Accelerating Restarted GMRES with Mixed Precision Arithmetic

Neil Lindquist, Piotr Luszczek, and Jack Dongarra, *Fellow, IEEE*

Abstract—The generalized minimum residual method (GMRES) is a commonly used iterative Krylov solver for sparse, non-symmetric systems of linear equations. Like other iterative solvers, data movement dominates its run time. To improve this performance, we propose running GMRES in reduced precision with key operations remaining in full precision. Additionally, we provide theoretical results linking the convergence of finite precision GMRES with classical Gram-Schmidt with reorthogonalization and its infinite precision counterpart which helps justify the convergence of this method to double-precision accuracy. We tested the mixed-precision approach with a variety of matrices and preconditioners on a GPU-accelerated node. Excluding the ILU(0) preconditioner, we achieved average speedups ranging from $8\,\%$ to $61\,\%$ relative to comparable double-precision implementations, with the simpler preconditioners achieving the higher speedups.

Index Terms—Linear Systems, Multiple precision arithmetic

1 Introduction

The generalized minimum residual method (GMRES) is an iterative solver for sparse, non-symmetric systems of linear equations [1]. Like most iterative solvers, GMRES consists mostly of matrix-vector and vector-vector operators, resulting in a low computational intensity. Hence, reducing data movement is necessary to improve performance. Towards this end, we present an implementation of GMRES using a mix of single- and double-precision that is designed to achieve the same final accuracy as double-precision GMRES while reducing data movement.

In short, GMRES constructs a basis for the Krylov subspace using Arnoldi's procedure [2], then finds the solution vector from that subspace that minimizes the 2-norm of the resulting residual. One particularly useful modification to GMRES is restarting, which is used to limit the memory usage and computational requirements of the growing Krylov basis [3]. We focus on two schemes for orthogonalizing the Krylov basis in the Arnoldi procedure, modified Gram-Schmidt (MGS) and classical Gram-Schmidt with reorthogonalization (CGSR). The former is often used due to its lower computational and data-access costs [4], while the latter better retains orthogonality and can be implemented using matrix-vector products [5]. For CGSR, we always reorthogonalize once, which is sufficient to provide numerically stability [6]. Additionally, we focused on a left-preconditioned version of GMRES but expect the results to hold for right-preconditioned versions too. Algorithm 1 shows the formulation of GMRES we used.

Based on our previous work, we focus on a specific mixed-precision strategy that has been successful regarding both accuracy and CPU performance [7]. This approach uses single precision everywhere except to compute the residual and update the solution. Specifically, it uses double precision for lines 3 and 25 in Alg. 1 and uses single precision for everything else. (I.e. computation done in double precision is labeled with u_{high} and computation done in single precision is labeled with u_{low}). This requires a copy of A in single precision, and two vector type conversions per restart ($\boldsymbol{z_k}$ and $\boldsymbol{u_k}$) but has the advantage of otherwise using existing uniform-precision kernels. Our previous work included experiments with the effects on convergence when reducing various parts of GMRES and using different restart strategies; those experiments guided the design of our mixed-precision scheme. Herein, we extend the support for this approach by providing stronger theoretical analysis of the reduced precision inner solver and new experimental results on a GPU-accelerated node with a variety of preconditioners.

2 Previous Work

The idea of using multiple precisions to improve performance has been applied successfully to a variety of problems, particularly in recent years [8]. Furthermore, it is a well-established method for improving the performance of solving systems of linear equations, particularly for dense linear systems [9], [10].

One approach to using multiple precisions in GMRES is to store the preconditioner in reduced precision and doing the rest of the computation in full precision [11]. The approximate nature of the preconditioner means that reducing the floating-point precision has a limited reduction in quality. One interesting variation of this is to precondition a full-precision GMRES with a reduced-precision GMRES (possibly with a preconditioner of its own) [12]. This is similar to our approach, but we use iterative refinement as the outer solver instead of GMRES.

There has recently been useful theoretical work for varying the working precision as the iteration progresses in a

• *Neil Lindquist, Piotr Luszczek, and Jack Dongarra are with the Innovative Computing Laboratory, University of Tennessee, Knoxville, TN, 37996. E-mail: {nlindqu1,luszczek,dongarra}@icl.utk.edu*
• *Jack Dongarra is also with Oak Ridge National Laboratory, Oak Ridge, TN 37831 and the University of Manchester, Manchester M13 9PL, United Kingdom*

Algorithm 1 Restarted GMRES in mixed-precision with left preconditioning [3]

```
1:  $A \in \mathbb{R}^{n\times n}, \quad \boldsymbol{x_0}, \boldsymbol{b} \in \mathbb{R}^n, \quad M^{-1} \approx A^{-1}$
2:  for $k = 1, 2, \dots$ do
3:      $\boldsymbol{z_k} \leftarrow \boldsymbol{b} - A\boldsymbol{x_k}$                                  $(u_{\text{high}})$
4:      If $\|\boldsymbol{z_k}\|_2$ is small enough, stop                      $(u_{\text{high}})$
5:      $\boldsymbol{r_k} \leftarrow M^{-1}\boldsymbol{z_k}$                                      $(u_{\text{low}})$
6:      $\beta \leftarrow \|\boldsymbol{r_k}\|_2, \quad s_0 \leftarrow \beta$                             $(u_{\text{low}})$
7:      $\boldsymbol{v_1} \leftarrow \boldsymbol{r_k}/\beta, \quad V_1 \leftarrow [\boldsymbol{v_1}]$                          $(u_{\text{low}})$
8:      $j \leftarrow 0$
9:      loop until the restart condition is met
10:         $j \leftarrow j + 1$
11:         $\boldsymbol{w} \leftarrow M^{-1}A\boldsymbol{v_j}$                                 $(u_{\text{low}})$
12:         $\boldsymbol{w}, h_{1,j}, \dots, h_{j,j} \leftarrow \mathrm{orth}(\boldsymbol{w}, V_j)$   ▷ MGS or CGSR
13:         $h_{j+1,j} \leftarrow \|\boldsymbol{w}\|_2$                                 $(u_{\text{low}})$
14:         $\boldsymbol{v_{j+1}} \leftarrow \boldsymbol{w}/h_{j+1,j}$                               $(u_{\text{low}})$
15:         $V_{j+1} \leftarrow [V_j, \boldsymbol{v_{j+1}}]$                              $(u_{\text{low}})$
16:         for $i = 1, \dots, j-1$ do
17:             $\begin{bmatrix} h_{i,j} \\ h_{i+1,j} \end{bmatrix} \leftarrow \begin{bmatrix} \alpha_i & \beta_i \\ -\beta_i & \alpha_i \end{bmatrix} \times \begin{bmatrix} h_{i,j} \\ h_{i+1,j} \end{bmatrix}$    $(u_{\text{low}})$
18:         end for
19:         $\begin{bmatrix} \alpha_j \\ \beta_j \end{bmatrix} \leftarrow \text{rotation\_matrix}\left(\begin{bmatrix} h_{j,j} \\ h_{j+1,j} \end{bmatrix}\right)$        $(u_{\text{low}})$
20:         $\begin{bmatrix} s_j \\ s_{j+1} \end{bmatrix} \leftarrow \begin{bmatrix} \alpha_j & \beta_j \\ -\beta_j & \alpha_j \end{bmatrix} \times \begin{bmatrix} s_j \\ 0 \end{bmatrix}$          $(u_{\text{low}})$
21:         $\begin{bmatrix} h_{j,j} \\ h_{j+1,j} \end{bmatrix} \leftarrow \begin{bmatrix} \alpha_j & \beta_j \\ -\beta_j & \alpha_j \end{bmatrix} \times \begin{bmatrix} h_{j,j} \\ h_{j+1,j} \end{bmatrix}$      $(u_{\text{low}})$
22:     end loop
23:     $H \leftarrow \{h_{i,\ell}\}_{1\le i,\ell\le j}, \quad \boldsymbol{s} \leftarrow [s_1, \dots s_j]^T$
24:     $\boldsymbol{u_k} \leftarrow V_j H^{-1}\boldsymbol{s}$                                    $(u_{\text{low}})$
25:     $\boldsymbol{x_{k+1}} \leftarrow \boldsymbol{x_k} + \boldsymbol{u_k}$                                $(u_{\text{high}})$
26: end for

27: procedure MGS$(\boldsymbol{w}, V_j)$
28:     $[\boldsymbol{v_1}, \dots, \boldsymbol{v_j}] \leftarrow V_j$
29:     for i = 1, 2, . . . , j do
30:         $h_{i,j} \leftarrow \boldsymbol{w} \cdot \boldsymbol{v_i}$                                  $(u_{\text{low}})$
31:         $\boldsymbol{w} \leftarrow \boldsymbol{w} - h_{i,j}\boldsymbol{v_i}$                              $(u_{\text{low}})$
32:     end for
33:     return $\boldsymbol{w}, h_{1,j}, \dots, h_{j,j}$
34: end procedure

35: procedure CGSR$(\boldsymbol{w}, V_j)$
36:     $\boldsymbol{h} \leftarrow {V_j}^T\boldsymbol{w}$                                      $(u_{\text{low}})$
37:     $\boldsymbol{w} \leftarrow \boldsymbol{w} - V_j\boldsymbol{h}$                                  $(u_{\text{low}})$
38:     $\boldsymbol{g} \leftarrow {V_j}^T\boldsymbol{w}$                                      $(u_{\text{low}})$
39:     $\boldsymbol{w} \leftarrow \boldsymbol{w} - V_j\boldsymbol{g}$                                  $(u_{\text{low}})$
40:     $[h_{0,j}, \dots, h_{j,j}]^T \leftarrow \boldsymbol{h} + \boldsymbol{g}$                         $(u_{\text{low}})$
41:     return $\boldsymbol{w}, h_{1,j}, \dots, h_{j,j}$
42: end procedure
```

non-restarted GMRES [13]. Notably, part of this work shows that computing the Arnoldi process in finite-precision allows GMRES to converge at approximately the same rate as GMRES computed exactly until an accuracy related to round-off error is reached. Our theoretical results in Sec. 3 are based on some of these ideas. There exists further theoretical work on reducing the accuracy of just the matrix vector products in GMRES and other Krylov solvers as the number of inner iterations progresses [14], [15], [16]. These approaches may avoid the need to restart, unlike our work; however, they increase the complexity of implementation and achieving high accuracy may require estimating the smallest singular value.

