SIAM J. SCI. COMPUT.
Vol. 41, No. 4, pp. A2536–A2551

SQUEEZING A MATRIX INTO HALF PRECISION, WITH AN APPLICATION TO SOLVING LINEAR SYSTEMS*

NICHOLAS J. HIGHAM†, SRIKARA PRANESH†, AND MAWUSSI ZOUNON‡

Abstract. Motivated by the demand in machine learning, modern computer hardware is increasingly supporting reduced precision floating-point arithmetic, which provides advantages in speed, energy, and memory usage over single and double precision. Given the availability of such hardware, mixed precision algorithms that work in single or double precision but carry out part of a computation in half precision are now of great interest for general scientific computing tasks. Because of the limited range of half precision arithmetic, in which positive numbers lie between 6×10^{-8} and 7×10^4, a straightforward rounding of single or double precision data into half precision can lead to overflow, underflow, or subnormal numbers being generated, all of which are undesirable. We develop an algorithm for converting a matrix from single or double precision to half precision. It first applies two-sided diagonal scaling in order to equilibrate the matrix (that is, to ensure that every row and column has ∞-norm 1), then multiplies by a scalar to bring the largest element within a factor $\theta \leq 1$ of the overflow level, and finally rounds to half precision. The second step ensures that full use is made of the limited range of half precision arithmetic, and θ must be chosen to allow sufficient headroom for subsequent computations. We apply the new algorithm to GMRES-based iterative refinement (GMRES-IR), which solves a linear system $Ax = b$ with single or double precision data by LU factorizing A in half precision and carrying out iterative refinement with the correction equations solved by GMRES preconditioned with the low precision LU factors. Previous implementations of this algorithm have used a crude conversion to half precision that our experiments show can cause slow convergence of GMRES-IR for badly scaled matrices or failure to converge at all. The new conversion algorithm computes ∞-norms of rows and columns of the matrix and its cost is negligible in the context of LU factorization. We show that it leads to faster convergence of GMRES-IR for badly scaled matrices and thereby allows a much wider class of problems to be solved.

Key words. diagonal scaling, half precision arithmetic, fp16, overflow, underflow, subnormal numbers, iterative refinement, linear system, mixed precision, GMRES, preconditioning

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

DOI. 10.1137/18M1229511

1. Introduction.

The landscape of scientific computing is changing, because of the growing availability and usage of low precision floating-point arithmetic. The 2008 revision of IEEE standard 754 introduced a 16-bit floating point format, known as half precision (fp16) [19]. Although defined only as a storage format, it has been widely adopted for computing, and is supported by the NVIDIA P100 and V100 GPUs and the AMD Radeon Instinct MI25 GPU. On such hardware, half precision operations run at least twice as fast as single precision ones, and up to 8 times faster on the NVIDIA V100 because of its tensor cores. The Summit machine at Oak Ridge National Laboratory, which heads the June 2018, November 2018, and June 2019 TOP 500 lists (www.top500.org), has 4608 nodes, with 6 NVIDIA V100 GPUs per

*Submitted to the journal's Methods and Algorithms for Scientific Computing section November 28, 2018; accepted for publication (in revised form) June 7, 2019; published electronically August 1, 2019.

https://doi.org/10.1137/18M1229511

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).

‡Numerical Algorithms Group, Suite 21A, Manchester One, 53 Portland Street, Manchester, M1 3LD, UK (mawussi.zounon@nag.co.uk).

node [28], [32]. To obtain the best performance from such hardware, it is clearly crucial to develop algorithms that can exploit half precision arithmetic. Indeed exaop performance has already been achieved on Summit through careful use of the tensor cores [11], [21].

As regards future machines, Fujitsu has announced that the A64FX Arm processor that will power Japan's first exascale computer supports half precision arithmetic running at twice the speed of single precision [9].

Another form of half precision arithmetic is the bfloat16 format[1] used by Google in its tensor processing units. The bfloat16 format allocates more bits to the exponent than fp16 and fewer to the significand, so it has a larger range but less precision. Intel gives a detailed specification of the bfloat16 format, which will be supported in its forthcoming Nervana Neural Network Processor [30] and Cooper Lake processors [10], in a recent white paper [20].

While machine learning is one of the main drivers for the development of half precision in hardware [13], [33], half precision is also being used in other applications such as weather and climate modelling [6], [29].

Here, we are concerned with the fundamentally important problem of solving a general linear system of equations $Ax = b$. We suppose that $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$ are stored in a working precision of double precision (fp64) or single precision (fp32), and that we want to solve $Ax = b$. The standard approach is to compute an LU factorization in the working precision and solve two triangular systems, obtaining a computed solution $\widehat{x}_w$ say. Our aim is to solve the problem substantially faster by exploiting half precision arithmetic, yet obtain a computed solution with the same level of backward and forward errors as $\widehat{x}_w$, with minimal restrictions on A.

Algorithms satisfying most of these requirements have already been developed, based on iterative refinement [3], [4], [14], [15], [16]. Such algorithms round the entries of A to half precision, compute the LU factors in half precision, compute a solution using the low accuracy LU factors, and then use iterative refinement to generate a solution of working precision quality. Haidar et al. [14], [15] show that on NVIDIA GPUs, using the tensor core features, this approach leads to a speedup of up to 4 over highly optimized double precision solvers, with a reduction in power consumption of up to 80%. In this context, standard iterative refinement converges only for very well conditioned matrices ($\kappa(A) \lesssim 10^4$, where $\kappa(A) = \|A\|\|A^{-1}\|$), but the range of solvable problems is greatly enlarged by using GMRES [31] preconditioned with the low precision LU factors to solve the update equations [3], [4].

However, this usage of half precision has the drawback that the elements of the matrix A may overflow or underflow when rounded to half precision. To see why, consider Table 1.1, which shows key characteristics of half, single, and double precision arithmetic, as well as bfloat16. When rounded to fp16, any double precision number with magnitude on the interval $[6.6 \times 10^4, 1.8 \times 10^{308}]$ will overflow and any double precision number with magnitude on the interval $[2.2 \times 10^{-308}, 5.9 \times 10^{-8}]$ will underflow. Many matrices of practical interest have entries in these ranges.

