these cases are considerably less common and in general we have usually observed BiCGStab-IR to slow down the convergence.

In the next section, we investigate how to potentially enhance the convergence speed by using a more tailored approach that takes into account the specific properties of BiCGStab.

3.3 Mixed precision flying restart (BiCGStab-FR)

In this section we propose a mixed precision version of the flying restart variant of BiCGStab (BiCGStab-FR), originally proposed in 1996 by Sleijpen and van der Vorst [1996] in a uniform precision setting.

This BiCGStab-FR method is outlined in Algorithm 5. It shares some similarities with BiCGStab-IR: it also takes the form of an inner–outer scheme, where the inner loop is essentially a BiCGStab solver in low precision u_{low} and the outer loop in precision u_{high} is a restart mechanism that periodically resets the inner solution x_i to zero and the inner right-hand side to the residual r_i, and accumulates the inner solutions in the outer solution y. Due to these similarities, BiCGStab-FR

should also be able to converge to an accuracy of level u_{high}, provided the matrix is not too ill conditioned. However, there is a key difference with BiCGStab-IR: unlike BiCGStab-IR, which resets everything in the inner BiCGStab solver at each new outer iteration, BiCGStab-FR only resets the solution and the right-hand side, but not the internal state (specifically, the vector p_0) of the BiCGStab solver. For this reason, we may expect BiCGStab-FR to converge faster than BiCGStab-IR.

Algorithm 5: BiCGStab-FR in mixed precision (FR-u_{low}/u_{high}).

Input: A, b, $i_{\max}$, $j_{\max}$, ϵ_{in}, ϵ
Output: y

Initialize $\bar{r_0}$ arbitrarily
Set $y = x_0 = 0$, $j = 1$ and $p_0 = r_0 = b$
Construct preconditioner M (Prec u_{low})
for ($i = 1$ **to** $i_{\max}$)
  Solve: $M\hat{p} = p_{i-1}$
  $\alpha = (\bar{r_0}, r_{i-1})/(\bar{r_0}, A\hat{p})$
  $s = r_{i-1} - \alpha A\hat{p}$
  Solve: $M\hat{s} = s$ (Prec u_{low})
  $\omega = (A\hat{s}, s)/(A\hat{s}, A\hat{s})$
  $x_i = x_{i-1} + \alpha\hat{p} + \omega\hat{s}$
  $r_i = s - \omega A\hat{s}$
  if ($\|r_i\| \leq \epsilon_{\text{in}}$ *or* $j \geq j_{\max}$) **then**
    $r_i = b - Ax_i$
    $y = y + x_i$ (Prec u_{high})
    $x_i = 0, \quad b = r_i, \quad j = 1$
  if ($\|r_i\| \leq \epsilon$) **then** exit loop.
  $\beta = \alpha(\bar{r_0}, r_i)/\omega(\bar{r_0}, r_{i-1})$ (Prec u_{low})
  $p_i = r_i + \beta(p_{i-1} - \omega A\hat{p})$
  $j = j + 1$
return $y + x_i$

We illustrate experimentally the different behaviors of BiCGStab-IR and BiCGStab-FR in Figure 3 for matrix sherman4. The figure confirms that mixed precision BiCGStab-FR (Algorithm 5) successfully converges to the same accuracy as fp64 BiCGStab, and converges faster than mixed precision BiCGStab-IR. Note that this matrix represents a favorable case for BiCGStab-FR; in our experiments of section 4, we will illustrate that there are other matrices for which BiCGStab-FR may not necessarily converge faster, or even converges slower than BiCGStab-IR. However, the general trend that we have observed across a wide range of matrices is that BiCGStab-FR is usually more robust than BiCGStab-IR. Moreover, it presents a significant other advantage: it is much simpler to choose the restart criterion, as we explain in the next section.

3.4 Choice and impact of the restart criterion

The BiCGStab-IR and BiCGStab-FR methods presented in the previous sections take the form of inner–outer schemes based on different restart mechanisms. Selecting an appropriate restart criterion is crucial, as it significantly affects the robustness and performance of the solvers. Indeed, we need to restart frequently enough in order to converge to high accuracy and avoid stagnating at low accuracy, but we also wish to perform as much computation as possible in low precision in order to achieve performance gains.

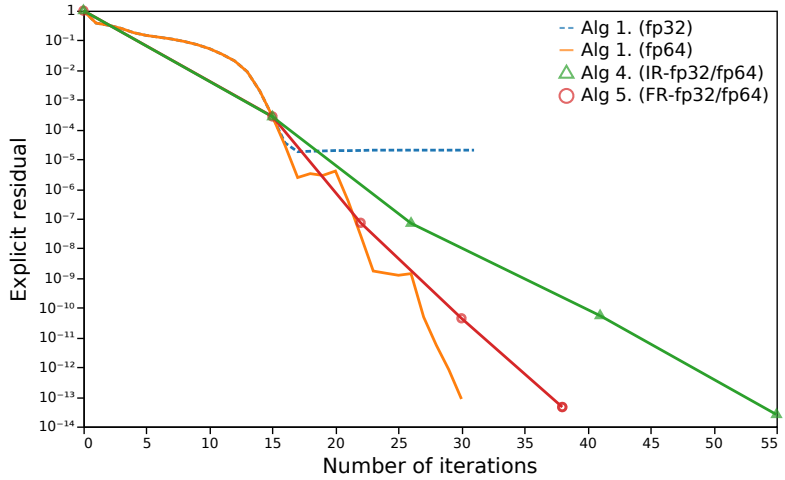
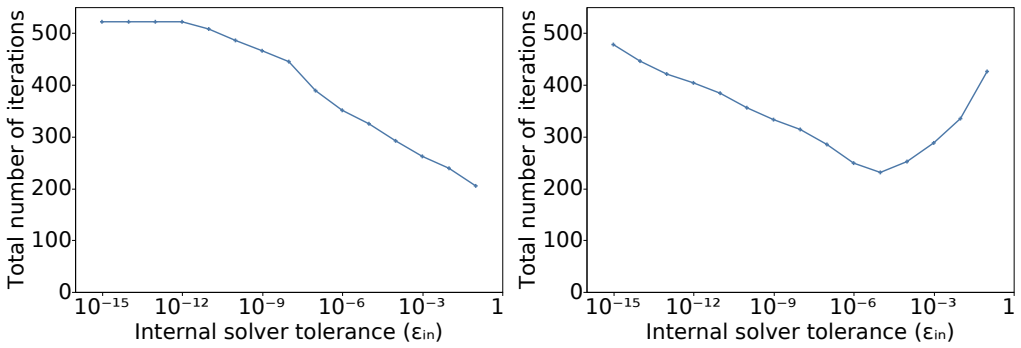


Fig. 3. Illustration of the third strategy (BiCGStab-FR) for matrix sherman4, with preconditioner ILU0.