For restarted GMRES, there have been a few works involving using multiple precisions in the GMRES algorithm. The first is using mixed-precision iterative refinement where the inner solver is a single-precision GMRES [17], [18]. This approach is similar to what we tested; however, that work tests only limited configurations of GMRES and matrices. The second approach is to store just the Krylov basis in reduced precision, which was tested with both floating- and fixed-point formats and 32- and 16-bits per value [19]. It was successful in providing a speedup with the 32-bit floating-point version providing the best median speedup at 1.4 times. However, the scheme, as described, requires custom, high-performance, mixed-precision kernels, which increases the cost of implementation due to the limited availability of existing mixed-precision routines.

One final work with GMRES is the use of integer arithmetic [20]. While integer GMRES did not involve a reduction in data movement, it does show GMRES achieving a full-precision solution with limited iteration overhead when the solver uses an alternative data format. Relatedly, there has been work to use data compression techniques in flexible GMRES, although only for the non-orthogonalized Krylov vectors [21].

Mixed precision approaches have also been used for other iterative solvers. Like GMRES, mixed precision approaches include a reduced precision preconditioner [22], [23] and using a reduced precision solver inside iterative refinement [17]. However, with Krylov methods, iterative refinement discards the subspace at each restart; so, the strategy of "reliable updates" has been proposed, which retains information about the Krylov subspace across restarts [24], [25]. Finally, there has been some exploration of the use of alternate data representations, such as data compression, in iterative solvers [26], [27], [28].

3 Numerical Properties of Mixed Precision GMRES

It is important to understand the effects of reductions in precision on the accuracy of the final solutions. Note that restarted GMRES is equivalent to iterative refinement using a non-restarted GMRES as the inner solver, which provides a powerful tool for recovering accuracy. This equivalence can be seen by noting that lines 5 through 24 in Alg. 1 are equivalent to a non-restarted GMRES with an initial guess of $\boldsymbol{x_k}$ and the remaining form iterative refinement of the non-restarted GMRES.

First, consider the parts of the solver that must remain in full precision. Notably, the linear system being solved, $A\boldsymbol{x} = \boldsymbol{b}$, must be stored in full precision. In general, reductions in the accuracy of A and $\boldsymbol{b}$ directly change the problem being solved and can introduce a backward error on the order of the reduced precision. If $\boldsymbol{x}$ is stored in reduced precision, a similar forward error will be introduced. As an extension of this, adding the updates from the inner solve to $\boldsymbol{x}$ must be done in full precision, to ensure $\boldsymbol{x}$ remains in full precision. Finally, the residual $\boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{x}$ must be computed in full precision, otherwise errors smaller than the reduced precision accuracy cannot be corrected.

Next, consider the effects of reducing precision in the computation of the error correction. For stationary iterative refinement, it is well-known that the error correction can be computed in reduced precision while still achieving a full precision final solution [9]. For GMRES, the theory is less developed. Note that single-precision GMRES is backwards stable to single precision [29], [30]. Then, mixed-precision iterative refinement can use this GMRES to compute a double-precision backwards-stable solution [31]. Thus, the approach should be backwards-stable to full-precision. However, this analysis ignores the possibility of restarting before a single-precision accurate solution is achieved which is a significant issue in practice due to the increasing memory cost of GMRES.

Recent work has shown that MGS-GMRES converges at approximately the same rate whether the Arnoldi process is computed in finite precision or exactly until the relative residual is below a roundoff-dependent threshold [13]. Unfortunately, it is difficult to turn this into a useful general-purpose bound. Towards this end, we provide the following theorem for CGSR-GMRES which takes advantage of the better orthogonality of CGSR to describe the accuracy of the finite precision solution relative to the exact precision one [5].

Theorem 1. *Let $\overline{\boldsymbol{\chi_j}}$ and $\boldsymbol{\chi_j^{(e)}}$ be the solutions computed by j iterations of finite precision GMRES and exact GMRES, respectively, without restarting. Let $\boldsymbol{b}$ be the right-hand side vector. Suppose $u < 10^{-3}$ and $c_4(n,j)u\kappa(A\overline{V_j}) < 1$ for a particular $c_4(n,j) \in \mathcal{O}(n^2j^3)$ with $\overline{V_j}$ being the computed Krylov basis. Let $\delta_+ = (1+\sqrt{u})^{1/2}$ and $\delta_- = (1-\sqrt{u})^{1/2}$. Then,*

$$\begin{aligned}
\|\boldsymbol{b} - A\overline{\boldsymbol{\chi_j}}\|_2 \leq{}& \delta_+^2\delta_-^{-2}\|\boldsymbol{b} - A\boldsymbol{\chi_j^{(e)}}\|_2 \\
&+ 9\delta_+ uj\|A\|_2\|\overline{\boldsymbol{\chi_j}}\|_2 \\
&+ \delta_+^2\delta_-^{-1}\gamma_p j^{1/2}\|A\|_F\|\boldsymbol{\chi_j^{(W)}}\|_2 \\
&+ \delta_+\delta_-^{-1}c_1(n,j)u\|A\|_F\|\boldsymbol{\chi_j^{(W)}}\|_2 \\
&+ \frac{c_1(n,j)u\|A\|_2\|\overline{\boldsymbol{\chi_j}}\|_2}{\delta_- - \gamma_j j^{1/2}\delta_+} \\
&+ \frac{j^{1/2}\delta_+\|A\|_F\|\overline{\boldsymbol{\chi_j}}\|_2}{\delta_- - \gamma_j j^{1/2}\delta_+} \\
&\times \Bigg(\gamma_p + \gamma_j \\
&\qquad + (\gamma_j + 9uj\gamma_j + 9uj^{1/2}) \\
&\qquad \times \delta_+\delta_-^{-1}(2+\gamma_p)\Bigg).
\end{aligned}$$

where $c_1(n,j) \in \mathcal{O}(nj)$ and $\boldsymbol{\chi_j^{(W)}}$ is the solution computed by j iterations of a particular weighted-GMRES.

The polynomials $c_1(n,j)$ and $c_4(n,j)$ are the same as those in Giraud et al.'s error analysis of CGSR [6]. The proof is provided in the appendix. Note that $\overline{\boldsymbol{\chi_j}}$ and $\boldsymbol{\chi_J^{(e)}}$ are equivalent to $\boldsymbol{u_k}$ from Alg. 1 when the restart condition is met after j inner iterations.

When $u = 2^{-24}$, $\|\boldsymbol{\chi_j^{(e)}}\| \approx \|\overline{\boldsymbol{\chi_j}}\| \approx \|\boldsymbol{\chi_j^{(W)}}\|$, $j^{3/2}u \ll 1$, and $p^{3/2}u \ll 1$, this can be simplified to

$$\begin{aligned}
&\frac{\|\boldsymbol{b} - A\overline{\boldsymbol{\chi_j}}\|_2}{\|A\|_F\|\overline{\boldsymbol{\chi_j}}\|_2 + \|\boldsymbol{b}\|_2} \\
&\qquad\lesssim 1.1\frac{\|\boldsymbol{b} - A\boldsymbol{\chi_j^{(e)}}\|_2}{\|A\|_F\|\boldsymbol{\chi_j^{(e)}}\|_2 + \|\boldsymbol{b}\|_2} \\
&\qquad\qquad + (4j^{3/2} + 30j + 3pj^{1/2} + 3c_1(n,j))u.
\end{aligned}$$

In other words, finite precision CGSR-GMRES converges at effectively the same rate as its exact counterpart until an error of approximately $(4j^{3/2} + 30j + 3pj^{1/2} + 3c_1(n,j))u$ is reached. This implies that for restarted GMRES, if full precision GMRES can converge, then reduced precision GMRES should either converge similarly or satisfy the backward error threshold. In the latter case, the backward stability of iterative refinement will result in a backward stable solution [31]. In the former cases, we expect similar behavior, but differences in vector directions may reduce effectiveness when the solver restarts.

It should be noted that the low-order polynomials c_1 and c_4 can quickly become onerous for realistically sized matrices. Fortunately, the bounds provided by this theorem are much worse than will occur in practice. First, they assume round-off error will always accumulate without cancellation. However, the recent work on probabilistic error bounds for dot-products has shown that a relative error of about $u\sqrt{n}$ holds with probability close to 1, compared the formal worst-case bound of un [32]. Second, the error bounds assume that the dot product summation is done sequentially. However, GPU accelerated systems distribute the work across many threads which helps reduce the accumulation of errors [33].

4 Implementation Considerations

Periodic restarting is a key component in obtaining full precision accuracy in this approach as it relies on iterative refinement. For many problems, both the memory constraints and the increasing computational load will force the solver to restart before the accuracy of the inner solver is achieved. However, for some problems, GMRES can produce a full precision solution in relatively few iterations; in these cases, waiting to restart until after a fixed number of inner iterations will result in a stalled improvement when the round-off error has overwhelmed any meaningful contributions to the computed solution. We have previously discussed and tested various restart strategies [7]. In that work, we found that an effective restart strategy was to initiate the first restart when the residual approximation improves by a factor of 10^{-6}, then initiate subsequent restarts after the same number of inner iterations.

TABLE 1
Properties of the matrices tested without a preconditioner. †Condition estimator reached 200 000 iterations before the LSQR convergence criteria.

