Exploiting Lower Precision Arithmetic in Solving Symmetric Positive Definite Linear Systems and Least Squares Problems

Higham, Nicholas and Pranesh, Srikara

2019

MIMS EPrint: **2019.20**

Manchester Institute for Mathematical Sciences
School of Mathematics

The University of Manchester

Reports available from: http://eprints.maths.manchester.ac.uk/
And by contacting: The MIMS Secretary
School of Mathematics
The University of Manchester
Manchester, M13 9PL, UK

ISSN 1749-9097

EXPLOITING LOWER PRECISION ARITHMETIC IN SOLVING SYMMETRIC POSITIVE DEFINITE LINEAR SYSTEMS AND LEAST SQUARES PROBLEMS*

NICHOLAS J. HIGHAM† AND SRIKARA PRANESH†

Abstract. What is the fastest way to solve a linear system $Ax = b$ in arithmetic of a given precision when A is symmetric positive definite and otherwise unstructured? The usual answer is by Cholesky factorization, assuming that A can be factorized. We develop an algorithm that can be faster, given an arithmetic of precision lower than the working precision as well as (optionally) one of higher precision. The arithmetics might, for example, be of precisions half, single, and double; half and double, possibly with quadruple; or single and double, possibly with quadruple. We compute a Cholesky factorization at the lower precision and use the factors as preconditioners in GMRES-based iterative refinement. To avoid breakdown of the factorization we shift the matrix by a small multiple of its diagonal. We explain why this is preferable to the common approach of shifting by a multiple of the identity matrix. We also incorporate scaling in order to avoid overflow and reduce the chance of underflow when working in IEEE half precision arithmetic. We extend the algorithm to solve a linear least squares problem with a well conditioned coefficient matrix by forming and solving the normal equations. In both algorithms most of the work is done at low precision provided that iterative refinement and the inner iterative solver converge quickly. We explain why replacing GMRES by the conjugate gradient method causes convergence guarantees to be lost, but show that this change has little effect on convergence in practice. Our numerical experiments confirm the potential of the new algorithms to provide faster solutions in environments that support multiple precisions of arithmetic.

Key words. symmetric positive definite matrix, Cholesky factorization, diagonal scaling, half precision arithmetic, mixed precision, bfloat16, fp16, overflow, underflow, iterative refinement, linear system, least squares problem, normal equations, GMRES, conjugate gradient method, preconditioning

AMS subject classifications. 65F05, 65F08, 65F35, 65F10

1. Introduction.

The standard way to solve a linear system $Ax = b$ whose coefficient matrix is symmetric positive definite but otherwise not specially structured is via a Cholesky factorization, assuming that one can be computed and stored. Normally, all computations are carried out at the working precision, which is typically double precision (64 bits). If multiple precisions of floating-point arithmetic are available can we solve the problem more quickly? This question is timely because of the increasing availability of half precision (16-bit) arithmetic, to add to the single precision (32-bit) arithmetic that has been available for many years.

The 2008 revision of the IEEE-754 standard for floating-point arithmetic [27] proposed a half precision format, which we denote by fp16. As shown in Table 1.1, it has an 11-bit significand and a 5-bit exponent. Fp16 arithmetic is increasingly supported by accelerators, including the NVIDIA P100 (2016), V100 (2017), and A100 (2020) GPUs and the AMD Radeon Instinct family of accelerators.[1] On these devices half precision is up to twice as fast as single precision, or up to 8 and 16 times faster than single precision on the NVIDIA V100 and A100, respectively, when their tensor cores are exploited. Of the 500 machines on the June 2020 TOP500 list, 150 systems

*Version of July 8, 2020. **Funding**: This work was supported by MathWorks, Engineering and Physical Sciences Research Council grant EP/P020720/1, and the Royal Society. The opinions and views expressed in this publication are those of the authors, and not necessarily those of the funding bodies.

†School of Mathematics, University of Manchester, Manchester, M13 9PL, UK (nick.higham@manchester.ac.uk, srikara.pranesh@manchester.ac.uk).

[1] https://www.amd.com/en/graphics/servers-radeon-instinct-mi

TABLE 1.1
Parameters for four floating-point arithmetics, given to three significant figures: number of bits in significand (including implicit most significant bit) and exponent (signif., exp.), unit roundoff u, smallest positive (subnormal) number $x_{\min}^s$, smallest normalized positive number $x_{\min}$, and largest finite number $x_{\max}$.

	(signif., exp.)	u	$x_{\min}^s$	$x_{\min}$	$x_{\max}$
bfloat16	$(8, 8)$	3.91×10^{-3}	9.18×10^{-41} [a]	1.18×10^{-38}	3.39×10^{38}
fp16	$(11, 5)$	4.88×10^{-4}	5.96×10^{-8}	6.10×10^{-5}	6.55×10^{4}
fp32	$(24, 8)$	5.96×10^{-8}	1.40×10^{-45}	1.18×10^{-38}	3.40×10^{38}
fp64	$(53, 11)$	1.11×10^{-16}	4.94×10^{-324}	2.22×10^{-308}	1.80×10^{308}

[a] Existing implementations of bfloat16 do not support subnormal numbers.

employ accelerators, among which around 80 percent support fp16.[2] Furthermore, the Fugaku machine that heads the June 2020 Top 500 list[3] is based on the A64FX Arm processor, which supports fp16 [12], and the Frontier exascale machine at Oak Ridge National Laboratory will use custom-built AMD Radeon Instinct GPUs with mixed precision capabilities [35].

Another half precision format is bfloat16, originally proposed by Google and formalized by Intel [28]. Bfloat16 has an 8-bit significand, and an 8-bit exponent. The exponent width is the same as that of single precision and hence bfloat16 has a range similar to single precision. Bfloat is supported by the Google Tensor Processing Unit[4] (TPU), the NVIDIA A100 GPU, the Intel Cooper Lake processor, and the Armv8-A architecture [3].

It has already been demonstrated that the fp16 arithmetic and the tensor cores of an NVIDIA V100 can be exploited to solve a general dense linear system $Ax = b$ up to four times faster than by an optimized double precision solver, with a reduction in energy consumption of up to 80 percent [18], [20]. The key idea, proposed in [7], [8], is to factorize A in half precision arithmetic and use GMRES-based iterative refinement (GMRES-IR), with the computed LU factors as preconditioners, to compute a numerically stable solution at the working precision of single or double. Furthermore, the same GMRES-IR algorithm has demonstrated a performance up to 3.7 times higher than that of an optimized double precision solver at scale on the Summit machine that headed the November 2019 TOP500 list [6], [19]. Similar improvements are also shown for a combination of complex single and half precision arithmetic in [1].

In principle, adapting GMRES-IR to exploit positive definite matrices might seem straightforward: replace LU factorization by Cholesky factorization and GMRES by the conjugate gradient (CG) method. However, Cholesky factorization can fail because the matrix can lose definiteness when it is rounded to lower precision. Furthermore, when CG is used existing error analysis can only guarantee iterative refinement to converge for very well conditioned matrices.

In this work we develop a scaling and shifting algorithm to perform lower precision Cholesky factorization of a matrix given in a higher precision. We use the computed Cholesky factorization to develop a GMRES-IR algorithm for symmetric positive definite systems. We then extend the algorithm to solve a linear least squares problem $\min_x \|b - Ax\|_2$ via the normal equations, where A is assumed to be well conditioned.

[2] https://www.top500.org/statistics/list/
[3] https://www.top500.org/lists/top500/2020/06/
[4] https://cloud.google.com/tpu/

To ensure that Cholesky factorization succeeds in lower precision we need to increase the diagonal, either before or after rounding to the lower precision. A natural way to do so is with the shift $A \leftarrow A + \mu I$, for some $\mu > 0$. We will show that it is better to transform $A \leftarrow A + \mu' D^2$, where $D = \mathrm{diag}(a_{ii}^{1/2})$. Perturbing A by a multiple of D^2 produces a smaller perturbation than perturbing by a multiple of I and it results in better performance of GMRES-IR.

In fact, we will work with the unit diagonal matrix $H = D^{-1}AD^{-1}$ and will round a shifted and scaled version of this matrix. The scalings are necessary for fp16, to allow a wider dynamic range of data to be supported, but they will usually not be required for bfloat16 or single precision.

Our algorithms contain three precisions: the working precision, with unit roundoff u; a lower precision, with unit roundoff $u_\ell > u$, at which matrix factorization is done; and a potentially higher precision, with unit roundoff $u_r \le u$, which we will use in iterative refinement. We define

$$\gamma_n = \frac{nu}{1-nu}, \quad nu < 1, \qquad \gamma_n^{(\ell)} = \frac{nu_\ell}{1-nu_\ell}, \quad nu_\ell < 1.$$

While we are particularly interested in the case where u_ℓ represents half precision, our algorithms and analysis are quite general.

