Combining sparse approximate factorizations with mixed precision iterative refinement

PATRICK AMESTOY, Mumps Technologies, ENS Lyon, France
ALFREDO BUTTARI, CNRS, IRIT, France
NICHOLAS J. HIGHAM, Department of Mathematics, The University of Manchester, UK
JEAN-YVES L'EXCELLENT, Mumps Technologies, ENS Lyon, France
THEO MARY, Sorbonne Université, CNRS, LIP6, France
BASTIEN VIEUBLÉ, INPT, IRIT, France

The standard LU factorization-based solution process for linear systems can be enhanced in speed or accuracy by employing mixed precision iterative refinement. Most recent work has focused on dense systems. We investigate the potential of mixed precision iterative refinement to enhance methods for sparse systems based on approximate sparse factorizations. In doing so we first develop a new error analysis for LU- and GMRES-based iterative refinement under a general model of LU factorization that accounts for the approximation methods typically used by modern sparse solvers, such as low-rank approximations or relaxed pivoting strategies. We then provide a detailed performance analysis of both the execution time and memory consumption of different algorithms, based on a selected set of iterative refinement variants and approximate sparse factorizations. Our performance study uses the multifrontal solver MUMPS, which can exploit block low-rank (BLR) factorization and static pivoting. We evaluate the performance of the algorithms on large, sparse problems coming from a variety of real-life and industrial applications showing that the proposed approach can lead to considerable reductions of both the time and memory consumption.

CCS Concepts: • **Mathematics of computing → Solvers**; **Mathematical software performance**; **Computations on matrices**; • **Computing methodologies → Linear algebra algorithms**.

Additional Key Words and Phrases: iterative refinement, GMRES, linear system, mixed precision, multiple precision, rounding error analysis, floating-point arithmetic, sparse direct solver, multifrontal method, preconditioning, parallelism

ACM Reference Format:
Patrick Amestoy, Alfredo Buttari, Nicholas J. Higham, Jean-Yves L'Excellent, Theo Mary, and Bastien Vieublé. 2022. Combining sparse approximate factorizations with mixed precision iterative refinement. 1, 1 (January 2022), 29 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Direct methods for the solution of sparse linear systems $Ax = b$, where $A \in \mathbb{R}^{n \times n}$ and $x, b \in \mathbb{R}^n$, are widely used and generally appreciated for their robustness and accuracy. These desirable

Authors' addresses: Patrick Amestoy, Mumps Technologies, ENS Lyon, 46 Allée d'Italie, Lyon, France, patrick.amestoy@mumps-tech.com; Alfredo Buttari, CNRS, IRIT, 2 Rue Charles Camichel, Toulouse, France, alfredo.buttari@irit.fr; Nicholas J. Higham, Department of Mathematics, The University of Manchester, M13 9PL, Manchester, UK, nick.higham@manchester.ac.uk; Jean-Yves L'Excellent, Mumps Technologies, ENS Lyon, 46 Allée d'Italie, Lyon, France, jean-yves.l.excellent@mumps-tech.com; Theo Mary, Sorbonne Université, CNRS, LIP6, Paris, France, theo.mary@lip6.fr; Bastien Vieublé, INPT, IRIT, 2 Rue Charles Camichel, Toulouse, France, bastien.vieuble@irit.fr.

XXXX-XXXX/2022/1-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

properties, however, come at the cost of high operational complexity and memory consumption and a limited scalability on large scale, parallel supercomputers compared with iterative solvers. In order to mitigate some of these limitations, we can use various approaches to trade off some accuracy and robustness for lower complexity and memory consumption or better computational efficiency and scalability. These include the use of low-rank approximations or relaxed numerical pivoting. Furthermore, the recent appearance and increasingly widespread adoption of low precision arithmetics offer additional opportunities for reducing the computational cost of sparse direct solvers. In the cases where these approaches lead to a poor quality solution, they can be combined with other lightweight algorithms that attempt to recover the lost accuracy. Arguably the most well known and oldest of such algorithms is iterative refinement, whose basic idea is to improve the accuracy of a given computed solution $\widehat{x}$ by iteratively repeating three steps:

(1) compute the residual $r = b - A\widehat{x}$;
(2) solve $Ad = r$;
(3) update $\widehat{x} \leftarrow \widehat{x} + d$.

While same precision can be used on all three refinement steps in order to improve numerical stability [35, 46], multiple arithmetics can be conveniently mixed in order to achieve better accuracy, robustness, or performance. The method, originally proposed by Wilkinson [49, 50] for fixed-point arithmetic and later extended by Moler [41] to floating-point computations, uses higher precision for computing the residuals, which allows the method to converge to a more accurate solution. Langou et al. [37] and Buttari et al. [15] redefined iterative refinement as a way to accelerate a direct solver by computing the LU factorization of A in single precision instead of double while keeping a double precision accuracy on the solution.

In recent years, the emergence of lower precision floating-point arithmetic in hardware, in particular the half precision fp16 and bfloat16 arithmetics, has generated renewed interest in mixed precision algorithms in general and in mixed precision iterative refinement in particular [33]. Recent work has explored the use of mixed precision iterative refinement methods that employ a factorization computed in low precision [17]. Furthermore, novel mixed precision iterative refinement variants [3, 17] that can use up to five different precisions have been proposed, offering a wide range of options with different tradeoffs between accuracy, robustness, and performance. Several of these variants of iterative refinement have been implemented on modern hardware, notably supporting half precision such as GPUs, and have been shown to be highly successful at accelerating the solution of dense linear systems [26–28, 39]. Unlike for dense systems, there have been few previous efforts to accelerate sparse direct solvers with iterative refinement, and most date back to the two-precision variants of the late 2000s [13, 14].

In this article we investigate the potential of mixed precision arithmetic to accelerate the solution of large sparse linear systems by combining state-of-the-art iterative refinement variants with state-of-the-art sparse factorizations taking into account the use of numerical approximations. First, we tackle this subject from a theoretical point of view and extend the error analysis in [3] to the case of approximate factorizations. Then we address the issues related to a high performance parallel implementation of mixed precision iterative refinement for sparse linear systems and provide an in-depth analysis of experimental results obtained on real-life problems.

The article is structured as follows. For the sake of completeness, in section 2 we present relevant information on sparse direct solvers and approximate factorization methods, and give details on the different iterative refinement algorithms that we work with. In section 3 we explain how the specific features of sparse direct solvers influence the behavior of iterative refinement and the design choices that have to be made when considering which variant to use and what the differences are with respect to the dense case. We provide, in section 4, an error analysis for iterative refinement using a general approximate factorization model with LU factorization or GMRES as the solver on the update step. In section 5, we illustrate the effectiveness of a subset of modern iterative refinement variants, with and without numerical approximations, for the parallel solution of large scale sparse linear systems by implementing them on top of the MUMPS sparse direct solver [7, 8]. This includes a performance analysis of the standard factorization, as well as the use of two types of approximate factorizations and their combination on problems coming from a variety of real life industrial and academic applications.

2 BACKGROUND

2.1 Sparse direct methods

Sparse direct solvers compute the solution of a sparse linear system through the factorization of the associated matrix. In this work we deal with Gaussian elimination types of factorizations, that is, LU, Cholesky or LDL^T. Computing the factorization of a sparse matrix is made difficult by the occurrence of *fill-in*, which is when zero coefficients of the sparse matrix are turned into nonzeros by the factorization process. Because of fill-in, in general, it is not possible to define the complexity of a sparse matrix factorization; however, for a 3D cubic problem it can be shown that, if the matrix is permuted using nested dissection [24], $O(n^2)$ floating-point operations are required and the size of the resulting factors is $O(n^{4/3})$ (respectively, $O(n^{3/2})$ and $O(n \log(n))$ for a 2D, square problem), assuming n is the size of the matrix.

In the case of unsymmetric or symmetric indefinite problems, pivoting must be used to contain element growth and make the solution process backward stable. However, for both dense and sparse systems, pivoting reduces the efficiency and scalability of the factorization on parallel computers because it requires communication and synchronizations. In the case of sparse factorizations, pivoting has the additional drawback of introducing additional fill-in. Moreover, because this fill-in depends on the unfolding of the factorization it cannot be predicted beforehand, so pivoting requires the use of dynamic data structures and may lead to load unbalance in a parallel setting. For this reason, few sparse direct solvers employ robust pivoting techniques. Although in many cases the overhead imposed by pivoting can be modest, when targeting large scale parallel computers and/or numerically difficult problems performance may be severely affected.

2.2 Approximate factorizations

In order to improve performance and/or reduce the complexity, sparse direct solvers often compute approximate factorizations. In this work we mainly focus on two approximate factorization techniques, described in the next two paragraphs, which can be combined. The error analysis presented in section 4, though, is more general and potentially applies to other approximate factorization methods.

Block low-rank (BLR). In several applications, we can exploit the data sparsity by partitioning dense matrices (for example appearing during a sparse matrix factorization) into blocks of low numerical rank. Sparse direct solvers exploiting this property to accelerate the computations have been proposed and shown to be highly effective in a variety of applications [5, 7, 25, 43, 45]. We will in particular focus on the block low-rank (BLR) format [4, 6, 7], which is based on a flat block partitioning of the matrix into low-rank blocks. The LU factorization of a BLR matrix can be efficiently computed by adapting the usual partitioned LU factorization to take advantage of the low-rank property of the blocks (see, for example, [7] for a detailed description of the algorithms). The use of the BLR method in a sparse direct solver can reduce, at best, the operational complexity to $O(n^{4/3})$ and the factors size to $O(n \log n)$ for a 3D problem (respectively, $O(n \log n)$ and $O(n)$ for a 2D problem) [6]. The constants hidden in the big O complexities depend on the ranks of the blocks, which are determined by a threshold, τ_b in this article, that controls the accuracy of the approximations. A larger threshold leads to lower memory and operational costs, but also to lower accuracy. This makes iterative refinement a particularly attractive method because it allows us to recover a satisfactory accuracy in the cases where a large threshold is employed to reduce the time and memory consumption of the factorization.

Static pivoting. Unlike partial pivoting, static pivoting, first proposed by Li and Demmel [38], does not apply permutations on the rows or columns of the sparse matrix. Instead when a pivot is found to be too small with respect to a prescribed threshold $\tau_s \|A\|_\infty$, it is replaced with $\tau_s \|A\|_\infty$. Static pivoting improves the use of BLAS3 operations and improves parallelism with respect to partial pivoting, whose scalability suffers from the communications needed to identify the pivots at each elimination stage. Moreover, the use of static pivoting in a sparse direct solver does not introduce additional fill-in, as partial pivoting does, and, consequently, is less prone to load unbalance. It must be noted that static pivoting has a twofold effect on the accuracy and stability of the factorization. A small value for τ_s makes the factorization more accurate but might lead to large element growth, while a large value controls element growth but reduces the accuracy of the factorization. Several previous studies [12, 22, 38] have proposed to remedy the instability introduced by static pivoting by using fixed precision iterative refinement.

2.3 Iterative refinement

Iterative refinement is an old algorithm, but major evolutions were recently proposed and we summarize here the most up-to-date forms that are based on the LU factorization of the matrix A.

Algorithm 1 LU based iterative refinement in three precisions (LU-IR)

```
Input: an $n \times n$ matrix $A$ and a right-hand side $b$.
Output: an approximate solution to $Ax = b$.
Compute the LU factorization $A \approx \widehat{L}\widehat{U}$ at precision $u_f$.
Initialize $x_0$ (to, e.g., $\widehat{U}^{-1}\widehat{L}^{-1}b$).
while not converged do
    Compute $r_i = b - Ax_i$ at precision $u_r$.
    Solve $Ad_i = r_i$ by $d_i = \widehat{U}^{-1}\widehat{L}^{-1}r_i$ at precision $u_f$.
    Compute $x_{i+1} = x_i + d_i$ at precision $u$.
end while
```

Algorithm 2 GMRES based iterative refinement in five precisions (GMRES-IR)

```
Input: an $n \times n$ matrix $A$ and a right-hand side $b$.
Output: an approximate solution to $Ax = b$.
Compute the LU factorization $A \approx \widehat{L}\widehat{U}$ at precision $u_f$.
Initialize $x_0$ (to, e.g., $\widehat{U}^{-1}\widehat{L}^{-1}b$).
while not converged do
    Compute $r_i = b - Ax_i$ at precision $u_r$.
    Solve $\widehat{U}^{-1}\widehat{L}^{-1}Ad_i = \widehat{U}^{-1}\widehat{L}^{-1}r_i$ by GMRES at precision $u_g$ with matrix–vector products with
        $\widetilde{A} = \widehat{U}^{-1}\widehat{L}^{-1}A$ at precision $u_p$.
    Compute $x_{i+1} = x_i + d_i$ at precision $u$.
end while
```

The historical and most common form of iterative refinement solves the correction equation $Ad = r$ by substitution using the computed LU factors of the matrix in precision u_f. The computation of the residual is done in precision u_r and the update is done in working precision u. We refer to this kind of iterative refinement as LU-based iterative refinement or LU-IR, which is described in Algorithm 1. However, the use of low precision arithmetic to accelerate the LU factorization also restricts substantially the ability of LU-IR to handle moderately ill-conditioned problems. To overcome this limitation and extend the applicability of low precision factorizations, Carson and Higham [16] proposed an alternative form of iterative refinement that can handle much more ill-conditioned matrices by solving the system $Ad = r$ by the GMRES method preconditioned with the computed LU factors, as described in Algorithm 2. The GMRES carries out its operations in precision u_g except the preconditioned matrix–vector products which are applied in a precision u_p. We refer to this method as GMRES-based iterative refinement or GMRES-IR. As an example, if the factorization is carried out in fp16 arithmetic, then LU-IR is only guaranteed to converge if $\kappa(A) \ll 2 \times 10^3$, whereas GMRES-IR is guaranteed to converge if $\kappa(A) \ll 3 \times 10^7$ in the case where the GMRES precisions (u_g and u_p) correspond to double precision arithmetic.