Overflow is unrecoverable, because LU factorization cannot produce useful results for a matrix with infinities amongst its entries. Underflow during the rounding could cause a serious loss of information; moreover the rounded matrix could have a zero row or column and hence be structurally singular. It is also desirable to avoid producing subnormal numbers, which lie between the underflow threshold and the smallest normalized floating-point number, as they have less precision than normalized num-

[1] https://en.wikipedia.org/wiki/Bfloat16_floating-point_format

TABLE 1.1

Parameters for bfloat16, fp16, fp32, and fp64 arithmetic, to three significant figures: unit roundoff u, smallest positive (subnormal) number $x_{\min}^s$, smallest normalized positive number $x_{\min}$, and largest finite number $x_{\max}$. In Intel's bfloat16 specification subnormal numbers are not supported [20].

	u	$x_{\min}^s$	$x_{\min}$	$x_{\max}$
bfloat16	3.91×10^{-3}	9.18×10^{-41}	1.18×10^{-38}	3.39×10^{38}
fp16	4.88×10^{-4}	5.96×10^{-8}	6.10×10^{-5}	6.55×10^{4}
fp32	5.96×10^{-8}	1.40×10^{-45}	1.18×10^{-38}	3.40×10^{38}
fp64	1.11×10^{-16}	4.94×10^{-324}	2.22×10^{-308}	1.80×10^{308}

bers and they incur a performance penalty if handled in software.[2] The limited range of fp16 may therefore create a fundamental problem in using it in this context—and indeed more generally.

In the work on iterative refinement in [14], [15], [16] elements that overflow during conversion to fp16 are mapped to the nearest finite number, $\pm x_{\max}$. As we will show, for badly scaled real-life matrices this approach can lead to slow convergence, so a more sophisticated strategy is needed.

In this work we investigate how to convert a matrix A from single or double precision to half precision without overflow and with a reduced chance of underflow and of subnormal numbers being generated. Essentially, we squeeze the matrix into half precision by a two stage process: first we apply a two-sided diagonal scaling to ensure that for every row and column the largest absolute value of the elements is 1; then we multiply by a scalar in order to expand the range of the matrix entries to occupy most of the fp16 range.

We focus on fp16 arithmetic in this work, but our algorithms and analysis apply equally well to bfloat16, which is, however, much less prone to overflow and underflow.

In the next section we present algorithms for handling the conversion to fp16 without reference to any specific problem, as the approaches presented are widely applicable. In section 3 we focus on mixed precision iterative refinement for the $Ax = b$ application. Then in section 4 we perform numerical experiments to test the effectiveness of the proposed scaling algorithms. In section 5 we discuss an alternative scaling algorithm that employs a rank-1 update and explain its pros and cons. We close with some concluding remarks in section 6.

2. Algorithms for converting a matrix to half precision. One response to overflow in converting a matrix to fp16 is simply to map any elements too large for fp16 to $\pm x_{\max}$. Algorithm 2.1 is a little more general: it maps any number outside the interval $[-\theta x_{\max}, \theta x_{\max}]$ to the nearest point on that interval, where $\theta \in (0, 1]$ is a parameter. We will explain the role of θ in section 2.1. Here, $\mathrm{fl_h}$ denotes the operator that rounds to fp16 and sign is the function that maps positive real numbers to 1, negative real numbers to -1, and 0 to 0. Algorithm 2.1, with $\theta = 1$, is the approach used in [14], [15], [16].

Another approach is to scale the matrix before rounding, as in Algorithm 2.2, which again ensures that the largest entry in magnitude is $\theta x_{\max}$.

Algorithms 2.1 and 2.2 have the following drawbacks. Algorithm 2.1 makes a potentially large perturbation for every element that overflows, so it can make a large change to the matrix. When $\max_{i,j} |a_{ij}| > x_{\max}$, Algorithm 2.2 reduces every element

[2] https://devblogs.nvidia.com/cuda-pro-tip-flush-denormals-confidence/, https://en.wikipedia.org/wiki/Denormal_number

Algorithm 2.1 (round then replace infinities). This algorithm rounds $A \in \mathbb{R}^{n\times n}$ to the fp16 matrix $A^{(h)}$, mapping any elements of modulus larger than $\theta x_{\max}$ to $\pm\theta x_{\max}$.

1: $A^{(h)} = \mathrm{fl_h}(A)$
2: For every i and j such that $|a_{ij}^{(h)}| \geq \theta x_{\max}$, set $a_{ij}^{(h)} = \mathrm{sign}(a_{ij})\theta x_{\max}$.

Algorithm 2.2 (scale by scalar then round). This algorithm rounds $A \in \mathbb{R}^{n\times n}$ to the fp16 matrix $A^{(h)}$, scaling all elements to avoid overflow. $\theta \in (0, 1]$ is a parameter.

1: $a_{\max} = \max_{i,j} |a_{ij}|$
2: $\mu = \theta x_{\max}/a_{\max}$
3: $A^{(h)} = \mathrm{fl_h}(\mu A)$

in magnitude, so it increases the risk of underflow. We illustrate the algorithms with the matrix, adapted from [12, p. 45],

$$A = \begin{bmatrix} 1 & 1 & \alpha \\ 1 & -1 & \alpha \\ 1 & 1 & 0 \end{bmatrix}, \quad \alpha \gg 1, \tag{2.1}$$

and we take $\theta = 1$ in both algorithms. For $\alpha > x_{\max}$, Algorithm 2.1 produces

$$A^{(h)} = \begin{bmatrix} 1 & 1 & x_{\max} \\ 1 & -1 & x_{\max} \\ 1 & 1 & 0 \end{bmatrix},$$

which is a large rank-1 change to A if $\alpha \gg x_{\max}$. Algorithm 2.2 produces $A^{(h)} = \mathrm{fl_h}((x_{\max}/\alpha)A)$, which (in view of Table 1.1) has first and second columns consisting entirely of subnormal numbers if

$$10^9 \approx \frac{x_{\max}}{x_{\min}} < \alpha \leq \frac{x_{\max}}{x_{\min}^s} \approx 10^{12}$$

and has zero first and second columns (hence is singular) if $\alpha \geq 2 \times 10^{12}$.

A second weakness of Algorithms 2.1 and 2.2 is that they do nothing to avoid underflow of elements of A. Consider the matrix

$$A(\delta) = \begin{bmatrix} 1 & \delta & \delta \\ \delta & \delta & -\delta \\ 1 & -\delta & \delta \end{bmatrix}, \quad 0 < \delta < 1, \tag{2.2}$$

and take $\theta = 1$ in both algorithms. If $\delta \leq 10^{-8}$, Algorithm 2.1 produces $A^{(h)} = A(0)$, which has zero second and third columns, while Algorithm 2.2 produces the same result if $\delta \leq 9 \times 10^{-13}$.