In the next section we explore how to perturb a positive definite matrix to ensure that Cholesky factorization succeeds in floating-point arithmetic. In section 3 we develop a scaling and shifting algorithm to perform low precision Cholesky factorization. In section 4 we use the low precision Cholesky factors as preconditioners in GMRES-IR and explain why the underlying convergence analysis does not apply when CG is used in place of GMRES. In section 5 we develop a normal equations-based solver built on GMRES-IR for well-conditioned least squares problems. In section 6 we perform numerical experiments with real-life matrices to investigate the numerical behavior of GMRES-IR and the GMRES-IR-based least squares solver. Conclusions are given in Section 7.

2. Shifting for Cholesky factorization. A key task in this work is to round a positive definite matrix stored in single precision or double precision to a lower precision and then compute a Cholesky factorization. The difficulty is that Cholesky factorization may fail because the rounded matrix is indefinite or not sufficiently positive definite. Our solution is to increase the diagonal before factorizing, but what type of perturbation should we apply and how large should it be? In this section we address this question in a general form, working with just one precision u; we consider the effect of rounding to a lower precision before factorizing in the next section.

We take A to be a symmetric positive definite matrix whose smallest eigenvalue $\lambda_{\min}(A)$ is possibly as small as $u\lambda_{\max}(A)$, where $\lambda_{\max}(A)$ is the largest eigenvalue of A. Thus for floating-point computation, A is on the border between being positive definite and indefinite. In fact, our algorithms can also be applied when $\lambda_{\min}(A) < 0$ with $\lambda_{\min}(A) = O(u\lambda_{\max}(A))$, since backward error considerations show that the algorithms cannot distinguish this case from the positive definite case. In either case, we wish to increase the diagonal so that Cholesky factorization is guaranteed to succeed.

One approach is based on Wilkinson's result that Cholesky factorization of a positive definite matrix $A \in \mathbb{R}^{n\times n}$ succeeds if $d_n\kappa_2(A)u \le 1$, where $d_n = 20n^{3/2}$ [38] and $\kappa_2(A) = \lambda_{\max}(A)/\lambda_{\min}(A)$. Let $A(\epsilon) = A + \epsilon I$, where $\epsilon > 0$. Applying Wilkinson's result to $A(\epsilon)$, we have that Cholesky factorization on $A(\epsilon)$ will succeed if

$d_n(\lambda_{\max}(A)+\epsilon)u \le \lambda_{\min}(A)+\epsilon$, that is,

$$\epsilon \ge \frac{d_n\lambda_{\max}(A)u - \lambda_{\min}(A)}{1-d_nu} \approx d_n\lambda_{\max}(A)u - \lambda_{\min}(A). \tag{2.1}$$

We note that this is consistent with the analysis in [16], [42, Thm. 1].

Now write

$$H = D^{-1}AD^{-1}, \quad D = \operatorname{diag}\bigl(a_{ii}^{1/2}\bigr). \tag{2.2}$$

This is the natural equilibration for a positive definite matrix, since $h_{ii} \equiv 1$ and $|h_{ij}| \le 1$ for $i \ne j$. Also, $\kappa_2(H)$ is within a factor n of the minimal 2-norm condition number over all symmetric two-sided nonsingular diagonal scalings of A [21, Cor. 7.6], [36]. Demmel obtained the following result [11], [21, Thms. 10.5, 10.7].

THEOREM 2.1. *Let $A = DHD \in \mathbb{R}^{n\times n}$ be symmetric positive definite, where $D = \operatorname{diag}\bigl(a_{ii}^{1/2}\bigr)$. If*

$$\lambda_{\min}(H) > n\gamma_{n+1}/(1-\gamma_{n+1}) \tag{2.3}$$

then Cholesky factorization applied to A succeeds (barring underflow and overflow). If Cholesky factorization succeeds it produces a nonsingular $\widehat{R}$ satisfying

$$\widehat{R}^T\widehat{R} = A + \Delta A, \qquad |\Delta A| \le (1-\gamma_{n+1})^{-1}\gamma_{n+1}dd^T,$$

where $d_i = a_{ii}^{1/2}$.

This result shows that to guarantee the success of the factorization we can shift H to satisfy (2.3) instead of shifting A subject to (2.1). The matrix $H(\epsilon) = H + \epsilon I$ has constant diagonal $1+\epsilon$, so $H(\epsilon) = D_1WD_1$, where $D_1 = (1+\epsilon)^{1/2}I$ and $W = (1+\epsilon)^{-1}H(\epsilon)$ has unit diagonal.

Hence by Theorem 2.1, Cholesky factorization on $H(\epsilon)$ will succeed if $\lambda_{\min}(W) > n\gamma_{n+1}/(1-\gamma_{n+1})$, that is, if

$$(1+\epsilon)^{-1}(\lambda_{\min}(H)+\epsilon) > n\gamma_{n+1}/(1-\gamma_{n+1}). \tag{2.4}$$

Solving for ϵ gives

$$\epsilon > \frac{n\gamma_{n+1}/(1-\gamma_{n+1}) - \lambda_{\min}(H)}{1 - n\gamma_{n+1}/(1-\gamma_{n+1})} \approx n(n+1)u - \lambda_{\min}(H). \tag{2.5}$$

Now $DH(\epsilon)D = DHD + \epsilon D^2 = A + \epsilon D^2$, and $D^2 = \operatorname{diag}(a_{ii})$. Dropping the $\lambda_{\min}$ terms in (2.1) and (2.5), we have two different perturbations to A that allow Cholesky factorization to succeed:

$$A \leftarrow A + \Delta A_1, \quad \Delta A_1 = d_n\lambda_{\max}(A)uI, \tag{2.6}$$

$$A \leftarrow A + \Delta A_2, \quad \Delta A_2 = n(n+1)u\operatorname{diag}(a_{ii}). \tag{2.7}$$

How do these perturbations compare? Ignoring the constants d_n and $n(n+1)$, we have

$$1 \le \frac{\|\Delta A_1\|_2}{\|\Delta A_2\|_2} = \frac{\lambda_{\max}(A)}{\max_i a_{ii}} \le n,$$

where the lower bound is attained for $A = I$ and the upper bound can be approached arbitrarily closely, since it is an equality for the matrix of ones (which is positive

semidefinite). Hence, modulo the constants, ΔA_2 generally has the smaller norm. However, we will not know $\lambda_{\max}(A)$ in practice so will have to estimate it; we will replace it by the lower bound $\max_i a_{ii}$.

We now have two perturbations, $\Delta A_1 = (c_n \max_i a_{ii})I$ and $\Delta A_2 = c_n' \operatorname{diag}(a_{ii})$, where c_n and c_n' are suitable constants. We note that while ΔA_2 makes the same relative perturbation to each diagonal element a_{ii}, ΔA_1 makes a large relative perturbation to any a_{ii} with $a_{ii} \ll \max_i a_{ii}$. This is important because Theorem 2.1 shows that the backward error matrix for Cholesky factorization is bounded relative to the diagonal. Consider the example [14]

$$A = \begin{bmatrix} 10^{40} & \times & \times \\ \times & 10^{20} & \times \\ \times & \times & 1 \end{bmatrix},$$

where the elements marked $\times$ can have any values that yield a positive definite matrix. For double precision (dropping the constant c_n), $\Delta A_1 \approx 10^{40} u I \approx 10^{24} I$ and the $(3,3)$ element of $A + \Delta A_1$ is $\mathrm{fl}(1 + 10^{24}) = 10^{24}$, meaning that a_{33} has been lost. This is a serious loss of information, because ΔA_1 subjects a_{33} to a relative perturbation of order 1 yet by Theorem 2.1 the backward error matrix for Cholesky factorization causes only a relative perturbation of order nu in a_{33}.

Our conclusion is that in a situation where one wishes to increase the diagonal of a positive definite matrix A in order that Cholesky factorization succeeds, adding a multiple of $\operatorname{diag}(a_{ii})$ is better than addding a multiple of I if one wishes to make the smallest relative change to the elements and to minimize the loss of information. In all previous work that we are aware of where a diagonal perturbation is made before beginning Cholesky factorization, a multiple of I has been added—for example [16], [42, Thm. 1].

3. Low precision Cholesky factorization. We now assume that we are given a symmetric positive definite matrix $A \in \mathbb{R}^{n\times n}$ in floating-point arithmetic of precision u and wish to compute a Cholesky factorization in floating-point arithmetic with precision $u_\ell > u$. The most practically important cases are where $(u_\ell, u) =$ (half, single), (half, double), or (single, double). Naively, we might first form $A^{(\ell)} = \mathrm{fl}_\ell(A)$, where fl_ℓ denotes the operation of rounding to precision u_ℓ, and then compute the Cholesky factorization of $A^{(\ell)}$ in precision u_ℓ. For half precision this procedure can fail for two reasons. First, if fp16 is used then the limited range might cause overflow during the rounding. Second, for both bfloat16 and fp16, $A^{(\ell)}$ might not be (sufficiently) positive definite, because a matrix whose smallest eigenvalue is safely bounded away from zero with respect to single precision or double precision may become numerically indefinite under the perturbation induced by rounding to half precision. The second issue can also be encountered when a double precision matrix is rounded to single precision. To overcome these problems we will use scaling and shifting.