Matrix	Rows	Nonzeros	Condition Lower Bound	RHS Provided
af_0_k101	5.0×10^5	1.8×10^7	5.5×10^5 †	yes
af_shell9	5.0×10^5	1.8×10^7	1.2×10^6 †	yes
apache2	7.2×10^5	4.8×10^6	3.0×10^6 †	no
atmosmodj	1.3×10^6	8.8×10^6	6.4×10^3	yes
BenElechi1	2.5×10^5	1.3×10^7	1.3×10^6 †	yes
bone010	9.9×10^5	4.8×10^7	1.6×10^6 †	no
Bump_2911	2.9×10^6	1.3×10^8	7.5×10^6 †	no
cage13	4.5×10^5	7.5×10^6	1.1×10^1	no
cage14	1.5×10^6	2.7×10^7	9.6×10^0	no
crankseg_1	5.3×10^4	1.1×10^7	1.4×10^7 †	no
CurlCurl_2	8.1×10^5	8.9×10^6	4.1×10^5 †	no
CurlCurl_4	2.4×10^6	2.7×10^7	3.4×10^5 †	no
ecology2	1.0×10^6	5.0×10^6	3.2×10^7 †	no
F1	3.4×10^5	2.7×10^7	7.1×10^5 †	yes
FEM_3D_thermal2	1.5×10^5	3.5×10^6	2.5×10^3	no
G3_circuit	1.6×10^6	7.7×10^6	6.0×10^6 †	no
hood	2.2×10^5	9.9×10^6	3.8×10^5 †	no
language	4.0×10^5	1.2×10^6	5.9×10^2	no
marine1	4.0×10^5	6.2×10^6	3.8×10^5 †	yes
mc2depi	5.3×10^5	2.1×10^6	1.3×10^{14}	no
ns3Da	2.0×10^4	1.7×10^6	5.6×10^2	yes
parabolic_fem	5.3×10^5	3.7×10^6	2.1×10^5 †	yes
poisson3Db	8.6×10^4	2.4×10^6	2.6×10^3	yes
pwtk	2.2×10^5	1.2×10^7	6.9×10^5 †	no
rajat31	4.7×10^6	2.0×10^7	4.0×10^6	no
stomach	2.1×10^5	3.0×10^6	2.9×10^1	no
t2em	9.2×10^5	4.6×10^6	2.2×10^5 †	no
thermal2	1.2×10^6	8.6×10^6	1.5×10^6 †	yes
tmt_unsym	9.2×10^5	4.6×10^6	2.3×10^8 †	no
torso2	1.2×10^5	1.0×10^6	2.0×10^1	no
torso3	2.6×10^5	4.4×10^6	9.5×10^1	no
venkat01	6.2×10^4	1.7×10^6	1.3×10^5 †	yes

Total memory usage is an important constraint, particularly when considering the smaller memory sizes of GPU accelerators. Consider GMRES implemented for matrices in compressed sparse row (CSR) format and restarting after, at most, m inner iterations. The matrix entries, right-hand side, and solution take combined $12n_{nz}+\mathcal{O}(n)$ bytes. The double-precision GMRES requires an additional $8nm + \mathcal{O}(n + m^2)$ bytes. Instead, the mixed precision variant requires an additional $4n_{nz} + 4nm + \mathcal{O}(n + m^2)$ bytes. For problems with many nonzeros per row relative to m, the mixed-precision approach will require a larger total allocation, which may be onerous on memory-constrained systems. Storing the low-order and high-order bytes separately would circumvent this issue [34]. Regardless of matrix storage, the mixed-precision approach always has a smaller increase in allocation for an increase in m. On some large problems, this may allow for a larger basis and, thus, take better advantage of superlinear convergence in GMRES [35].

5 Experimental Results

To test the performance of our approach on GPUs, we implemented a restarted GMRES using the Kokkos performance portability library [36] and NVIDIA's cuBLAS and cuSPARSE libraries. Kokkos was chosen for ease of use and is not expected to perform significantly different from a CUDA implementation. To limit expensive memory transfers between CPU and GPU, all computation is done on the GPU and only the high level control flow is done on the CPU.

Various matrices in CSR format from the SuiteSparse collection with more than a million nonzero elements were tested [37]. Furthermore, we only used matrices that converged with fewer than 300 restarts for a given configuration. If a file ending with _b was provided by SuiteSparse, the first column was used as the right-hand side. For other matrices, the right-hand side was computed from a solution where each element was randomly selected from the uniform range $[0, 1)$. The matrices tested are shown in Table 1. In addition to structural properties, the table contains lower bounds of the condition numbers of these matrices, computed by testing forward error vectors of LSQR [38]. Many of the poorly conditioned matrices reached the iteration limit before LSQR's convergence criteria, so they may have significantly worse conditioning than these lower bounds imply.

We tested the effectiveness of the mixed-precision approach for a variety of preconditioners. First is the identity matrix to test unpreconditioned GMRES. While in practice some form of preconditioner is almost always used, not using a preconditioner makes it easier to compute condition numbers. Second is a scalar Jacobi preconditioner. This is a diagonal matrix where each element is the inverse of the corresponding diagonal element of A to provide a measure of row scaling. Third is an incomplete LU factorization without fill in (ILU(0)). This is a powerful preconditioner formed through Gaussian Elimination, except only entries that are nonzero in A are computed. However, the sparse triangular solves needed to apply the factorization cannot be parallelized to efficiently use a GPU. So, fourth, we test ILU(0), except with the triangular solves replaced with Jacobi iterations [39], [40]. Because we are focusing on how well the mixed-precision approach works for different types of preconditioners, we test each preconditioner in isolation and do not compare them. Furthermore, each preconditioner is computed in double precision, then converted to the appropriate precision to limit differences caused by preconditioner differences. In addition to testing double-precision GMRES and the proposed mixed-precision GMRES, we also tested a completely single-precision implementation and reducing the precision of just the preconditioner. In order to use existing uniform-precision kernels, the input and output of the preconditioner are converted to single precision.

Each test was run to reach a backward error of

$$\frac{\|\boldsymbol{b} - A\boldsymbol{x}\|_2}{\|A\|_F\|\boldsymbol{x}\|_2 + \|\boldsymbol{b}\|_2} \leq 10^{-10}.$$

Any test requiring more than 300 restarts was considered a failure, with the tested matrices chosen to all succeed using the double-precision implementation. At most 100 inner iterations were run before restarting, with three restart strategies tested:

1) just the inner-iteration count,
2) the residual approximation improving by a factor of 10^{-10}, and
3) the residual approximation improving by a factor of 10^{-6} for the first restart and the same number of inner iterations after that.

Note that mixed precision was only tested with the third strategy to prevent the choice of restart strategy from inappropriately benefiting it. For each configuration we ran the code three times, plotting the median, with error bars for the minimum and maximum. Run times include constructing the preconditioner and any type conversions. The restart strategy with the smallest median was plotted. Speedups were computed as the inverse of the geometric mean of the normalized mixed-precision times.

Our implementation is available at bitbucket.org/icl/mixed-precision-gmres under the tag `TPDS`. We used Kokkos 3.1.01, CUDA 10.2.89, MKL 2019.3.199, and GCC 7.3.0. Code was run on a machine with a single NVIDIA V100 GPU and two Haswell 10-core Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz processors. Each CPU core has a $32\,\mathrm{KiB}$ L1 instruction cache, a $32\,\mathrm{KiB}$ L1 data cache, and a $256\,\mathrm{KiB}$ L2 cache. Each processor has a single $25\,\mathrm{MiB}$ L3 cache, and the entire node has $32\,\mathrm{GiB}$ of memory. The V100 card has 80 streaming multiprocessors, a $128\,\mathrm{KiB}$ L1 cache for each multiprocessor, a $6\,\mathrm{MiB}$ shared L2 cache, and a $16\,\mathrm{GiB}$ memory.

First, the results when no preconditioner is used are shown in Fig. 1. The average speedups for the mixed-precision approach were $18\,\%$ for MGS and $61\,\%$ for CGSR. Furthermore, it provided a speedup for most of the tested matrices and with CGSR almost doubled the performance on many of the matrices. It only reduced performance for `atmosmodj` using MGS and `language`. The single-precision implementation only satisfied the target accuracy on 16 of the 23 problems; for these problems, it had average speedups of $0\,\%$ and $35\,\%$, respectively. The total number of inner iterations was mostly consistent between the double- and mixed-precision implementations, as is shown in Table 2, with the single-precision implementations needing similar or more iterations. The most notable exception is `rajat31`, which converged in significantly fewer iterations with the mixed- and single-precision implementations, contributing to its much higher speedups. We have been unable to determine the source of this behavior; we speculate that the floating-point error happens to perturb the Krylov subspace to better contain the solution. For `language`, the baseline implementation converged after 29 iterations without restarting, whereas the mixed-precision implementation restarted once after 29 iterations for a total of 58 iterations. Of the matrices with provided right-hand sides, `atmosmodj` had a significant increase in the number of iterations, but mixed-precision was still able to achieve a speedup with CGSR; `thermal2` also had an increase in iterations for MGS, but still performed on par with the baseline. For the matrices with comparable iteration counts, many can achieve approximately a 2× speedup with CGSR; however, some only obtained modest speedups. These reduced speedups correlate with the matrices with a high number of nonzeros per row relative to the size of the basis; thus, transferring the column-index array of the matrix will have a larger influence on the runtime. As per Table 1, even matrices with condition larger than the inverse of single-precision unit roundoff could be solved more efficiently with the mixed-precision approach; although, this may depend on the right-hand side. Finally, the matrices with right-hand sides from the SuiteSparse collection overall behaved similar to those with generated right-hand sides.