To address these issues we now consider a more sophisticated class of algorithms that carry out two-sided diagonal scaling prior to converting to fp16: they replace A by RAS, where

$$R = \mathrm{diag}(r_i), \quad S = \mathrm{diag}(s_i), \qquad r_i, s_i > 0, \quad i = 1\colon n.$$

Such scaling algorithms have been developed in the context of linear systems and linear programming problems. Despite the large literature on scaling such problems,

no clear conclusions are available on when or how one should scale; see [8] for a recent experimental study. In any case, our use of these scalings is different from that in previous studies, where the aim of scaling has been to reduce a condition number or to speed up the convergence of an iterative method applied to the scaled matrix. We scale in order to help squeeze a single or double precision matrix into half precision, with a particular application to using the resulting half precision LU factors for iterative refinement.

Our usage of two-sided diagonal scaling is given in Algorithm 2.3.

Algorithm 2.3 (two-sided diagonal scaling then round). This algorithm rounds $A \in \mathbb{R}^{n\times n}$ to the fp16 matrix $A^{(h)}$, scaling all elements to avoid overflow. $\theta \in (0,1]$ is a parameter.

1: Apply any two-sided diagonal scaling algorithm to A, to obtain diagonal matrices R, S.
2: Let β be the maximum magnitude of any entry of RAS.
3: $\mu = \theta x_{\max}/\beta$
4: $A^{(h)} = \mathrm{fl_h}(\mu(RAS))$

We now consider two different algorithms for determining R and S; both algorithms are carried out at the working precision. We first consider row and column equilibration, implemented in Algorithm 2.4. This scaling ensures that every row and column has maximum element in modulus equal to 1—that is, each row and column is equilibrated. The LAPACK routines `xyyEQU` carry out this form of scaling [2].

Algorithm 2.4 (row and column equilibration). Given $A \in \mathbb{R}^{n\times n}$, which is assumed to have no zero row or column, this algorithm computes nonsingular diagonal matrices R and S such that $B = RAS$ has the property that $\max_k |b_{ik}| = \max_k |b_{ki}| = 1$ for all i.

1: **for** $i = 1\colon n$ **do**
2: $\quad r_i = \|A(i,:)\|_\infty^{-1}$
3: **end for**
4: $R = \mathrm{diag}(r)$
5: $\widetilde{A} = RA$ % $\widetilde{A}$ is row equilibrated.
6: **for** $j = 1\colon n$ **do**
7: $\quad s_j = \|\widetilde{A}(:,j)\|_\infty^{-1}$
8: **end for**
9: $S = \mathrm{diag}(s)$

We note that Algorithm 2.4 applies the row scaling before the column scaling. If the column scaling is applied first a different result may be obtained, and while the result has the same characteristic scaling property the conditioning may be very different. For the matrix (2.1), Algorithm 2.4 yields

$$RAS = \begin{bmatrix} \alpha^{-1} & \alpha^{-1} & 1 \\ \alpha^{-1} & -\alpha^{-1} & 1 \\ 1 & 1 & 0 \end{bmatrix}, \quad \kappa_\infty(RAS) \approx \alpha,$$

whereas scaling columns then rows gives

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 0 \end{bmatrix}, \quad \kappa_\infty(RAS) \approx 1.$$

However, in general there is no reason to prefer to scale by rows or columns first.

If the input matrix is symmetric then Algorithm 2.4 will generally destroy symmetry. A symmetry-preserving two-sided scaling is proposed by Knight, Ruiz, and Uçar [22]. The algorithm, given in Algorithm 2.5, is iterative and scales simultaneously on both sides rather than sequentially on one side then the other as in Algorithm 2.4. It has the properties that

1. if $A = A^T$ then $R = S$;
2. the algorithm is permutation invariant: if it produces the scaling RAS for A then it produces the scaling $P_1 R P_1^T (P_1 A P_2) P_2^T S P_2$ for $P_1 A P_2$;
3. it is linearly convergent, with asymptotic error constant $1/2$.

Algorithm 2.5 (symmetry-preserving row and column equilibration). Given $A \in \mathbb{R}^{n\times n}$, which is assumed to have no zero row or column, this algorithm computes nonsingular diagonal matrices R and S such that $B = RAS$ has the property that $\max_k |b_{ik}| = \max_k |b_{ki}| = 1$ for all i and $R = S$ if $A = A^T$. tol is a convergence tolerance.

```
1:  R = I, S = I
2:  repeat
3:      for i = 1: n do
4:          r_i = ||A(i,:)||_∞^{-1/2}
5:          s_i = ||A(:,i)||_∞^{-1/2}
6:      end for
7:      A = diag(r) A diag(s)
8:      R = diag(r) R
9:      S = S diag(s)
10: until max_i |r_i − 1| ≤ tol and max_i |s_i − 1| ≤ tol
```

Applied to A in (2.1), Algorithm 2.5 gives (converging after one iteration)

$$RAS = \begin{bmatrix} \alpha^{-1/2} & \alpha^{-1/2} & 1 \\ \alpha^{-1/2} & -\alpha^{-1/2} & 1 \\ 1 & 1 & 0 \end{bmatrix}, \quad \kappa_\infty(RAS) \approx \alpha^{1/2},$$

which in terms of conditioning and the size of the smallest nonzero entry is intermediate between the matrices from Algorithm 2.4 and the variant that scales the columns first.

We note that β in line 2 of Algorithm 2.3 is equal to 1 for Algorithms 2.4 and 2.5. This is immediate for Algorithm 2.4 and is true for Algorithm 2.5 as long as at least one iteration is performed (of course, row and column equilibration in Algorithm 2.5 is achieved only in the limit).

Other two-sided diagonal scaling algorithms exist, including Hungarian scaling [18] and other algorithms discussed by Larsson [23]. We have tried several of them and not found other algorithms to have any advantage over Algorithms 2.4 or 2.5 in our linear system application, so we do not discuss them here.

2.1. Discussion. How do Algorithm 2.1, Algorithm 2.2, and Algorithm 2.3 with Algorithm 2.4 or Algorithm 2.5 compare for converting a matrix to fp16? Depending on the usage of the fp16 matrix $A^{(h)}$, several possible criteria may be of interest, of which we mention three.

- As few elements as possible should underflow or become nonzero but unnormalized.

TABLE 3.1

Bounds on $\kappa_\infty(A)$ such that GMRES-IR with precisions given in the first three columns is guaranteed to converge with the limiting backward or forward errors shown in the final three columns, where "single" and "double" denote quantities of the order of the unit roundoffs for single precision and double precision, respectively.