3.1. Scaling. The first step is to scale the matrix A to $H = D^{-1}AD^{-1}$, $D = \operatorname{diag}(a_{ii}^{1/2})$, as in (2.2). The matrix D will be kept at precision u. The aim is to reduce the dynamic range in order to avoid overflow and reduce the chance of underflow in conversion to the lower precision. This is needed for fp16, but it is usually not necessary for bfloat16 and single precision. For the rest of the presentation we will always scale, for simplicity of notation. We note that we could choose D to have diagonal elements that are powers of 2, to avoid rounding errors, but in our experience doing so brings no practical benefits.

Two-sided diagonal scaling before rounding to low precision was used in [25] in the context of LU factorization. The scaling used there equilibrates rows and columns; our scaling with D is the natural analogue of that for symmetric positive definite matrices. For Cholesky factorization we need an extra ingredient to ensure a successful factorization, which we consider next.

3.2. Shifting. We now convert H to the lower precision u_ℓ, incorporating a shift to ensure that the lower precision matrix is sufficiently positive definite for Cholesky factorization to succeed, as discussed in section 2. We will shift H by $c_n u_\ell I$, where c_n is a positive integer constant, to obtain $G = H + c_n u_\ell I$. Since the diagonal of H is I, this shift incurs no rounding error and it produces the same result whether we shift in precision u then round or round then shift in precision u_ℓ.

For fp16, in view of the narrow range we will also multiply the shifted matrix by a scalar to bring it close to the overflow level $x_{\max}$, in order to minimize the chance of underflow and of subnormal numbers being produced. So our final precision-u_ℓ matrix is constructed as

$$\begin{aligned} G &= H + c_n u_\ell I, \\ \beta &= 1 + c_n u_\ell, \quad \mu = \theta x_{\max}/\beta, \\ A^{(\ell)} &= \mathrm{fl}_\ell(\mu G), \end{aligned} \tag{3.1}$$

where $\theta \in (0,1)$ is a parameter. Note that $\beta = \max_{ij} |g_{ij}|$, so the largest absolute value of any element of $A^{(\ell)}$ is $\theta x_{\max}$. Note also that since the growth factor for Cholesky factorization is 1 (see, e.g., [21, Prob. 10.4]), there is no danger of overflow during Cholesky factorization of $A^{(\ell)}$.

3.3. Analysis. We now give some analysis to guide the choice of c_n. For simplicity we will ignore the rounding errors in forming H at precision u.

We consider Cholesky factorization of the matrix $A^{(\ell)} = \widehat{H} + \epsilon I$, where $\widehat{H} = \mathrm{fl}_\ell(H)$ and $\epsilon = c_n u_\ell$ (since, as noted above, it does not matter whether we shift or round first, as the shift causes no rounding errors). We ignore the scale factor μ in (3.2), which has negligible effect on the success of the factorization.

We apply the analysis of section 2 to $A^{(\ell)}$. From (2.5) with u replaced by u_ℓ, we know that Cholesky factorization on $A^{(\ell)}$ in precision u_ℓ will succeed if

$$c_n u_\ell = \epsilon > n(n+1)u_\ell - \lambda_{\min}(\widehat{H}). \tag{3.2}$$

How should we choose the constant c_n? Since $\lambda_{\min}(\widehat{H}) \gtrsim -u_\ell$ can be expected, we will take $\epsilon = c_n u_\ell$ with a suitable integer constant c_n. The inequality (3.2) suggests the choice $c_n = n(n+1)$, but this choice is not practical, as then $c_n u_\ell > 1$ for fp16 when $n \geq 45$, so we would be making a perturbation of order 1 to a matrix with elements bounded by 1, and we could not expect to obtain a useful factorization. Clearly, the rounding error analysis that leads to the sufficient condition (2.3) for the success of Cholesky factorization is pessimistic, and based on the arguments in [23, Thm. 3.8] we can reasonably expect $c_n = n$ to be a more realistic constant. However, even this value is uncomfortably large for half precision. We therefore take the pragmatic approach of taking c_n to be a small constant c. If Cholesky factorization fails we will increase c and try again. We will determine experimentally how large c should be for a range of problems of interest.

We note that since $A^{(\ell)}$ has elements bounded by $1 + O(u_\ell)$ and its smallest eigenvalue will be of order $\max(\lambda_{\min}(H), u_\ell)$ we expect $\kappa_2(A^{(\ell)}) \lesssim n/\max(\lambda_{\min}(H), u_\ell)$. In

practice, we have found this upper bound to be often within two orders of magnitude of $\kappa_2(A^{(\ell)})$.

Based on these arguments we propose Algorithm 3.1. For bfloat16 and single precision we do not need to scale by the matrix D and so we can simplify Algorithm 3.1: lines 1–4 can be deleted and line 5 can be replaced by $A^{(\ell)} = \mathrm{fl}_\ell(A + cu_\ell \operatorname{diag}(a_{ii}))$.

Algorithm 3.1 (Cholesky factorization in precision u_ℓ). Given a symmetric positive definite $A \in \mathbb{R}^{n\times n}$ in precision u this algorithm computes an approximate Cholesky factorization $R^TR \approx \mu D^{-1}AD^{-1}$ at precision u_ℓ, where $D = \operatorname{diag}(a_{ii}^{1/2})$. The scalar $\theta \in (0, 1]$ and the positive integer c are parameters.

1: $D = \operatorname{diag}(a_{ii}^{1/2})$, $H = D^{-1}AD^{-1}$ % Set $h_{ii} \equiv 1$ instead of computing it.
2: $G = H + cu_\ell I$
3: $\beta = 1 + cu_\ell$
4: $\mu = \theta x_{\max}/\beta$
5: $A^{(\ell)} = \mathrm{fl}_\ell(\mu G)$
6: Attempt Cholesky factorization $A^{(\ell)} = R^TR$ in precision u_ℓ.
7: **if** Cholesky factorization failed **then**
8: $c \leftarrow 2c$, goto line 2
9: **end if**

3.4. Modified Cholesky factorization. An alternative to Algorithm 3.1 is (ignoring scaling) to compute a modified Cholesky factorization of $A^{(\ell)}$, or to do so if the initial Cholesky factorization fails. Given a symmetric and possibly indefinite A, modified Cholesky factorizations compute a Cholesky or block LDL^T factorization of $A + E$ for a perturbation E whose size is related to how close A is to being positive definite [15]. We will not pursue modified Cholesky factorization for three reasons. First, our specific situation allows a detailed analysis of how to shift and does not require the general machinery of modified Cholesky factorization. Second, Cholesky factorization is faster, because modified Cholesky factorization algorithms either require pivoting (for block LDL^T factorization) or are inherently level-1 BLAS-based and so will suffer a performance penalty. Third, modified Cholesky factorization is much less widely available in program libraries than Cholesky factorization, and is not, to our knowledge, available in libraries for accelerators, such as the MAGMA library[5] [13], [34].

Instead of shifting A by a multiple of its diagonal we could shift individual diagonal elements during the Cholesky factorization, as necessary, to avoid taking the square roots of nonpositive quantities. However, this approach can lead to much larger perturbations, because insufficient perturbations early in the factorization can require larger ones later. To determine appropriate sizes for these selective perturbations one would essentially need to develop a modification of a modified Cholesky factorization algorithm. See [32, sect. 3] for further details.

4. GMRES-based iterative refinement for $Ax = b$. We now use the lower precision Cholesky factorization computed by Algorithm 3.1 to solve a symmetric positive definite linear system $Ax = b$ by iterative refinement. Algorithm 4.1 is an adaptation of GMRES-IR [7], [8] from general linear systems to symmetric positive definite systems, using the approximate Cholesky factors from Algorithm 3.1 as a

[5] https://icl.cs.utk.edu/magma/

preconditioner. This algorithm makes use of a third precision $u_r \leq u$, which in practice will be either $u_r = u$ or $u_r = u^2$. For bfloat16 or single precision, with the simplifications to Algorithm 3.1 mentioned above, we can remove the D^{-1} factors in Algorithm 4.1 and set $\mu = 1$.

The convergence properties of Algorithm 4.1 are essentially the same as those of standard GMRES-IR, as analyzed in [7], [8]. The perturbation on line 2 of Algorithm 3.1 is no larger than the backward error for Cholesky factorization at precision u_ℓ, so it does not affect the analysis. Table 4.1 gives the largest values of $\kappa_\infty(A)$ for which the limiting backward errors and forward errors stated in the table are guaranteed to be achieved, for several different combinations of precisions; the lines with $u_r < u$ follow from [7], [8] while the lines with $u_r = u$ are from [22] and, for the backward error column, are pessimistic.