With the rising number of available precisions in hardware, Carson and Higham [17] reestablished the use of extra precision in the computation of the residual, bridging the gap between traditional iterative refinement targeting accuracy improvements and iterative refinement targeting a faster factorization. This leads to the use of up to three different precisions in LU-IR (Algorithm 1). Finally Amestoy et al. [3] have analyzed Algorithm 2 in five precisions to allow for an even more flexible choice of precisions and to be able to best exploit the range of arithmetics available in the target hardware.

3 SPECIFIC FEATURES OF ITERATIVE REFINEMENT WITH SPARSE DIRECT SOLVERS

The most important difference between iterative refinement for dense and sparse linear systems lies in its practical cost. As explained in section 2.1, a key property of sparse direct solvers is that they generate *fill-in*, that is, the LU factors of A are typically much denser than A itself. Therefore, as the size of the matrix grows, the storage for A becomes negligible compared with that for its LU factors. Note that this still holds for data sparse solvers despite the reduced asymptotic complexity. For example, as explained in section 2.2, BLR sparse direct solvers reduce the size of the LU factors to at best $O(n \log n)$ entries, but with the constants hidden in the big O, the size of the LU factors typically remains several orders of magnitude larger than that of the original matrix.

A crucial consequence of the existence of fill-in is that, with a lower precision factorization ($u_f > u$), LU-IR (Algorithm 1) can achieve not only higher speed but also lower memory consumption than a standard sparse direct solver run entirely in precision u. This is because the LU factors, which account for most of the memory footprint, need be stored only in precision u_f. We emphasize that LU-IR does not reduce the memory footprint in the case of dense linear systems, since in this case the matrix A and the LU factors require the same number of entries, and A must be stored at least in precision u. In fact, since a copy of A must be kept in addition to its LU factors, iterative refinement for dense linear systems actually consumes more memory than a standard in-place LU factorization in precision u.

Similar comments apply to the cost of the matrix–vector products Ax_i in the computation of the residual (step 4 of Algorithms 1 and 2). Whereas for a dense matrix this cost is comparable with that of the LU triangular solves (step 5 with LU-IR), when the matrix is sparse it becomes, most of the time, negligible. In particular, this means that we have more flexibility to choose the precision u_r, especially when the target precision u is double precision: performing the matrix–vector products in high precision ($u_r = u^2$) does not necessarily have a significant impact on the performance, even for arithmetics usually not supported in hardware, such as quadruple precision (fp128). This is illustrated and further discussed in section 5.

To summarize, LU-IR is attractive for sparse linear systems because it can lead to memory gains and because the most costly step of the iterative phase, the triangular solves with the LU factors, is carried out in the low precision u_f.

Unfortunately these last points are not guaranteed to be met when using GMRES-IR because the triangular solves have to be applied in precision $u_p < u_f$. As a consequence the cost of the iterations is higher and the factors need to be casted in precision u_p. As an extreme case, setting $u_p = u^2$ as originally proposed by Carson and Higham [17] would make the iterative phase significantly costly compared with the factorization. Therefore, the five-precision analysis of Amestoy et al. [3], which allows for setting $u_p > u^2$, is even more relevant in the sparse case. In this article, we therefore focus on variants where $u_p \geq u$.

Finally, another specific feature of sparse direct solvers is related to pivoting. While partial pivoting is the most common approach for dense linear systems, sparse direct solvers commonly use other approaches (for example static pivoting, see section 2.2) that better preserve the sparsity of the LU factors and limit the communications in parallel contexts. While partial pivoting guarantees the practical stability of the resolution, these methods do not. However, combined with iterative refinement, a sparse direct solver can achieve a satisfactory stability under suitable conditions.

4 ERROR ANALYSIS OF ITERATIVE REFINEMENT WITH A GENERAL APPROXIMATE FACTORIZATION

Carson and Higham [17] provided an error analysis of a general form of iterative refinement using an arbitrary linear solver. They then specialized this analysis to LU-IR and GMRES-IR, under the assumption that the LU factors are computed with standard Gaussian elimination with partial pivoting. However, as explained above, modern sparse direct solvers often depart from this assumption, because they typically do not implement partial pivoting, and because they take advantage of data sparsity resulting in numerical approximations. This affects the error analysis of LU-IR and GMRES-IR and the conditions under which they are guaranteed to converge. For this reason, in this section we propose a new error analysis under a general approximate factorization model. Our model can be applied to at least BLR, static pivoting and their combined use, and we expect it to cover several other approximate approaches used in direct solvers. Moreover, although in this article we are particularly motivated by sparse applications, the results of this section carry over to the dense case.

4.1 Preliminaries and notations

We use the standard model of floating-point arithmetic [29, sect. 2.2]. For any integer k we define

$$\gamma_k = \frac{ku}{1 - ku}.$$

A superscript on γ denotes that u carries that superscript as a subscript; thus $\gamma_k^f = ku_f/(1 - ku_f)$, for example. We also use the notation $\widetilde{\gamma}_k = \gamma_{ck}$ to hide modest constants c.

The error bounds obtained by our analysis depend on some constants related to the problem dimension n. We refer to these constants as c_k for $k = 1, 2, 3 \ldots$ As these constants are known to be pessimistic [18, 30, 31], for the sake of the readability, we do not always keep track of their precise value. When we drop constants c_k from an inequality we write the inequality using "$\ll$". A convergence condition expressed as "$\kappa(A) \ll \theta$" can be read as "$\kappa(A)$ is sufficiently less than θ". Finally, we also use the notation $\lesssim$ when dropping second order terms in the error bounds.

We consider a sparse linear system $Ax = b$, where $A \in \mathbb{R}^{n\times n}$ is nonsingular and $b \in \mathbb{R}^n$. We denote by p the maximum number of nonzeros in any row of the matrix $[A\ b]$.

The forward error of an approximate solution $\widehat{x}$ is $\varepsilon_{\mathrm{fwd}} = \|x - \widehat{x}\|/\|x\|$, while the (normwise) backward error of $\widehat{x}$ is [29, sec. 7.1]

$$\varepsilon_{\mathrm{bwd}} = \min\{\, \epsilon : (A + \Delta A)\widehat{x} = b + \Delta b,\ \|\Delta A\| \le \epsilon\|A\|,\ \|\Delta b\| \le \epsilon\|b\|\,\} = \frac{\|b - A\widehat{x}\|}{\|A\|\,\|\widehat{x}\| + \|b\|}.$$

We also use Wilkinson's growth factor ρ_n defined in [29, p. 165].

Our error analysis uses the ∞-norm, denoted by $\|\cdot\|_\infty$, and we write $\kappa_\infty(A) = \|A\|_\infty\|A^{-1}\|_\infty$ for the corresponding condition number of A. We use unsubscripted norms or condition numbers when the constants depending on the problem dimensions have been dropped, since the norms are equivalent.

4.2 Error analysis

In analyzing iterative refinement we aim to show that under suitable conditions the forward error and backward error decrease until they reach a certain size called the limiting forward error or backward error. We informally refer to "convergence", meaning that errors decrease to a certain limiting accuracy, while recognizing that the error does not necessarily converge in the formal sense.

Let us first recall the known results on LU-IR and GMRES-IR from the error analysis in [3, 17], based on the assumption of a standard LU factorization computed by Gaussian elimination with partial pivoting. In the case of LU-IR (Algorithm 1) the convergence condition for both the forward and backward errors is

$$\kappa(A)u_f \ll 1. \tag{4.1}$$

In the case of GMRES-IR (Algorithm 2) we have instead

$$(u_g + u_p\kappa(A))\kappa(A)^2u_f^2 \ll 1, \tag{4.2}$$

for the forward error to converge and

$$(u_g + u_p\kappa(A))(1 + \kappa(A)u_f)\kappa(A) \ll 1, \tag{4.3}$$

for the backward error to converge. These are both significantly less restrictive conditions than the LU-IR one (4.1).

Provided the corresponding conditions are met, the forward and backward errors will reach their limiting values

$$\varepsilon_{\mathrm{fwd}} \le pu_r\frac{\||A^{-1}||A||x|\|}{\|x\|} + u \tag{4.4}$$

and

$$\varepsilon_{\mathrm{bwd}} \le pu_r + u, \tag{4.5}$$

respectively. Note that these limiting values are solver independent (as long as iterative refinement converges).

We now turn to the main objective of this section, which is to derive conditions analogous to (4.1), (4.2), and (4.3) under a more general model of an approximate LU factorization. Specifically, our model makes the following two assumptions.

- The approximate factorization performed at precision u_s provides computed LU factors satisfying
$$A = \widehat{L}\widehat{U} + \Delta A^{(1)}, \quad |\Delta A^{(1)}| \lesssim c_1\epsilon\|A\|_\infty ee^T + c_2u_s|\widehat{L}||\widehat{U}|, \tag{4.6}$$
where e is the vector of ones and ϵ is a parameter quantifying the quality of the approximate factorization.

- The triangular solve $\widehat{T}y = v$, where $\widehat{T}$ is one of the approximately computed LU factors, performed at precision u_s provides a computed solution $\widehat{y}$ satisfying

$$(\widehat{T} + \Delta\widehat{T})\widehat{y} = v + \Delta v, \quad |\Delta\widehat{T}| \lesssim c_3 u_s |\widehat{T}|, \quad |\Delta v| \lesssim c_4 u_s |v|. \tag{4.7}$$

From (4.6) and (4.7), it follows that the solution of the linear system $Ay = v$ provides a computed solution $\widehat{y}$ satisfying

$$\begin{aligned} &(A + \Delta A^{(2)})\widehat{y} = v + \Delta v, \\ &|\Delta A^{(2)}| \lesssim c_1 \epsilon \|A\|_\infty ee^T + (c_2 + 2c_3) u_s |\widehat{L}||\widehat{U}|, \\ &|\Delta v| \lesssim c_4 u_s (|v| + |\widehat{L}||\widehat{U}||\widehat{y}|). \end{aligned} \tag{4.8}$$

4.2.1 Error analysis for LU-IR. We want to determine the convergence conditions for LU-IR (Algorithm 1). We can apply [17, Cor. 3.3] and [17, Cor. 4.2] respectively for the convergence of the forward and normwise backward errors of the system $Ax = b$, and for both we need respectively a bound on the forward and backward errors of the computed solution $\widehat{d_i}$ of the correction equation $Ad_i = \widehat{r}_i$. Note that for LU-IR, the factorization (4.6) and the LU solves (4.7) are performed in precision u_f.

Considering the solution of the linear system $Ad_i = \widehat{r}_i$, (4.8) yields

$$d_i - \widehat{d_i} = A^{-1}\Delta A^{(2)}\widehat{d_i} - A^{-1}\Delta\widehat{r}_i.$$

Taking norms, we obtain

$$\frac{\|d_i - \widehat{d_i}\|_\infty}{\|\widehat{d_i}\|_\infty} \lesssim (c_1\epsilon + c_4 u_f)\|A^{-1}\|_\infty \|A\|_\infty + (c_2 + 2c_3 + c_4) u_f \|A^{-1}\|_\infty \||\widehat{L}||\widehat{U}|\|_\infty.$$

Using [29, Lem. 9.6]

$$\||\widehat{L}||\widehat{U}|\|_\infty \leq (1 + 2(n^2 - n)\rho_n)(\|A\|_\infty + \|\Delta A^{(1)}\|_\infty), \tag{4.9}$$

where ρ_n is the growth factor for $A + \Delta A^{(1)}$. Dropping second order terms finally gives

$$\frac{\|d_i - \widehat{d_i}\|_\infty}{\|\widehat{d_i}\|_\infty} \lesssim c_5(\epsilon + \rho_n u_f)\kappa_\infty(A). \tag{4.10}$$

In the same fashion we can show that

$$\|\widehat{r}_i - A\widehat{d_i}\|_\infty \lesssim c_6(\epsilon + \rho_n u_f)(\|A\|_\infty \|\widehat{d_i}\|_\infty + \|\widehat{r}_i\|_\infty). \tag{4.11}$$

Dropping constants and applying [17, Cor. 3.3] and [17, Cor. 4.2] using (4.10) and (4.11) guarantees that as long as

$$(\rho_n u_f + \epsilon)\kappa(A) \ll 1 \tag{4.12}$$

the forward and the normwise backward errors of the system $Ax = b$ will converge to their limiting values (4.4) and (4.5).

As a check, if we set $\epsilon = 0$ (no approximation) and drop ρ_n (negligible element growth), we recover (4.1).

Before commenting in section 4.2.3 on the significance of these new LU-IR convergence conditions, we first similarly derive the GMRES-IR conditions.