				Backward error		
u_h	u	u_r	$\kappa_\infty(A)$	Normwise	Componentwise	Forward error
half	single	double	10^8	single	single	single
half	double	quad	10^{12} *	double	double	double

* This bound is from the forward error analysis in [4]; the backward error analysis requires only $\kappa_\infty(A) \leq 10^{16}$.

- Key properties of the original matrix, such as singular values or condition number, should be preserved as much as possible.
- The inverses of the computed LU factors of $A^{(h)}$ should form an effective preconditioner for A.

In section 4 we give numerical experiments that focus on the first and last of these criteria. We look at the percentage of nonzero elements of a matrix that underflow after scaling and rounding to fp16 as well as the performance of GMRES-based iterative refinement.

Now we discuss the parameter θ in Algorithms 2.1, 2.2, and 2.3. The best choice will be problem-dependent. The aim is to take θ close to 1 in order to maximize the use of the fp16 range and thereby to reduce the chance of underflow (which in the worst case could make the matrix singular) and of producing subnormal numbers. However, θ needs to be sufficiently less than 1 to allow headroom for subsequent computations not to overflow. It is unusual in scientific computing to work close to the overflow level, but given the constant relative spacing of the floating-point number system there is no reason not to do so. This reasoning is analogous to the "expose to the right" approach in digital photography,[3] whereby at the capture stage one maximizes the exposure without overexposing, thereby making full use of the dynamic range representable in a digital image.

3. Application to $Ax = b$. Now we employ the algorithms of the previous section within the solution of $Ax = b$ using GMRES-based iterative refinement (GMRES-IR) in three precisions [3], [4]. GMRES-IR carries out iterative refinement at the working precision u using LU factors computed at a precision $u_h \geq u$, and with residuals computed at a precision $u_r \leq u$. It solves the update equation in iterative refinement, which has the residual as right-hand side, using GMRES preconditioned with the LU factors. Table 3.1 shows the limiting backward and forward errors that are guaranteed by the analysis of [3], [4] for $\kappa_\infty(A)$ satisfying the specified bounds.

We convert the matrix to fp16 by one of Algorithms 2.1–2.3 before it is factorized. The overall algorithm is Algorithm 3.1, and the choices of precisions of interest here are (u_h, u, u_r) = (half, double, quad) and (u_h, u, u_r) = (half, single, double). The preconditioned matrix MA is not formed explicitly, but rather its action on a vector is obtained by two triangular solves and a multiplication with A, both at precision u_r.

In Algorithm 3.1, the refinement and the GMRES solves work with the original system, so backward errors and residuals are measured on the original data. It would be possible to work with the diagonally scaled system, but since norms are not in-

[3] https://en.wikipedia.org/wiki/Exposing_to_the_right

Algorithm 3.1 (GMRES-IR) Let $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$ be given in precision u. This algorithm solves $Ax = b$ using GMRES-based iterative refinement with the conversion to fp16 done by one of Algorithms 2.1, 2.2, and 2.3. $\theta \in (0, 1]$ is a parameter.

Obtain $A^{(h)}$ from one of
  (a) Algorithm 2.1. Set $\mu = 1$, $R = I$, $S = I$.
  (b) Algorithm 2.2. Set $R = I$, $S = I$.
  (c) Algorithm 2.3 together with Algorithm 2.4 or Algorithm 2.5.
$b^{(h)} = \mathrm{fl_h}(Rb)$
Compute an LU factorization $A^{(h)} \approx \widehat{L}\widehat{U}$ in precision u_h.
Solve $A^{(h)}y_0 = b^{(h)}$ in precision u_h using the LU factors and form $x_0 = \mu S y_0$ at precision u.
for $i = 0\colon i_{\max} - 1$ **do**
  Compute $r_i = b - Ax_i$ at precision u_r and round r_i to precision u.
  Solve $MAd_i = Mr_i$ by GMRES at precision u, where $M = \mu S\widehat{U}^{-1}\widehat{L}^{-1}R$ and where matrix–vector products with MA are computed at precision u_r, and store d_i at precision u.
  $x_{i+1} = x_i + d_i$ at precision u.
  if converged **then**
    return x_{i+1}, **quit**
  end if
end for

variant under diagonal scaling this would change the convergence test and hence the attained backward and forward errors, so it should only be done with knowledge of the problem. This is why LAPACK, the MATLAB backslash, and other standard solvers do not automatically scale linear systems. For further discussion of the role of scaling in solving $Ax = b$ see [17, sect. 9.8].

Now we consider the effect of the scaling in Algorithm 3.1. A complication in analyzing LU factors of diagonally scaled matrices is that scaling might change the pivot sequence. Therefore to simplify the analysis we assume that the pivot sequence is unaffected by R and S. We consider a matrix whose entries are within the normalized fp16 range and assume that μ and the diagonal elements of R and S are powers of 2. Then $\mathrm{fl_h}(\mu RAS) = \mu R\,\mathrm{fl_h}(A)S = \mu RA^{(h)}S$. If $A^{(h)} = L_1U_1$ and $\mu RA^{(h)}S = LU$ are LU factorizations then $A^{(h)} = R^{-1}LR \cdot \mu^{-1}R^{-1}US^{-1} \equiv \widetilde{L}\widetilde{U}$ is also an LU factorization. Since an LU factorization is unique, $\widetilde{L} = L_1$ and $\widetilde{U} = U_1$. Hence the matrix M in Algorithm 3.1 satisfies (in exact arithmetic)

$$M = \mu SU^{-1}L^{-1}R = \mu SU^{-1}R \cdot R^{-1}L^{-1}R = \widetilde{U}^{-1}\widetilde{L}^{-1} = U_1^{-1}L_1^{-1}.$$

The latter matrix corresponds to the unscaled problem, so the matrix MA to which GMRES-IR is applied is the same with or without scaling, and so the scaled and unscaled algorithms are equivalent. There is even a numerical equivalence, stemming from the fact that the rounding errors in the LU factorizations of RAS and A scale in the same way for a fixed pivot sequence, given our assumptions on R and S [17, sect. 9.8]. In practice, the pivot sequences for A and RAS may be different, but in this case it is hard to compare the matrices M obtained with and without scaling. Nevertheless, the purpose of scaling is to fit A into fp16, and this argument shows that if it accelerates iterative refinement it will do so indirectly, through allowing better LU factors to be computed.

The initial linear system that we solve for x_0 in Algorithm 3.1, which reduces to triangular systems with $\widehat{L}$ and $\widehat{U}$, may need scaling in order to avoid overflow and unnecessary underflow. How to scale triangular systems is well understood [1], [7], [24], so we do not discuss it here. In our numerical experiments we did not scale the triangular systems. We also note that any scaling necessary for b is best done at the triangular solve stage.