As described in Algorithms 4 and 5, the restart criterion can be defined from various conditions. In particular, we introduced two parameters that control the restart frequency, ϵ_{in} and $i_{\max}$: we restart whenever the norm of the residual becomes smaller than ϵ_{in}, or when $i_{\max}$ inner iterations have been performed.

The restart criterion therefore introduces parameters that should be tuned for optimal performance. Importantly, we next explain why the convergence of BiCGStab-FR is much less sensitive to these parameters than that of BiCGStab-IR, and is therefore easier to optimize and more robust overall. We illustrate this key advantage of BiCGStab-FR in Figure 4, which plots the total number of iterations depending on the choice of ϵ_{in} (for this experiment, we do not limit the maximum number of inner iterations, that is, we set $i_{\max} = \infty$).



(a) BiCGStab-IR (b) BiCGStab-FR

Fig. 4. Number of iterations before convergence w.r.t. restart parameter
Matrix: SparseSuite HB/sherman4 Preconditioner: None

In the case of BiCGStab-IR, as illustrated in Figure 4a, there exists an optimal value of ϵ_{in} to be found. Indeed, if ϵ_{in} is too small, the inner solver stagnates while trying to achieve ϵ_{in} accuracy,

leading to a large number of inner iterations; but if ϵ_{in} is too large, the inner loop does not make sufficient progress at each outer iteration, leading to a large number of outer iterations. Unfortunately, this optimal value of ϵ_{in} tends to be strongly matrix dependent, and so it is difficult to optimize the convergence speed of BiCGStab-IR in practice.

In contrast, the behavior of BiCGStab-FR with respect to ϵ_{in} is quite different, as shown in Figure 4b. In this case, the total number of iterations steadily decreases as ϵ_{in} increases. This is because a larger ϵ_{in} minimizes the chance of the inner solver stagnating, while not preventing the outer loop to convergence since no information is lost between restarts. Therefore, the convergence speed can be improved simply by increasing ϵ_{in} and thus the number of restarts. Since each restart however requires recomputing the explicit residual R in precision u_{high}, there is therefore a clear tradeoff between the total number of iterations and the cost of these iterations. Thus, BiCGStab-FR is much more robust to the choice of ϵ_{in}, which merely consists in balancing convergence speed and computational cost.

4 Numerical experiments

In this section, we present extensive experiments on a wide range of real-life matrices, and we analyze the performance of the three mixed precision strategies described in the previous section.

We first describe our experimental setup in section 4.1. Then, in section 4.2, we illustrate different possible behaviors in terms of convergence. In section 4.3, we carry out a detailed performance analysis of the underlying kernels. Finally, in section 4.4, we put everything together and evaluate the overall performance of the solver.

4.1 Experimental setup

We use matrices derived from both sparse matrix benchmarks in the SuiteSparse collection [Davis and Hu 2011], and real-life applications. Table 1 summarizes the characteristics of the sparse matrices used. Regarding reservoir simulation and CO2 storage application, the matrices and associated right-hand sides were extracted from black-oil multi-phase porous media flow simulator. At each Newton step, the Jacobian matrix and the right-hand side define the linear system to be solved. This nonlinear system arises from the discretization of the Darcy equations. We consider different classes of sparse matrices in our study, including block sparse matrices with block sizes of 3×3 and 2×2. For the SuiteSparse matrices, the right-hand side is generated such that the solution to the system is the unit vector.

To improve numerical stability and accuracy, and to mitigate the range limitations inherent in finite precision arithmetic, we employ diagonal scaling on all matrices before running the solver. For the SuiteSparse (scalar) matrices, all the rows are divided by the value in the diagonal. For the reservoir simulation (block) matrices, after deriving P such that $PB = I$, with B a diagonal block of the matrix and I the identity matrix, all the blocks of the matrix on the same block row as B are multiplied by P. This scaling reduces the risk of values falling outside the representative range of the fp32 format, and thus the risk of overflow and underflow. Once the computations are complete, the solution can be rescaled back to its original range if necessary.

We will compare uniform precision BiCGStab (Algorithm 1) with the three mixed precision strategies described in the previous section, namely, BiCGStab with lower precision preconditioning (Algorithm 3), BiCGStab-IR (Algorithm 4), and BiCGStab-FR (Algorithm 5). We implemented all these algorithms in C++ within the MCGSolver library [Anciaux-Sedrakian et al. 2022]. For all numerical experiments, the tolerance ϵ for the relative residual error was set to 10^{-11}. The inner tolerance ϵ_{in} in BiCGStab-IR and BiCGStab-FR was reasonably tuned for each matrix: we evaluated seven values of ϵ_{in} between 10^{-1} and 10^{-7} and chose the one giving the best solve time. We do not restrict the maximum number of inner or outer iterations, that is, we set $i_{\max} = j_{\max} = \infty$.

Table 1. Matrices

Origin	Matrix	n	# nonzeros	Block size
CO2 storage	Geoxim100	1 200 006	97 345 446	1×1
CO2 storage	Geoxim200	1 200 006	97 345 446	1×1
Reservoir Simulation	IvaskBO	148 716	4 256 343	3×3
Reservoir Simulation	GCS	185 498	1 094 385	3×3
Reservoir Simulation	Spe10[Christie and Blunt 2001]	2 188 842	6 421 171	2×2
SuiteSparse	cage15	5 154 859	99 199 551	1×1
SuiteSparse	atmosmodl	1 489 752	10 319 760	1×1
SuiteSparse	sherman4	1104	3786	1×1
SuiteSparse	CoupCons3D	416 800	17 277 420	1×1
SuiteSparse	Cube_Coup_dt6	2 164 760	124 406 070	1×1
SuiteSparse	Emilia_923	923 136	40 373 538	1×1
SuiteSparse	Geo_1438	1 437 960	60 236 322	1×1
SuiteSparse	ML_Laplace	377 002	27 582 698	1×1
SuiteSparse	ss	1 652 680	34 753 577	1×1
SuiteSparse	ss1	205 282	845 089	1×1
SuiteSparse	wang3	26 064	177 168	1×1
SuiteSparse	Zhao1	33 861	166 453	1×1