4.2.2 Error analysis for GMRES-IR. We now determine the convergence conditions of GMRES-IR (Algorithm 2). We proceed similarly as for LU-IR, seeking bounds on the forward and backward errors of the computed solution $\widehat{d_i}$ of the correction equation $Ad_i = \widehat{r}_i$. One difference lies in the fact that the GMRES solver is applied to the preconditioned system $\widetilde{A}d_i = \widehat{s}_i$ where $\widetilde{A} = \widehat{U}^{-1}\widehat{L}^{-1}A$ and $s_i = \widehat{U}^{-1}\widehat{L}^{-1}\widehat{r}_i$. Our analysis follows closely the analysis of [3, sec. 3.2], so we mainly focus on the changes coming from the use of a more general approximate factorization model and refer the reader to that work for the full details.

We first need to bound the error introduced in forming the preconditioned right-hand side s_i in precision u_p. Computing s_i implies two triangular solves (4.7) which differ from the GMRES-IR original analysis by having an error term on the right-hand side. Adapting [3, Eqs. (3.11)-(3.13)] with (4.6) and (4.7) and using (4.9) provides the bound

$$\|s_i - \widehat{s}_i\|_\infty \lesssim c_7 u_p \rho_n \kappa_\infty(A)\|s_i\|_\infty. \tag{4.13}$$

As in [3] we compute the error of the computation of the preconditioned matrix–vector product $z_i = \widetilde{A}\widehat{v}_i$ in order to use [3, Thm. 3.1]. We obtain z_i through a standard matrix–vector product with A followed by two triangular solves (4.7) with $\widehat{L}$ and $\widehat{U}$. The computed $\widehat{z}_i$ satisfies $\widehat{z}_i = z_i + f_i$, where f_i carries the error of the computation. With a very similar reasoning as for deriving [3, Eq. (3.14)], considering our new assumptions, we obtain the bound

$$\|f_i\|_2 \lesssim c_8 u_p \rho_n \kappa_\infty(A)\|\widetilde{A}\|_F\|\widehat{v}_i\|_2. \tag{4.14}$$

Apart from the constants and the presence of the growth factor ρ_n which can be arbitrarily large without assumptions on the pivoting, (4.14) and (4.13) are similar to [3, Eq. (3.14)] and [3, Eq. (3.13)] and meet the assumptions of [3, Thm. 3.1] which can be used to compute a bound of $\|s_i - \widetilde{A}\widehat{d_i}\|_\infty$.

We can finally bound the normwise relative backward error of the system $\widetilde{A}\widehat{d_i} = s_i$ [3, Eq. (3.17)] by

$$\frac{\|s_i - \widetilde{A}\widehat{d_i}\|_\infty}{\|\widetilde{A}\|_\infty\|\widehat{d_i}\|_\infty + \|s_i\|_\infty} \lesssim c_9\big(u_g + u_p\rho_n\kappa_\infty(A)\big) \tag{4.15}$$

and the relative error of the computed $\widehat{d_i}$ [3, Eq. (3.18)] by

$$\frac{\|d_i - \widehat{d_i}\|_\infty}{\|d_i\|_\infty} \lesssim c_9\big(u_g + u_p\rho_n\kappa_\infty(A)\big)\kappa_\infty(\widetilde{A}). \tag{4.16}$$

In addition, the backward error of the original correction equation $Ad_i = \widehat{r}_i$ can be bounded using $\widehat{r}_i - A\widehat{d_i} = \widehat{L}\widehat{U}(s_i - \widetilde{A}\widehat{d_i})$ and (4.15), yielding

$$\|\widehat{r}_i - A\widehat{d_i}\|_\infty \lesssim c_9(u_g + u_p\rho_n\kappa_\infty(A))(\|\widetilde{A}\|_\infty\|A\|_\infty\|\widehat{d_i}\|_\infty + \kappa_\infty(A)\|\widehat{r}_i\|_\infty). \tag{4.17}$$

It is essential to study the conditioning of the preconditioned matrix $\widetilde{A}$ in order to express the convergence conditions according to the conditioning of the original matrix $\kappa(A)$. Using the same reasoning as for [16, Eq. 3.2] we obtain

$$\begin{aligned}\|\widetilde{A}\|_\infty &\lesssim 1 + c_{10}u_f\rho_n\kappa_\infty(A) + c_1\epsilon\kappa_\infty(A),\\ \kappa_\infty(\widetilde{A}) &\lesssim (1 + c_{10}u_f\rho_n\kappa_\infty(A) + c_1\epsilon\kappa_\infty(A))^2.\end{aligned}$$

Dropping constants and applying [17, Cor. 3.3] and [17, Cor. 4.2] using (4.16) and (4.17) guarantees that as long as

$$(u_g + u_p\rho_n\kappa(A))(u_f\rho_n + \epsilon)^2\kappa(A)^2 \ll 1 \qquad \text{(backward error)}, \tag{4.18}$$

$$(u_g + u_p\rho_n\kappa(A))((u_f\rho_n + \epsilon)\kappa(A) + 1)\kappa(A) \ll 1 \qquad \text{(forward error)}, \tag{4.19}$$

the forward and the normwise backward errors of the system $Ax = b$ will converge to their limiting values (4.4) and (4.5).

As a check, with $\epsilon = 0$ and dropping ρ_n, we recover (4.2) and (4.3).

4.2.3 Summary of the error analysis and interpretation. We summarize the analysis in the following theorem.

THEOREM 4.1. *Let $Ax = b$ be solved by LU-IR (Algorithm 1) or GMRES-IR (Algorithm 2) using an approximate LU factorization satisfying* (4.6)–(4.7). *Then the forward error will reach its limiting value* (4.4) *provided that*

$$(u_f \rho_n + \epsilon)\kappa(A) \ll 1 \qquad \textit{(LU-IR)}, \tag{4.20}$$

$$(u_g + u_p \rho_n \kappa(A))(u_f \rho_n + \epsilon)^2 \kappa(A)^2 \ll 1 \qquad \textit{(GMRES-IR)}, \tag{4.21}$$

and the backward error will reach its limiting value (4.5) *provided that*

$$(u_f \rho_n + \epsilon)\kappa(A) \ll 1 \qquad \textit{(LU-IR)}, \tag{4.22}$$

$$(u_g + u_p \rho_n \kappa(A))((u_f \rho_n + \epsilon)\kappa(A) + 1)\kappa(A) \ll 1 \qquad \textit{(GMRES-IR)}. \tag{4.23}$$

We now comment on the significance of this result. Compared with the original convergence conditions (4.1)–(4.3), the new conditions of Theorem 4.1 include two new terms. The first is the growth factor ρ_n that, without any assumption on the pivoting strategy, cannot be assumed to be small. This shows that a large growth factor can prevent iterative refinement from converging. The second is ϵ which reflects the degree of approximation used by the factorization. The terms $\rho_n u_f + \epsilon$ show that we can expect the approximation to impact the convergence of iterative refinement when $\epsilon \gtrsim \rho_n u_f$ (ignoring the difference between the constants in front of each term). It is important to note that the instabilities introduced by element growth and numerical approximations combine additively, rather than multiplicatively (there is no $\epsilon \rho_n$ term). In particular, this means that the usual wisdom that it is not useful to use a very high precision for an approximate factorization ($u_f \ll \epsilon$) is no longer true in presence of large element growth. This is a key property that we confirm experimentally in section 5.

Note that the left-hand side quantities in the convergence conditions (4.20)–(4.23) also bound the convergence rate of the refinement: thus smaller quantities will in general lead to faster convergence.

4.2.4 Convergence conditions for BLR and static pivoting. We now apply the above analysis to the use of BLR approximations and static pivoting.

The BLR format approximates the blocks of the matrix by replacing them by low-rank matrices. The ranks are determined by a threshold, τ_b in this article, that controls the accuracy of the approximations. Higham and Mary [32] have carried out error analysis of the BLR LU factorization and obtained a backward error bound of order $\tau_b \|A\| + u_f \|\widehat{L}\| \|\widehat{U}\|$. One issue is that their analysis derives normwise bounds, whereas our model (4.6) and (4.7) requires componentwise bounds. However, we have checked that, at the price of slightly larger constants and a more complicated analysis, analogous componentwise bounds can be obtained. Therefore, using the componentwise version of [32, Thm. 4.2] or [32, Thm. 4.3] for (4.6) and of [32, Thm. 4.4] for (4.7), we conclude that Theorem 4.1 applies with $\epsilon = \tau_b$.

We now turn to static pivoting, assuming a strategy that replaces pivots smaller in absolute value than $\tau_s \|A\|_\infty$ by $\tau_s \|A\|_\infty$, where τ_s is a threshold that controls the accuracy of the factorization. With such a strategy we are actually solving a perturbed system

$$Mx = b, \quad M = A + E, \tag{4.24}$$

where E is a diagonal matrix having nonzero entries equal to $\tau_s\|A\|_\infty$ in the positions corresponding to pivots that were replaced. By applying [29, Thm. 9.3] to (4.24) we meet the condition (4.6) with $\epsilon = \tau_s$, while condition (4.7) is met since the triangular solves are standard. Therefore Theorem 4.1 applies with $\epsilon = \tau_s$.

Finally, we can also derive convergence conditions for the case where BLR and static pivoting are combined. This amounts to using BLR approximations on the perturbed system (4.24), and so Theorem 4.1 applies with $\epsilon = \tau_b + \tau_s$.

5 PERFORMANCE ANALYSIS

We have implemented a selected set of iterative refinement variants and we analyze in this section their practical performance for the solution of large scale, real-life sparse problems on parallel computers.

After describing our implementation details and our experimental setting in section 5.1 and 5.2, we compare, in section 5.3, five variants with the case of a plain fp64 factorization plus solves. We then carry out detailed analyses of the time and memory performance of these variants in sections 5.3.1 and 5.3.2, respectively. In section 5.4 we investigate the use of four BLR, static pivoting, and BLR with static pivoting variants combined with iterative refinement.

5.1 Implementation details

To perform our experiments we implemented both LU-IR and GMRES-IR for their execution on parallel architectures. In the following we describe our implementation choices.

For the sparse LU factorization and LU triangular solves, we rely on the MUMPS solver [7, 9], which implements the multifrontal method [23]. It must be noted that most of our analysis readily applies to other sparse factorization approaches, such as the right- or left-looking supernodal method used, for example, in SuperLU [20], PaStiX [34], or PARDISO [44]. The only exception is the memory consumption analysis (section 5.3.2), where we rely on features of the multifrontal method. The default pivoting strategy used in the MUMPS solver is threshold partial pivoting [21] which provides great stability; alternatively, static pivoting (as described in section 2.2) can be used, where possible, to improve performance. MUMPS also implements the BLR factorization method described in section 2.2; for a detailed description of the BLR feature of MUMPS, we refer to [4, 7].

For the GMRES solver, we have used an in-house implementation of the unrestarted MGS-GMRES method. This code does not use MPI parallelism, but is multithreaded; as a result, all computations are performed on a single node, using multiple cores, except for the solves in the preconditioning which are operated through a call to the corresponding MUMPS routine which benefits from MPI parallelism. This also implies that the original system matrix and all the necessary vectors (including the Krylov basis) are centralized on MPI rank zero. The GMRES method is stopped when the scaled residual falls below a prescribed threshold τ_g.

In the GMRES-IR case, the solves require the LU factors to be in a different precision than what was computed by the factorization (that is, $u_f \neq u_p$). Two options are possible to handle this requirement. The first is to make an explicit copy of the factors by casting the data into precision u_p; the second is to make the solve operations blockwise, as is commonly done to take advantage of BLAS-3 operations, and cast the blocks on the fly using temporary storage. This second approach has the advantage of not increasing the memory consumption (only a small buffer is needed to cast blocks of the factors on the fly) and may even positively affect performance on memory-bound applications such as the solve [11]. However this approach requires in-depth modifications of the MUMPS solve step and we leave it for future work. In this article we thus rely on the first approach, and assume the cast is performed in-place so as to minimize the storage overhead. In the same fashion as the factors, we also cast the original matrix in precision u_r to perform the matrix–vector products in the residual computation.

For the symmetric matrices, we use the LDL^T factorization. It must be noted that the matrix-vector product is not easily parallelizable when a compact storage format is used for symmetric matrices (such as one that stores only the upper or lower triangular part); for this reason, we choose to store symmetric matrices with a non-compact format in order to make the residual computation more efficiently parallelizable.

The code implementing the methods has been written in Fortran 2003, supports real and complex arithmetics, and supports both multithreading (through OpenMP) and MPI parallelism (through MUMPS). The results presented below were obtained with MUMPS version 5.4.0; the default settings were used except we used the advanced multithreading described in [40]. We used the Metis [36] tool version 5.1.0 for computing the fill-reducing permutation. BLAS and LAPACK routines are from the Intel Math Kernel Library version 18.2 and the Intel C and Fortran compilers version 18.2 were used to compile our code as well as the necessary third party packages. The code was compiled with the "flush to zero" option to avoid inefficient computations on subnormal numbers; this issue is discussed in [51]. Since commonly available BLAS libraries do not support quadruple precision arithmetic, we had to implement some operations (copy, norms) by taking care of multithreading them.

5.2 Experimental setting

Throughout our experiments we analyze several variants of iterative refinement that use different combinations of precisions and different kinds of factorization, with and without approximations such as BLR or static pivoting.