Second are the results for a scalar Jacobi preconditioner, shown in Fig. 2. The average speedups for the mixed-precision implementation were $12\,\%$ and $50\,\%$ for MGS and CGSR, respectively. The single-precision preconditioner outperformed the double-precision implementation in some cases. However, it failed on `mc2depi` for CGSR and provided an average slowdown of $8\,\%$ for both orthogonalization schemes on the successful matrices. The single-precision implementation failed to satisfy the target accuracy in 11 of the 30 problems; the remaining problems had average speedups of $-8\,\%$ (i.e., a slowdown) and $23\,\%$, respectively. For MGS, there were five matrices where the mixed-precision implementation failed to outperform double precision. For CGSR, mixed-precision outperformed double-precision for all matrices except `language`. The total number of inner iterations for a matrix was mostly consistent between the tested configurations, as is shown in Table 3, with `language` again taking twice the iterations and `pwtk` taking fewer iterations with MGS compared to CGSR. Like the non-preconditioned results, better speedups were achieved on matrices that had a small number of nonzeros per row relative to the size of the basis, which relates to the costs of matrix-vector products compared to orthogonalization.

Third are the results for an ILU(0) preconditioner, shown in Fig. 3. The average speedup for mixed-precision was

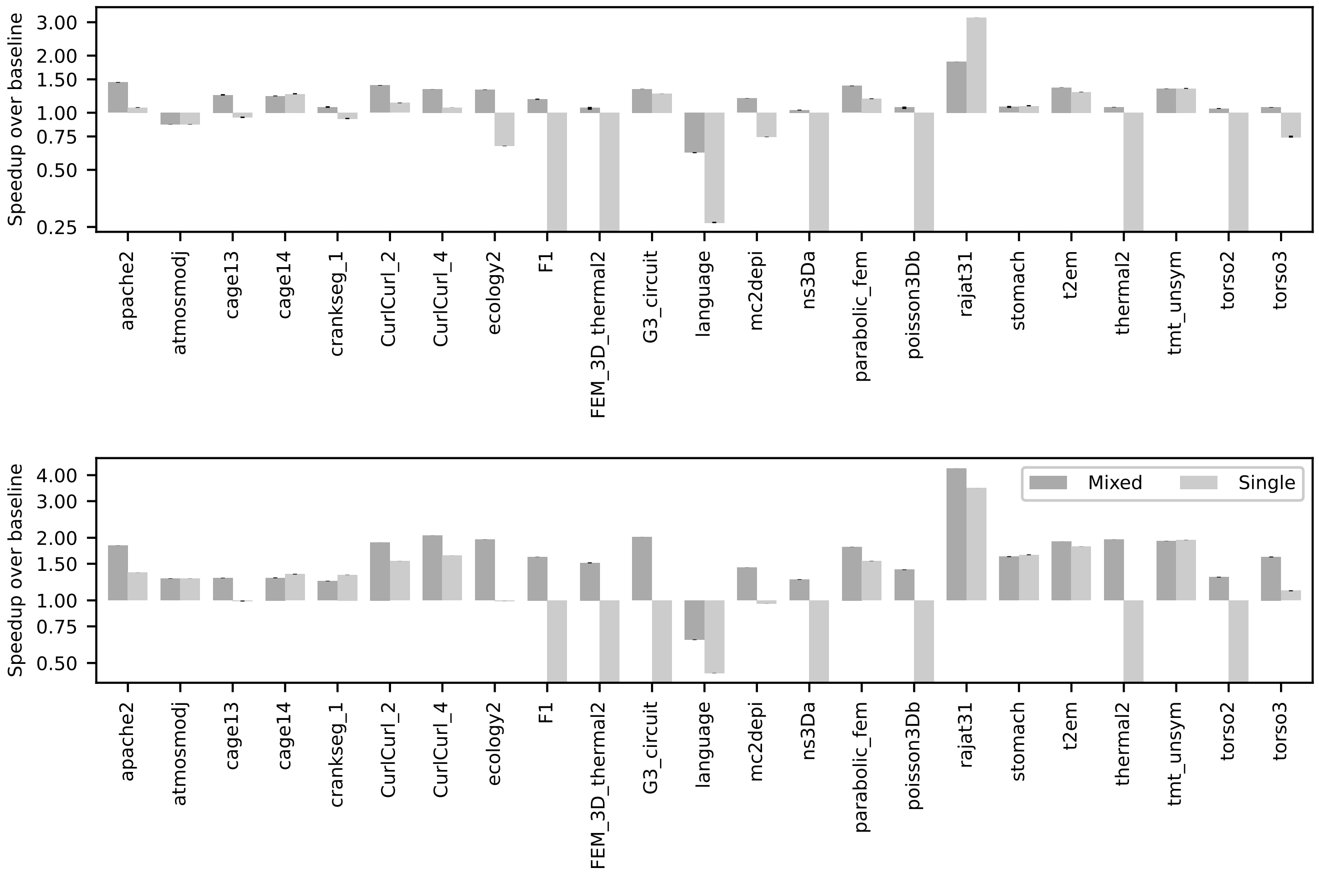


Fig. 1. Relative performance of unpreconditioned GMRES with MGS (top) or CGSR (bottom).

TABLE 2
Inner Iteration counts for unpreconditioned GMRES.

	Double		Mixed		Single	
Matrix	MGS	CGSR	MGS	CGSR	MGS	CGSR
apache2	21 400	21 400	21 500	21 500	29 200	29 300
atmosmodj	200	200	300	300	300	300
cage13	30	30	30	30	45	45
cage14	30	30	30	30	30	30
crankseg_1	6 300	6 200	6 300	6 300	7 300	5 900
CurlCurl_2	9 900	9 900	9 900	9 900	12 300	12 300
CurlCurl_4	21 100	21 100	21 100	21 100	26 400	26 400
ecology2	900	900	900	900	1 800	1 800
F1	29 200	29 200	29 200	29 200	-	-
FEM_3D_thermal2	300	300	300	300	-	-
G3_circuit	28 200	28 200	27 500	28 200	29 200	-
language	29	29	58	58	145	87
mc2depi	10 400	10 200	12 100	12 900	19 500	19 500
ns3Da	1 400	1 400	1 400	1 400	-	-
parabolic_fem	3 500	3 500	3 500	3 500	4 100	4 100
poisson3Db	300	300	300	300	-	-
rajat31	4 000	4 000	2 900	2 000	1 700	2 500
stomach	300	300	300	300	300	300
t2em	4 800	4 800	4 800	4 800	5 100	5 100
thermal2	21 100	28 500	25 600	28 500	-	-
tmt_unsym	500	500	500	500	500	500
torso2	80	80	80	80	-	-
torso3	200	200	200	200	300	300

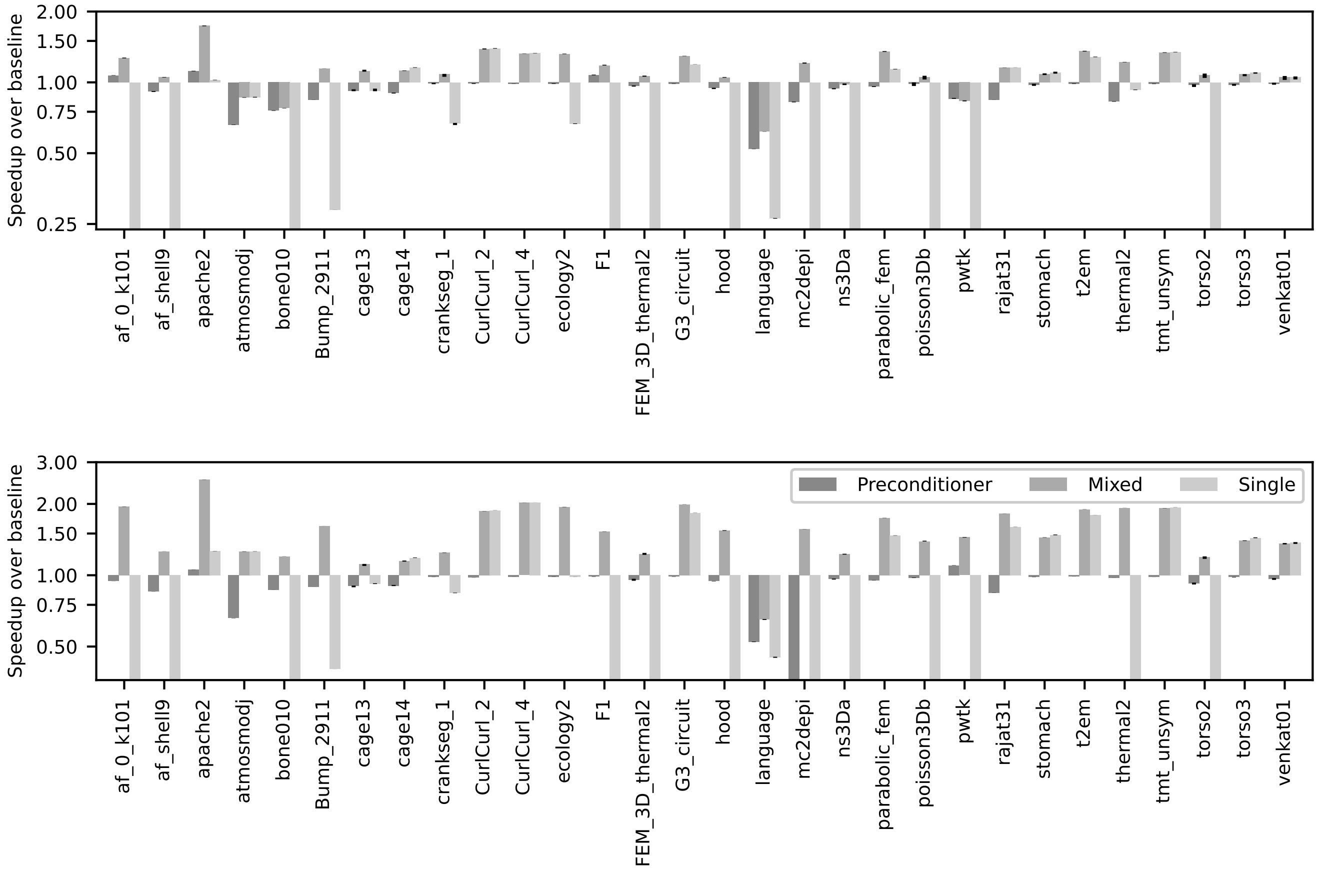


Fig. 2. Relative performance of GMRES with a scalar Jacobi preconditioner and MGS (top) or CGSR (bottom).