Finally, we consider how to choose the parameter θ in Algorithms 2.1, 2.2, and 2.3. We need to ensure that the elements of L and U do not overflow. Assume that we are using LU factorization with partial pivoting, $PA = LU$. The lower triangular matrix L has elements bounded in modulus by 1 and $|u_{ij}| \le \rho_n \max_{i,j} |a_{ij}|$ for all i and j, where ρ_n is the growth factor [17, sects. 9.3, 9.4]. Therefore we need $\theta \le \rho_n^{-1}$. In practice, ρ_n is not large, so to avoid overflow in the LU factors we might take $\theta = 0.1$ (say), for which the final fp16 matrix satisfies $\max_{i,j} |a_{ij}| = 0.1 x_{\max}$.

As well as avoiding overflow we also wish to avoid zero pivots. A pivot u_{kk} underflows to zero if $|u_{kk}| < x^s_{\min}$. From the inequality

$$\begin{aligned} |u_{kk}^{-1}| &= |e_k^T U^{-1} e_k| = |e_k^T A^{-1} P^T L e_k| = |(PA^{-T} e_k)^T L e_k| \\ &\le \|PA^{-T} e_k\|_1 \|Le_k\|_\infty \le \|A^{-1}\|_\infty, \end{aligned}$$

we see that if u_{kk} underflows then $\|A^{-1}\|_\infty^{-1} \le |u_{kk}| < x^s_{\min}$, in which case

$$\kappa_\infty(A) = \|A\|_\infty \|A^{-1}\|_\infty \ge \max_{i,j} |a_{ij}| \|A^{-1}\|_\infty \ge \frac{\theta x_{\max}}{x^s_{\min}}. \tag{3.1}$$

For half precision the lower bound is 1.09×10^{11} with $\theta = 0.1$, and in practice we will usually have $\kappa_\infty(A) \ll 10^{11}$ after applying diagonal scaling. Therefore we can conclude that it is unlikely that a pivot will underflow.

4. Numerical experiments. In this section we perform numerical experiments to evaluate the different approaches to dealing with the conversion from double or single precision to half precision within GMRES-IR. The convergence test that we use in line 9 of Algorithm 3.1 is

$$\frac{\|b - Ax_{i+1}\|_\infty}{\|A\|_\infty \|x_{i+1}\|_\infty + \|b\|_\infty} \le nu, \tag{4.1}$$

in which the quantity on the left is the normwise backward error [17, Thm. 7.1]. We first demonstrate the ineffectiveness of Algorithms 2.1 and 2.2 and then we study the performance of Algorithm 2.3. Throughout this section performance is measured in terms of the total number of GMRES iterations required in Algorithm 3.1. We use the MATLAB codes for GMRES-based iterative refinement from https://github.com/eccarson/ir3, with minor modifications. The inner GMRES 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, and a maximum of 10 iterative refinement steps are performed. Finally we take $\text{tol} = 10^{-4}$ in Algorithm 2.5.

We use Moler's `fp16` class for half precision computations [25], [26], and for quadruple precision we use the Advanpix Multiprecision Computing Toolbox [27] with the setting `mp.Digits(34)`, which is compliant with the IEEE 754-2008 standard [19]. We take $\theta = 0.1$ in all the numerical experiments. Two combinations of precisions are considered: (u_h, u, u_r) = (half, double, quad) and (u_h, u, u_r) = (half, single, double).

The numerical experiments were performed in MATLAB R2018b on a Mac laptop with Intel Core i5, and 8 Gb of RAM. We use 13 matrices from the SuiteSparse Matrix

TABLE 4.1

Selected matrices from the SuiteSparse Matrix Collection. "∗" denotes that the matrix is symmetric. The categories that the matrices come from are 1–2*: chemical process simulation,* 3–6*: CFD,* 7*: materials problem,* 8*: optimal control,* 9–11*: structural problems, and* 12–13*:* 2D/3D *problem sequence.*

Index	Matrix	n	$\kappa_\infty(A)$	$\max_{i,j}\lvert a_{ij}\rvert$	$\min_{i,j}\{\lvert a_{ij}\rvert : a_{ij} \neq 0\}$
1	west0132	132	1.05e+12	3.16e+05	3.31e-05
2	west0167	167	6.76e+10	3.16e+05	3.30e-05
3	pores_1	30	2.49e+06	2.46e+07	4.00e+00
4	saylr1	238	1.59e+09	3.06e+08	7.19e-04
5	steam1	240	3.11e+07	2.17e+07	1.48e-07
6	steam3	80	7.64e+10	1.73e+10	1.08e-01
7	arc130	130	1.20e+12	1.05e+05	7.71e-31
8∗	tumorAntiAngiogenesis_2	305	1.98e+10	5.15e+05	8.50e-06
9∗	bcsstk01	48	1.59e+06	2.47e+09	3.33e+03
10∗	lund_a	147	5.44e+06	1.50e+08	1.22e-04
11∗	nos1	237	2.53e+07	1.22e+09	8.00e+05
12	fs_183_3	183	1.59e+14	8.45e+08	2.35e-10
13	fs_183_1	183	1.08e+14	8.22e+08	1.81e-25

Collection[4] [5], which are selected as representative of all 36 $n \times n$ matrices in the collection with $n \leq 400$ and maximum absolute value of a matrix entry greater than $x_{\max}$ for fp16. We restrict to $n \leq 400$ because the time required to LU factorize in half precision is very high for larger n, because of the overheads of the `fp16` class. Table 4.1 lists the matrices along with their properties and the underlying application. The right-hand side of each system is generated as `randn(n,1)`, and the random number generator is seeded for reproducibility. Our test codes are available at https://github.com/SrikaraPranesh/fp16Scaling.

We first consider the use of Algorithms 2.1 and 2.2 in GMRES-IR. The results are displayed in Table 4.2. We see that Algorithm 2.1 often requires a large number of GMRES iterations for iterative refinement to converge. Algorithm 2.2 requires fewer GMRES iterations than Algorithm 2.1, but iterative refinement fails to converge for matrices 12 and 13, because underflow causes the matrices to be singular in fp16. From these numerical experiments we conclude that Algorithms 2.1 and 2.2 are not reliable ways to convert a matrix to fp16.