We note that we are using GMRES-IR instead of standard iterative refinement, for which the update equation is solved by substitution with the Cholesky factors, because it can solve a much wider range of problems. In the table for standard iterative refinement corresponding to Table 4.1 (see [8, Table 7.1]), $\kappa_\infty(A)$ is limited to $1/u_\ell$, which is a severe restriction.

If the refinement process converges quickly and GMRES converges quickly within each refinement step then for large n the dominant cost in Algorithm 4.1 is the cost of computing the Cholesky factors at precision u_ℓ: this is the only $O(n^3)$ flops part of the algorithm. Hence we can expect significant speedups over a solution with Cholesky factorization computed at precision u.

Algorithm 4.1 (Cholesky-based GMRES-IR for a positive definite system) Let a symmetric positive definite $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$ be given in precision u. This algorithm solves $Ax = b$ by GMRES-based iterative refinement, using the approximate Cholesky factorization computed by Algorithm 3.1. The scalar $\theta \in (0, 1]$ and the positive integer c are parameters.

1: Run Algorithm 3.1, obtaining the Cholesky factorization $A^{(\ell)} = R^T R$ in precision u_ℓ.
2: $b^{(\ell)} = \mathrm{fl}_\ell(D^{-1}b)$
3: Solve $A^{(\ell)}y_0 = b^{(\ell)}$ in precision u_ℓ using the Cholesky factors and form $x_0 = \mu D^{-1}y_0$ at precision u.
4: **for** $i = 0\colon i_{\max} - 1$ **do**
5: Compute $r_i = b - Ax_i$ at precision u_r and round r_i to precision u.
6: Solve $MAd_i = Mr_i$ by GMRES at precision u, where $M = \mu D^{-1}R^{-1}R^{-T}D^{-1}$ and matrix–vector products with MA are computed at precision u_r, and store d_i at precision u.
7: $x_{i+1} = x_i + d_i$ at precision u.
8: **if** converged **then**
9: return x_{i+1}, **quit**
10: **end if**
11: **end for**

We now analyze the spectrum of the preconditioner in Algorithm 4.1, under the simplifying assumption that there are no rounding errors, so that the only source of error is the perturbation $cu_\ell I$ in forming $G = H + cu_\ell I$ in Algorithm 3.1. Then we

TABLE 4.1

Bounds on $\kappa_\infty(A)$ such that Algorithm 4.1 using IEEE arithmetic precisions given in the first three columns is guaranteed to converge with the indicated limiting backward or forward errors, where "half", "single", and "double" denote quantities of the order of the unit roundoffs for ha;f precision, single precision, and double precision, respectively. Here, $\mathrm{cond}(A,x) = \| |A^{-1}||A||x| \|_\infty / \|x\|_\infty$. In the u_ℓ column, half can be replaced by bfloat16, in which case the bound on $\kappa_\infty(A)$ must be reduced by a factor 8.

			Backward error		Forward error	
u_ℓ	u	u_r	$\kappa_\infty(A)$	Limit	$\kappa_\infty(A)$	Limit
half	single	single	10^3	single	10^4	$\mathrm{cond}(A,x) \times$ single
half	single	double	10^7	single	10^7	single
half	double	double	10^6	double	10^7	$\mathrm{cond}(A,x) \times$ double
half	double	quad	10^{16}	double	10^{11}	double
single	double	double	10^7	double	10^{10}	$\mathrm{cond}(A,x) \times$ double
single	double	quadruple	10^{16}	double	10^{15}	double

have

$$\begin{aligned} MA &= \mu D^{-1} R^{-1} R^{-T} D^{-1} A \\ &= \mu D^{-1} \big(\mu(H + c u_\ell I)\big)^{-1} D^{-1} A \\ &= D^{-1} \big(H + c u_\ell I\big)^{-1} H D \\ &= D^{-1} F D, \end{aligned}$$

where F has eigenvalues $\lambda_i/(\lambda_i + c u_\ell)$ and the λ_i are the eigenvalues of H. Hence the eigenvalues of MA satisfy

$$\lambda_i(MA) - 1 = \frac{\lambda_i}{\lambda_i + c u_\ell} - 1 = \frac{-c u_\ell}{\lambda_i + c u_\ell}. \tag{4.1}$$

From our assumption that $|\lambda_{\min}(A)| \gtrsim u \lambda_{\max}(A)$, it can be shown that $|\lambda_{\min}(H)| \gtrsim u$ and so for $u_\ell \geq u$ we are assured that $|\lambda_i(MA) - 1| \lesssim 1$ and moreover that $|\lambda_i(MA) - 1| \approx c u_\ell$ for λ_i of order 1. However, since MA is nonnormal no conclusions can be drawn about the speed of convergence of GMRES.

GMRES is a method for general linear systems, so the refinement phase of Algorithm 4.1 does not exploit the symmetry or definiteness of A. The CG method is a natural alternative to GMRES. However, the convergence theory in [7] exploits the backward stability of GMRES. Preconditioned CG is not guaranteed to be backward stable unless the matrix or the preconditioner is well conditioned [17, Eq. (34)], so the analysis no longer applies if we use CG in place of GMRES. The same is true for preconditioned MINRES, since the analysis of [17] is applicable to any iterative method that is based on three-term recurrence relations for the solution and residual. The essential problem is that preconditioning must be two-sided if it is to preserve symmetry and this does not allow a favorable error analysis, whereas in GMRES-IR a left preconditioner is used. Nevertheless, in section 6 we will investigate empirically the behavior of Algorithm 4.1 with GMRES replaced by the CG method on line 6.

5. Cholesky-based GMRES-IR for the least squares problem. The ideas of the previous sections can be adapted to the linear least squares problem $\min_x \|Ax - b\|_2$, where $A \in \mathbb{R}^{m \times n}$ with $m \geq n$ has full rank. The normal equations method solves the normal equations

$$A^T A x = A^T b$$

using Cholesky factorization of A^TA. In general, this method is deprecated by numerical analysts because it has a backward error bound of order $\kappa_2(A)u$ [21, sect. 20.4] and the Cholesky factorization can break down for $\kappa_2(A) > u^{-1/2}$, but it is used by statisticians with some justification [26]. Here, we assume that A is (sufficiently) well conditioned. We propose the GMRES-IR-based least squares solver given in Algorithm 5.1. Another GMRES-IR-based least squares solver that is applicable to a wider range of A and is based on QR factorization is developed by Carson, Higham, and Pranesh [9].

Algorithm 5.1 (Cholesky-based GMRES-IR for the least squares problem) Let a full rank $A \in \mathbb{R}^{m\times n}$, where $m \geq n$, and $b \in \mathbb{R}^m$ be given in precision u. This algorithm solves the least squares problem $\min_x \|b - Ax\|_2$ using Cholesky-based GMRES-IR. The scalar $\theta \in (0, 1]$ and the positive integer c are parameters.

1: Compute $B = AS$, where $S = \mathrm{diag}(1/\|a_j\|_2)$, with a_j the jth column of A.
2: $\mu = \theta x_{\max}$
3: $B^{(\ell)} = \mathrm{fl}_\ell(\mu^{1/2}B)$
4: Compute $C = {B^{(\ell)}}^T B^{(\ell)}$ in precision u_ℓ.
5: Compute the Cholesky factorization $C + cu_\ell \,\mathrm{diag}(c_{ii}) = R^TR$ in precision u_ℓ.
6: **if** Cholesky factorization failed **then**
7: $\quad c \leftarrow 2c$, goto line 5
8: **end if**
9: Form $b^{(\ell)} = \mathrm{fl}_\ell(SA^Tb)$.
10: Solve $R^TRy_0 = b^{(\ell)}$ in precision u_ℓ and form $x_0 = \mu S y_0$ at precision u.
11: **for** $i = 0\colon i_{\max} - 1$ **do**
12: $\quad$ Compute $r_i = A^T(b - Ax_i)$ at precision u_r and round r_i to precision u.
13: $\quad$ Solve $MA^TAd_i = Mr_i$ by GMRES at precision u, where $M = \mu SR^{-1}R^{-T}S$ and matrix–vector products with A^TA are computed at precision u_r, and store d_i at precision u.
14: $\quad x_{i+1} = x_i + d_i$ at precision u.
15: $\quad$ **if** converged **then**
16: $\quad\quad$ return x_{i+1}, **quit**
17: $\quad$ **end if**
18: **end for**

We make some comments on the algorithm. Line 1 produces a matrix B with columns of unit 2-norm. The computation $C = {B^{(\ell)}}^T B^{(\ell)}$ on line 4 produces a symmetric positive definite matrix with constant diagonal elements $\mu = \theta x_{\max}$, so overflow cannot occur for $\theta < 1$. The shift on line 5 is analogous to that in Algorithm 3.1, but here the matrix C is already well scaled and in precision u_ℓ so there is no need to scale C to have unit diagonal.