TABLE 3
Inner Iteration counts for GMRES preconditioned with Jacobi.

	Double		Preconditioner		Mixed		Single	
Matrix	MGS	CGSR	MGS	CGSR	MGS	CGSR	MGS	CGSR
af_0_k101	18 200	14 300	16 800	14 900	19 800	12 700	-	-
af_shell9	21 000	20 500	22 700	23 600	27 600	28 400	-	-
apache2	11 700	11 700	10 400	10 900	9 800	8 500	16 700	17 200
atmosmodj	200	200	300	300	300	300	300	300
bone010	14 600	19 000	19 100	21 500	25 600	28 200	-	-
Bump_2911	3 500	3 500	4 100	3 900	4 100	4 100	16 400	16 600
cage13	22	19	22	22	22	22	33	33
cage14	22	22	22	19	22	22	22	22
crankseg_1	800	800	800	800	800	800	1 300	1 200
CurlCurl_2	1 500	1 500	1 500	1 500	1 500	1 500	1 500	1 500
CurlCurl_4	1 900	1 900	1 900	1 900	1 900	1 900	1 900	1 900
ecology2	800	800	800	800	800	800	1 600	1 600
F1	3 600	3 600	3 300	3 600	3 600	3 800	-	-
FEM_3D_thermal2	60	60	60	60	60	60	-	-
G3_circuit	1 100	1 100	1 100	1 100	1 100	1 100	1 200	1 200
hood	3 900	3 900	4 100	4 100	4 100	4 000	-	-
language	29	29	58	58	58	58	145	87
mc2depi	11 000	10 600	13 200	-	12 600	12 200	-	-
ns3Da	14 300	14 400	14 800	14 600	15 000	14 900	-	-
parabolic_fem	3 600	3 600	3 700	3 700	3 700	3 700	4 400	4 400
poisson3Db	400	400	400	400	400	400	-	-
pwtk	13 500	18 400	15 700	16 600	17 700	20 200	-	-
rajat31	600	600	700	700	700	700	700	800
stomach	130	100	130	100	130	130	130	130
t2em	4 800	4 800	4 800	4 800	4 800	4 800	5 100	5 100
thermal2	21 400	25 300	25 400	25 500	22 700	25 500	29 800	-
tmt_unsym	500	500	500	500	500	500	500	500
torso2	56	47	56	56	56	56	-	-
torso3	134	100	100	100	134	134	134	134
venkat01	96	96	96	96	96	96	96	96

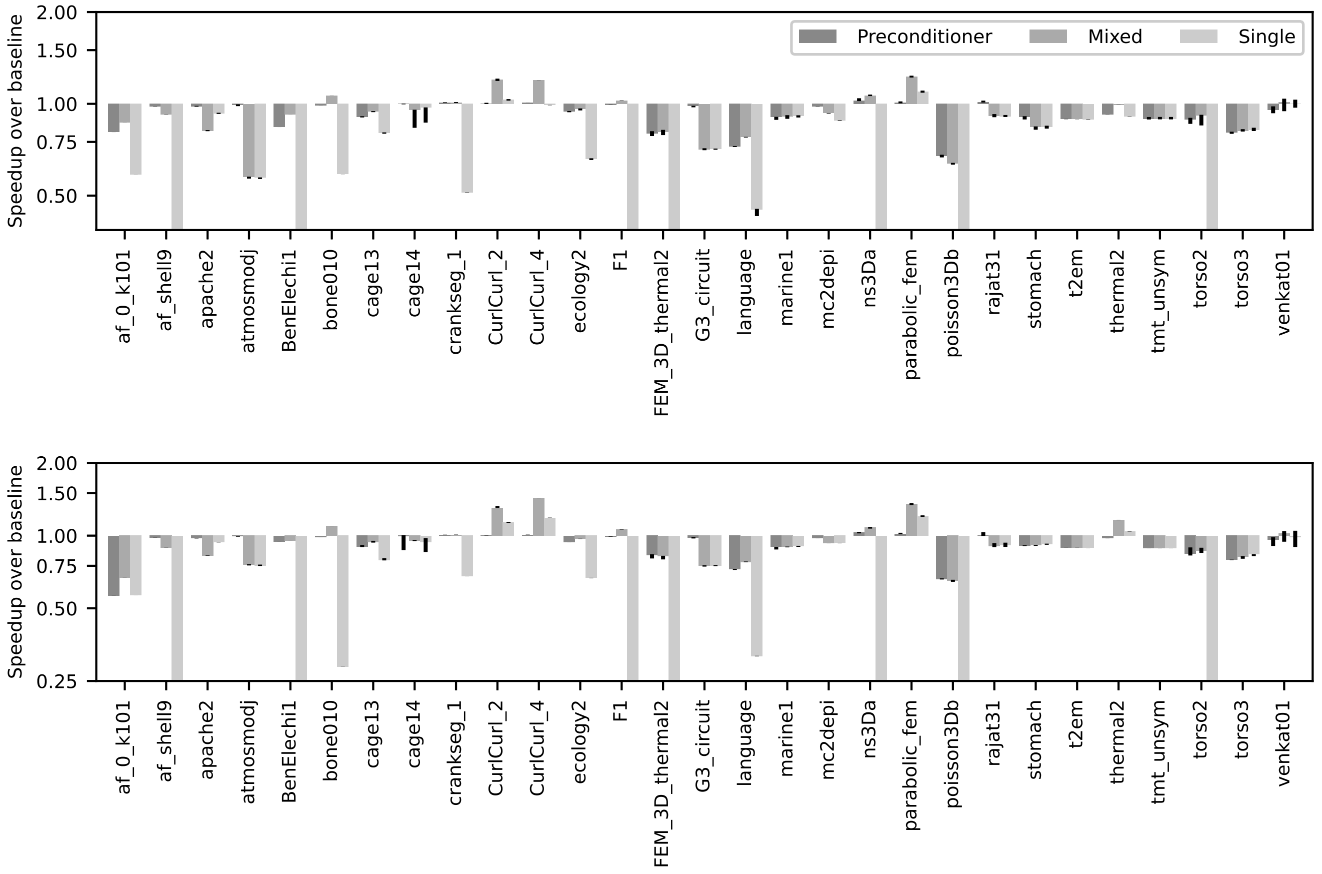
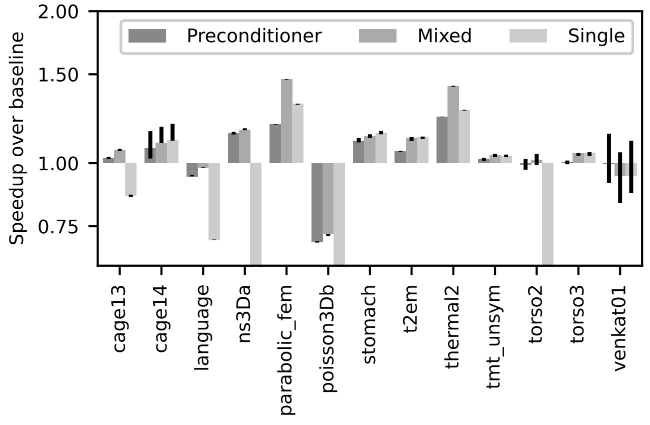


Fig. 3. Relative performance of CGSR GMRES with an ILU(0) preconditioner and MGS (top) or CGSR (bottom).

TABLE 4
Inner Iteration counts for GMRES preconditioned with ILU(0).

	Double		Preconditioner		Mixed		Single	
Matrix	MGS	CGSR	MGS	CGSR	MGS	CGSR	MGS	CGSR
af_0_k101	4 500	4 400	5 500	7 700	5 200	6 600	7 700	7 800
af_shell9	2 200	2 200	2 200	2 200	2 400	2 500	-	-
apache2	600	600	600	600	800	800	700	700
atmosmodj	82	82	82	82	200	164	200	164
BenElechi1	3 900	4 300	4 600	4 500	4 200	4 500	-	-
bone010	1 200	1 200	1 200	1 200	1 200	1 200	2 200	4 700
cage13	7	7	8	8	8	8	12	12
cage14	7	7	7	7	8	8	8	8
crankseg_1	200	200	200	200	200	200	400	300
CurlCurl_2	600	600	600	600	600	600	700	700
CurlCurl_4	1 400	1 400	1 400	1 400	1 400	1 400	1 700	1 700
ecology2	200	200	200	200	200	200	300	300
F1	1 000	1 000	1 000	1 000	1 000	1 000	-	-
FEM_3D_thermal2	10	10	12	12	12	12	-	-
G3_circuit	200	200	200	200	300	300	300	300
language	9	9	14	14	14	14	28	42
marine1	300	300	300	300	300	300	300	300
mc2depi	1 700	1 700	1 600	1 600	1 700	1 700	1 800	1 700
ns3Da	200	200	200	200	200	200	-	-
parabolic_fem	800	800	800	800	800	800	900	900
poisson3Db	100	100	174	174	174	174	-	-
rajat31	10	10	9	9	18	18	18	18
stomach	17	17	18	18	20	18	20	18
t2em	600	600	600	600	600	600	600	600
thermal2	4 800	5 100	5 100	5 100	5 200	5 100	5 700	5 700
tmt_unsym	200	200	200	200	200	200	200	200
torso2	11	11	12	12	12	12	-	-
torso3	35	35	48	48	48	48	48	48
venkat01	12	12	12	12	12	12	12	12



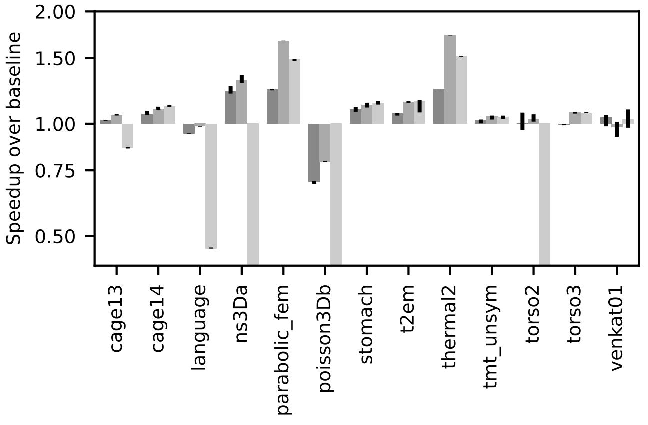


Fig. 4. Relative performance of CGSR GMRES with an ILU(0) preconditioner using five Jacobi iterations for triangular solves and MGS (top) or CGSR (bottom).