Next we consider the performance of Algorithm 3.1 using Algorithms 2.4 and 2.5 for the scaling. The results, in Table 4.3, show that both diagonal scalings perform very well. We make two observations. First, in the (half, single, double) case, no iterations are required for many of the matrices. This is because one or both of $\|A\|_\infty$ and $\|x\|_\infty$ are so large that (4.1) is satisfied for the initial solution. Second, we note that convergence is achieved in every case and with generally fewer iterative refinement steps and GMRES iterations than for Algorithms 2.1 and 2.2.

To explore these results further, we note that the analysis in [3], [4] actually shows that convergence of iterative refinement will be achieved provided that $\kappa_\infty(MA)u$ is sufficiently less than 1, where $MA = \mu S\widehat{U}^{-1}\widehat{L}^{-1}R$ is the preconditioned matrix in Algorithm 3.1. (The bounds in Table 3.1 are a weakening of this condition.) Table 4.4 shows the values of $\kappa_\infty(MA)$ for a working precision of double; for single precision the quantities are broadly similar. We see that $\kappa_\infty(MA)u \ll 1$ for matrices 1–11.

For matrices 12 and 13 the preconditioned matrix MA is extremely ill conditioned, like A, but Algorithms 2.4 and 2.5 nevertheless converge, even though $\kappa_\infty(MA)$ is of

[4]Formerly known as the University of Florida Sparse Matrix Collection.

TABLE 4.2

Total number of GMRES iterations required by GMRES-IR (Algorithm 3.1), using Algorithm 2.1 and Algorithm 2.2, for single and double as working precision. Failure to converge is denoted by "–". Numbers in parentheses indicates the number of iterative refinement steps.

Index	(half, single, double)		(half, double, quad)	
	Algorithm 2.1	Algorithm 2.2	Algorithm 2.1	Algorithm 2.2
1	4 (1)	2 (1)	11 (2)	7 (2)
2	3 (1)	2 (1)	10 (2)	8 (2)
3	27 (2)	2 (1)	66 (3)	7 (2)
4	31 (2)	0 (0)	235 (3)	14 (2)
5	94 (2)	0 (0)	258 (3)	4 (2)
6	8 (1)	2 (1)	106 (3)	3 (1)
7	0 (0)	1 (1)	4 (2)	2 (1)
8	0 (0)	3 (1)	25 (3)	10 (2)
9	94 (3)	0 (0)	119 (3)	7 (2)
10	409 (5)	0 (0)	334 (3)	13 (3)
11	212 (2)	0 (0)	562 (3)	21 (2)
12	0 (0)	– (–)	9 (2)	– (–)
13	0 (0)	– (–)	8 (2)	– (–)

TABLE 4.3

Total number of GMRES iterations required by GMRES-IR (Algorithm 3.1), using Algorithm 2.4 and Algorithm 2.5, for single and double as working precision. Numbers in parentheses indicate the number of iterative refinement steps.

Index	(half, single, double)		(half, double, quad)	
	Algorithm 2.4	Algorithm 2.5	Algorithm 2.4	Algorithm 2.5
1	0 (0)	0 (0)	2 (1)	2 (1)
2	0 (0)	0 (0)	4 (2)	4 (2)
3	2 (1)	2 (1)	6 (2)	5 (2)
4	0 (0)	0 (0)	16 (2)	14 (2)
5	0 (0)	0 (0)	2 (1)	2 (1)
6	0 (0)	0 (0)	2 (1)	2 (1)
7	0 (0)	0 (0)	2 (1)	2 (1)
8	0 (0)	0 (0)	8 (2)	8 (2)
9	0 (0)	0 (0)	9 (3)	10 (3)
10	1 (1)	0 (0)	11 (3)	11 (3)
11	0 (0)	0 (0)	36 (3)	22 (2)
12	0 (0)	0 (0)	9 (2)	2 (1)
13	0 (0)	0 (0)	7 (2)	2 (1)

order u^{-1}. We also note that a small number of GMRES iterations are required in every case, as shown in Table 4.3.

We note that in the previous works [4], [14], [15] the forward error test $\|x - x_*\|_\infty / \|x_*\|_\infty < u$, where x_* is the exact solution, was also used as a convergence criterion for GMRES-IR. We have run all our experiments with this forward error test along with (4.1); we found no major differences, with the forward error test typically requiring 1–3 additional GMRES iterations.

When diagonal scalings are applied in practice, they are often rounded to the nearest powers of 2 in order to avoid rounding errors in their application; this is done in the LAPACK `xyyEQUB` routines, for example. We have not rounded the scalings in the results reported here. We did try such rounding, but it had no effect on the number of GMRES iterations, which we attribute to rounding errors in applying the scaling (which is done at the working precision) being negligible compared with those in the rounding to fp16.

Based on these numerical experiments with GMRES-IR we conclude that Algo-

TABLE 4.4
Condition number of MA in Algorithm 3.1 for a working precision of double, where $M = \mu S\widehat{U}^{-1}\widehat{L}^{-1}R$ and $\widehat{L}$ and $\widehat{U}$ are the LU factors of $A^{(h)}$ computed in fp16.

		Algorithm 2.4	Algorithm 2.5
Index	$\kappa_\infty(A)$	$\kappa_\infty(MA)$	$\kappa_\infty(MA)$
1	1.05e+12	8.03e+05	8.70e+05
2	6.76e+10	1.39e+03	5.18e+03
3	2.49e+06	2.48e+00	2.13e+00
4	1.59e+09	8.78e+01	9.56e+01
5	3.11e+07	1.01e+00	2.21e+00
6	7.64e+10	2.67e+00	5.62e+02
7	1.20e+12	6.24e+04	8.95e+04
8	1.98e+10	5.20e+03	9.56e+03
9	1.60e+06	7.40e+00	2.99e+01
10	5.44e+06	2.02e+03	1.55e+03
11	2.53e+07	4.64e+04	4.95e+02
12	1.59e+14	7.24e+14	3.97e+16
13	1.08e+14	2.24e+14	3.06e+16

rithms 2.4 and 2.5 are both very effective as scaling algorithms within Algorithm 2.3 and are significantly better than Algorithms 2.1 and 2.2. If a symmetry-preserving LU-type factorization is to be computed then Algorithm 2.5 should be preferred, as it preserves symmetry.