Computations were performed in common software environments: GCC/11.2.0, hwloc/2.5.0 and imkl/2021.4.0. Flush-to-zero was used to set denormalized floating-point numbers to zero, via the compilation flags `-ffast-math` for GCC. The variant "NoVec" used in section 4.3 was generated by deactivating all the vectorization options of the compiler, the exact list of options is:

```
-O2 -fgcse-after-reload -fipa-cp-clone -floop-interchange -floop-unroll-and-jam
-fpeel-loops -fpredictive-commoning -fsplit-loops -fsplit-paths -funswitch-loops
-ftree-loop-distribution -ftree-partial-pre -fversion-loops-for-strides
```

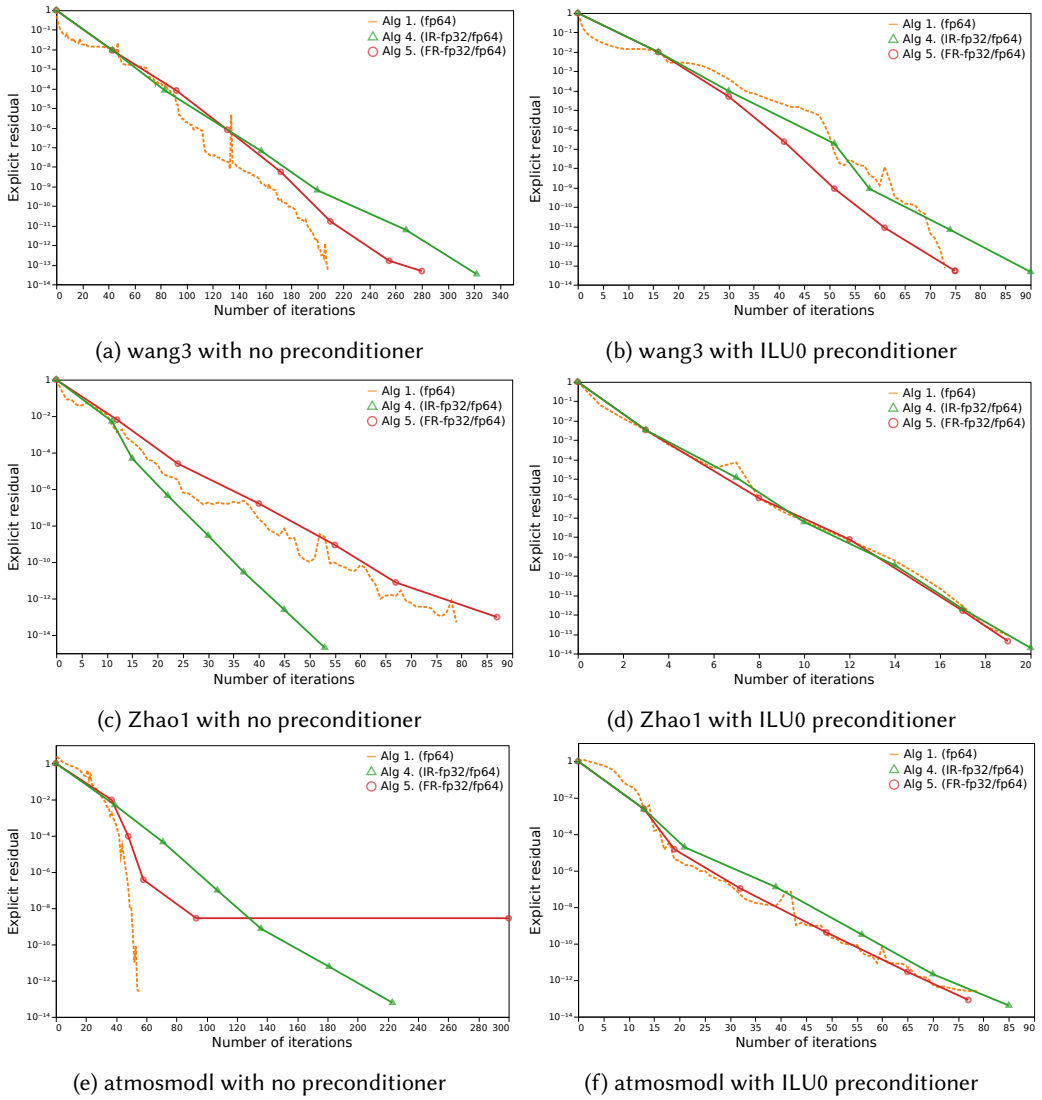
We ran all the experiments on the IFPEN Orion cluster equiped with 108 bi-socket compute nodes, AMD Genoa 9534 CPUs with 64 cores running at 2.45 GHz frequency.

4.2 Convergence rate: effect of the variant and preconditioning

We have explained that the nonlinear convergence behavior of BiCGStab may lead to significant differences in convergence rate between BiCGStab, BiCGStab-IR, and BiCGStab-FR. Previously, we have illustrated this for matrix sherman4, for which BiCGStab was faster than BiCGStab-FR, which in turn was faster than BiCGStab-IR. Here, we show that there is no systematic trend and that the relative convergence rate of each variant compared with the other two strongly depends on the matrix. Since preconditioning effectively changes the matrix being solved, the preconditioner used (if any) can also affect this convergence comparison.

Figure 5 provides a comparative illustration of convergence behavior, highlighting the previously mentioned phenomena through three additional matrices, wang3, Zhao1, and atmosmodl, examined with and without a preconditioner. For wang3 with no preconditioner (Figure 5a), we observe the same relative order as for sherman4: BiCGStab, BiCGStab-FR, and BiCGStab-IR, from fastest to slowest; however, if an ILU0 preconditioner is used (Figure 5b), BiCGStab-FR becomes faster than BiCGStab. For Zhao1, BiCGStab-IR converges much faster than the other two variants (Figure 5c); however, ILU0 preconditioning makes all variants converge at approximately the same rate (Figure 5d). For atmosmodl with no preconditioner (Figure 5e), it can be observed that BiCGStab-FR

initially converges faster than BiCGStab-IR, then the convergence stagnates at 10^{-7}, after which the trend reverses. As shown in Figure 5e, BiCGStab-IR exhibits steady but slower convergence, requiring nearly four times as many iterations to reach the same level of accuracy as BiCGStab. Similar to the case of wang3, Figure 5f illustrates that ILU0 preconditioning enables all variants to converge at almost the same rate, with BiCGStab-FR being the fastest.