TABLE 5
Inner Iteration counts for GMRES preconditioned using ILU(0) with five Jacobi iterations for triangular solves.

	Double		Preconditioner		Mixed		Single	
Matrix	MGS	CGSR	MGS	CGSR	MGS	CGSR	MGS	CGSR
cage13	7	7	8	8	8	8	12	12
cage14	7	7	8	8	8	8	8	8
language	9	9	14	14	14	14	21	35
ns3Da	200	200	200	200	200	200	-	-
parabolic_fem	800	800	800	800	800	800	900	900
poisson3Db	100	100	174	174	174	174	-	-
stomach	21	21	22	22	22	22	22	22
t2em	800	800	800	800	800	800	800	800
thermal2	5 100	5 100	5 000	5 100	5 100	5 100	5 700	5 800
tmt_unsym	200	200	200	200	200	200	200	200
torso2	11	11	12	12	12	12	-	-
torso3	48	48	64	64	64	64	64	64
venkat01	16	16	16	16	16	16	16	16

$-9\,\%$ and $-7\,\%$ for MGS and CGSR, respectively (i.e., a slowdown). For the single-precision preconditioner, the speedups were $-8\,\%$ and $-9\,\%$ instead. There are a few factors that we attribute the lack of improvement to. The single-precision implementation failed to produce an accurate enough solution to 7 of the problems; the remaining problems had average speedups of $-11\,\%$ and $-10\,\%$, respectively. First, both sparse triangular solves have limited parallelism for the GPU to exploit, particularly when there are few nonzeros per row; this results in the GPU bandwidth being underutilized and limited benefit by reducing the size of the data. Furthermore, the poor performance of the triangular solves causes them to make up a large part of the performance. Second, the factorization is done in double-precision for both implementations, making it a fixed cost in the performance. Third, because of the effectiveness of the preconditioner, a high percentage of matrices can be solved without restarting in double-precision; however, the mixed-precision implementation must always restart at least once. These restarts incur overhead by computing the solution update and new residual, and reduce the rate of convergence for some matrices, increasing the iteration count, by discarding information on the old Krylov subspace and interfering with GMRES's superlinear convergence [35]. Table 4 shows the relevant iteration counts and that the baseline can converge without restarting for 11 out of the 29 matrices.

Finally, are the results for an ILU(0) preconditioner with five Jacobi iterations for triangular solves, shown in Fig. 4. The speedup was $8\,\%$ and $13\,\%$ for MGS and CGSR, respectively. Additionally, the single-precision preconditioner was able to achieve some improvement overall, with speedups of $3\,\%$ and $4\,\%$, respectively. The single-precision implementation failed to produce an accurate enough solution to 3 of the problems; the remaining problems had average speedups of $4\,\%$ and $4\,\%$, respectively. Note that while the triangular solves have been improved, the other factors limiting the improvement of the regular ILU(0) preconditioner remain. Table 5 shows the iteration counts and that the baseline only needed to restart on 5 of the 13 matrices.

While testing the performance, we discovered that using CGSR provides better performance in GPU-accelerated GMRES compared to MGS, despite the extra orthogonalization. However, results showing this performance difference do not appear in literature, outside of a few assertions that CGSR is better for GPU-accelerated systems [19], [41]. The overall speedup is higher for simpler preconditioners and the mixed-precision implementation, as shown in Table 6. Recall that MGS requires j dot-products alternated with j vector additions for the jth inner iteration while CGSR merely requires four matrix-vector products. Thus, CGSR launches significantly fewer kernels which reduces

TABLE 6
Average speedup of CGSR-GMRES versus MGS-GMRES for various configurations.

Preconditioner	Double Speedup	Mixed Speedup	Single Speedup
Identity	36%	87%	181%
Jacobi	34%	79%	116%
ILU(0)	4%	7%	4%
ILU(0) with Jacobi	8%	13%	4%

overhead; furthermore, high GPU utilization is easier to obtain with larger kernels than smaller ones. The better speedup for mixed- and single-precision implementations likely comes from reductions in the cost of the kernel's execution making the kernel launches more costly relative to the total time. Similarly, when GMRES uses a cheaper preconditioner, it spends a higher percentage of its runtime doing the orthogonalization which results in a better speedup for switching from MGS to CGSR.

6 Conclusion

Like previous works with similar uses of precision [7], [18], [19], [20], our mixed-precision implementation never required more than twice the total inner iterations than the double-precision implementation, and usually much less than twice the total inner iterations when many restarts are needed. This reinforces the ideas provided by Sec. 3. Furthermore, looking at the matrices tested without a preconditioner, as listed in Table 1, none of the matrices satisfy $n^2u < 1$, let alone $\mathcal{O}(n^2j^3)u\kappa(A\overline{V_j}) < 1$. So, $c_4(n,j)$ from Thm. 1 can likely be significantly improved, which correlates with the analysis of the types of dot-product bounds used.

Between the theoretical results in Sec. 3 and the experimental results in Sec. 5, there is strong evidence that this mixed-precision approach for GMRES retains double-precision accuracy. Performance improvement was less clear cut, with the ILU(0) preconditioner seeing a slowdown. However, tests with GPU-friendly preconditioners and baselines that restarted consistently showed speedups, especially with CGSR orthogonalization.

There are three main future directions for this work. The first direction is to understand the performance of mixed-precision GMRES on multi-GPU and distributed systems. These systems are important for solving problems too large to be solved effectively or even fit on a single compute unit. However, they have additional communication costs to coordinate and exchange data. The second direction is to investigate the use of alternative data representations. 16-bit floating-point formats are one possibility but reduce the accuracy by half compared to single precision. However, alternative techniques, such as compression, may allow using less than 32-bits per value without significantly reducing accuracy. Furthermore, it may be possible to reduce the memory needed for matrix indices. The third direction is to extend these ideas to other formulations of GMRES and other Krylov methods. One particularly important class of GMRES variants are the communication avoiding and pipelined algorithms, which try to reduce communication overheads when running on distributed systems.

Appendix
Proof of Theorem 1

Let $\overline{\cdot}$ denote values computed by finite precision GMRES and $\cdot^{(e)}$ denote values computed by exact GMRES. Because of, e.g., Line 3 in Alg. 1, we can assume the initial guess is $\mathbf{0} \in \mathbb{R}^n$ without loss of generality.

We start with Arnoldi's procedure, as described by Lines 6–15 of Alg. 1. In finite precision, it produces $\overline{V_{j+1}}$, $\overline{H_j}$ such that

$$A\overline{V_j} + E_j = \overline{V_{j+1}}\,\overline{H_j} \qquad \text{and} \qquad \overline{V_{j+1}}^T\overline{V_{j+1}} = I - F. \quad (1)$$

Note that

$$E_j = [\mathcal{E}_1\boldsymbol{v_1}, \mathcal{E}_2\boldsymbol{v_2}, \ldots, \mathcal{E}_j\boldsymbol{v_j}] + \Delta H$$

where $|\mathcal{E}_i| \leq \gamma_p|A|$ for $i = 1, \ldots, j$, p is the maximum number of nonzeros in any row of A [33], $\|\Delta H\|_2 \leq c_1(n,j)u\|A\|_2$, and $c_1(n,j) \in \mathcal{O}(nj^{3/2})$ [6, (25)]. So,

$$\|E_j\|_2 \leq \gamma_p\||A|\|_2\||\overline{V_j}|\|_2 + c_1(n,j)u\|A\|_2.$$

By the assumption on the conditioning, $\|F\|_2 \leq \sqrt{u}$ [6, Thm. 2, (32), and (38)]. So, there exists a symmetric positive definite matrix W such that $\overline{V_{j+1}}^T W\overline{V_{j+1}} = I$ and $\kappa(W) \leq (1+\sqrt{u})/(1-\sqrt{u})$ [13, Lemma 2]. Thus,

$$\begin{aligned}\sqrt{1-\sqrt{u}} \leq \sigma_1(\overline{V_{j+1}}) \leq \sigma_1(\overline{V_j}) \\ \leq \sigma_j(\overline{V_j}) \leq \sigma_{j+1}(\overline{V_{j+1}}) \leq \sqrt{1+\sqrt{u}}\end{aligned}$$

and

$$\|\overline{H_j}\|_2 \leq \left(\sqrt{\tfrac{1+\sqrt{u}}{1-\sqrt{u}}}(1+\gamma_p) + c_1(n,j)u\right)\|A\|_F.$$

The final computed solution is $\overline{\boldsymbol{\chi_j}} = \overline{V_{j+1}}\overline{\boldsymbol{y_j}} + \boldsymbol{\delta\chi}$ with $|\boldsymbol{\delta\chi}| \leq \gamma_j|\overline{V_j}||\overline{\boldsymbol{y_j}}|$ [33]. Then,

$$\|\overline{\boldsymbol{y_j}}\|_2 \leq \left(\sqrt{1-\sqrt{u}} - \gamma_j j^{1/2}\sqrt{1+\sqrt{u}}\right)^{-1}\|\overline{\boldsymbol{\chi_j}}\|_2$$

and

$$\|\boldsymbol{b} - A\overline{\boldsymbol{\chi_j}}\|_2 \leq \|\boldsymbol{b} - A\overline{V_j}\overline{\boldsymbol{y_j}}\|_2 + \|A\boldsymbol{\delta\chi}\|_2.$$