In all experiments, the working precision is set to double ($u = \text{D}$) and GMRES is used in fixed precision ($u_g = u_p$) for a reason explained below. The factorization precision (u_f), the residual precision (u_r), and the precisions inside GMRES (u_g and u_p) may vary according to the experiments. Alongside the text, we define an iterative refinement variant with the solver employed (LU or GMRES) and the set of precisions u_f, u, and u_r (and u_g, u_p if GMRES is the solver used). If the solver employed is LU we refer to it as an LU-IR variant and if it is GMRES we call it a GMRES-IR variant. We use the letters S, D, and Q to refer to single, double, and quadruple precision arithmetic. We compare the iterative refinement variants to a standard double precision direct solver, namely, MUMPS, which we refer to as DMUMPS (Double precision MUMPS).

The values of the BLR threshold τ_b and the static pivoting threshold τ_s are specified alongside the text. For simplicity we set τ_g, the threshold used to stop GMRES, to 10^{-6} in all the experiments, even though it could be tuned on a case by case basis for optimized performance.

We do not cover all combinations of precisions of LU-IR and GMRES-IR; rather, we focus our study on a restricted number of combinations of u_f, u_g, and u_p, all meaningful in the sense of [3, sect. 3.4]. This is motivated by several reasons.

- Hardware support for half precision is still limited and the MUMPS solver on which we rely for this study does not currently support its use for the factorization; this prevents us from experimenting with $u_f = \text{H}$.
- Setting $u_p = \text{Q}$ might lead to excessively high execution time and memory consumption. In addition, it has been noticed in [3] that in practice this brings only a marginal improvement in the convergence compared with the case $u_p = \text{D}$ on a wide set of real life problems.
- In our experiments we rarely observed the Krylov basis to exceed more than a few dozen vectors except in section 5.4 for very high thresholds τ_b and τ_s. Hence setting $u_g > u_p$ to

reduce the memory footprint associated with the Krylov basis is not a priority for this study and we focused on the case $u_g = u_p$.

In sparse direct solvers, the factorization is commonly preceded by a so called analysis step to prepare the factorization. We do not report results on this step since:

- Its behavior is independent of the variants and precisions chosen.
- It can be performed once and reused for all problems that share the same structure.
- The fill-reducing permutation can be more efficiently computed when the problem geometry is known (which is the case in numerous applications).
- The efficiency of this step is very much implementation-dependent.

All the experiments were run on the Olympe supercomputer of the CALMIP supercomputing center of Toulouse, France. It is composed of 360 bi-processors nodes equipped with 192GB of RAM and 2 Intel Skylake 6140 processors (2.3Ghz, 18 cores) each. All experiments were done using 18 threads per MPI process because this was found to be the most efficient combination. Depending on the matrix, we use 2 or 4 MPI processes (that is, 1 or 2 nodes) for the problem to fit in memory; the number of MPI processes for each matrix is specified in Table 1 and is the same for all the experiments.

Table 1 shows the matrices coming from the SuiteSparse Matrix Collection [19] (not bold) and industrial applications provided by industrial partners (bold) that were used for our experiments. These matrices are chosen such that a large panel of applications and a large range of condition numbers are covered. The data reported in the last three columns of the table are computed by the MUMPS solver with the settings described above. As MUMPS applies a scaling for numerical stability on the input matrix, the displayed condition number is therefore the one of the scaled matrix.

In all tests the right-hand side vector was set to $b = Ax$ with a generated x vector having all its components set to 1, which also served as the reference solution to compute the forward error.

We give a short description of the matrices provided by our industrial partners:

- **ElectroPhys10M**: Cardiac electrophysiology model [42].
- **DrivAer6M**: Incompressible CFD, pressure problem, airflow around an automobile [48].
- **tminlet3M**: Noise propagation in an airplane turbine [10].
- **perf009ar**: Elastic design of a pump subjected to a constant interior pressure. It was provided by Électricité de France (EDF), who carries out numerical simulations for structural mechanics applications using Code_Aster[1].
- **elasticity-3d**: Linear elasticity problem applied on a beam combosed of hereogenous materials [2].
- **lfm_aug5M**: Electromagnetic modelling, stabilized formulation for the low frequency solution of Maxwell's equation [47].
- **CarBody25M**: structural mechanics, car body model.
- **thmgas**: coupled thermal, hydrological, and mechanical problem.

5.3 Performance of LU-IR and GMRES-IR using standard LU factorization

In this first set of experiments we analyze the time and memory savings that different iterative refinement variants without approximate factorization are able to achieve and we show how the specific features discussed in section 3, the choice of a multifrontal solver, and the matrix properties can affect the performance of the method.

[1] http://www.code-aster.org

Table 1. Set of matrices from SuiteSparse and industrial applications used in our experiments. N is the dimension; NNZ the number of nonzeros in the matrix; Arith. the arithmetic of the matrix (R: real, C: complex); Sym. the symmetry of the matrix (1: symmetric, 0: general); MPI the number of MPI processes used for the experiments with this matrix;$\kappa(A)$ the condition number of the matrix; Fact. flops the number of flops required for the factorization; Slv. flops the number of flops required for one LU solve.

ID	Name	N	NNZ	Arith.	Sym.	MPI	$\kappa(A)$	Fact. (flops)	Slv. (flops)
1	**ElectroPhys10M**	1.0E+07	1.4E+08	R	1	4	1E+01	3.9E+14	8.6E+10
2	ss	1.7E+06	3.5E+07	R	0	2	1E+04	4.2E+13	1.2E+10
3	nlpkkt80	1.1E+06	2.9E+07	R	1	2	2E+04	1.8E+13	7.4E+09
4	Serena	1.4E+06	6.4E+07	R	1	2	2E+04	2.9E+13	1.1E+10
5	Geo_1438	1.4E+06	6.3E+07	R	1	2	6E+04	1.8E+13	1.0E+10
6	Chevron4	7.1E+05	6.4E+06	C	0	2	2E+05	2.2E+10	1.6E+08
7	ML_Geer	1.5E+06	1.1E+08	R	0	2	2E+05	4.3E+12	4.1E+09
8	Transport	1.6E+06	2.4E+07	R	0	2	3E+05	1.1E+13	5.2E+09
9	Bump_2911	2.9E+06	1.3E+08	R	1	2	7E+05	2.0E+14	3.9E+10
10	**DrivAer6M**	6.1E+06	5.0E+07	R	1	2	9E+05	6.5E+13	2.6E+10
11	vas_stokes_1M	1.1E+06	3.5E+07	R	0	2	1E+06	1.5E+13	6.3E+09
12	Hook_1489	1.5E+06	6.1E+07	R	1	2	2E+06	8.3E+12	6.2E+09
13	Queen_4147	4.1E+06	3.3E+08	R	1	2	4E+06	2.7E+14	5.7E+10
14	dielFilterV2real	1.2E+06	4.8E+07	R	1	2	6E+06	1.1E+12	2.3E+09
15	Flan_1565	1.6E+06	1.2E+08	R	1	2	1E+07	3.9E+12	6.2E+09
16	**tminlet3M**	2.8E+06	1.6E+08	C	0	4	3E+07	1.1E+14	2.1E+10
17	**perf009ar**	5.4E+06	2.1E+08	R	1	2	4E+08	1.9E+13	1.9E+10
18	Pflow_742	7.4E+05	3.7E+07	R	1	2	3E+09	1.4E+12	2.1E+09
19	Cube_Coup_dt0	2.2E+06	1.3E+08	R	1	2	3E+09	9.9E+13	2.7E+10
20	**elasticity-3d**	5.2E+06	1.2E+08	R	1	2	4E+09	1.5E+14	5.2E+10
21	fem_hifreq_circuit	4.9E+05	2.0E+07	C	0	2	4E+09	4.3E+11	7.6E+08
22	**lfm_aug5M**	5.5E+06	3.7E+07	C	1	4	6E+11	2.2E+14	4.7E+10
23	Long_Coup_dt0	1.5E+06	8.7E+07	R	1	2	6E+12	5.2E+13	1.7E+10
24	**CarBody25M**	2.4E+07	7.1E+08	R	1	2	9E+12	9.6E+12	2.6E+10
25	**thmgas**	5.5E+06	3.7E+07	R	0	4	8E+13	1.1E+14	3.5E+10

In Table 2 we present the execution time and memory consumption of five iterative refinement variants and DMUMPS for the set of the test matrices of Table 1. We classify the variants into two categories; in the first, we have variants that achieve a forward error equivalent to that obtained with the double precision direct solver DMUMPS (the ones using $u_r = \text{D}$) and, in the second, those whose forward error is of order 10^{-16}, the double precision unit roundoff (the ones using $u_r = \text{Q}$). Actually, for the first category, LU-IR and GMRES-IR can provide a better accuracy on the solution than DMUMPS, which is why we stop their iterations when they reach a forward error of the same order as the solution obtained with DMUMPS. We denote by a "—" the failure of a method to converge. For each matrix the best execution time and memory consumption of the three variants is highlighted in bold.

Some general conclusions can be drawn from the results in this table. The LU-IR variants with single precision factorization generally achieve the lowest execution times, except for few cases where iterative refinement underperforms for reasons we will discuss in section 5.3.1 or where

Table 2. Execution time (in seconds) and memory consumption (in GBytes) of IR variants and DMUMPS for a subset of the matrices listed in Table 1. The solver and the precisions u_f, u_r, u_g, and u_p are specified in the table for each IR variant, u is fixed to u = D.

Solver	DMUMPS	LU	GMRES	LU	LU	GMRES	DMUMPS	LU	GMRES	LU	LU	GMRES
u_f		S	S	D	S	S		S	S	D	S	S
u_r		D	D	Q	Q	Q		D	D	Q	Q	Q
u_p=u_g		—	D	—	—	D		—	D	—	—	D
ID	Time eq. DMUMPS (s)			Time eq. 10^{-16} (s)			Mem eq. DMUMPS (GB)			Mem eq. 10^{-16} (GB)		
1	265.2	**154.0**	166.5	269.4	**155.9**	168.2	272.0	**138.0**	171.3	272.0	**138.0**	173.5
2	52.7	**31.7**	33.4	53.7	**33.3**	36.3	64.8	**33.1**	46.1	64.8	**33.1**	46.7
3	31.0	**23.1**	25.9	31.5	**24.8**	28.0	28.2	**14.2**	14.9	28.2	**14.2**	15.4
4	44.3	**31.2**	32.8	45.2	**32.7**	35.4	40.9	**20.7**	21.9	40.9	**20.7**	23.0
5	28.2	**22.3**	27.0	29.0	**23.7**	27.5	33.4	**16.9**	19.9	33.4	**16.9**	21.0
6	2.1	**1.7**	3.4	2.4	**2.1**	3.5	1.8	**1.0**	1.3	1.8	**1.0**	1.5
7	13.1	**9.6**	11.0	13.7	**11.1**	11.7	21.9	**11.3**	16.4	21.9	**11.3**	18.2
8	17.2	**10.9**	12.6	17.6	**12.1**	12.7	28.1	**14.3**	21.0	28.1	**14.3**	21.4
9	205.4	**129.3**	144.5	208.5	**136.3**	155.8	135.7	**68.4**	77.8	135.7	**68.4**	79.9
10	91.8	**67.6**	77.9	94.6	**75.0**	79.2	81.6	**41.7**	52.9	81.6	**41.7**	53.7
11	25.3	**15.2**	16.0	26.0	**16.5**	17.7	34.1	**17.3**	25.2	34.1	**17.3**	25.8
12	15.2	**10.7**	12.7	15.9	**12.2**	14.9	19.8	**10.2**	12.5	19.8	**10.2**	13.5
13	284.2	**165.2**	184.7	288.6	**177.9**	201.4	178.0	**89.8**	114.5	178.0	**89.8**	119.7
14	**4.2**	4.4	5.7	**4.7**	8.4	7.9	7.1	**3.7**	4.6	7.1	**3.7**	5.4
15	10.4	**8.4**	10.1	**11.2**	13.6	12.7	18.1	**9.3**	12.4	18.1	**9.3**	14.3
16	294.5	**136.2**	157.9	299.3	180.3	**180.22**	241.0	**121.0**	169.9	241.0	**121.0**	175.1
17	**46.1**	57.5	52.0	**50.6**	235.1	73.1	55.6	**28.9**	38.1	55.6	**28.9**	41.4
18	**5.6**	74.8	16.6	**6.3**	164.3	24.3	6.6	**3.5**	4.4	6.6	**3.5**	4.9
19	114.5	**68.7**	73.8	116.4	**74.0**	79.2	89.9	**45.3**	54.0	89.9	**45.3**	56.1
20	156.7	—	**118.6**	**160.3**	—	179.4	153.0	—	**103.6**	153.0	—	**105.5**
21	**7.5**	—	22.9	**8.0**	—	33.5	8.4	—	**6.7**	8.4	—	**7.3**
22	536.2	**254.5**	269.3	546.9	**271.7**	307.2	312.0	**157.0**	187.5	312.0	**157.0**	188.7
23	67.2	**46.6**	49.0	70.0	**55.1**	59.5	52.9	**26.7**	33.1	52.9	**26.7**	34.5
24	**62.9**	—	109.8	**71.6**	—	170.4	77.6	—	**54.3**	77.6	—	**65.6**
25	97.6	**65.4**	79.8	103.1	**90.2**	92.2	192.0	**97.7**	141.7	192.0	**97.7**	142.3

convergence is not achieved. They also always achieve the lowest memory consumption when they converge, which comes at no surprise because most of the memory is consumed in the factorization step.