There are two reasons why explicitly forming $C = {B^{(\ell)}}^T B^{(\ell)}$ in Algorithm 5.1 is reasonable from the numerical stability point of view. First, C is used to form a preconditioner, so the usual problems with forming a cross product matrix (loss of significance and condition squaring) are less of a concern. Second, if we are working in fp16 on an NVIDIA GPU with tensor cores we can accumulate block fused multiply-add operations in single precision when forming C; this leads to a more accurate C, as shown by the error analysis of Blanchard et al. [5].

For the computed $\widehat{R}$ we have

$$\widehat{R}^T\widehat{R} \approx {B^{(\ell)}}^T B^{(\ell)} \approx \mu S A^T A S,$$

or

$$(A^TA)^{-1} \approx \mu S \widehat{R}^{-1}\widehat{R}^{-T} S,$$

so we are preconditioning with an approximation to the inverse of A^TA. We apply the preconditioned operator MA^TA to vectors at precision u_r. Computing $y = A^TAx$ costs $4mn$ flops and $SR^{-1}R^{-T}Sy$ costs another $2n^2+2n$ flops, making $4mn+2n^2+2n$ flops in total. For $m \gg n$ and large n, computing $y = A^TAx$ costs a factor $n/4$ fewer flops than the mn^2 flops needed to form A^TA, while for $m \approx n$ the difference is a factor $n/6$. For large n, even allowing for the fact that the flops we are comparing are at different precisions, as long as GMRES converges quickly the cost of the refinement stage should be negligible compared with the cost of forming A^TA and computing the Cholesky factorization.

What can be said about the convergence of Algorithm 5.1? It differs from Algorithm 4.1 mainly in that it works with A^TA rather than A, so at worst we can expect convergence for $\kappa_\infty(A)$ bounded by the square root of the condition number bounds given in Table 4.1 and with limiting backward error and forward errors a factor $\kappa_\infty(A)$ larger than those in the table, since in translating a backward error from $A^TAx = A^Tb$ to $Ax - b$ we gain a factor as much as $\kappa_\infty(A)$ [26]. However, this is a pessimistic viewpoint because it is known that iterative refinement for the normal equations (and the closely related seminormal equations) works better than a simple condition squaring viewpoint suggests [4, sects. 2.9.3, 6.6.5]. Moreover, if $u_r < u$ we expect to benefit from applying the operator MA^TA in precision u_r. Since we are in any case targetting well conditioned matrices we will not attempt a detailed error analysis.

Related to our work is the Cholesky–QR algorithm for computing a QR factorization $A = QR$. It forms the cross-product matrix A^TA, computes the Cholesky factorization $A^TA = R^TR$, then obtains the orthogonal factor Q as $Q = AR^{-1}$, and this process can be iterated for better numerical stability; see, for example, [16], [39], [40], [41]. Our work differs in that we do not compute Q, we carry out the Cholesky factorization in lower precision than the working precision, and we solve a least squares problem using preconditioned iterative refinement.

Finally, we note that unless A is extremely badly scaled, for bfloat16 or single precision arithmetic we can set $S = I$ and $\mu = 1$ in Algorithm 4.1, since there will usually be no danger of underflow or overflow.

6. Numerical experiments. In this section we perform numerical experiments to investigate how to choose c in Algorithms 3.1 and 5.1, to study the behavior of Algorithms 4.1 and Algorithm 5.1, and to test variants of the algorithms that use the CG method in place of GMRES.

The `chop` function[6] of Higham and Pranesh [24] is used to simulate low precision computation and the AdvanPix Multiprecision Computing Toolbox [30] with `mp.Digits(34)` is used for quadruple precision computation. For half precision we consider only fp16 in our experiments. All the experiments are performed in MATLAB 2019a on a Lenovo ThinkStation with Intel Xeon W-2123 CPU and 32 Gb RAM. The test codes are available at https://github.com/SrikaraPranesh/fp16_Cholesky.

[6] https://github.com/higham/chop.

TABLE 6.1
Symmetric positive definite test matrices chosen from the SuiteSparse Matrix Collection.

Index	Matrix	n	$\kappa_2(A)$	$\max_{i,j} \lvert a_{ij}\rvert$	$\min_{i,j}\{\lvert a_{ij}\rvert : a_{ij} \neq 0\}$
1	Trefethen_300	300	1.77e+03	1.99e+03	1.00e+00
2	mesh2e1	306	2.90e+02	2.46e+02	7.22e-07
3	mesh2em5	306	2.47e+02	1.47e+02	1.59e-06
4	plat362	362	2.18e+11	4.57e-01	3.53e-21
5	mhdb416	416	3.99e+09	1.67e+00	1.39e-19
6	bcsstk06	420	7.57e+06	2.42e+09	7.70e-34
7	bcsstk07	420	7.57e+06	2.42e+09	7.70e-34
8	bcsstm06	420	3.46e+06	7.60e+03	2.20e-03
9	bcsstm07	420	7.62e+03	1.74e+03	9.81e-33
10	nos5	468	1.10e+04	4.27e+05	8.88e-16
11	bcsstk20	485	3.89e+12	1.21e+16	4.88e-04
12	bcsstm20	485	2.55e+05	4.78e+07	1.87e+02
13	494_bus	494	2.42e+06	2.00e+04	1.70e-01
14	Trefethen_500	500	3.19e+03	3.57e+03	1.00e+00

For the precisions $(u_\ell,\ u,\ u_r)$ we take (half, single, double), (half, double, double), (half, double, quad), and (single, double, double).

6.1. Linear systems. We first consider linear systems $Ax = b$ with symmetric positive definite A. Iterative refinement in Algorithm 4.1 is terminated when the normwise backward error satisfies

$$\frac{\|b - A\widehat{x}\|_\infty}{\|A\|_\infty\|\widehat{x}\|_\infty + \|b\|_\infty} \leq nu, \tag{6.1}$$

where $\widehat{x}$ is the computed solution. We tried both GMRES and CG as the iterative solver on line 6 of Algorithm 4.1, and for CG we used the standard symmetrized form of the preconditioned iteration [31, Alg. 9.2]; both the iterations are terminated based on a backward error criterion for the preconditioned system with tolerance 10^{-2} and 10^{-4} for working precisions of single and double, respectively. The right-hand side vectors are generated using `randn(n,1)`, and the random number generator is seeded for reproducibility. In the scaling, we take $\theta = 0.1$. We consider all the symmetric positive definite matrices of dimension 300 to 500 from the SuiteSparse Matrix Collection [10]; their properties are displayed in Table 6.1. Note that several of these matrices are badly scaled. The SuiteSparse matrices are given in double precision. We store them as dense matrices and apply Cholesky factorization without row or column interchanges.

First we investigate the choice of c in Algorithm 3.1 and compare Algorithm 3.1 with an alternative that shifts A: in place of lines 1–5 it carries out the operations

1: $B = A + cu_\ell(\max_i a_{ii})I$
2: $D = \operatorname{diag}(b_{ii}^{1/2})$, $C = D^{-1}BD^{-1}$
3: $\mu = \theta x_{\max}$
4: $A^{(\ell)} = \mathrm{fl}_\ell(\mu C)$

In Table 6.2 we display the minimum positive integer values of c, denoted by c_H for Algorithm 3.1 (since it shifts H) and c_A for the modified algorithm (since it shifts A) such that the Cholesky factorization in fp16 arithmetic succeeds. From the table we see that c_A and c_H are identical for all but two matrices and are always at most 2. The same value $c = 2$ was also sufficient to ensure the success of Cholesky factorization in single precision arithmetic. Based on these results we use $c = 2$ in the

rest of the experiments in this section.

In Table 6.3 we display the iteration counts for the GMRES-based Algorithm 4.1 and its CG-based variant. Several points can be noted.

1. The iteration counts for the GMRES and CG variants are broadly similar, except that the CG variant fails to converge for matrix 4.

2. Both variants work extremely well for (half, single, double), requiring at most one step of refinement and very few iterations of the iterative solver, except for matrix 4. For (half, double, quad) the iteration counts are higher. Note that rounding to single precision and the shift both have a regularizing effect, so the matrix for working precision single is in general better conditioned that that for working precision double.

3. The results for GMRES-IR are consistent with Table 4.1—indeed convergence is achieved for somewhat larger values of $\kappa_\infty(A)$ than the table predicts.

4. "0" iterations denotes that the initial solution computed using the fp16 Cholesky factors satisfied the normwise backward error criterion and no iterative refinement steps were necessary, which is because $\|A\|_\infty\|x\|_\infty$ is so large that (6.1) is satisfied by the initial solution itself; analogous behavior was observed in [25, sect. 4].