By (1),

$$\|\boldsymbol{b} - A\overline{V_j}\overline{\boldsymbol{y_j}}\|_2 = \|\overline{V_{j+1}}(\beta\boldsymbol{e_1} - \overline{H_j}\overline{\boldsymbol{y_j}})\|_2 + \|E_j\overline{\boldsymbol{y_j}}\|_2.$$

Thus,

$$\begin{aligned}&\|\overline{V_{j+1}}(\beta\boldsymbol{e_1} - \overline{H_j}\overline{\boldsymbol{y_j}})\|_2 \\ &\qquad = \sqrt{1+\sqrt{u}}\|\beta\boldsymbol{e_1} - \overline{H_j}\overline{\boldsymbol{y_j}}\|_2.\end{aligned}$$

Next, consider the least squares problem solved by lines 16–24 in Alg. 1. Let $\overline{G_i}$ and G_i be the computed and exact Givens rotation matrices to eliminate the subdiagonal elements in rows $i = 2, \ldots, j+1$ of $\overline{H_j}$, and let $\overline{R}$ and R be the resulting computed and exact triangular matrices. Furthermore, let Q be the product of rotation matrices, and let $\overline{\boldsymbol{q}}$ and $\boldsymbol{q}$ be the computed and exact values of $Q^T\beta e_1$. Thus, $\|\overline{R} - R\|_2 \leq 9uj\|\overline{H_j}\|_2$ and $\|\overline{\boldsymbol{q}} - \boldsymbol{q}\|_2 \leq 9uj\beta$ [33, Lemmas 19.8, 3.6, and 3.4]. So,

$$\begin{aligned}\|\beta\boldsymbol{e_1} - \overline{H_j}\overline{\boldsymbol{y_j}}\|_2 \leq \|\overline{\boldsymbol{q}} - \overline{R}\overline{\boldsymbol{y_j}}\|_2 \\ + 9uj\beta + \|(\overline{R} - R)\overline{\boldsymbol{y_j}}\|_2\end{aligned}$$

Then, $\overline{\boldsymbol{y_j}}$ satisfies

$$(\overline{R} + \Delta R)\overline{\boldsymbol{y_j}} = \overline{q}[1:j]$$

where $|\Delta R| \le \gamma_j |\overline{R}|$. So,

$$\begin{aligned}\|\overline{\boldsymbol{q}} - \overline{R}\overline{\boldsymbol{y_j}}\|_2 &= \left\| \begin{bmatrix}\Delta R\overline{\boldsymbol{y_j}} \\ \overline{\boldsymbol{q}}[j+1]\end{bmatrix}\right\|_2 \\ &\le \|\Delta R \overline{\boldsymbol{y_j}}\|_2 + |\overline{\boldsymbol{q}}[j+1]| \\ &= \min_{\boldsymbol{y}}(\|\beta \boldsymbol{e_1} - \overline{H_j}\boldsymbol{y}\|_2) \\ &\qquad + 9uj\beta + \gamma_j \||\overline{R}|\|_2 \|\overline{\boldsymbol{y_j}}\|_2\end{aligned}$$

Additionally,

$$\||\overline{R}|\|_2 \le (j^{1/2} + 9j^{3/2}u)\|\overline{H_j}\|_2$$

Next, we bound this minimization. Note that for any $\boldsymbol{y} \in \mathbb{R}^j$

$$\begin{aligned}\|\beta \boldsymbol{e_1} - \overline{H_j}\boldsymbol{y}\|_2 &= \|W^{1/2}\overline{V_{j+1}}(\beta \boldsymbol{e_1} - \overline{H_j}\boldsymbol{y})\|_2 \\ &\le \sqrt{\tfrac{1}{1-\sqrt{u}}}\|\boldsymbol{b} - \overline{V_{j+1}}\,\overline{H_j}\boldsymbol{y}\|_2 \qquad ([13, (35)]) \\ &\le \sqrt{\tfrac{1}{1-\sqrt{u}}}\left(\|\boldsymbol{b} - A\overline{V_j}\boldsymbol{y}\|_2 + \|E_j\|_2\|\boldsymbol{y}\|_2\right).\end{aligned}$$

Additionally,

$$\begin{aligned}&\left\|\arg\min_{\boldsymbol{y}}(\|\boldsymbol{b} - A\overline{V_j}\boldsymbol{y}\|_2 + \|E_j\|_2\|\boldsymbol{y}\|_2)\right\|_2 \\ &\quad\le \left\|\arg\min_{\boldsymbol{y}}(\|\boldsymbol{b} - A\overline{V_j}\boldsymbol{y}\|_2)\right\|_2 \\ &\quad\le \sqrt{1+\sqrt{u}}\|\boldsymbol{\chi_j^{(W)}}\|_2\end{aligned}$$

where $\boldsymbol{\chi_j^{(W)}}$ is the solution computed exactly by j iterations of W-GMRES. So,

$$\begin{aligned}&\min_{\boldsymbol{y}}(\|\boldsymbol{b} - A\overline{V_j}\boldsymbol{y}\|_2 + \|E_j\|_2\|\boldsymbol{y}\|_2) \\ &\quad\le \min_{\boldsymbol{y}}(\|\boldsymbol{b} - A\overline{V_j}\boldsymbol{y}\|_2) + \sqrt{1+\sqrt{u}}\|E_j\|\|\boldsymbol{\chi_j^{(W)}}\|\end{aligned}$$

Because exact W-GMRES computes $\min_{\boldsymbol{y}}(\|\boldsymbol{b} - A\overline{V_j}\boldsymbol{y}\|_2)$,

$$\min_{\boldsymbol{y}}\|\boldsymbol{b} - A\overline{V_j}\boldsymbol{y}\|_2 \le \sqrt{\tfrac{1+\sqrt{u}}{1-\sqrt{u}}}\|\boldsymbol{b} - A\boldsymbol{\chi_j^{(e)}}\|_2.$$

Finally, combining the preceding inequalities gives

$$\begin{aligned}\|\boldsymbol{b} - A\overline{\boldsymbol{\chi_j}}\|_2 \le\ & \tfrac{1+\sqrt{u}}{1-\sqrt{u}}\|\boldsymbol{b} - A\boldsymbol{\chi_j^{(e)}}\|_2 \\ &+ 9uj\sqrt{1+\sqrt{u}}\|A\|_2\|\overline{\boldsymbol{\chi_j}}\|_2 \\ &+ \sqrt{\tfrac{1+\sqrt{u}}{1-\sqrt{u}}}\|A\|_F\|\boldsymbol{\chi_j^{(W)}}\|_2 \\ &\quad\times\left(\gamma_p\sqrt{j(1+\sqrt{u})} + c_1(n,j)u\right) \\ &+ \frac{c_1(n,j)u\|A\|_2\|\overline{\boldsymbol{\chi_j}}\|_2}{\sqrt{1-\sqrt{u}} - \gamma_j j^{1/2}\sqrt{1+\sqrt{u}}} \\ &+ \frac{j^{1/2}\sqrt{1+\sqrt{u}}\|A\|_F\|\overline{\boldsymbol{\chi_j}}\|_2}{\sqrt{1-\sqrt{u}} - \gamma_j j^{1/2}\sqrt{1+\sqrt{u}}} \\ &\quad\times\Bigg(\gamma_p + \gamma_j \\ &\qquad + (\gamma_j + 9uj\gamma_j + 9uj^{1/2}) \\ &\qquad\times\sqrt{\tfrac{1+\sqrt{u}}{1-\sqrt{u}}}(2+\gamma_p)\Bigg). \qquad \square\end{aligned}$$

Acknowledgments

This material is based upon work supported by the University of Tennessee grant MSE E01-1315-038 as Interdisciplinary Seed funding and in part by UT Battelle subaward 4000123266. This material is also based upon work supported by the National Science Foundation under OAC Grant No. 2004541.

References

[1] Y. Saad and M. H. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 7, no. 3, pp. 856–869, 1986.

[2] W. E. Arnoldi, "The principle of minimized iteration in the solution of the matrix eigenvalue problem," *Quarterly of Applied Mathematics*, vol. 9, pp. 17–29, 1951.

[3] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. Philadelphia, PA, USA: SIAM Press, 2003.

[4] C. C. Paige and Z. Strakos, "Residual and backward error bounds in minimum residual Krylov subspace methods," *SIAM J. Sci. Comput.*, vol. 23, no. 6, pp. 1898–1923, Jun. 2001.

[5] L. Giraud, J. Langou, and M. Rozloznik, "The loss of orthogonality in the Gram-Schmidt orthogonalization process," *Comput. Math. Appl.*, vol. 50, no. 7, pp. 1069–1075, Oct. 2005.

[6] L. Giraud, J. Langou, M. Rozložník, and J. van den Eshof, "Rounding error analysis of the classical Gram-Schmidt orthogonalization process," *Numerische Mathematik*, vol. 101, no. 1, pp. 87–100, Jul. 2005.

[7] N. Lindquist, P. Luszczek, and J. Dongarra, "Improving the performance of the GMRES method using mixed-precision techniques," in *Driving Scientific and Engineering Discoveries through the Convergence of HPC, Big Data and AI*. Oak Ridge, TN, USA: Springer, Aug. 2020.

[8] A. Abdelfattah, H. Anzt, E. G. Boman, E. Carson, T. Cojean, J. Dongarra, A. Fox, M. Gates, N. J. Higham, X. S. Li, J. Loe, P. Luszczek, S. Pranesh, S. Rajamanickam, T. Ribizel, B. F. Smith, K. Swirydowicz, S. Thomas, S. Tomov, Y. M. Tsai, and U. M. Yang, "A survey of numerical linear algebra methods utilizing mixed-precision arithmetic," *The International Journal of High Performance Computing Applications*, Mar. 2021.