Since the GMRES-IR variants with single precision factorization typically require more LU solves to achieve convergence than the LU-IR variants with single precision factorization, they usually have a higher execution time. Their memory consumption is also higher because in our implementation the factors are cast to double precision. These variants, however, generally provide a more robust and reliable solution with respect to the LU-IR (u_f = s) ones. As a result, GMRES-IR variants can solve problems where LU-IR do not achieve convergence. In such cases, for our matrix set, their execution time can be higher than that of variants that employ double precision factorization (DMUMPS or LU-IR with u_f = D and u_r = Q); however their memory footprint usually remains smaller.

Overall, Table 2 shows that the GMRES-IR variants provide a good compromise between performance and robustness: unlike LU-IR (u_f = s), they converge for all matrices in our set, while still achieving a significantly better performance than double precision based factorization variants.

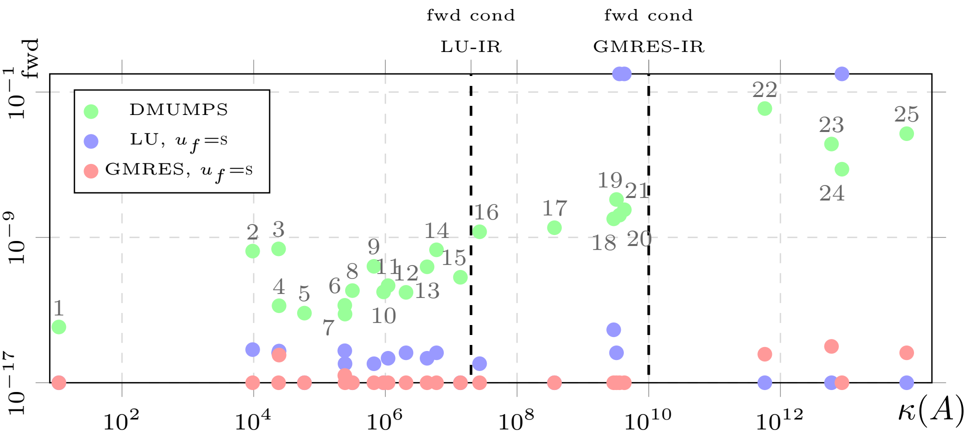

Fig. 1. Forward error achieved by three IR variants for the matrices used in Table 2 (denoted by their ID) as a function of their condition number $\kappa(A)$. We fix $u = u_g = u_p =$ D and $u_r =$ Q. The vertical dashed lines show the forward convergence condition for LU-IR ($u_f =$ S, $u =$ D) and for GMRES-IR ($u_f =$ S, $u = u_g = u_p =$ D).

It is also worth noting that, with respect to variants with $u_r =$ D, variants with $u_r =$ Q can achieve a forward error of order 10^{-16} with only a small additional overhead in both time (because the residual is computed in quadruple rather than double precision and a few more iterations are required) and memory consumption (because the matrix is stored in quadruple precision). As a result, these variants can produce a more accurate solution than a standard double precision direct solver (DMUMPS) with a smaller memory consumption and, in most cases, faster. We illustrate the accuracy improvement in Figure 1, which reports the forward error achieved by variants DMUMPS and LU-IR and GMRES-IR with $u_r =$ Q.

In order to provide more insight into the behavior of each variant, we next carry out a detailed analysis of time and memory consumption in sections 5.3.1 and 5.3.2, respectively.

5.3.1 Detailed execution time analysis. The potential gain in execution time of mixed precision iterative refinement comes from the fact that the most time consuming operation, the LU factorization, is carried in low precision arithmetic and high precision is only used in iterative refinement steps which involve low complexity operations. For this gain to be effective, the cost of the refinement iterations must not exceed the time reduction resulting from running the factorization in low precision. This is very often the case. First of all, on current processors (such as the model used for our experiments) computations in single precision can be up to twice as fast as those in double precision. Additionally, operations performed in the refinement steps have a lower asymptotic complexity compared with the factorization. Nonetheless, in practice, the overall time reduction can vary significantly depending on a number of parameters. First of all, the ratio between the complexity of the factorization and that of the solution phase is less favorable on 2D problems than on 3D problems (see section 2.1). Second, the single precision factorization may be less than twice as fast as the double precision one; this may happen, for example, on small problems where the overhead of symbolic operations in the factorization (data indexing, handling of data structures, etc.) is relatively high or, in a parallel setting, because the single precision factorization is less scalable than the double precision one due to the relatively lower weight of floating-point operations with respect to that of symbolic ones. It must also be noted that although the factorization essentially

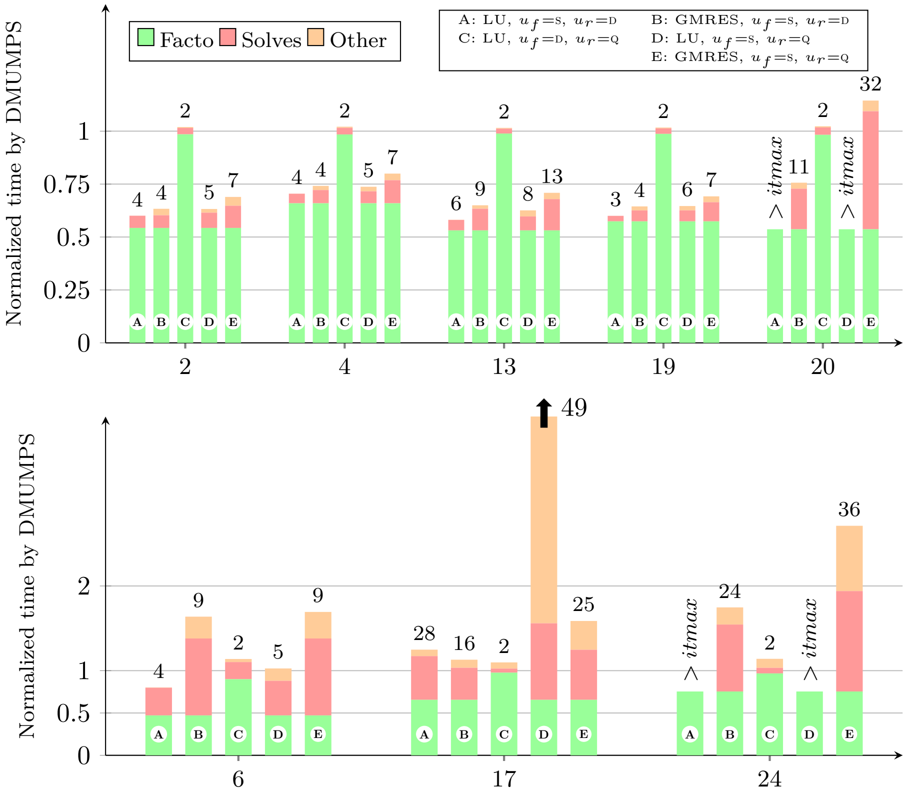

Fig. 2. Execution time for different LU-IR and GMRES-IR variants normalized by that of DMUMPS, for a subset of our test matrices (denoted by their ID on the x-axis). Each bar shows the time breakdown into LU factorization, LU solves, and other computations. We print on top of each bar the total number of calls to LU solves. We fix $u = u_g = u_p =$ D. Variants with u_r = D provide a forward error equivalent to the one obtained with DMUMPS (A and B), while variants with u_r = Q provide a forward error of order 10^{-16} (C, D, and E).

relies on efficient BLAS-3 operations, the operations done in the iterative refinement, in particular the LU solves, rely on memory-bound BLAS-2 operations and are thus less efficient. Finally, in the case of badly conditioned problems, iterative refinement may require numerous iterations to achieve convergence.

Figure 2 shows the execution time of variants encountered in Table 2 normalized with respect to that of DMUMPS for a selected subset of matrices from our test set; each bar also shows the time breakdown into LU factorization, LU solves and all the rest which includes computing the residual and, for the GMRES-based variants, casting the factors, computing the Krylov basis, orthonormalizing it, etc. The values on top of each bar are the number of LU solve operations; note that for GMRES-based variants, multiple LU solve operations are done in each refinement iteration.

In the first row of this figure we report problems that behave well, in the sense that all the parameters mentioned above align in the direction that leads to a good overall time reduction. For these problems the single precision factorization is roughly twice as fast as the double precision one, the complexity of the solve is much lower than that of the factorization (three orders of magnitude in all cases, as reported in Table 1), and relatively few iterations are needed to achieve convergence.

For all these problems, the complexity of the matrix–vector product is more than two orders of magnitude lower than that of the solve (see columns "NNZ" and "Slv." of Table 1). As a result, the computation of the residual only accounts for a small portion of the total execution time—even for variants with $u_r = \textsc{q}$, for which it is carried out in slow quadruple precision arithmetic (which is not supported by our hardware). This is a very desirable property since these variants greatly improve the forward error with only a modest overhead. The figure clearly shows, however, that despite their relatively low complexity, the operations in iterative refinement are relatively slow and, therefore, the gain is considerably reduced when many solves are necessary. This issue is exacerbated in the case of GMRES-IR variants, because the solves are carried out in double instead of single precision as for LU-IR variants ($u_f = \textsc{s}$).

In the second row of Figure 2 we report some cases where mixed precision iterative refinement does not reduce execution time. Matrix 6 is a relatively small 2D problem where the cost of the solve and the matrix–vector product relative to that of the factorization is high; as a result, even for a moderate number of refinement iterations, variant DMUMPS achieves the best execution time and all other variants are much slower. Matrix 17 is one where the single precision factorization is only 1.6 times faster than the double precision one and, additionally, it produces little fill-in (as shown by the small ratio Slv./NNZ in Table 1) and so the relative cost of computing the residual in quadruple precision is high. Finally, matrix 24 is badly conditioned and variants based on single precision factorization either do not converge or require so many iterations that the execution time is higher than that of DMUMPS. It is however worth noting that on these particular matrices variants based on single precision factorization may be slower than DMUMPS but at a significantly reduced memory cost (as shown in Table 2).

5.3.2 Detailed memory consumption analysis. One distinctive feature of the multifrontal method in comparison with left or right-looking ones is in the way it uses memory. In addition to the memory needed to store the factors which grows monotonically throughout the factorization, the multifrontal method also needs a temporary workspace which we refer to as *active memory*. As a result, the peak memory consumption achieved in the course of the factorization is generally higher than the memory needed to store the factors. It must also be noted that parallelism does not have any effect on the memory needed to store the factors but generally increases the size of the active memory: this is because more temporary data is generated at the same time. For a thorough discussion of the memory consumption in the multifrontal method we refer the reader to the paper by Agullo et al. [1].

In the rest of this article we refer to the difference between the peak memory consumption and the size of factors as *active memory overhead*. We assume that at the end of the factorization all the active memory is freed and only the factors are left in memory. It is only at this moment that the original problem matrix is cast to quadruple precision for computing the residual at each refinement iteration. Therefore, the active memory overhead and the memory required to store a quadruple precision version of the matrix do not accumulate. In our implementation, the GMRES-IR variants with $u_p = u^2 = \textsc{d}$ also require the factors to be cast to double precision which we do upon completion of the factorization, when the active memory is freed. We also report the size of the Krylov basis in the GMRES solver: although in most of our experiments this is completely negligible, there might be cases (we will show one) where the number of GMRES iterations is sufficiently high to make the memory consumed by the Krylov basis relevant. Finally, we do not count the memory consumption of the solution, residual and correction vectors.

All these assumptions lead us to Figure 3 where we present the normalized memory consumption of certain LU-IR and GMRES-IR variants relative to that of variant DMUMPS for a selected subset of problems. We do not include variants using $u_r = \textsc{d}$ because they behave very similarly to variants

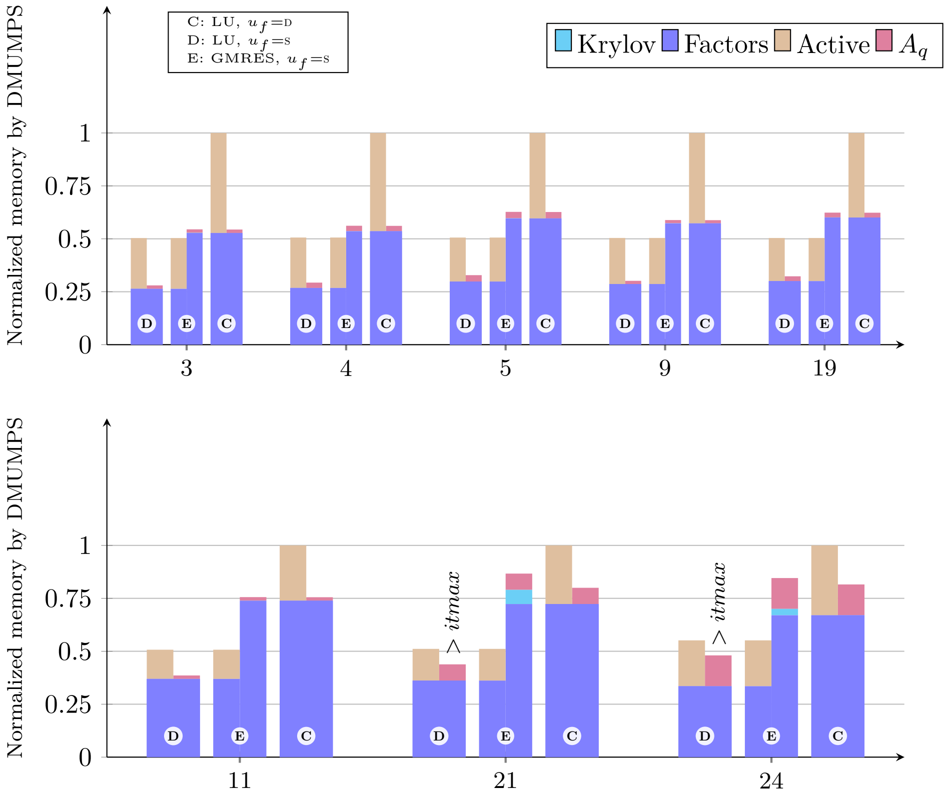

Fig. 3. Memory consumption for different LU-IR and GMRES-IR variants normalized by that of DMUMPS, for a subset of our test matrices (denoted by their ID on the x-axis). The bars show the memory breakdown in factors memory, active memory overhead, the storage for the Krylov basis (GMRES-IR variant only), and the storage for the matrix in quadruple precision. Each variant bar is split into two subbars corresponding to the peak consumption during the factorization and solve phases, respectively (and thus the overall required memory by the variant is the maximum of the two peaks). We fix $u = u_g = u_p =$ D and $u_r =$ Q. All the variants (C, D, and E) provide a forward error of order 10^{-16}.

with $u_r =$ Q. For each problem and variant the bar is split in two parts showing the memory consumption during and after the factorization, respectively.