(a) wang3 with no preconditioner

(b) wang3 with ILU0 preconditioner

(c) Zhao1 with no preconditioner

(d) Zhao1 with ILU0 preconditioner

(e) atmosmodl with no preconditioner

(f) atmosmodl with ILU0 preconditioner

Fig. 5. ExpRes = $\frac{\|Ax-b\|}{\|b\|}$ Kernel: AVX512-BCSP Cluster: IFPEN Orion

We conclude that there is no clear winner between the two proposed methods: sometimes BiCGStab-IR performs better, and sometimes BiCGStab-FR does. It is therefore worth considering both methods when targeting a mixed precision solver.

4.3 Performance analysis of the kernels

Before evaluating the performance of the full mixed precision solver, we analyze the performance of the main computational kernels: the sparse matrix–vector product (SpMV) and the application of the preconditioner via sparse triangular solve (SpTRSV). Indeed, we must first assess how to effectively accelerate these operations by switching them to low precision arithmetic, namely, from fp64 to fp32. As we will show, this is far from trivial, because the performance gains strongly depend on multiple factors.

First, the performance of CPU computations is significantly enhanced by the use of SIMD instructions, which operate on vectors of data. The size of hardware SIMD registers and data type dictates the vector size. Currently, processors are equipped with registers that can be of length 128, 256 (AVX, AVX2), or 512 (AVX512, SVE) bits. This allows for the simultaneous processing of either 2, 4, or 8 fp64 numbers, and 4, 8, or 16 fp32 numbers, respectively.

Second, sparse matrix layouts have an important impact on performance. There are many different storage layouts in the literature [Chen et al. 2019; Lee et al. 2004; Williams et al. 2007], which aim to reduce their memory footprint and indirection or to improve the performance on modern computer architectures. According to the profile of the matrix and the number of elements per row, different layouts, such as Block CSR (BCSR) [Liu and Vinter 2015], Block Ellpack (BEll) or Block Compressed Sparse Packets (BCSP) [Kreutzer et al. [n. d.]], are better suited to a given hardware architecture (CPUs with different SIMD instructions, GPUs). The challenge is to maintain memory alignment, in order to benefit from the vectorization gains. With the BCSR layout, it is difficult to rearrange data in order to expoit the full potential of SIMD computation units. SIMD is better used with BCSR if block size is, at least or a multiple of, the SIMD vector size The BCSP layout, which is an adaptation of Sell-C-σ format [Kreutzer et al. [n. d.]], allows for more efficient SpMV implementations using different SIMD instructions. Moreover, it is important to note that sparse matrix layouts also need to store some indices of the matrix coefficient as integers in order to reconstruct the original matrix. These indices occupy a memory space that is independent of the precision used for the floating-point numbers, and thus have an impact on the maximum theoretical improvement in performance obtainable by reducing the precision. When the BCSR or BCSP format is used for scalar matrices, it means the blocksize is equal to 1.

Let us now analyze how all these parameters can interact and affect the performance when switching the precision from fp64 to fp32. Figure 6 shows the time cost of the SpMV kernel in fp64 and fp32, depending on the sparse matrix layout (BCSR or BCSP) and the SIMD instructions used (no vectorization, AVX2 or AVX512). For each matrix, the time costs are normalized with respect to the fp64 operation with BCSR layout and no vectorization.

We observe that, while the fp64/fp32 reduction in time is in all cases significant, it strongly varies depending on the parameter combination. In particular, in the baseline reference case (NoVec-BCSR), the execution is reduced by only about 30%. Using AVX2 (AVX2-BCSR) has almost no effect on the cage15 matrix, whereas it significantly improves the fp32 performance on matrices GCS and IvaskBO. This can be explained by the block structure of these two matrices, which thus present some potential for vectorization even with the BCSR layout. Switching to the BCSP layout (AVX2-BCSP) allows for more efficiently exploiting vectorization and thus further improves performance, especially for cage15. Overall, the fp64/fp32 improvement in performance for this AVX2-BCSP variant are much larger than for the baseline variant NoVec-BCSR, up to 40%. This can be explained by the fact that SIMD instructions can accommodate twice as many fp32 numbers than fp64 ones, and thus fp32 operations benefit from vectorization more than fp64 ones. Finally, switching from AVX2 to AVX512 improves the fp64 performance, but not the fp32 one. While this

AVX512-BCSP variant does not lead to the largest fp64/fp32 improvement, it gives the highest absolute performance and will therefore be used as the reference kernel for the next tests.

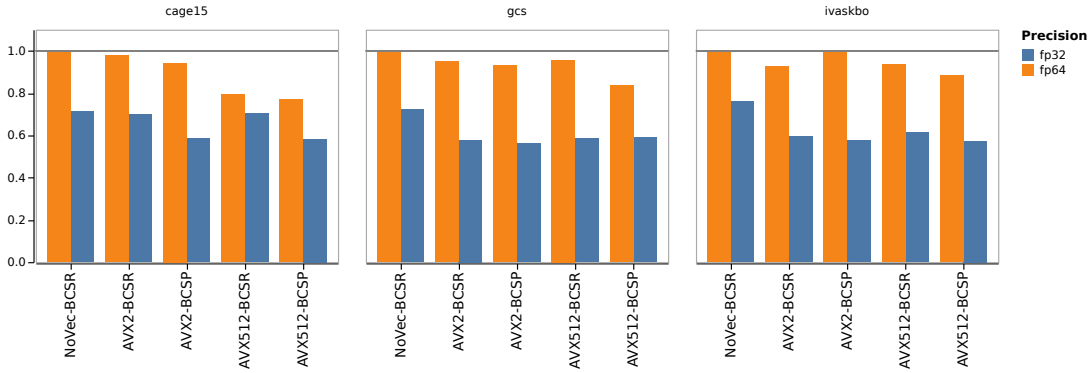


Fig. 6. Normalized time of one SpMV operation in fp32 and fp64 for different SIMD instructions and sparse matrix layouts (lower is better).

We perform a similar experiment for the SpTRSV kernel in Figure 7, which shows the normalized time cost of applying the ILU preconditioner. The AVX2 and AVX512 kernels compute the lower and upper trisolve with a manual function using SIMD intrinsics, whereas the AVX512-mkl kernel uses the special function `mkl_sparse_trsv` from the `mkl_spblas` library.

We observe again that the use of vectorization leads to a small time reduction of the fp64 operation except for the AVX512-mkl kernel, and that the reduction is much more significant for the fp32 one. The AVX512-mkl kernel gives comparable acceleration in fp32 for the block matrices (GCS and IvaskBO) but fails to give the same acceleration as other kernels with the point matrix (cage15).