[9] J. H. Wilkinson, *Rounding Errors in Algebraic Processes*. Princeton, NJ, USA: Prentice-Hall, 1963.

[10] A. Buttari, J. Dongarra, J. Langou, J. Langou, P. Luszczek, and J. Kurzak, "Mixed precision iterative refinement techniques for the solution of dense linear systems," *Int. J. High Perform. Comput. Appl.*, vol. 21, no. 4, pp. 457–466, Nov. 2007.

[11] L. Giraud, A. Haidar, and L. T. Watson, "Mixed-precision preconditioners in parallel domain decomposition solvers," in *Domain Decomposition Methods in Science and Engineering XVII*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 357–364.

[12] M. Baboulin, A. Buttari, J. Dongarra, J. Kurzak, J. Langou, J. Langou, P. Luszczek, and S. Tomov, "Accelerating scientific computations with mixed precision algorithms," *CoRR*, vol. abs/0808.2794, 2008.

[13] S. Gratton, E. Simon, D. Titley-Peloquin, and P. Toint, "Exploiting variable precision in GMRES," *SIAM Journal on Scientific Computing (to appear)*, 2020.

[14] J. van den Eshof and G. L. G. Sleijpen, "Inexact Krylov subspace methods for linear systems," *SIAM Journal on Matrix Analysis and Applications*, vol. 26, no. 1, pp. 125–153, Jan. 2004.

[15] A. Bouras and V. Frayssé, "Inexact matrix-vector products in Krylov methods for solving linear systems: A relaxation strategy," *SIAM Journal on Matrix Analysis and Applications*, vol. 26, no. 3, pp. 660–678, Jan. 2005.

[16] V. Simoncini and D. B. Szyld, "Theory of inexact Krylov subspace methods and applications to scientific computing," *SIAM Journal on Scientific Computing*, vol. 25, no. 2, pp. 454–477, Jan. 2003.

[17] H. Anzt, V. Heuveline, and B. Rocker, "Mixed precision iterative refinement methods for linear systems: Convergence analysis based on Krylov subspace methods," in *Proceedings of the 10th International Conference on Applied Parallel and Scientific Computing - Volume 2*, ser. PARA'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 237–247.

[18] ——, "An error correction solver for linear systems: Evaluation of mixed precision implementations," in *High Performance Computing for Computational Science – VECPAR 2010*, J. M. L. M. Palma, M. Daydé, O. Marques, and J. C. Lopes, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 58–70.

[19] J. I. Aliaga, H. Anzt, T. Grützmacher, E. S. Quintana-Ortí, and A. E. Tomás, "Compressed basis GMRES on high performance GPUs," *arXiv:2009.12101 [cs]*, Sep. 2020.

[20] T. Iwashita, K. Suzuki, and T. Fukaya, "An integer arithmetic-based sparse linear solver using a GMRES method and iterative refinement," in *2020 IEEE/ACM 11th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*. Atlanta, GA: IEEE Computer Society Press, Sep. 2020.

[21] E. Agullo, F. Cappello, S. Di, L. Giraud, X. Liang, and N. Schenkels, "Exploring variable accuracy storage through lossy compression techniques in numerical linear algebra: A first application to flexible GMRES," Inria Bordeaux Sud-Ouest, Research Report RR-9342, 2020.

[22] A. Buttari, J. Dongarra, J. Kurzak, P. Luszczek, and S. Tomov, "Using mixed precision for sparse matrix computations to enhance the performance while achieving 64-bit accuracy," *ACM Trans. Math. Softw.*, vol. 34, no. 4, pp. 17:1–17:22, Jul. 2008.

[23] M. Emans and A. van der Meer, "Mixed-precision AMG as linear equation solver for definite systems," *Procedia Computer Science*, vol. 1, no. 1, pp. 175–183, 2010.

[24] M. Clark, R. Babich, K. Barros, R. Brower, and C. Rebbi, "Solving lattice QCD systems of equations using mixed precision solvers on GPUs," *Computer Physics Communications*, vol. 181, no. 9, pp. 1517–1528, 2010.

[25] R. Strzodka and D. Goddeke, "Pipelined mixed precision algorithms on FPGAs for fast and accurate PDE solvers from low precision components," in *2006 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa, CA, USA, 2006, pp. 259–270.

[26] H. Anzt, G. Flegar, T. Grützmacher, and E. S. Quintana-Ortí, "Toward a modular precision ecosystem for high-performance computing," *The International Journal of High Performance Computing Applications*, vol. 33, no. 6, pp. 1069–1078, Nov. 2019.

[27] O. S. Lawlor, "In-memory data compression for sparse matrices," in *Proceedings of the 3rd Workshop on Irregular Applications: Architectures and Algorithms*, ser. IA^3 '13. New York, NY, USA: Association for Computing Machinery, Nov. 2013, pp. 1–6.

[28] N. Lindquist, "Reducing Memory Access Latencies using Data Compression in Sparse, Iterative Linear Solvers," Bachelor's Thesis, Saint John's University, Apr. 2019.

[29] C. C. Paige, M. Rozloznik, and Z. Strakos, "Modified Gram-Schmidt (MGS), least squares, and backward stability of MGS-GMRES," *SIAM Journal on Matrix Analysis and Applications*, vol. 28, no. 1, pp. 264–284, 2006.

[30] J. Drkošová, A. Greenbaum, M. Rozložník, and Z. Strakoš, "Numerical stability of GMRES," *BIT Numerical Mathematics*, vol. 35, no. 3, pp. 309–330, Sep. 1995.

[31] E. Carson and N. J. Higham, "Accelerating the solution of linear systems by iterative refinement in three precisions," *SIAM Journal on Scientific Computing*, vol. 40, no. 2, pp. A817–A847, Jan. 2018.

[32] N. J. Higham and T. Mary, "A new approach to probabilistic rounding error analysis," *SIAM Journal on Scientific Computing*, vol. 41, no. 5, pp. A2815–A2835, Jan. 2019.

[33] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed. Society for Industrial and Applied Mathematics, 2002.

[34] H. Anzt, J. Dongarra, and E. S. Quintana-Ortí, "Adaptive precision solvers for sparse linear systems," in *Proceedings of the 3rd International Workshop on Energy Efficient Supercomputing*, ser. E2SC '15. New York, NY, USA: ACM, 2015, pp. 2:1–2:10.

[35] H. A. Van der Vorst and C. Vuik, "The superlinear convergence behaviour of GMRES," *Journal of Computational and Applied Mathematics*, vol. 48, no. 3, pp. 327–341, 1993.

[36] H. C. Edwards, C. R. Trott, and D. Sunderland, "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns," *Journal of Parallel and Distributed Computing*, vol. 74, no. 12, pp. 3202–3216, 2014.

[37] T. A. Davis and Y. Hu, "The University of Florida Sparse Matrix Collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, Dec. 2011.

[38] H. Avron, A. Druinsky, and S. Toledo, "Spectral condition-number estimation of large sparse matrices," *Numerical Linear Algebra with Applications*, vol. 26, no. 3, May 2019.

[39] H. Anzt, E. Chow, and J. Dongarra, "Iterative sparse triangular solves for preconditioning," in *Euro-Par 2015: Parallel Processing*, J. L. Träff, S. Hunold, and F. Versaci, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 650–661.

[40] E. Chow, H. Anzt, J. Scott, and J. Dongarra, "Using Jacobi iterations and blocking for solving sparse triangular systems in incomplete factorization preconditioning," *Journal of Parallel and Distributed Computing*, vol. 119, pp. 219–230, Sep. 2018.

[41] J. Dubois, C. Calvin, and S. Petiton, "Performance and numerical accuracy evaluation of heterogeneous multicore systems for Krylov orthogonal basis computation," in *High Performance Computing for Computational Science – VECPAR 2010*, ser. Lecture Notes in Computer Science, J. M. L. M. Palma, M. Daydé, O. Marques, and J. C. Lopes, Eds. Berlin, Heidelberg: Springer, 2011, pp. 45–57.

Neil Lindquist is a graduate student at the Innovative Computing Laboratory in the University of Tennessee, Knoxville's Tickle College of Engineering. He earned his B.A. degree in Computer Science and Mathematics from Saint John's University in Collegeville, Minnesota. Neil's research interests include numerical linear algebra, and the effects of data representation on performance and accuracy.

Piotr Luszczek is a Research Assistant Professor at the Innovative Computing Laboratory in the University of Tennessee, Knoxville's Tickle College of Engineering. Piotr earned his B.S. and M.Sc. degrees in Computer Science from AGH University of Science and Technology in Kraków, Poland, and his Ph.D. in Computer Science from the University of Tennessee Knoxville. Piotr's research interests include benchmarking, numerical linear algebra for high-performance computing, automatic performance tuning for modern hardware, and stochastic models for performance. Piotr has over a decade of experience developing HPC numerical software for large scale, distributed memory systems with multi-core processors and hardware accelerators. Currently, Piotr serves as a co-PI for the ECP xSDK project, the primary goal of which is to improve access to world-class software on exascale machines. In the course of his career, Piotr has achieved a Google Scholar h-index of 38 and an i10-index of 105.

Jack Dongarra is an American University Distinguished Professor of Computer Science in the Electrical Engineering and Computer Science Department at the University of Tennessee. He holds the position of a Distinguished Research Staff member in the Computer Science and Mathematics Division at Oak Ridge National Laboratory, and is an Adjunct Professor in the Computer Science Department at Rice University. Dongarra holds the Turing Fellowship in the schools of Computer Science and Mathematics at the University of Manchester. He is a Faculty Fellow at Texas A&M University's Institute for Advanced Study. Dongarra is the founding director of Innovative Computing Laboratory.