In the first row we report problems that behave well, which corresponds to the most common case as shown in Table 2. It shows, as expected, that LU-IR with single precision factorization consumes half as much memory as DMUMPS because the memory needed to store the problem matrix in quadruple precision does not exceed the active memory overhead. Thus, the highest memory consumption corresponds to the single precision factorization peak. GMRES-based variant ($u_p =$ D) casts the factors to double precision which exceeds the peak of the single precision factorization. Nonetheless, even if on top of this we have to add the memory needed to store the quadruple precision matrix, the overall consumption is lower than the double precision factorization peak by a factor which can be almost up to 50% on this set, making the memory consumption of the GMRES-IR variant almost identical to that of the LU-IR one in a few cases (such as matrices 3 and 4). As for the LU-IR variant with $u_f =$ D, it clearly does not bring any improvement with respect to

DMUMPS but no loss either because the memory for storing the matrix in quadruple precision is much lower than the active memory overhead.

In the second row of Figure 3 we report problems where the memory reduction is not as good, for four different reasons, the last two of which are exclusive to GMRES-IR.

(1) In the case of matrix 24, the single precision factorization consumes more than half the memory of the double precision one (about 55%). This is because the relative weight of the symbolic data structures, which is counted as part of the active memory overhead and does not depend on the factorization precision, is high for this matrix.
(2) In the case of matrices 21 and 24 the factorization generates little fill-in which makes the relative weight of the quadruple precision matrix copy significant compared with the size of the factors. Here this storage remains less than the active memory overhead and so the overall memory consumption of LU-IR is not impacted; however, it does impact GMRES-IR, leading to less memory savings.
(3) In the case of matrices 11 and 21 (and to a lesser extent 24) the active memory overhead represents a smaller fraction of the total memory, further reducing the memory savings for GMRES-IR.
(4) Finally, matrix 21 (and to a lesser extent 24) is one of the few matrices having a non-negligible Krylov basis memory footprint, showing that an increase in the number iterations for the GMRES to converge diminishes here the potential memory gain.

5.4 Performance of LU-IR and GMRES-IR using approximate factorizations

In this set of experiments we are interested in studying the performance of LU-IR and GMRES-IR while using BLR, static pivoting, and BLR with static pivoting. For each experiment, we use a selected set of matrices from Table 1 which are representative of different types of behavior that can be encountered in practice.

These approximation techniques have two conflicting effects on the performance: if, on the one hand, they reduce the time and memory of the factorization, on the other hand, they increase the number of refinement iterations.

5.4.1 BLR factorization. In Table 3 we present the execution time and memory consumption of four iterative refinement variants using low rank BLR factorization for different values of the compression threshold τ_b. All variants provide a forward error on the solution equivalent to the one of DMUMPS. If τ_b = "full-rank", the factorization is run without BLR, this is a standard factorization as in section 5.3. It should be noted that in this case the double precision factorization LU-IR variant is equivalent to DMUMPS and we will refer to it as DMUMPS in the text. We denote by "—" the cases where convergence is not reached and, for each matrix, the best execution time and memory consumption is highlighted in bold. We choose to work with the default BLR settings of MUMPS, in which case the data in the active memory is not compressed with low rank approximations. It should be noted, however, that MUMPS also offers the possibility to compress the data in the active memory; this would result in additional memory savings, but will badly affect the execution time because it adds the compression overhead without reducing the operational complexity.

The experimental results of Table 3 are in good agreement with the theoretical convergence conditions of Theorem 4.1 developed in section 4. We can clearly see how the robustness of the presented variants is related to both the condition number $\kappa(A)$ of the matrix (specified for each matrix in Table 1) and the BLR threshold τ_b. Convergence is not achieved for excessively large values of the BLR threshold; the largest τ_b value for which convergence is achieved depends on the matrix condition number and, in general, it is smaller for badly conditioned problems. In the case

Table 3. Execution time (in seconds), memory consumption (in GBytes) and number of LU solve calls of IR variants for a subset of the matrices listed in Table 1 and depending on the compression threshold τ_b. We fix $u_r = u =$ D.

	Solver	LU	LU	GMRES	GMRES	LU	LU	GMRES	GMRES	LU	LU	GMRES	GMRES
	u_f	D	S	S	S	D	S	S	S	D	S	S	S
	u_p=u_g	—	—	D	S	—	—	D	S	—	—	D	S
ID	τ_b	Time (s)				Memory (GB)				Nb LU solves			
1	full-rank	265.2	154.0	166.5	163.3	272.0	138.0	171.3	138.0	1	3	5	7
	1E-10	101.0	70.5	73.7	72.7	158.0	84.5	84.6	84.6	2	3	5	7
	1E-08	93.1	70.0	72.8	72.1	157.0	80.6	82.2	82.2	2	3	5	7
	1E-06	91.1	**64.9**	68.2	68.5	149.0	77.8	79.6	79.6	3	3	6	10
	1E-04	88.3	66.3	69.3	70.8	143.0	73.6	77.0	77.0	4	4	7	13
	1E-02	89.8	71.4	75.0	128.1	147.0	73.6	73.6	73.6	9	9	11	125
	1E-01	97.8	73.6	81.0	119.5	147.0	**71.8**	73.4	73.4	19	18	24	115
	5E-01	130.1	92.3	87.1	130.0	139.0	72.8	74.6	74.6	63	58	36	141
16	full-rank	294.5	136.2	157.9	176.3	241.0	121.0	169.9	121.0	1	7	15	56
	1E-10	232.9	158.8	174.0	181.2	188.0	118.0	169.9	118.0	2	7	16	55
	1E-08	204.9	149.7	165.3	182.7	171.0	114.0	161.9	114.0	3	7	17	79
	1E-06	179.0	**88.3**	98.8	105.5	154.0	82.4	93.8	82.8	5	7	16	54
	1E-04	—	—	105.6	116.3	—	—	**70.9**	**70.9**	—	—	69	181
17	full-rank	46.1	57.5	52.0	110.0	55.6	28.9	38.1	28.9	1	28	16	92
	1E-10	**32.9**	40.3	36.9	83.1	38.7	20.5	25.6	20.5	2	22	15	94
	1E-08	33.6	41.5	37.3	88.0	37.1	19.7	22.8	19.7	4	26	16	107
	1E-06	—	—	40.9	187.8	—	—	20.0	**18.6**	—	—	25	280
	1E-04	—	—	658.3	—	—	—	36.6	—	—	—	949	—
	1E-02	—	—	2224.1	—	—	—	98.1	—	—	—	3338	—
22	full-rank	536.2	254.5	269.3	353.6	312.0	157.0	187.5	157.0	1	4	5	46
	1E-10	313.3	199.8	210.0	230.9	240.0	141.0	147.6	144.0	2	4	7	37
	1E-08	260.2	119.2	130.1	162.3	218.0	112.0	116.0	116.0	3	4	9	60
	1E-06	223.2	100.4	110.1	131.3	199.0	107.0	107.0	107.0	4	4	9	47
	1E-04	212.3	**95.8**	105.4	124.7	200.0	101.0	103.0	103.0	22	20	19	65
	1E-02	—	—	482.6	1111.0	—	—	**96.8**	**96.8**	—	—	367	1763
24	full-rank	**62.9**	—	109.8	—	77.6	—	54.3	—	1	—	24	—
	1E-10	63.3	—	90.8	—	65.5	—	44.0	—	3	—	23	—
	1E-08	68.9	—	91.3	—	64.8	—	**41.8**	—	6	—	23	—
	1E-06	—	—	299.4	—	—	—	55.8	—	—	—	140	—
25	full-rank	97.6	65.4	79.8	79.6	192.0	97.7	141.7	97.7	1	4	7	10
	1E-10	88.9	63.7	75.8	69.5	137.0	70.9	110.5	71.0	2	4	7	7
	1E-08	81.3	**59.5**	66.1	66.7	131.0	67.5	92.1	67.6	3	4	7	7
	1E-06	85.1	61.4	65.6	70.8	118.0	61.4	70.4	61.5	8	8	9	13
	1E-04	—	—	147.5	131.4	—	—	53.7	53.7	—	—	53	48
	1E-02	—	—	1043.9	2380.8	—	—	45.5	45.5	—	—	523	1259
	1E-01	—	—	3340.5	3155.2	—	—	48.9	**43.7**	—	—	1399	1649
	5E-01	—	—	2746.0	3932.7	—	—	49.1	**43.7**	—	—	1403	2094

of GMRES-IR variants, which are more robust, the BLR threshold can be pushed to larger values without breaking convergence.

The use of BLR generally results in substantial reductions of the execution time. As the BLR threshold increases, the operational complexity and, consequently, the execution time of the factorization and solve operations decreases; conversely, the number of iterations increases up to the point where convergence may not be achieved anymore. The optimal BLR threshold value which delivers the lowest execution time obviously depends on the problem. It must be noted that even though the GMRES-IR variants achieve convergence for larger τ_b values, this leads to an excessive number of iterations whose cost exceeds the improvement provided by BLR; as a result, these variants are slower than LU-IR ones (u_f = S but also u_f = D) in all cases. Consequently, the single precision factorization LU-IR variant generally achieves the best execution time on this set of problems, similarly to what was observed in section 5.3, with a few exceptions. On matrix 17 the double precision factorization LU-IR variant is the best due to the fact that similarly to the full rank case (see section 5.3.1) the BLR factorization is less than twice as fast when single precision is used instead of double for the same τ_b value; additionally, a substantial number of iterations is needed to achieve convergence. It is worth mentioning that on this matrix the GMRES-IR variant with $u_g = u_p$ = D is faster than the single precision factorization LU-IR variant (36.9s versus 40.3s) and consumes less memory than the double precision factorization LU-IR variant (20.0GB versus 37.1GB). On matrix 24, DMUMPS is the fastest variant as in the full rank case; this is due to the fact that, on this problem, BLR does not achieve a good reduction of the operational complexity and, therefore, of the execution time.

As for the storage, the use of BLR leads to a different outcome with respect to the case where a full-rank factorization is used (see section 5.3) where the single precision factorization LU-IR variant is the best. This is due to the combination of two factors. First, when BLR is used, the relative weight of the active memory is higher because it corresponds to data which is not compressed due to the choice of parameters we have made; consequently, the memory consumption peak is often reached during the factorization rather than during the iterative refinement. Second, the memory consumption of the factorization decreases monotonically when the BLR threshold is increased. As a result of these two effects, the GMRES-IR variants achieve the lowest memory consumption on this set of problems, because they can preserve convergence for larger values of τ_b than the LU-IR variants can. For example, on matrix 16 the GMRES-IR variant with $u_g = u_p$ = D consumes almost 15% less memory than the LU-IR one with u_f = S (70.9GB versus 82.4GB), on matrix 25 the GMRES-IR variant with $u_g = u_p$ = D consumes almost 30% less memory than variant LU-IR with u_f = S (43.7GB versus 61.4GB), and on matrix 24 the GMRES-IR variant with $u_g = u_p$ = D consumes more than 35% less memory than variant LU-IR with u_f = D (41.8GB versus 64.8GB). It is worth pointing out that the value of τ_b for which GMRES-IR achieves the lowest possible memory consumption is not always the largest value for which convergence is still possible. This is because for a large number of iterations the memory needed to store the Krylov basis may exceed the savings obtained with BLR. This problem can be overcome or mitigated by choosing an appropriate value for the τ_g threshold or, similarly, using a restarted GMRES method; we leave this analysis for future work.

We finally compare the two GMRES-IR variants $u_g = u_p$ = D and $u_g = u_p$ = S. When $u_g = u_p$ = S, GMRES-IR avoids the cast of the LU factors from single to double precision, and thus reduces memory consumption compared with $u_g = u_p$ = D. However, as explained above, the relative weight of the factors with respect to the active memory is smaller as τ_b increases, and so the reduction achieved by GMRES-IR with $u_g = u_p$ = S grows smaller until the point where both variants achieve a similar memory consumption. On our matrix set, for the values of τ_b where the LU-IR with u_f =

Table 4. Execution time (in seconds) and memory consumption (in GBytes) of IR variants for a subset of the matrices listed in Table 1 and depending on the perturbation τ_s. $\underline{\rho_n} = \max\{\max|L|, \max|U|\}/\max|A|$ is a lower bound of the growth factor. We fix $u_r = u =$ D.