The use of vectorization is thus critical to obtain the best fp64/fp32 reduction in timing, which ranges between 0.7 and 0.5 times the non-vectorized fp64 version depending on the matrix. The variability is linked to the block size of the matrices, with larger block sizes leading to greater improvements.

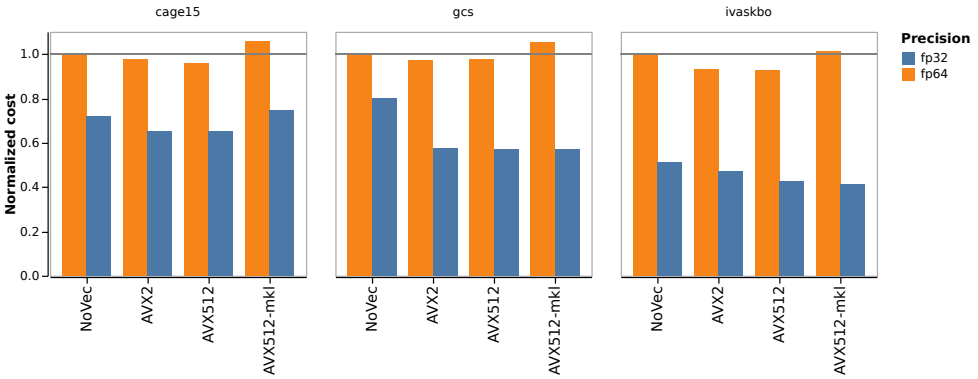


Fig. 7. Normalized time of one preconditioner application (ILU0) in fp32 and fp64 for different SIMD instructions and sparse matrix layouts (lower is better)

We conclude from this performance analysis that the SpMV and SpTRSV kernels can be significantly accelerated by switching from fp64 to fp32, provided suitable sparse matrix layout and SIMD instructions are used. Ensuring large kernel speedups is critical, since they will determine the actual speedups that are achievable for the full mixed precision solver. We will proceed with the AVX512-BCSP kernel for the subsequent experiments, as it provides the most reliable improvement in performance across the tested operations and matrices.

4.4 Performance evaluation of the mixed precision solver

We now evaluate the performance of the full solver for the different mixed precision strategies previously described. Figure 8 presents a detailed performance analysis for three representative matrices already considered. We complement this detailed analysis with Table 2, which reports more concise performance results on a wider range of real-life matrices.

Figure 8 compares six solver variants:

- Uniform double precision ("fp64") BiCGStab, which serves as the baseline reference time (normalized to be equal to 1 for each matrix);
- Single precision preconditioning ("precond-fp32"), corresponding to Algorithm 3 with fp32 as u_{low} and fp64 as u_{high}; this variant allows for analyzing separately the performance improvements due to the use of single precision in the preconditioner and the rest of the solver;
- Mixed precision BiCGStab-IR ("IR-fp32/fp64") and BiCGStab-FR ("FR-fp32/fp64"), corresponding to Algorithms 4 and 5, respectively, with fp32 as u_{low} and fp64 as u_{high};
- Uniform double precision BiCGStab-IR ("IR-fp64") and BiCGStab-FR ("FR-fp64"), corresponding to Algorithms 4 and 5, respectively, with fp64 as both u_{low} and u_{high}; these last two variants allow for analyzing the effect of restarting on the convergence independently of the use of mixed precision.

For each solver variant, the figure plots the time breakdown across different computational phases:

- "Precond": the application of the preconditioner in the inner loop;
- "SpMV": the sparse matrix–vector products in the inner loop;
- "Other inner": the rest of the inner loop operations (dot products, vector additions, etc.);
- "Outer": the outer loop operations, which consist in computing the residual via an SpMV.

We do not include in this breakdown the time for constructing the preconditioner; for most matrices, it accounts for less than 5% of the total solver time. For a few outliers, it accounts for a substantially higher fraction because the solver converges in a very small number of iterations, leading to a short solve phase and thus a higher relative cost for the preconditioner construction.

In addition to the time breakdown, Figure 8 also plots the total number of iterations taken by each variant to converge to the required accuracy.

Figure 8 illustrates that the precond-fp32, IR-fp32/fp64, and FR-fp32/fp64 variants can all significantly reduce the total solution time compared with the fp64 solver baseline. The precond-fp32 variant employs fp32 only in the preconditioning phase, and so "Precond" is the only phase that is accelerated. The figure shows that this is achieved while maintaining the same number of iterations for all matrices and without any extra cost, so precond-fp32 is consistently faster than the fp64 baseline. In contrast, the IR-fp32/fp64 and FR-fp32/fp64 variants also employ fp32 for the rest of the inner loop operations, and thus the "SpMV" and "Other inner" phases are also accelerated, potentially leading to greater time savings. However, IR and FR require additional outer loop operations in fp64, and may also affect the total number of iterations, so a matrix-dependent tradeoff arises. We can also see on Figure 8 by looking at IR-fp64 and FR-fp64 that the number of iterations when

choosing $u_{low} = u_{high}$ is very close to the one of the mixed precision variant indicating that the convergence behavior is mainly dictated by the restarting strategy.

In the cases of cage15 and IvaskBO matrices, all variants achieved the required accuracy with the same number of iterations, resulting in an equal number of preconditioner applications. Despite this, the precond-fp32 variant requires slightly more time for the "Precond" phase due to the need to convert the input vector (to which the preconditioner is applied) from fp64 to fp32, and the resulting output vector from fp32 back to fp64. This conversion occurs at each iteration and thus leads to a higher cost per iteration compared with the IR-fp32/fp64 and FR-fp32/fp64 variants, for which the inner loop operations are all performed in fp32 and thus require no conversions (conversions are only needed at each outer iteration).

The IR-fp32/f64 variant for the GCS matrix also spends more time in the preconditioning phase, however, this can be attributed to the increased number of iterations, as the cost per application remains constant.

For the cage15 matrix, the time spent in the 'Outer SpMV + Residual' phase is significantly greater for the IR-fp32/fp64 variant than for the FR-fp32/f64 variant. This can be explained by the number of restarts needed to computed the solution, which is respectively 3 for IR and 2 for FR.