We look at some information that gives further insight into these results. Table 6.4 displays $\kappa_2(H)$ for the diagonally scaled matrix H in Algorithm 3.1. The values of $\kappa_2(H)$ are smaller than those of $\kappa_2(A)$ and a factor 10^8 smaller for matrix 5. After conversion to fp16, and before shifting, $\lambda_{\min}(H)$ was negative for matrices 4 and 11, which is consistent with the $\kappa_2(H)$ values. The table also displays the values of the condition numbers of $M_H A$ and $M_A A$, the products (explicitly computed in double precision) of the preconditioners with A, where M_H is obtained from Algorithm 3.1 and M_A from the modified algorithm. The matrices in Table 6.3 with the larger GMRES and CG iteration counts are those with the larger values of $\kappa_2(M_H A)$.

In Table 6.5 we display results for the same experiment but now using the modified version of Algorithm 3.1 in Algorithm 4.1, which shifts A instead of H. Comparing with Table 6.3, we see slightly larger numbers of iterative refinement steps but many more steps for GMRES and CG. This is consistent with Table 6.4, in which the values of $\kappa_2(M_A A)$ are mostly larger than the values of $\kappa_2(M_H A)$. Clearly, the Cholesky factors obtained by shifting A are not nearly as effective preconditioners as those obtained by shifting H.

Since quadruple precision is not supported by hardware, $u_r = u$ was used in GMRES-IR in [18], [20]. We show results for (half, double, double) and (single, double, double) in Table 6.6. We see that (half, double, double) in Table 6.6 generally results in more iterative refinement steps and GMRES or CG iterations than (half, double, quad) in Table 6.3, which is because the preconditioned operator is being applied at a lower precision in Table 6.6. However, the (single, double, double) column shows convergence in significantly fewer iterative refinement steps and GMRES or CG iterations than (half, double, quad) in Table 6.3; here, the higher quality preconditioner outweighs the lesser precision at which it is applied; the last column of Table 6.4 confirms the quality of the preconditioner.

From these experiments we can see that CG performs as well as GMRES in practice within Algorithm 4.1, but we recall that the existing error analysis guarantees convergence of the iterative refinement process only for GMRES, not for CG.

6.2. Least squares problems. To study the behavior of Algorithm 5.1, we consider all full rank rectangular matrices $A \in \mathbb{R}^{m\times n}$ from the SuiteSparse Matrix Collection [10], with $m > n$, $20 \le m \le 2000$, $n \le 400$, and $\kappa_2(A) \le 10^5$. The matrix

TABLE 6.2
Minimum positive integer constants c_H and c_A in Algorithm 3.1 (which shifts H) and its modified version (which shifts A) to ensure the success of Cholesky factorization in fp16 arithmetic.

Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14
c_A	0	0	0	1	0	1	1	0	0	0	1	0	1	0
c_H	0	0	0	2	0	1	1	0	0	0	2	0	1	0

TABLE 6.3
Total number of GMRES and CG iterations in Algorithm 4.1 and its CG variant, for single and double working precisions. Numbers in parentheses denote the number of iterative refinement steps. Failure to converge is denoted by "–".

Index	(half, single, double)		(half, double, quad)	
	GMRES-IR	CG-IR	GMRES-IR	CG-IR
1	0 (0)	0 (0)	4 (2)	4 (2)
2	2 (1)	2 (1)	6 (2)	6 (2)
3	2 (1)	2 (1)	6 (2)	6 (2)
4	362 (1)	362 (1)	382 (2)	– (–)
5	0 (0)	2 (1)	3 (2)	3 (1)
6	0 (0)	8 (1)	25 (2)	26 (2)
7	0 (0)	6 (1)	34 (3)	34 (3)
8	0 (0)	0 (0)	1 (1)	1 (1)
9	0 (0)	0 (0)	4 (2)	4 (2)
10	3 (1)	4 (1)	18 (3)	19 (3)
11	0 (0)	0 (0)	124 (2)	71 (2)
12	0 (0)	0 (0)	1 (1)	1 (1)
13	0 (0)	0 (0)	27 (2)	20 (2)
14	0 (0)	0 (0)	4 (2)	4 (2)

names and properties are listed in Table 6.7. Iterative refinement is terminated when the Frobenius norm backward error

$$\eta_F(\widehat{x}) := \frac{\min\{\, \|[\Delta A,\, \xi\Delta b]\|_F : \|(A+\Delta A)\widehat{x} - (b+\Delta b)\|_2 = \min\,\}}{\|[A,\, \xi b]\|_F} \tag{6.2}$$

is less than nu, where ξ is a parameter that we set to 1. This backward error is computed from the formula [21, sect. 20.7], [37]

$$\eta_F(\widehat{x}) = \frac{\min\{\phi, \sigma_{\min}\left([A \quad \phi(I - rr^{\dagger})]\right)\}}{\|[A,\, \xi b]\|_F}, \tag{6.3}$$

where $r = b - A\widehat{x}$ and

$$\phi = \sqrt{\mu}\frac{\|r\|_2}{\|\widehat{x}\|_2}, \quad \mu = \frac{\xi^2\|\widehat{x}\|_2^2}{1+\xi^2\|\widehat{x}\|_2^2}.$$

The right-hand side vector is generated using `randn(m,1)` and $\theta = 0.1$ is used.

In Table 6.8 we show the minimum values of c on line 5 of Algorithm 5.1 such that Cholesky factorization succeeds in fp16 arithmetic. On the basis of these results, we select $c = 12$ when the low precision is half precision, but when the low precision is single precision we use $c = 2$, which is sufficient in this case.

We consider the solution of a least squares problem using Algorithm 5.1 and a variant that on line 13 uses CG instead of GMRES. Table 6.9 displays the numbers of iterations. Once again, CG-IR and GMRES-IR require similar numbers of iterations.

TABLE 6.4

For $n \times n$ matrices A in Table 6.1, quantities of interest in Algorithm 3.1. Recall that $H = D^{-1}AD^{-1}$, where $D = \mathrm{diag}(a_{ii}^{1/2})$. Here, M_H is the preconditioner from Algorithm 3.1 and M_A is the preconditioner from the modified version of Algorithm 3.1 that shifts A instead of H.

		(half, double, quad)		(single, double, double)
Index	$\kappa_2(H)$	$\kappa_2(M_H A)$	$\kappa_2(M_A A)$	$\kappa_2(M_H A)$
1	4.45e+00	1.00e+00	2.73e+00	1.00e+00
2	1.21e+02	1.10e+00	1.20e+00	1.00e+00
3	1.27e+02	1.09e+00	1.14e+00	1.00e+00
4	1.85e+11	8.52e+08	2.05e+08	2.73e+04
5	6.97e+01	6.23e+00	2.96e+06	1.00e+00
6	3.18e+04	8.72e+01	5.17e+03	1.00e+00
7	3.18e+04	8.72e+01	5.17e+03	1.00e+00
8	1.00e+00	1.00e+00	3.37e+03	1.00e+00
9	2.73e+01	1.10e+00	6.15e+00	1.00e+00
10	1.95e+03	2.63e+00	9.65e+00	1.00e+00
11	1.80e+08	2.32e+08	3.65e+09	1.04e+04
12	1.00e+00	1.00e+00	2.50e+02	1.00e+00
13	7.90e+04	6.82e+02	1.60e+03	1.02e+00
14	4.45e+00	1.00e+00	4.11e+00	1.00e+00

TABLE 6.5

Total number of GMRES and CG iterations in Algorithm 4.1 using modified version of Algorithm 3.1 that shifts A, for single and double working precisions. Numbers in parentheses denote the number of iterative refinement steps. Failure to converge is denoted by "–".

Index	(half, single, double)		(half, double, quad)	
	GMRES-IR	CG-IR	GMRES-IR	CG-IR
1	3 (1)	4 (1)	16 (3)	16 (3)
2	2 (1)	2 (1)	10 (3)	12 (3)
3	2 (1)	2 (1)	9 (3)	9 (3)
4	362 (1)	362 (1)	434 (2)	– (–)
5	416 (1)	416 (1)	492 (2)	397 (1)
6	129 (1)	132 (1)	529 (3)	356 (2)
7	87 (1)	91 (1)	503 (3)	516 (3)
8	11 (1)	12 (1)	45 (3)	23 (2)
9	5 (1)	7 (1)	32 (3)	35 (3)
10	7 (1)	10 (1)	37 (3)	37 (3)
11	485 (1)	485 (1)	485 (1)	– (–)
12	24 (1)	30 (1)	122 (3)	84 (2)
13	47 (1)	61 (1)	248 (3)	235 (3)
14	4 (1)	5 (1)	19 (3)	19 (3)

Table 6.10 shows that matrix 8, which has the highest iteration counts in Table 6.9 has the largest value of $\kappa_2(MA^TA)$. Once again, employing single precision for factorization significantly reduces the number of iterative refinement steps and GMRES or CG iterations. This is because of the superior quality of the preconditioner, as shown in third column of Table 6.10. We also tried (half, double, double); the results were similar to those for (half, double, quad).