At the end of section 2 we explained that the purpose of θ in Algorithm 2.3 is to increase the matrix entries in order to avoid subnormal numbers and underflow. To check the effect of θ we compared the results for $\theta = 0.1$, for which the largest entry in absolute value of the scaled matrix is $0.1x_{\max}$, and $\theta = 1/x_{\max}$, for which the entries lie on the interval $[-1, 1]$. We found that setting $\theta = 1/x_{\max}$ does not lead to any significant change in the number of GMRES iterations. However, as shown in Table 4.5, for some matrices the percentage of nonzero elements that underflow drops significantly for $\theta = 0.1$. Note that for matrices 7, 12, and 13, $\theta = 0.1$ results in a significant reduction in the percentage of nonzero entries that underflow when compared with $\theta = 1/x_{\max}$. The percentage of entries that become subnormal is very small (always less than 1 percent), but again $\theta = 0.1$ leads to a smaller percentage than $\theta = 1/x_{\max}$.

5. Another scaling strategy. One can consider other transformations, in addition to Algorithm 2.3, in an attempt to fully exploit the fp16 range. Let A denote the matrix after the initial diagonal scaling using either Algorithm 2.4 or Algorithm 2.5 and define

$$a_{\min} = \min_{i,j} a_{ij}, \quad a_{\max} = \max_{i,j} a_{ij}.$$

In this section we explore the idea of carrying out a linear transformation of the elements of A to a different interval $[z_1, z_2]$ within the fp16 range. To that end, we define the matrix C by

$$\begin{aligned} c_{ij} &= z_2 \left(\frac{a_{ij} - a_{\min}}{a_{\max} - a_{\min}} \right) + z_1 \left(\frac{a_{ij} - a_{\max}}{a_{\min} - a_{\max}} \right) \\ &= a_{ij} \left(\frac{z_2 - z_1}{a_{\max} - a_{\min}} \right) + \frac{z_1 a_{\max} - z_2 a_{\min}}{a_{\max} - a_{\min}}. \end{aligned}$$

Hence

$$C = \alpha A + \beta e e^T, \tag{5.1}$$

TABLE 4.5
Percentage of nonzero entries which underflow during conversion from double precision to half precision for $\theta = 1/x_{\max}$ (for which $\max_{i,j} |a_{ij}^{(h)}| = 1$) and $\theta = 0.1$ (for which $\max_{i,j} |a_{ij}^{(h)}| = 0.1x_{\max}$).

Index	Algorithm 2.4		Algorithm 2.5	
	$\theta = 1/x_{\max}$	$\theta = 0.1$	$\theta = 1/x_{\max}$	$\theta = 0.1$
1	0.00	0.00	0.00	0.00
2	0.00	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00
5	2.85	0.00	2.85	0.00
6	0.00	0.00	0.00	0.00
7	44.84	36.45	44.36	30.67
8	0.00	0.00	0.00	0.00
9	0.00	0.00	0.00	0.00
10	5.59	0.00	5.96	0.00
11	0.00	0.00	0.00	0.00
12	16.37	3.37	10.76	0.00
13	23.85	11.72	21.04	4.81

where $e = [1,\ 1,\ \ldots,\ 1]^T$ and

$$\alpha = \frac{z_2 - z_1}{a_{\max} - a_{\min}}, \quad \beta = \frac{z_1 a_{\max} - z_2 a_{\min}}{a_{\max} - a_{\min}}.$$

One possible choice of target interval is $[z_1, z_2] = [-\theta x_{\max}, \theta x_{\max}]$, where $\theta \in (0, 1]$, which stretches the range to within a factor θ of the whole fp16 range (which Algorithm 2.3 does not guarantee to do). To demonstrate the advantage of this approach, consider the matrix

$$A = \begin{bmatrix} 1-\epsilon & 1 \\ 1 & 1 \end{bmatrix}, \quad \epsilon = 2^{-k}, \quad 12 \le k \le 16,$$

which is stored exactly in double precision arithmetic and for which $\kappa_2(A) \approx 4/\epsilon \le 2.6 \times 10^5$. Note that A is equilibrated and so diagonal scaling by Algorithms 2.4 or 2.5 does not alter the matrix entries. Therefore $\mathrm{fl_h}(A)$ is the matrix of 1s and so is singular, even though A is relatively well conditioned as a double precision matrix. If we use the rank-1 update scaling with $[z_1, z_2] = [-\theta x_{\max}, \theta x_{\max}]$ then the scaled matrix in (5.1) is given by

$$C = \theta x_{\max} \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix}, \quad \kappa_\infty(A) = 2. \tag{5.2}$$

The transformation (5.1) preserves information in the ϵ term and greatly improves the condition number of the matrix. However, a drawback is that it fills in any zero elements, so destroys sparsity. We also note that when $z_1 = -z_2$ and $a_{\min} = -a_{\max}$, $\beta = 0$, and so $C = \alpha A$ and the condition number does not change.

Another possibility is $[z_1, z_2] = [\mu x_{\min}, \theta x_{\max}]$, where $\mu \in \{0, \widetilde{\theta}\}$, with $\widetilde{\theta} \ge 1$. This transformation maps the a_{ij} onto the nonnegative interval $[\mu x_{\min}, \theta x_{\max}]$. We could alternatively use this choice of $[z_1, z_2]$ and modify $a_{\min}$ and $a_{\max}$ so that

$$a_{\min} = \min_{i,j} |a_{ij}|, \quad a_{\max} = \max_{i,j} |a_{ij}|.$$

Here, we can set $\mu = 0$ to preserve sparsity. Now the transformation maps the nonnegative a_{ij} onto $[\mu x_{\min}, \theta x_{\max}]$, but the nonpositive elements are not constrained and can overflow.

In the context of a system of linear equations $Ax = b$, the transformation (5.1) yields

$$(C - \beta ee^T)x = \alpha b.$$

This equation can be solved using the Sherman–Morrison formula to give

$$x = \left(C^{-1} - \frac{\beta C^{-1} ee^T C^{-1}}{1 - \beta e^T C^{-1} e} \right) \alpha b, \tag{5.3}$$

which requires the solution of two linear systems with C, which can be done by GMRES-IR with the need for just one LU factorization.

Our experience in the context of GMRES-IR is that this transformation generally does not provide any benefits over Algorithm 2.3 on its own, so we will not pursue it here. In other contexts the transformation could prove useful. For example, it can potentially reduce the condition number, as demonstrated in (5.2), and if the matrix has positive entries then using (5.1) with $[z_1, z_2] = [\widetilde{\theta} x_{\min}, \theta x_{\max}]$ prevents entries from overflowing, underflowing, or becoming subnormal in the conversion to fp16.

6. Conclusions. Converting a floating-point matrix to lower precision is not a trivial task when the lower precision format has a much narrower range than the original one, especially when the target is the fp16 arithmetic that is increasingly available in hardware. Overflow and underflow in the conversion are undesirable, as is the generation of subnormal numbers.