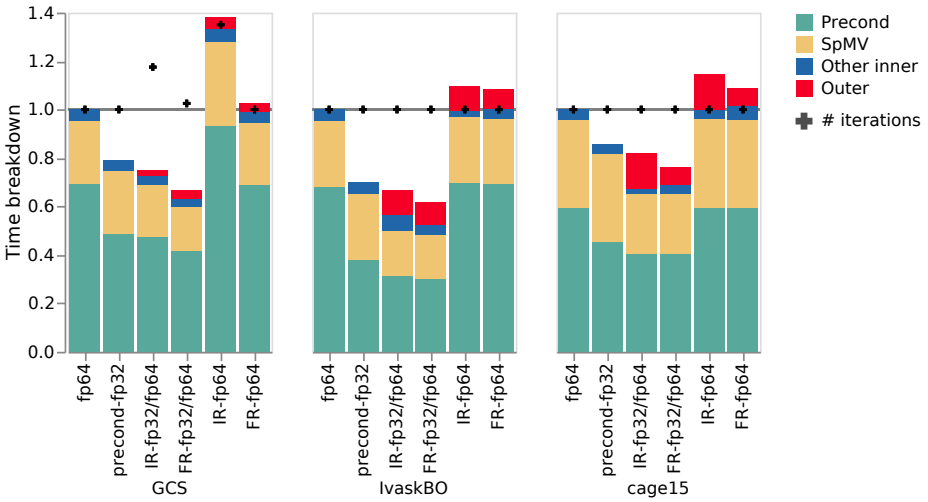


Fig. 8. Performance of the different (uniform and mixed precision) solver variants, with time breakdown across computational phases (lower is better).

Table 2 presents performance results for a wider range of matrices. For each matrix, it reports the solution time normalized with respect the fp64 baseline, alongside the number of iterations (and restarts where applicable). The table highlights the effectiveness of both IR-fp32/f64 and FR-fp32/f64 strategies across the set of matrices, which achieve performance improvements of up to 45% (0.55 normalized time) compared with the baseline. Both strategies, IR-fp32/fp64 and FR-fp32/fp64 achieve the best computational time (indicated in bold) on a comparable number of matrices The FR-fp32/fp64 method performed better for problems where the solution is attainable in under 100 iterations with exceptions of Geoxim100 and CoupCons3D where both time were really close between IR-fp32/f64 and FR-fp32/fp64 (respectively 0.69 / 0.74 and 0.75 / 0.79). The precond-fp32 method also allows for some performance improvements over the fp64 baseline, although

the improvement is expectedly more limited. It achieved the best performance for Geo_1438 and Emilia_923 matrices because the other variants performed very poorly, needing between 1.5 and 2.5 times more iterations to converge. It also achieved the best ratio compared to fp64 for the matrice cage15 where all variants converged in 5 iterations, leaving too few steps to amortize the extra cost of double-precision restarts. Overall, the table confirms the significant performance gains that can be achieved via mixed precision.

Table 2. Performance of the different solver variants across a wider range of matrices; "time" indicates the solution time normalized with respect to fp64 baseline (with the best variant in bold); "#its" indicates the total number of iterations and, if applicable, the number of restarts between parentheses.

Matrix	fp64	precond-fp32		IR-fp32/fp64		FR-fp32/fp64	
	#its	time	#its	time	#its	time	#its
Geoxim100	29	0.80	29	**0.69**	28 (6)	0.74	30 (6)
Geoxim200	239	0.95	284	**0.81**	285 (7)	1.13	395 (14)
IvaskBO	11	0.66	11	0.63	11 (6)	**0.56**	11 (5)
GCS	40	0.76	40	0.73	47 (6)	**0.65**	41 (10)
SPE10	207	1.08	296	**0.75**	262 (7)	0.77	266 (6)
cage15	5	0.86	5	0.84	5 (3)	**0.77**	5 (2)
atmosmodl	79	0.98	78	0.85	85 (6)	**0.77**	77 (5)
CoupCons3D	27	0.85	27	**0.75**	25 (6)	0.79	28 (5)
Cube_Coup_dt6	263	0.83	254	**0.82**	294 (7)	0.84	301 (6)
Emilia_923	364	**1.04**	441	1.19	580 (14)	1.85	903 (22)
Geo_1438	330	**0.96**	380	1.16	519 (14)	1.16	515 (30)
ML_Laplace	435	1.26	689	**0.89**	570 (12)	1.10	699 (25)
ss	137	0.99	151	**0.70**	132 (7)	0.81	155 (6)
ss1	4	0.95	4	0.92	3 (3)	**0.83**	3 (3)
wang3	73	1.15	73	1.13	90 (6)	**0.94**	75 (6)
Zhao1	19	0.98	19	1.04	20 (6)	**0.93**	19 (4)

5 Conclusion

In this article, we have proposed and evaluated three mixed precision strategies for accelerating BiCGStab solvers for sparse linear systems, showcasing the potential of mixed precision computations to boost efficiency while maintaining numerical accuracy. Our results, validated on real-life and benchmark matrices, show possible decrease in computation time of up to 45%, thanks to careful tuning and leveraging of SIMD optimizations.

This work opens the way towards further optimizations of BiCGStab solvers in mixed precision. In particular, while we have here focused on fp32 as the low precision, even lower precisions like fp16 or bfloat16 could be employed to further enhance solver performance, especially on specialized architectures such as GPU accelerators.

Acknowledgments

This work was partially supported by projects of the French National Agency for Research (ANR): InterFLOP (ANR-20-CE46-0009), NumPEx Exa-MA (ANR-22-EXNU-0002), and MixHPC (ANR-23-CE46-0005-01).

References

Ahmad Abdelfattah, Hartwig Anzt, Erik G. Boman, Erin Carson, Terry Cojean, Jack Dongarra, Mark Gates, Thomas Grützmacher, Nicholas J. Higham, Sherry Li, Neil Lindquist, Yang Liu, Jennifer Loe, Piotr Luszczek, Pratik Nayak, Sri Pranesh, Siva Rajamanickam, Tobias Ribizel, Barry Smith, Kasia Swirydowicz, Stephen Thomas, Stanimire Tomov, Yaohung M. Tsai, Ichitaro Yamazaki, and Urike Meier Yang. 2020. A Survey of Numerical Methods Utilizing Mixed Precision Arithmetic. (July 2020). https://doi.org/10.48550/ARXIV.2007.06674 arXiv:2007.06674 [cs.MS]

P. R. Amestoy, O. Boiteau, A. Buttari, M. Gerest, F. Jézéquel, J.-Y. L'Excellent, and T. Mary. 2022. Mixed Precision Low-Rank Approximations and their Application to Block Low-Rank LU Factorization. *IMA J. Numer. Anal.* 43, 4 (2022), 2198–2227. https://doi.org/10.1093/imanum/drac037

Patrick R. Amestoy, Alfredo Buttari, Nicholas J. Higham, Jean-Yves L'Excellent, Theo Mary, and Bastien Vieublé. 2023. Combining Sparse Approximate Factorizations with Mixed Precision Iterative Refinement. *ACM Trans. Math. Software* 49, 1 (2023). https://doi.org/10.1145/3582493

P. R. Amestoy, A. Buttari, N. J. Higham, J.-Y. L'Excellent, T. Mary, and B. Vieublé. 2024. Five-precision GMRES-based iterative refinement. *SIAM J. Matrix Anal. Appl.* 45, 1 (2024), 529–552. https://doi.org/10.1137/23M1549079

Patrick R. Amestoy, Antoine Jego, Jean-Yves L'Excellent, Theo Mary, and Grégoire Pichon. 2025. BLAS-based Block Memory Accessor with Applications to Mixed Precision Sparse Direct Solvers. https://hal.science/hal-05019106 HAL EPrint hal-05019106.