Finally, we note that the results are consistent with our expectations from the convergence analysis. The most extreme case is matrix 8 for (half single, double), since it has condition number of order 10^4, which is at the limit of what the second row of Table 4.1 can accept to guarantee convergence (recall that we need to take the square root of the condition number bound). The refinement nevertheless converges.

TABLE 6.6
Total number of GMRES and CG iterations in Algorithm 4.1 for double precision and $u_r = u$. Numbers in parentheses denote the number of iterative refinement steps. Failure to converge is denoted by "–".

Index	(half, double, double)		(single, double, double)	
	GMRES-IR	CG-IR	GMRES-IR	CG-IR
1	3 (3)	3 (3)	1 (1)	1 (1)
2	6 (3)	8 (4)	2 (2)	2 (2)
3	6 (3)	6 (3)	1 (1)	1 (1)
4	562 (3)	– (–)	61 (2)	38 (1)
5	3 (2)	3 (2)	0 (0)	0 (0)
6	38 (5)	32 (4)	2 (1)	2 (1)
7	39 (5)	32 (4)	2 (1)	2 (1)
8	1 (1)	1 (1)	1 (1)	1 (1)
9	4 (4)	4 (4)	1 (1)	1 (1)
10	13 (4)	16 (4)	2 (2)	2 (2)
11	157 (3)	105 (3)	4 (1)	6 (1)
12	1 (1)	1 (1)	1 (1)	1 (1)
13	40 (4)	32 (4)	3 (2)	2 (1)
14	3 (3)	3 (3)	1 (1)	1 (1)

TABLE 6.7
Matrices $A \in \mathbb{R}^{m\times n}$ chosen from the SuiteSparse Matrix Collection.

Index	Matrix	(m,n)	$\kappa_2(A)$	$\max_{i,j} \lvert a_{ij}\rvert$	$\min_{i,j}\{\, \lvert a_{ij}\rvert : a_{ij} \neq 0 \,\}$
1	divorce	(50,9)	1.94e+01	1.00e+00	1.00e+00
2	Cities	(55,46)	2.07e+02	7.10e+01	1.00e+00
3	ash219	(219,85)	3.02e+00	1.00e+00	1.00e+00
4	WorldCities	(315,100)	6.60e+01	1.00e+00	1.00e+00
5	ash331	(331,104)	3.10e+00	2.37e+06	1.85e+04
6	ash608	(608,188)	3.37e+00	2.46e+07	4.00e+00
7	ash958	(958,292)	3.20e+00	1.00e+00	1.00e+00
8	illc1033	(1033,320)	1.89e+04	1.57e+02	3.04e-05
9	well1033	(1033,320)	1.66e+02	1.95e+00	1.24e-02

TABLE 6.8
Minimum positive integer constant c in line 5 of Algorithm 5.1 to ensure the success of Cholesky factorization in fp16 arithmetic.

Index	1	2	3	4	5	6	7	8	9
c	0	4	0	0	0	0	0	12	2

TABLE 6.9
Total number of GMRES and CG iterations in Algorithm 5.1 and its variant, for single and double working precisions. Numbers in parentheses denote the number of iterative refinement steps.

Index	(half, single, double)		(half, double, quad)		(single, double, double)	
	GMRES-IR	CG-IR	GMRES-IR	CG-IR	GMRES-IR	CG-IR
1	4 (2)	4 (2)	9 (3)	9 (3)	2 (2)	2 (2)
2	11 (2)	13 (2)	29 (3)	26 (3)	2 (2)	2 (2)
3	1 (1)	1 (1)	6 (3)	6 (3)	1 (1)	1 (1)
4	3 (1)	3 (1)	15 (3)	14 (3)	2 (2)	2 (2)
5	1 (1)	1 (1)	4 (2)	4 (2)	1 (1)	1 (1)
6	1 (1)	1 (1)	4 (2)	4 (2)	1 (1)	1 (1)
7	1 (1)	1 (1)	4 (2)	4 (2)	1 (1)	1 (1)
8	13 (1)	136 (1)	457 (3)	683 (3)	24 (3)	13 (2)
9	12 (1)	14 (1)	60 (3)	51 (3)	3 (2)	2 (1)

TABLE 6.10
Two norm condition number of the preconditioned matrix MA^TA in Algorithm 5.1. Half and single denote the precision in which the Cholesky factorization was computed.

Index	half	single
1	1.51e+00	1.00e+00
2	3.14e+01	1.00e+00
3	1.01e+00	1.00e+00
4	2.32e+00	1.00e+00
5	1.01e+00	1.00e+00
6	1.01e+00	1.00e+00
7	1.01e+00	1.00e+00
8	1.57e+07	3.03e+02
9	5.77e+01	1.00e+00

6.3. Performance results. Algorithm 4.1 has been implemented for GPUs by Abdelfattah, Tomov, and Dongarra [2]. Their implementation effectively takes (u_ℓ, u, u_r) equal to (half, double, double). The Cholesky factorization is computed in mixed fp16 and fp32 precisions: the factorization of the diagonal blocks and the multiple right-hand side triangular solves are done in fp32, while the matrix multiplications that update the trailing submatrix are done on the tensor cores, exploiting the ability in NVIDIA's Volta, Turing, and Ampere architectures to compute a fused block multiply-add with fp16 arguments in one clock cycle while accumulating the result in single precision. This approach exploits the speed of half precision arithmetic on the tensor cores while achieving close to single precision accuracy for large n, as shown by the error analysis in [5, sec. 4].

On an NVIDIA V100, Abdelfattah, Tomov, and Dongarra, with matrices of dimension up to 42,000, obtained speedups of up to 4.7 over a double precision solver, which is even larger than the speedup of 3.7 mentioned in section 1 for GMRES-IR for general linear systems.

7. Conclusions. As computer hardware increasingly supports multiple precisions of floating-point arithmetic, it is important to investigate whether scientific computing tasks can take advantage of these capabilities. Our focus here has been on exploiting a Cholesky factorization computed at low precision to deliver a solution of a linear system or least squares problem having accuracy and stability commensurate with a higher precision. An immediate obstacle is that rounding a symmetric positive definite matrix to lower precision can produce a matrix for which Cholesky factorization breaks down because the rounded matrix is indefinite or not sufficiently positive definite. We have shown that increasing the diagonal by a small relative amount proportional to the unit roundoff for the lower precision allows Cholesky factorization to succeed and yields computed factors good enough to act as preconditioners for GMRES-based iterative refinement. We have also shown that the standard approach of shifting by a multiple of the identity matrix is not as effective.

The proposed Cholesky-based GMRES-IR algorithm, Algorithm 4.1, is supported by rounding error analysis and can produce backward stable solutions for problems with condition numbers up to around the reciprocal of the unit roundoff of the working precision. The overall performance depends on the speed of convergence of GMRES, and is hard to predict, but in our tests on real-life matrices GMRES often converges quickly, especially for the (half, single, double) and (single, double, double) combinations of precisions.

We found experimentally that GMRES can be replaced by CG with little change

in the number of iterative refinement steps and inner iterations, despite the fact that the supporting error analysis is not applicable.

Our proposed use of low precision Cholesky factorization for solving well conditioned least squares problem via the normal equations, in Algorithm 5.1, also works well in our tests.

Both Algorithm 4.1 and Algorithm 5.1 can benefit from the availability of fast low precision arithmetic and so they promise to be significantly faster than an optimized double precision solver. Indeed, as we explained in section 6.3, Abdelfattah, Tomov, and Dongarra [2] have used Algorithm 4.1 to obtain a speedup of a factor 4.7 over a double precision solver on an NVIDIA V100, and their code is planned to be integrated into the MAGMA library [29], [33].

Acknowledgments. We thank the referees for their helpful suggestions.

REFERENCES

[1] A. Abdelfattah, S. Tomov, and J. Dongarra, *Towards half-precision computation for complex matrices: A case study for mixed-precision solvers on GPUs*, in 2019 IEEE/ACM 10th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA), 2019, pp. 17–24, https://doi.org/10.1109/ScalA49573.2019.00008.

[2] A. Abdelfattah, S. Tomov, and J. Dongarra, *Investigating the benefit of FP16-enabled mixed-precision solvers for symmetric positive definite matrices using GPUs*, in Computational Science—ICCS 2020, V. V. Krzhizhanovskaya, G. Závodszky, M. H. Lees, J. J. Dongarra, P. M. A. Sloot, and S. B. J. Teixeira, eds., no. 12138 in Lecture Notes in Computer Science, Springer International Publishing, 2020, pp. 237–250, https://doi.org/10.1007/978-3-030-50417-5_18.

[3] *Arm A64 Instruction Set Architecture Armv8, for Armv8-A Architecture Profile*, ARM Limited, Cambridge, UK, 2019, https://developer.arm.com/docs/ddi0596/e.

[4] Å. Björck, *Numerical Methods for Least Squares Problems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1996, https://doi.org/10.1137/1.9781611971484.