We have derived a new conversion algorithm that employs two-sided diagonal scaling along with a further scalar multiplication that moves the elements of largest magnitude close to the overflow threshold. Our particular interest is in GMRES-IR (Algorithm 3.1), which solves a linear system $Ax = b$, where A and b are given in double precision, using an LU factorization of an fp16 representation of A. Previous work has shown significant benefits in speed and energy usage over solving entirely in double precision [14], [15], [16], but in that work the conversion to fp16 was done by rounding and then mapping any infinities back to the nearest floating-point number. Our new conversion algorithm produces more reliable and faster convergence of GMRES-IR on badly scaled matrices, usually requiring fewer total GMRES iterations and fewer iterative refinement steps. It thereby allows a wider class of problems to be solved and so, given its negligible cost, is recommended for use with GMRES-IR.

REFERENCES

[1] E. Anderson, *Algorithm* 978: *Safe scaling in the level* 1 *BLAS*, ACM Trans. Math. Software, 44 (2017), pp. 12:1–12:28, https://doi.org/10.1145/3061665.

[2] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed., SIAM, Philadelphia, 1999, https://doi.org/10.1137/1.9780898719604.

[3] 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.

[4] 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.

[5] 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.

[6] A. Dawson, P. D. Düben, D. A. MacLeod, and T. N. Palmer, *Reliable low precision simulations in land surface models*, Clim. Dyn., 51 (2018), pp. 2657–2666, https://doi.org/10.1007/s00382-017-4034-x.

[7] J. W. Demmel and X. Li, *Faster numerical algorithms via exception handling*, IEEE Trans. Comput., 43 (1994), pp. 983–992, https://doi.org/10.1109/12.295860.
[8] J. M. Elble and N. V. Sahinidis, *Scaling linear optimization problems prior to application of the simplex method*, Comput. Optim. Appl., 52 (2012), pp. 345–371, https://doi.org/10.1007/s10589-011-9420-4.
[9] M. Feldman, *Fujitsu reveals details of processor that will power Post-K supercomputer*, https://www.top500.org/news/fujitsu-reveals-details-of-processor-that-will-power-post-k-supercomputer, Aug. 2018, accessed November 22, 2018.
[10] M. Feldman, *Intel lays out roadmap for next three Xeon products*, https://www.top500.org/news/intel-lays-out-roadmap-for-next-three-xeon-products/, Aug. 2018, accessed June 5, 2019.
[11] M. Feldman, *Record-breaking exascale application selected as Gordon Bell finalist*, https://www.top500.org/news/record-breaking-exascale-application-selected-as-gordon-bell-finalist/, Sept. 2018, accessed January 8, 2019.
[12] G. E. Forsythe and C. B. Moler, *Computer Solution of Linear Algebraic Systems*, Prentice–Hall, Englewood Cliffs, NJ, 1967.
[13] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, *Deep learning with limited numerical precision*, in Proceedings of the 32nd International Conference on Machine Learning, Vol. 37 of JMLR: Workshop and Conference Proceedings, 2015, pp. 1737–1746, http://www.jmlr.org/proceedings/papers/v37/gupta15.html.
[14] 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.
[15] 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, 2018, IEEE Press, pp. 47:1–47:11, https://doi.org/10.1109/SC.2018.00050.
[16] A. Haidar, P. Wu, S. Tomov, and J. Dongarra, *Investigating half precision arithmetic to accelerate dense linear system solvers*, in Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, ScalA '17 (Denver, CO), 2017, pp. 10:1–10:8, https://doi.org/10.1145/3148226.3148237.
[17] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed., SIAM, Philadelphia, 2002, https://doi.org/10.1137/1.9780898718027.
[18] J. Hook, J. Pestana, F. Tisseur, and J. Hogg, *Max-balanced Hungarian scalings*, SIAM J. Matrix Anal. Appl., 40 (2019), pp. 320–346, https://doi.org/10.1137/15M1024871.
[19] *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.
[20] 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.
[21] K. Kincade, *Team breaks exaop barrier with deep learning application*, https://phys.org/news/2018-10-team-exaop-barrier-deep-application.html, Oct. 2018, accessed May 16, 2019.
[22] P. A. Knight, D. Ruiz, and B. Uçar, *A symmetry preserving algorithm for matrix scaling*, SIAM J. Matrix Anal. Appl., 35 (2014), pp. 931–955, https://doi.org/10.1137/110825753.
[23] T. Larsson, *On scaling linear programs—some experimental results*, Optimization, 27 (1993), pp. 355–373, https://doi.org/10.1080/02331939308843895.
[24] C. C. K. Mikkelsen, A. B. Schwarz, and L. Karlsson, *Parallel robust solution of triangular linear systems*, Concurrency Computat.: Pract. Exper., (2018), https://doi.org/10.1002/cpe.5064.
[25] C. B. Moler, *Cleve Laboratory*, http://mathworks.com/matlabcentral/fileexchange/59085-cleve-laboratory.
[26] C. B. Moler, *"Half precision" 16-bit floating point arithmetic*, http://blogs.mathworks.com/cleve/2017/05/08/half-precision-16-bit-floating-point-arithmetic/, May 2017.
[27] *Multiprecision Computing Toolbox*, Advanpix, Tokyo, http://www.advanpix.com.
[28] *ORNL launches Summit supercomputer*, https://www.ornl.gov/news/ornl-launches-summit-supercomputer, June 2018, accessed June 30, 2018.
[29] T. N. Palmer, *More reliable forecasts with less precise computations: A fast-track route to*

cloud-resolved weather and climate simulators?, Phil. Trans. Roy. Soc. A, 372 (2014), https://doi.org/10.1098/rsta.2013.0391.

[30] N. Rao, *Beyond the CPU or GPU: Why enterprise-scale artificial intelligence requires a more holistic approach*, https://newsroom.intel.com/editorials/artificial-intelligence-requires-holistic-approach, May 2018, accessed November 5, 2018.

[31] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869, https://doi.org/10.1137/0907058.

[32] *Summit by the numbers*, https://www.olcf.ornl.gov/wp-content/uploads/2018/06/Summit_bythenumbers_FIN.png, June 2018, accessed June 30, 2018.

[33] A. Svyatkovskiy, J. Kates-Harbeck, and W. Tang, *Training distributed deep recurrent neural networks with mixed precision on GPU clusters*, in MLHPC'17: Proceedings of the Machine Learning on HPC Environments, ACM Press, New York, 2017, pp. 10:1–10:8, https://doi.org/10.1145/3146347.3146358.