Ani Anciaux-Sedrakian, Cédric Chevalier, Stéphane de Chaisemartin, Jean-Marc Gratien, Thomas Guignon, Pascal Havé, Nathalie Möller, and Xavier Tunc. 2022. Evaluation of the performance portability layer of different linear solver packages with alien, an open generic and extensible linear algebra framework.. In *ECCOMAS 2022: 8th European Congress on Computational Methods in Applied Sciences and Engineering*.

Hartwig Anzt, Jack Dongarra, Goran Flegar, Nicholas J. Higham, and Enrique S. Quintana-Ortí. 2018. Adaptive precision in block-Jacobi preconditioning for iterative sparse linear system solvers. *Concurrency Computat. Pract. Exper.* 31, 6 (March 2018). https://doi.org/10/gs7745

Mario Arioli and Iain Duff. 2008. Using FGMRES to obtain backward stability in mixed precision. *Electron. Trans. Numer. Anal.* 33 (01 2008).

M. Arioli, I. S. Duff, S. Gratton, and S. Pralet. 2007. A Note on GMRES Preconditioned by a Perturbed LDL^T Decomposition with Static Pivoting. *SIAM J. Sci. Comput.* 29, 5 (Jan. 2007), 2024–2044. https://doi.org/10.1137/060661545

Marc Baboulin, Alfredo Buttari, Jack Dongarra, Jakub Kurzak, Julie Langou, Julien Langou, Piotr Luszczek, and Stanimire Tomov. 2009. Accelerating scientific computations with mixed precision algorithms. *Comput. Phys. Comm.* 180, 12 (Dec. 2009), 2526–2533. https://doi.org/10.1016/j.cpc.2008.11.005

Alfredo Buttari, Jack Dongarra, Jakub Kurzak, Piotr Luszczek, and Stanimir Tomov. 2008. Using Mixed Precision for Sparse Matrix Computations to Enhance the Performance while Achieving 64-bit Accuracy. *ACM Trans. Math. Software* 34, 4 (July 2008), 1–22. https://doi.org/10.1145/1377596.1377597

Alfredo Buttari, Xin Liu, Theo Mary, and Bastien Vieublé. 2025. Mixed precision strategies for preconditioned GMRES: a comprehensive analysis. https://hal.science/hal-05071696 HAL EPrint hal-05071696.

Erin Carson and Ieva Daužickaitė. 2024. The stability of split-preconditioned FGMRES in four precisions. *Electron. Trans. Numer. Anal.* 60 (2024), 40–58. https://doi.org/10.1553/etna_vol60s40

Erin Carson and Nicholas J. Higham. 2017. A New Analysis of Iterative Refinement and Its Application to Accurate Solution of Ill-Conditioned Sparse Linear Systems. *SIAM J. Sci. Comput.* 39, 6 (Jan. 2017), A2834–A2856. https://doi.org/10.1137/17m1122918

Erin Carson and Nicholas J. Higham. 2018. Accelerating the Solution of Linear Systems by Iterative Refinement in Three Precisions. *SIAM J. Sci. Comput.* 40, 2 (Jan. 2018), A817–A847. https://doi.org/10/gs77zr

Erin Carson and Noaman Khan. 2023. Mixed Precision Iterative Refinement with Sparse Approximate Inverse Preconditioning. *SIAM J. Sci. Comput.* 45, 3 (June 2023), C131–C153. https://doi.org/10.1137/22m1487709

Donglin Chen, Jianbin Fang, Shizhao Chen, Chuanfu Xu, and Zheng Wang. 2019. Optimizing Sparse Matrix—Vector Multiplications on an ARMv8-Based Many-Core Architecture. *Int. J. Parallel Prog.* (2019). https://doi.org/10.1007/s10766-018-00625-8

M. A. Christie and M. J. Blunt. 2001. Tenth SPE Comparative Solution Project: A Comparison of Upscaling Techniques. *SPE Reservoir Evaluation & Engineering* 4, 04 (Aug. 2001), 308–317. https://doi.org/10.2118/72469-pa

Timothy A. Davis and Yifan Hu. 2011. The University of Florida sparse matrix collection. *ACM Trans. Math. Software* 38, 1 (Nov. 2011), 1–25. https://doi.org/10.1145/2049662.2049663

Stef Graillat, Fabienne Jézéquel, Theo Mary, Roméo Molina, and Daichi Mukunoki. 2024. Reduced-Precision and Reduced-Exponent Formats for Accelerating Adaptive Precision Sparse Matrix-Vector Product. In *Euro-Par 2024: Parallel Processing*. Springer Nature Switzerland, Cham, 17–30. https://doi.org/10.1007/978-3-031-69583-4_2

Thomas Grützmacher, Hartwig Anzt, and Enrique S. Quintana-Ortí. 2023. Using Ginkgo's memory accessor for improving the accuracy of memory-bound low precision BLAS. *Softw.: Pract. Exper.* 53, 1 (2023), 81–98. https://doi.org/10.1002/spe.3041

Azzam Haidar, Harun Bayraktar, Stanimire Tomov, Jack Dongarra, and Nicholas J. Higham. 2020. Mixed-Precision Iterative Refinement Using Tensor Cores on GPUs to Accelerate Solution of Linear Systems. *Proc. Roy. Soc. London A* 476, 2243 (2020), 20200110. https://doi.org/10.1098/rspa.2020.0110

Azzam Haidar, Stanimire Tomov, Jack Dongarra, and Nicholas J. Higham. 2018. Harnessing GPU Tensor Cores for Fast FP16 Arithmetic to Speed up Mixed-Precision Iterative Refinement Solvers. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis (SC18 (Dallas, TX))*. IEEE, Piscataway, NJ, USA, 47:1–47:11. https://doi.org/10.1109/SC.2018.00050

N. Higham. 1997. Iterative refinement for linear systems and LAPACK. *IMA J. Numer. Anal.* 17, 4 (Oct. 1997), 495–509. https://doi.org/10.1093/imanum/17.4.495

Nicholas J. Higham. 1991. Iterative refinement enhances the stability of QR factorization methods for solving linear equations. *BIT* 31, 3 (Sept. 1991), 447–468. https://doi.org/10.1007/bf01933262

Nicholas J. Higham and Theo Mary. 2022. Mixed Precision Algorithms in Numerical Linear Algebra. *Acta Numerica* 31 (2022), 347–414. https://doi.org/10.1017/s0962492922000022

Yongseok Jang, Pierre Jolivet, and Theo Mary. 2025. Mixed precision augmented GMRES. https://hal.science/hal-05163845 HAL EPrint hal-05163845.