[5] P. Blanchard, N. J. Higham, F. Lopez, T. Mary, and S. Pranesh, *Mixed precision block fused multiply-add: Error analysis and application to GPU tensor cores*, SIAM J. Sci. Comput., 42 (2020), pp. C124–C141, https://doi.org/10.1137/19M1289546.

[6] I. Buck, *World's fastest supercomputer triples its performance record.* https://blogs.nvidia.com/blog/2019/06/17/hpc-ai-performance-record-summit/, June 2019. Accessed June 24, 2019.

[7] E. Carson and N. J. Higham, *A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems*, SIAM J. Sci. Comput., 39 (2017), pp. A2834–A2856, https://doi.org/10.1137/17M1122918.

[8] E. Carson and N. J. Higham, *Accelerating the solution of linear systems by iterative refinement in three precisions*, SIAM J. Sci. Comput., 40 (2018), pp. A817–A847, https://doi.org/10.1137/17M1140819.

[9] E. Carson, N. J. Higham, and S. Pranesh, *Three-precision GMRES-based iterative refinement for least squares problems*, MIMS EPrint 2020.5, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, Feb. 2020, http://eprints.maths.manchester.ac.uk/2745/.

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

[11] J. W. Demmel, *On floating point errors in Cholesky*, Technical Report CS-89-87, Department of Computer Science, University of Tennessee, Knoxville, TN, USA, Oct. 1989. LAPACK Working Note 14.

[12] J. Dongarra, *Report on the Fujitsu Fugaku system*, Technical Report ICL-UT-20-06, Innovative Computing Laboratory, University of Tennessee, June 2020, https://www.icl.utk.edu/publications/report-fujitsu-fugaku-system.

[13] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, and I. Yamazaki, *Accelerating numerical dense linear algebra calculations with GPUs*, in Numerical Computations with GPUs, V. Kindratenko, ed., Springer International Publishing, Cham, 2014, pp. 3–28, https://doi.org/10.1007/978-3-319-06548-9_1.

[14] Z. Drmač, *Computing eigenvalues and singular values to high relative accuracy*, in Hand-

book of Linear Algebra, L. Hogben, ed., Chapman and Hall/CRC, Boca Raton, FL, USA, second ed., 2014, pp. 59.1–59.21.

[15] H.-R. Fang and D. P. O'Leary, *Modified Cholesky algorithms: A catalog with new approaches*, Math. Programming, 115 (2008), pp. 319–349, https://doi.org/10.1007/s10107-007-0177-6.

[16] T. Fukaya, R. Kannan, Y. Nakatsukasa, Y. Yamamoto, and Y. Yanagisawa, *Shifted CholeskyQR for computing the QR factorization of ill-conditioned matrices*, SIAM J. Sci. Comput., 42 (2020), pp. A477–A503, https://doi.org/10.1137/18M1218212.

[17] A. Greenbaum, *Estimating the attainable accuracy of recursively computed residual methods*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 535–551, https://doi.org/10.1137/S0895479895284944.

[18] A. Haidar, A. Abdelfattah, M. Zounon, P. Wu, S. Pranesh, S. Tomov, and J. Dongarra, *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, Y. Shi, H. Fu, Y. Tian, V. V. Krzhizhanovskaya, M. H. Lees, J. Dongarra, and P. M. A. Sloot, eds., Springer International Publishing, Cham, 2018, pp. 586–600, https://doi.org/10.1007/978-3-319-93698-7_45.

[19] A. Haidar, H. Bayraktar, S. Tomov, J. Dongarra, and N. J. Higham, *Mixed-precision solution of linear systems using accelerator-based computing*, Technical Report ICL-UT-20-05, Innovative Computing Laboratory, University of Tennessee, Knoxville, TN, USA, May 2020, https://www.icl.utk.edu/publications/mixed-precision-solution-linear-systems-using-accelerator-based-computing.

[20] A. Haidar, S. Tomov, J. Dongarra, and N. J. Higham, *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, SC '18 (Dallas, TX), Piscataway, NJ, USA, 2018, IEEE Press, pp. 47:1–47:11, https://doi.org/10.1109/SC.2018.00050.

[21] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second ed., 2002, https://doi.org/10.1137/1.9780898718027.

[22] N. J. Higham, *Error analysis for standard and GMRES-based iterative refinement in two and three-precisions*, MIMS EPrint 2019.19, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, Nov. 2019, http://eprints.maths.manchester.ac.uk/2735/.

[23] N. J. Higham and T. Mary, *A new approach to probabilistic rounding error analysis*, SIAM J. Sci. Comput., 41 (2019), pp. A2815–A2835, https://doi.org/10.1137/18M1226312.

[24] N. J. Higham and S. Pranesh, *Simulating low precision floating-point arithmetic*, SIAM J. Sci. Comput., 41 (2019), pp. C585–C602, https://doi.org/10.1137/19M1251308.

[25] N. J. Higham, S. Pranesh, and M. Zounon, *Squeezing a matrix into half precision, with an application to solving linear systems*, SIAM J. Sci. Comput., 41 (2019), pp. A2536–A2551, https://doi.org/10.1137/18M1229511.

[26] N. J. Higham and G. W. Stewart, *Numerical linear algebra in statistical computing*, in The State of the Art in Numerical Analysis, A. Iserles and M. J. D. Powell, eds., Oxford University Press, 1987, pp. 41–57.

[27] *IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2008 (revision of IEEE Std 754-1985)*, IEEE Computer Society, New York, 2008, https://doi.org/10.1109/IEEESTD.2008.4610935.

[28] Intel Corporation, *BFLOAT16—hardware numerics definition*, Nov. 2018, https://software.intel.com/en-us/download/bfloat16-hardware-numerics-definition. White paper. Document number 338302-001US.

[29] *Matrix algebra on GPU and multicore architectures (MAGMA)*. http://icl.cs.utk.edu/magma/.

[30] *Multiprecision Computing Toolbox*. Advanpix, Tokyo. http://www.advanpix.com.

[31] Y. Saad, *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second ed., 2003, https://doi.org/10.1137/1.9780898718003.

[32] R. B. Schnabel and E. Eskow, *A new modified Cholesky factorization*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 1136–1158, https://doi.org/10.1137/0911064.

[33] S. Tomov, J. Dongarra, and M. Baboulin, *Towards dense linear algebra for hybrid GPU accelerated manycore systems*, Parallel Comput., 36 (2010), pp. 232–240, https://doi.org/10.1016/j.parco.2009.12.005.

[34] S. Tomov, R. Nath, H. Ltaief, and J. Dongarra, *Dense linear algebra solvers for multicore with GPU accelerators*, in Proc. of the IEEE IPDPS'10, Atlanta, GA, April 19-23

2010, IEEE Computer Society, pp. 1–8, https://doi.org/10.1109/IPDPSW.2010.5470941. DOI: 10.1109/IPDPSW.2010.5470941.

[35] T. Trader, *Cray, AMD to Extend DOE's exascale frontier*. https://www.hpcwire.com/2019/05/07/cray-amd-exascale-frontier-at-oak-ridge/, May 2019. Accessed June 27, 2019.

[36] A. van der Sluis, *Condition numbers and equilibration of matrices*, Numer. Math., 14 (1969), pp. 14–23, https://doi.org/10.1007/BF02165096.

[37] B. Waldén, R. Karlson, and J. Sun, *Optimal backward perturbation bounds for the linear least squares problem*, Numer. Linear Algebra Appl., 2 (1995), pp. 271–286, https://doi.org/10.1002/nla.1680020308.

[38] J. H. Wilkinson, *A priori error analysis of algebraic processes*, in Proc. International Congress of Mathematicians, Moscow 1966, I. G. Petrovsky, ed., Mir Publishers, Moscow, 1968, pp. 629–640.

[39] Y. Yamamoto, Y. Nakatsukasa, Y. Yanagisawa, and T. Fukaya, *Roundoff error analysis of the CholeskyQR2 algorithm*, Electron. Trans. Numer. Anal., 44 (2015), pp. 306–326, http://emis.ams.org/journals/ETNA/vol.44.2015/pp306-326.dir/pp306-326.pdf.

[40] I. Yamazaki, S. Tomov, and J. Dongarra, *Mixed-precision Cholesky QR factorization and its case studies on multicore CPU with multiple GPUs*, SIAM J. Sci. Comput., 37 (2015), pp. C307–C330, https://doi.org/10.1137/14M0973773.

[41] I. Yamazaki, S. Tomov, and J. Dongarra, *Stability and performance of various singular value QR implementations on multicore CPU with a GPU*, ACM Trans. Math. Software, 43 (2016), pp. 10:1–10:18, https://doi.org/10.1145/2898347.

[42] Y. Yanagisawa, T. Ogita, and S. Oishi, *A modified algorithm for accurate inverse Cholesky factorization*, Nonlinear Theory and Its Applications, IEICE, 5 (2014), pp. 35–46, https://doi.org/10.1587/nolta.5.35.