	Solver	LU	LU	GMRES	LU	LU	GMRES	
	u_f	D	S	S	D	S	S	
	u_p=u_g	—	—	D	—	—	D	
ID	τ_s	Time (s)			Nb LU solves			$\underline{\rho_n}$
16	partial	294.5	136.2	157.9	1	7	15	4E2
	1E-10	258.1	**121.0**	141.6	2	9	16	2E4
	1E-08	258.1	**121.0**	141.5	2	9	16	2E4
	1E-06	258.1	**121.0**	144.7	2	9	18	2E4
	1E-04	—	—	1659.9	—	—	985	4E3
22	partial	536.2	**254.5**	269.3	1	4	5	5E4
	1E-10	—	—	—	—	—	—	2E9
	1E-08	508.0	—	—	7	—	—	2E7
	1E-06	490.3	—	—	3	—	—	2E5
	1E-04	499.2	—	773.2	5	—	124	2E3
	1E-02	1501.5	780.3	484.9	231	233	59	5E3

S does not converge, GMRES-IR with $u_g = u_p =$ S does not achieve significant memory reductions compared with GMRES-IR with $u_g = u_p =$ D (at best 7% on matrix 17, 18.6GB versus 20.0GB).

5.4.2 Static pivoting factorization. We now turn our attention to the use of static pivoting. We report in Table 4 the execution time and memory consumption of three iterative refinement variants for different values of the static pivoting threshold τ_s. All variants are stopped when they reach a forward error on the solution equivalent to the one of DMUMPS. If τ_s = "partial", the factorization is run in standard MUMPS threshold partial pivoting. It should be noted that in this case the double precision factorization LU-IR variant is equivalent to DMUMPS.

Once again the observed convergence behaviors are in good agreement with Theorem 4.1 as explained below. In the case of static pivoting, the execution time of the factorization does not depend on τ_s; in order to minimize the overall solution time, the goal is therefore to achieve the fastest possible convergence. This is a complex issue: a smaller perturbation τ_s does not always mean a faster convergence, because the value of τ_s also directly impacts the growth factor ρ_n. Thus, there is an optimal value of τ_s, which is clearly problem dependent, that leads to the fastest convergence by balancing the $u_f \rho_n$ and τ_s terms in the convergence condition. To confirm this, Table 4 reports $\underline{\rho_n}$, a lower bound on the true growth factor, that can be used as a cheap, but rough indicator of the behavior of ρ_n (the true ρ_n would be extremely expensive to compute for such large matrices). There is a clear trend of $\underline{\rho_n}$ decreasing as τ_s increases, which explains, for example, why on matrix 22 convergence is achieved for large τ_s. For many matrices in our set, such as matrix 16 in Table 4, static pivoting slightly accelerates the factorization without excessively deteriorating the convergence, and so allows for modest time gains overall. However, for some matrices such as matrix 22, static pivoting requires many iterations and can lead to significant slowdowns compared with partial pivoting. It is however interesting to note that, on matrix 22, if the use of partial pivoting is impossible (for instance because the available solver does not support it), the GMRES-IR variant provides the best overall execution time.

Table 5. Execution time (in seconds) and memory consumption (in GBytes) of IR variants for a subset of the matrices listed in Table 1 and depending on the perturbation τ_b for a fixed τ_s. The chosen τ_s is specified for each matrices. We fix $u_r = u =$ D.

		Solver	LU	LU	GMRES	LU	LU	GMRES
		u_f	D	S	S	D	S	S
		u_p=u_g	—	—	D	—	—	D
ID	τ_s	τ_b	Time (s)			Nb LU solves		
16	partial	1E-10	232.9	158.8	174.0	2	7	16
		1E-08	204.9	149.7	165.3	3	7	17
		1E-06	179.0	**88.3**	98.8	5	7	16
		1E-04	—	—	105.6	—	—	69
	10^{-8}	1E-10	196.9	139.9	152.2	2	9	17
		1E-08	181.9	133.7	149.8	3	9	21
		1E-06	137.9	**70.1**	80.0	6	12	18
		1E-04	—	—	90.9	—	—	71
22	partial	1E-10	313.3	199.8	210.0	2	4	7
		1E-08	260.2	119.2	130.1	3	4	9
		1E-06	223.2	100.4	110.1	4	4	9
		1E-04	212.3	**95.8**	105.4	22	20	19
		1E-02	—	—	482.6	—	—	367
	10^{-2}	1E-10	592.9	353.8	218.2	231	233	59
		1E-08	525.8	266.6	163.6	231	233	59
		1E-06	456.1	247.3	138.1	231	233	59
		1E-04	404.6	212.8	**123.1**	238	238	63
		1E-02	—	—	879.1	—	—	838
	10^{-6}	1E-10	253.5	—	—	3	—	—
		1E-08	200.2	—	—	3	—	—
		1E-06	**157.2**	—	—	4	—	—
		1E-04	166.4	—	—	33	—	—
		1E-02	—	—	—	—	—	—

5.4.3 BLR factorization with static pivoting. Finally in Table 5 we present the execution time and memory consumption of three iterative refinement variants (the same as in section 5.4.2) for different values of the BLR compression threshold τ_b and a fixed value of the static pivoting perturbation τ_s. All variants are stopped when they reach a forward error on the solution equivalent to the one of DMUMPS. If τ_s = partial, the factorization is run in standard MUMPS threshold partial pivoting and the results are then equivalent to the BLR results of section 5.4.1.

Theorem 4.1 applied to the case where BLR and static pivoting are used together states that the convergence conditions should be affected by the largest perturbations $\max(\tau_s, \tau_b)$ and the term $\rho_n u_f$ which depends on the growth factor. Our experiments confirm this: values of τ_s or τ_b for which a given variant was not converging with BLR or static pivoting alone still do not converge when they are combined, and, conversely, variants that were converging for BLR and static pivoting alone still converge when these two approximations are used together. Matrix 22 with $\tau_s = 10^{-6}$ illustrates an interesting point of the error bound $\max(\tau_s, \tau_b) + \rho_n u_f$: convergence is only achieved

Table 6. Best execution time and memory consumption improvements in comparison to DMUMPS amongst all the presented IR variants (full-rank, BLR, static pivoting, and BLR with static pivoting) for the industrial partners matrices (bold in Table 1) and matrix 13.

ID	DMUMPS		Best in time		Best in memory	
	Time (s)	Memory (s)	Time (s)	Memory (s)	Time (s)	Memory (s)
1	265.2	272.0	51.0 (5.2×)	73.0 (3.7×)	80.7 (3.3×)	71.2 (3.8×)
10	91.8	81.6	37.8 (2.4×)	26.9 (3.0×)	471.3 (0.2×)	22.4 (3.6×)
13	284.2	178.0	60.1 (4.7×)	50.7 (3.5×)	117.5 (2.4×)	50.3 (3.5×)
16	294.5	241.0	70.1 (4.2×)	82.4 (2.9×)	105.6 (2.8×)	70.9 (3.4×)
17	46.1	55.6	30.8 (1.5×)	38.6 (1.4×)	187.8 (0.2×)	18.6 (3.0×)
20	156.7	153.0	56.0 (2.8×)	41.9 (3.7×)	125.3 (1.3×)	39.0 (3.9×)
22	536.2	312.0	95.8 (5.6×)	101.0 (3.1×)	879.1 (0.6×)	91.6 (3.4×)
24	62.9	77.6	62.9 (1.0×)	77.6 (1.0×)	91.3 (0.7×)	41.8 (1.9×)
25	97.6	192.0	50.7 (1.9×)	67.5 (2.8×)	3155.2 (0.0×)	43.7 (4.4×)

for variant A that uses a double precision factorization (u_f = D), even for values of τ_b that are much larger than the unit roundoff of single precision. This shows that the rule of thumb that the factorizaton precision should be chosen as low as possible as long as $u_f \leq \tau_b$ is not true in presence of large element growth, since a smaller value of u_f can be beneficial to absorb a particularly large ρ_n.

While the reductions in execution time obtained by using static pivoting instead of partial pivoting were modest for the full-rank factorization, they are larger for the BLR factorization. Taking matrix 16 as an example, in full-rank the single precision factorization LU-IR variant is only 1.12 (136s/121s) times faster after the activation of the static pivoting (see Table 4), whereas in BLR it is 1.26 (88s/70s) times faster than (see Table 3). These better reductions are explained by the fact that in the BLR factorization, static pivoting also allows the panel reduction operation to be processed with low-rank operations [7], which leads to a reduction of flops and thus a faster execution time.

5.5 Performance summary

To summarize the results presented in the previous sections, we report in Table 6 the best execution time and memory consumption amongst all the previously reviewed iterative refinement variants, for the industrial partners matrices and matrix 13. All variants are stopped when they reach a forward error on the solution equivalent to the one of DMUMPS.

We obtain at best on matrix 22 a reduction of 5.6× in time and on matrix 25 a reduction of 4.4× in memory. A greater variability is observed in the speedup with respect to the memory gains. This is because numerous parameters affect the execution time which are related to the numerical properties of the problems as well as to the features of the computer architecture; in some extreme cases (such as matrix 24) no speedup is observed at all. As the best memory saving is sometimes obtained for aggressive values of the BLR threshold, the execution time can be deteriorated due to a high number of iterations. We however note that a balance between the two use cases can be struck to obtain large memory savings while keeping a reasonable execution time: taking matrix 25 as an example, we can achieve a 3.6× memory reduction (compared with the 2.8× reduction of the "best in time" variant) while only leading to a 0.7× slowdown (compared with the 0.03× slowdown of the "best in memory" variant).

6 CONCLUSIONS

We have evaluated the potential of mixed precision iterative refinement to improve the solution of large sparse systems with direct methods. Compared with dense systems, sparse ones present some challenges but also, as we have explained, some specific features that make the use of iterative refinement especially attractive. In particular, the fact that the LU factors are much denser than the original matrix makes the computation of the residual in quadruple precision arithmetic affordable, and leads to potentially large memory savings compared with a full precision solver. Moreover, iterative refinement can remedy potential instabilities in the factorization, which modern sparse solvers often introduce by using numerical approximations or relaxed pivoting strategies. In this study we have therefore sought to combine recently proposed iterative refinement variants using up to five precisions with state-of-the-art approximate sparse factorizations employing low-rank approximations and static pivoting.

From a theoretical point of view, the standard convergence bounds for LU-IR and GMRES-IR, obtained for a stable factorization with partial pivoting, needed to be adapted. We derived, in Theorem 4.1, new bounds that take into account numerical approximations in the factorization as well as a possibly large element growth due to relaxed pivoting. These bounds better correspond to the typical use of sparse solvers and we have observed them to be in good accordance with the experimental behavior, at least in the case of BLR approximations, static pivoting, and their combination.

We then provided an extensive experimental study of several iterative refinement variants combined with different types of approximate factorization using the multifrontal solver MUMPS. Our experiments demonstrate the potential of mixed precision arithmetic to reduce the execution time and memory consumption of sparse direct solvers, and shed light on important features of these methods. In particular, we have shown that LU-IR with a standard factorization in single precision is able to reduce the time and memory by up to 2× compared with a double precision solver. We have found GMRES-IR to usually be more expensive, but also more robust, which allows it to converge on very ill-conditioned problems and still achieve gains in memory, and sometimes even in time. Moreover, we have combined single precision arithmetic with BLR and static pivoting and analyzed how the convergence of iterative refinement depends on their threshold parameters. Overall, compared with the double precision solver, we have obtained reductions of up to 5.6× in time and 4.4× in memory all while preserving double precision accuracy. Moreover, we have shown that memory consumption can be even further reduced at the expense of time, by using GMRES-IR with more aggressive approximations.

These results open up promising avenues of research as half precision arithmetic becomes progressively available in hardware and supported by compilers.

ACKNOWLEDGMENTS

We thank our industrial partners and the EoCoE project for providing some of the test problems. All the experiments were performed on the Olympe supercomputer of the CALMIP center (project P0989).

REFERENCES

[1] Emmanuel Agullo, Patrick R. Amestoy, Alfredo Buttari, Abdou Guermouche, Jean-Yves L'Excellent, and François-Henry Rouet. 2016. Robust Memory-Aware Mappings for Parallel Multifrontal Factorizations. *SIAM J. Sci. Comput.* 38, 3 (2016), C256–C279. https://doi.org/10.1137/130938505 arXiv:https://doi.org/10.1137/130938505

[2] Hussam Al Daas, Laura Grigori, Pierre Jolivet, and Pierre-Henri Tournier. 2021. A Multilevel Schwarz Preconditioner Based on a Hierarchy of Robust Coarse Spaces. *SIAM J. Sci. Comput.* 43, 3 (2021), A1907–A1928. https://github.com/prj-/aldaas2019multi

[3] Patrick Amestoy, Alfredo Buttari, Nicholas J. Higham, Jean-Yves L'Excellent, Theo Mary, and Bastien Vieublé. 2021. *Five-Precision GMRES-based Iterative Refinement.* MIMS EPrint 2021.5. Manchester Institute for Mathematical Sciences, The University of Manchester, UK. 21 pages. http://eprints.maths.manchester.ac.uk/id/eprint/2807

[4] Patrick R. Amestoy, Cleve Ashcraft, Olivier Boiteau, Alfredo Buttari, Jean-Yves L'Excellent, and Clément Weisbecker. 2015. Improving Multifrontal Methods by Means of Block Low-Rank Representations. *SIAM J. Sci. Comput.* 37, 3 (2015), A1451–A1474. https://doi.org/10.1137/120903476 arXiv:https://doi.org/10.1137/120903476

[5] Patrick R. Amestoy, Romain Brossier, Alfredo Buttari, Jean-Yves L'Excellent, Théo Mary, Ludovic Métivier, Alain Miniussi, and Stéphane Operto. 2016. Fast 3D frequency-domain full waveform inversion with a parallel Block Low-Rank multifrontal direct solver: application to OBC data from the North Sea. *Geophysics* 81, 6 (2016), R363 – R383.