M. Jankowski and H. Woźniakowski. 1977. Iterative refinement implies numerical stability. *BIT* 17, 3 (Sept. 1977), 303–311. https://doi.org/10.1007/bf01932150

Moritz Kreutzer, Georg Hager, Gerhard Wellein, Holger Fehske, and Alan R. Bishop. [n. d.]. A Unified Sparse Matrix Data Format for Efficient General Sparse Matrix-Vector Multiplication on Modern Processors with Wide SIMD Units. *SIAM J. Sci. Comput.* ([n. d.]). https://doi.org/10.1137/130930352

Ronald Kriemann. 2023. Hierarchical Lowrank Arithmetic with Binary Compression. https://doi.org/10.48550/ARXIV.2308.10960

Ronald Kriemann. 2024. Performance of H-Matrix-Vector Multiplication with Floating Point Compression. (May 2024). https://doi.org/10.48550/ARXIV.2405.03456 arXiv:2405.03456 [cs.DC]

Julie Langou, Julien Langou, Piotr Luszczek, Jakub Kurzak, Alfredo Buttari, and Jack Dongarra. 2006. Exploiting the Performance of 32 bit Floating Point Arithmetic in Obtaining 64 bit Accuracy (Revisiting Iterative Refinement for Linear Systems). In *ACM/IEEE SC 2006 Conference (SC'06)*. IEEE. https://doi.org/10.1109/sc.2006.30

B.C. Lee, R.W. Vuduc, J.W. Demmel, and K.A. Yelick. 2004. Performance models for evaluation and automatic tuning of symmetric sparse matrix-vector multiply. In *International Conference on Parallel Processing, 2004. ICPP 2004.* https://doi.org/10.1109/ICPP.2004.1327917

Xiaoye S. Li and J.W. Demmel. 1998. Making Sparse Gaussian Elimination Scalable by Static Pivoting. In *Proceedings of the IEEE/ACM SC98 Conference*. IEEE, 34–34. https://doi.org/10.1109/sc.1998.10030

Neil Lindquist, Piotr Luszczek, and Jack Dongarra. 2022. Accelerating Restarted GMRES With Mixed Precision Arithmetic. 33, 4 (April 2022), 1027–1037. https://doi.org/10.1109/tpds.2021.3090757

Weifeng Liu and Brian Vinter. 2015. CSR5: An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication. (2015). https://doi.org/10.1145/2751205.2751209

Jennifer A. Loe, Christian A. Glusa, Ichitaro Yamazaki, Erik G. Boman, and Sivasankaran Rajamanickam. 2021. A Study of Mixed Precision Strategies for GMRES on GPUs. https://doi.org/10.48550/ARXIV.2109.01232

Daichi Mukunoki, Masatoshi Kawai, and Toshiyuki Imamura. 2023. Sparse Matrix-Vector Multiplication with Reduced-Precision Memory Accessor. In *2023 IEEE 16th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. 608–615. https://doi.org/10.1109/MCSoC60832.2023.00094

Yvan Notay. 2000. Flexible Conjugate Gradients. *SIAM J. Sci. Comput.* 22, 4 (Jan. 2000), 1444–1460. https://doi.org/10.1137/s1064827599362314

Eda Oktay and Erin Carson. 2023. Mixed precision GMRES-based iterative refinement with recycling. https://doi.org/10.48550/arXiv.2201.09827

Youcef Saad. 1993. A Flexible Inner-Outer Preconditioned GMRES Algorithm. *SIAM J. Sci. Comput.* 14, 2 (March 1993), 461–469. https://doi.org/10.1137/0914028

Yousef Saad. 2003. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898718003

Robert D. Skeel. 1980. Iterative refinement implies numerical stability for Gaussian elimination. *Math. Comp.* 35, 151 (1980), 817–832. https://doi.org/10.1090/s0025-5718-1980-0572859-4

G. L. G. Sleijpen and H. A. van der Vorst. 1996. Reliable updated residuals in hybrid Bi-CG methods. *Computing* 56, 2 (June 1996), 141–163. https://doi.org/10.1007/bf02309342

Kathryn Turner and Homer F. Walker. 1992. Efficient High Accuracy Solutions with GMRES(m). *SIAM J. Sci. Statist. Comput.* 13, 3 (May 1992), 815–825. https://doi.org/10.1137/0913048

H. A. van der Vorst. 1992. Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems. *SIAM J. Sci. Statist. Comput.* 13, 2 (March 1992), 631–644. https://doi.org/10.1137/0913035

Judith A. Vogel. 2007. Flexible BiCG and flexible Bi-CGSTAB for nonsymmetric linear systems. *Appl. Math. Comput.* 188, 1 (May 2007), 226–233. https://doi.org/10/d9k867

Samuel Williams, Leonid Oliker, Richard Vuduc, John Shalf, Katherine Yelick, and James Demmel. 2007. Optimization of sparse matrix-vector multiplication on emerging multicore platforms. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing (SC '07)*. ACM, 1–12. https://doi.org/10.1145/1362622.1362674

Yingqi Zhao, Takeshi Fukaya, and Takeshi Iwashita. 2023. Numerical Behavior of Mixed Precision Iterative Refinement Using the BiCGSTAB Method. *J. Inf. Process.* 31, 0 (2023), 860–874. https://doi.org/10.2197/ipsjjip.31.860

Yingqi Zhao, Takeshi Fukaya, Linjie Zhang, and Takeshi Iwashita. 2022. Numerical Investigation into the Mixed Precision GMRES (m) Method Using FP64 and FP32. *J. Inf. Process.* 30 (2022), 525–537. https://doi.org/10.2197/ipsjjip.30.525