[6] Patrick R. Amestoy, Alfredo Buttari, Jean-Yves L'Excellent, and Theo Mary. 2017. On the Complexity of the Block Low-Rank Multifrontal Factorization. *SIAM J. Sci. Comput.* 39, 4 (2017), A1710–A1740. https://doi.org/10.1137/16M1077192 arXiv:https://doi.org/10.1137/16M1077192

[7] Patrick R. Amestoy, Alfredo Buttari, Jean-Yves L'Excellent, and Theo Mary. 2019. Performance and Scalability of the Block Low-Rank Multifrontal Factorization on Multicore Architectures. *ACM Trans. Math. Software* 45 (2019), 2:1–2:26. Issue 1.

[8] Patrick R. Amestoy, Iain S. Duff, J. Koster, and Jean-Yves L'Excellent. 2001. A Fully Asynchronous Multifrontal Solver Using Distributed Dynamic Scheduling. *SIAM J. Matrix Anal. Appl.* 23, 1 (2001), 15–41.

[9] Patrick R. Amestoy, Iain S. Duff, Jean-Yves L'Excellent, and Jacko Koster. 2001. A Fully Asynchronous Multifrontal Solver Using Distributed Dynamic Scheduling. *SIAM J. Matrix Anal. Appl.* 23, 1 (2001), 15–41. https://doi.org/10.1137/S0895479899358194 arXiv:https://doi.org/10.1137/S0895479899358194

[10] Bernard Van Antwerpen, Yves Detandt, Diego Copiello, Eveline Rosseel, and Eloi Gaudry. 2014. *Performance improvements and new solution strategies of Actran/TM for nacelle simulations.* https://doi.org/10.2514/6.2014-2315 arXiv:https://arc.aiaa.org/doi/pdf/10.2514/6.2014-2315

[11] Hartwig Anzt, Jack Dongarra, Goran Flegar, Nicholas J. Higham, and Enrique S. Quintana-Ortí. 2019. Adaptive Precision in Block-Jacobi Preconditioning for Iterative Sparse Linear System Solvers. *Concurrency Computat. Pract. Exper.* 31, 6 (2019), e4460. https://doi.org/10.1002/cpe.4460

[12] Mario Arioli, Iain S. Duff, Serge Gratton, and Stephane Pralet. 2007. A Note on GMRES Preconditioned by a Perturbed LDL^T Decomposition with Static Pivoting. *SIAM J. Sci. Comput.* 29, 5 (2007), 2024–2044. https://doi.org/10.1137/060661545 arXiv:https://doi.org/10.1137/060661545

[13] 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 (2009), 2526–2533. https://doi.org/10.1016/j.cpc.2008.11.005

[14] 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. Softw.* 34, 4, Article 17 (July 2008), 22 pages. https://doi.org/10.1145/1377596.1377597

[15] Alfredo Buttari, Jack Dongarra, Julie Langou, Julien Langou, Piotr Luszczek, and Jakub Kurzak. 2007. Mixed Precision Iterative Refinement Techniques for the Solution of Dense Linear Systems. *Int. J. High Perform. Comput. Appl.* 21 (11 2007). https://doi.org/10.1177/1094342007084026

[16] 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 (2017), A2834–A2856. https://doi.org/10.1137/17M1122918

[17] 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 (2018), A817–A847. https://doi.org/10.1137/17M1140819

[18] Michael P. Connolly, Nicholas J. Higham, and Theo Mary. 2021. Stochastic Rounding and Its Probabilistic Backward Error Analysis. *SIAM J. Sci. Comput.* 43, 1 (Jan. 2021), A566–A585. https://doi.org/10.1137/20m1334796

[19] Timothy A. Davis and Yifan Hu. 2011. The University of Florida Sparse Matrix Collection. *ACM Trans. Math. Software* 38, 1 (Dec. 2011), 1:1–1:25. https://doi.org/10.1145/2049662.2049663

[20] James W. Demmel, Stanley C. Eisenstat, John R. Gilbert, Xiaoye S. Li, and Joseph W. H. Liu. 1999. A Supernodal Approach to Sparse Partial Pivoting. *SIAM J. Matrix Anal. Appl.* 20, 3 (1999), 720–755. https://doi.org/10.1137/S0895479895291765 arXiv:https://doi.org/10.1137/S0895479895291765

[21] Iain S Duff, Albert Maurice Erisman, and John Ker Reid. 1986. *Direct methods for sparse matrices.* Oxford University Press.

[22] Iain S. Duff and Stéphane Pralet. 2007. Towards Stable Mixed Pivoting Strategies for the Sequential and Parallel Solution of Sparse Symmetric Indefinite Systems. *SIAM J. Matrix Anal. Appl.* 29, 3 (2007), 1007–1024. https://doi.org/10.1137/050629598 arXiv:https://doi.org/10.1137/050629598

[23] Iain S. Duff and John K. Reid. 1983. The multifrontal solution of indefinite sparse symmetric linear systems. *ACM Trans. Math. Software* 9 (1983), 302–325.

[24] J. Alan George. 1973. Nested dissection of a regular finite-element mesh. *SIAM J. Numer. Anal.* 10, 2 (1973), 345–363.

[25] Pieter Ghysels, Xiaoye S. Li, François-Henry Rouet, Samuel Williams, and Artem Napov. 2016. An Efficient Multicore Implementation of a Novel HSS-Structured Multifrontal Solver Using Randomized Sampling. *SIAM J. Sci. Comput.* 38, 5 (2016), S358–S384. https://doi.org/10.1137/15M1010117

[26] Azzam Haidar, Ahmad Abdelfattah, Mawussi Zounon, Panruo Wu, Srikara Pranesh, Stanimire Tomov, and Jack Dongarra. 2018. The Design of Fast and Energy-Efficient Linear Solvers: On the Potential of Half-Precision Arithmetic and Iterative Refinement Techniques. In *Computational Science—ICCS* 2018, Yong Shi, Haohuan Fu, Yingjie Tian, Valeria V. Krzhizhanovskaya, Michael Harold Lees, Jack Dongarra, and Peter M. A. Sloot (Eds.). Springer, Cham, Switzerland, 586–600. https://doi.org/10.1007/978-3-319-93698-7_45

[27] 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

[28] 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 *Proceedings of the 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

[29] Nicholas J. Higham. 2002. *Accuracy and Stability of Numerical Algorithms* (second ed.). Society for Industrial and Applied Mathematics, Philadelphia, PA, USA. xxx+680 pages. https://doi.org/10.1137/1.9780898718027

[30] Nicholas J. Higham and Theo Mary. 2019. A New Approach to Probabilistic Rounding Error Analysis. *SIAM J. Sci. Comput.* 41, 5 (2019), A2815–A2835. https://doi.org/10.1137/18M1226312

[31] Nicholas J. Higham and Theo Mary. 2020. Sharper Probabilistic Backward Error Analysis for Basic Linear Algebra Kernels with Random Data. *SIAM J. Sci. Comput.* 42, 5 (2020), A3427–A3446. https://doi.org/10.1137/20M1314355

[32] Nicholas J. Higham and Theo Mary. 2020. Solving Block Low-Rank Linear Systems by LU Factorization is Numerically Stable. *IMA J. Numer. Anal.* (2020), 1–30. https://doi.org/10.1093/imanum/drab020

[33] Nicholas J. Higham and Theo Mary. 2021. *Mixed Precision Algorithms in Numerical Linear Algebra.* MIMS EPrint 2021.20. Manchester Institute for Mathematical Sciences, The University of Manchester, UK. 66 pages. http://eprints.maths.manchester.ac.uk/2841/ To appear in Acta Numerica.

[34] Pascal Hénon, Pierre Ramet, and Jean Roman. 1999. A Mapping and Scheduling Algorithm for Parallel Sparse Fan-In Numerical Factorization. 1059–1067. https://doi.org/10.1007/3-540-48311-X_148

[35] M. Jankowski and H. Woźniakowski. 1977. Iterative Refinement Implies Numerical Stability. *BIT* 17 (1977), 303–311. https://doi.org/10.1007/BF01932150

[36] George Karypis. 2013. METIS – *A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices – Version 5.1.0.* University of Minnesota.

[37] 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 *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing.* https://doi.org/10.1109/SC.2006.30

[38] Xiaoye S. Li and James W. Demmel. 1998. Making Sparse Gaussian Elimination Scalable by Static Pivoting. In *Proceedings of the 1998 ACM/IEEE Conference on Supercomputing.* IEEE Computer Society, Washington, DC, USA, 1–17.

[39] Florent Lopez and Theo Mary. 2021. Mixed Precision LU Factorization on GPU Tensor Cores: Reducing Data Movement and Memory Footprint. http://eprints.maths.manchester.ac.uk/2782/ MIMS EPrint 2020.20, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, September 2020..

[40] Jean-Yves L'Excellent and Wissam M. Sid-Lakhdar. 2014. A study of shared-memory parallelism in a multifrontal solver. *Parallel Comput.* 40, 3 (2014), 34–46. https://doi.org/10.1016/j.parco.2014.02.003

[41] Cleve B. Moler. 1967. Iterative Refinement in Floating Point. *J. ACM* 14, 2 (April 1967), 316–321. https://doi.org/10.1145/321386.321394

[42] Steven A. Niederer, Eric Kerfoot, Alan P. Benson, Miguel O. Bernabeu, Olivier Bernus, Chris Bradley, Elizabeth M. Cherry, Richard Clayton, Flavio H. Fenton, Alan Garny, Elvio Heidenreich, Sander Land, Mary Maleckar, Pras Pathmanathan, Gernot Plank, José F. Rodríguez, Ishani Roy, Frank B. Sachse, Gunnar Seemann, Ola Skavhaug, and Nic P. Smith. 2011. Verification of cardiac tissue electrophysiology simulators using an N-version benchmark. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 369, 1954 (2011), 4331–4351. https://doi.org/10.1098/rsta.2011.0139 arXiv:https://royalsocietypublishing.org/doi/pdf/10.1098/rsta.2011.0139

[43] Grégoire Pichon, Eric Darve, Mathieu Faverge, Pierre Ramet, and Jean Roman. 2018. Sparse supernodal solver using block low-rank compression: Design, performance and analysis. *Journal of Computational Science* 27 (2018), 255–270. https://doi.org/10.1016/j.jocs.2018.06.007

[44] Olaf Schenk, Klaus Gärtner, and Wolfgang Fichtner. 2000. Efficient Sparse LU Factorization with Left-Right Looking Strategy on Shared Memory Multiprocessors. *BIT* 40 (03 2000), 158–176. https://doi.org/10.1023/A:1022326604210

[45] Daniil Shantsev, Piyoosh Jaysaval, Sébastien de la Kethulle de Ryhove, Patrick R. Amestoy, Alfredo Buttari, Jean-Yves L'Excellent, and Théo Mary. 2017. Large-scale 3D EM modeling with a Block Low-Rank multifrontal direct solver. *Geophysical Journal International* 209, 3 (2017), 1558–1571.

[46] Robert D. Skeel. 1980. Iterative Refinement Implies Numerical Stability for Gaussian Elimination. *Math. Comp.* 35, 151 (1980), 817–832. http://www.jstor.org/stable/2006197

[47] Rita Streich, Christoph Schwarzbach, Michael Becken, and Klaus Spitzer. 2010. Controlled-source Electromagnetic Modelling Studies – Utility of Auxiliary Potentials for Low-frequency Stabilization. *Conference Proceedings, 72nd EAGE Conference* cp-161-00065 (2010). https://doi.org/10.3997/2214-4609.201400657

[48] Pascal Theissen, Kirstin Heuler, Rainer Demuth, Johannes Wojciak, Thomas Indinger, and Nikolaus Adams. 2011. Experimental Investigation of Unsteady Vehicle Aerodynamics under Time-Dependent Flow Conditions - Part 1. In *SAE 2011 World Congress & Exhibition*. SAE International. https://doi.org/10.4271/2011-01-0177

[49] James H. Wilkinson. 1948. *Progress Report on the Automatic Computing Engine*. Report MA/17/1024. Mathematics Division, Department of Scientific and Industrial Research, National Physical Laboratory, Teddington, UK. 127 pages. http://www.alanturing.net/turing_archive/archive/l/l10/l10.php

[50] James H. Wilkinson. 1963. *Rounding Errors in Algebraic Processes*. Notes on Applied Science No. 32, Her Majesty's Stationery Office, London. vi+161 pages. Also published by Prentice-Hall, Englewood Cliffs, NJ, USA. Reprinted by Dover, New York, 1994.

[51] Mawussi Zounon, Nicholas J. Higham, Craig Lucas, and Françoise Tisseur. 2022. Performance Impact of Precision Reduction in Sparse Linear Systems Solvers. *PeerJ Comput. Sci.* 8 (Jan. 2022), e778(1–22). https://doi.org/10.7717/peerj-cs.778