A half precision incomplete Cholesky factorization preconditioning

Jennifer Scott
University of Reading and STFC Rutherford Appleton Laboratory

Miroslav Tůma
Faculty of Mathematics and Physics, Charles University, Prague

Sparse Days, Toulouse, June 2024

Outline

1. Introduction (low precision, the problem $Ax = b$)
2. Breakdowns and their detection
3. Bounding factor entries
4. Conclusions

Motivation to use low precision

- Unsurprising motivation: using low precision to achieve higher speed in lower (work)space
- Traditionally: single precision (fp32) and double precision (fp64)
- Throughout 1990's, single was not much faster than double.
- Breakthrough in SSE units (Intel, 1999): single precision significantly accelerated
- Emergence of half precision (fp16) floating-point arithmetic: 2008 revision of the IEEE standard.
- Started as storage format, but soon in GPU accelerators. See discussions in Higham, 2017; Higham, Mary, 2022.
- BUT: fp16: limited range (largest positive number is 6.55×10^4); a variant: bfloat16 used by Google in its tensor processing units (larger range, less significant decimal digits)

Motivation to use low precision

- Unsurprising motivation: using low precision to achieve higher speed in lower (work)space
- Traditionally: single precision (fp32) and double precision (fp64)
- Throughout 1990's, single was not much faster than double.
- Breakthrough in SSE units (Intel, 1999): single precision significantly accelerated
- Emergence of half precision (fp16) floating-point arithmetic: 2008 revision of the IEEE standard.
- Started as storage format, but soon in GPU accelerators. See discussions in Higham, 2017; Higham, Mary, 2022.
- BUT: fp16: limited range (largest positive number is 6.55×10^4); a variant: bfloat16 used by Google in its tensor processing units (larger range, less significant decimal digits)

Motivation to use low precision

- Unsurprising motivation: using low precision to achieve higher speed in lower (work)space
- Traditionally: single precision (fp32) and double precision (fp64)
- Throughout 1990's, single was not much faster than double.
- Breakthrough in SSE units (Intel, 1999): single precision significantly accelerated
- Emergence of half precision (fp16) floating-point arithmetic: 2008 revision of the IEEE standard.
- Started as storage format, but soon in GPU accelerators. See discussions in Higham, 2017; Higham, Mary, 2022.
- BUT: fp16: limited range (largest positive number is 6.55×10^4); a variant: bfloat16 used by Google in its tensor processing units (larger range, less significant decimal digits)

Motivation to use low precision

- Unsurprising motivation: using low precision to achieve higher speed in lower (work)space
- Traditionally: single precision (fp32) and double precision (fp64)
- Throughout 1990's, single was not much faster than double.
- Breakthrough in SSE units (Intel, 1999): single precision significantly accelerated
- Emergence of half precision (fp16) floating-point arithmetic: 2008 revision of the IEEE standard.
- Started as storage format, but soon in GPU accelerators. See discussions in Higham, 2017; Higham, Mary, 2022.
- BUT: fp16: limited range (largest positive number is 6.55×10^4); a variant: bfloat16 used by Google in its tensor processing units (larger range, less significant decimal digits)

Introduction: low precision

Motivation to use low precision

- Unsurprising motivation: using low precision to achieve higher speed in lower (work)space
- Traditionally: single precision (fp32) and double precision (fp64)
- Throughout 1990's, single was not much faster than double.
- Breakthrough in SSE units (Intel, 1999): single precision significantly accelerated
- Emergence of half precision (fp16) floating-point arithmetic: 2008 revision of the IEEE standard.
- Started as storage format, but soon in GPU accelerators. See discussions in Higham, 2017; Higham, Mary, 2022.
- BUT: fp16: limited range (largest positive number is 6.55×10^4); a variant: bfloat16 used by Google in its tensor processing units (larger range, less significant decimal digits)

Motivation to use low precision

- Unsurprising motivation: using low precision to achieve higher speed in lower (work)space
- Traditionally: single precision (fp32) and double precision (fp64)
- Throughout 1990's, single was not much faster than double.
- Breakthrough in SSE units (Intel, 1999): single precision significantly accelerated
- Emergence of half precision (fp16) floating-point arithmetic: 2008 revision of the IEEE standard.
- Started as storage format, but soon in GPU accelerators. See discussions in Higham, 2017; Higham, Mary, 2022.
- BUT: fp16: limited range (largest positive number is 6.55×10^4); a variant: bfloat16 used by Google in its tensor processing units (larger range, less significant decimal digits)

Table: Parameters for bfloat16, fp16, fp32, and fp64 arithmetic: the number of bits in the significand and exponent, unit roundoff u, smallest positive (subnormal) number x^s_{min} , smallest normalized positive number x_{min}, and largest finite number x_{max}, all given to three significant figures. † In Intel's bfloat16 specification, subnormal numbers are not supported.

	Signif.	Exp.	u	x^s_{min}	x_{min}	x_{max}
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}
bfloat16	8	8	3.91×10^{-3}	†	1.18×10^{-38}	3.39×10^{38}

Solving ill-conditioned linear systems by preconditioned iterative methods in fp16

$$Ax = b$$

$A \in \mathbb{R}^{n \times n}$ is large, sparse and symmetric positive definite,

Problem 1: using input data

- Low precison means: matrix values have to be initially mapped to the set of values existing in fp16. (Haidar et al., 2017; Higham, Pranesh, Zounon, 2019; Higham, Pranesh, 2021)
- Doing this, we may loose (will loose) initially some information.
- This problem is demonstrated using our matrix test set (next slide)
- Our squeezing into fp16 is based on l2 scaling of columns of the lower triangular part of A, scaling is a must, nothing more needed (in our case)

Low precision and our data

Solving ill-conditioned linear systems by preconditioned iterative methods in fp16

$$Ax = b$$

$A \in \mathbb{R}^{n \times n}$ is large, sparse and symmetric positive definite,

Problem 1: using input data

- Low precison means: matrix values have to be initially mapped to the set of values existing in fp16. (Haidar et al., 2017; Higham, Pranesh, Zounon, 2019; Higham, Pranesh, 2021)
- Doing this, we may loose (will loose) initially some information.
- This problem is demonstrated using our matrix test set (next slide)
- Our squeezing into fp16 is based on l2 scaling of columns of the lower triangular part of A, scaling is a must, nothing more needed (in our case)

Solving ill-conditioned linear systems by preconditioned iterative methods in fp16

$$Ax = b$$

$A \in \mathbb{R}^{n\times n}$ is large, sparse and symmetric positive definite,

Problem 1: using input data

- Low precison means: matrix values have to be initially mapped to the set of values existing in fp16. (Haidar et al., 2017; Higham, Pranesh, Zounon, 2019; Higham, Pranesh, 2021)
- Doing this, we may loose (will loose) initially some information.
- This problem is demonstrated using our matrix test set (next slide)
- Our squeezing into fp16 is based on l2 scaling of columns of the lower triangular part of A, scaling is a must, nothing more needed (in our case)

Low precision and our data

Solving ill-conditioned linear systems by preconditioned iterative methods in fp16

$$Ax = b$$

$A \in \mathbb{R}^{n \times n}$ is large, sparse and symmetric positive definite,

Problem 1: using input data

- Low precison means: matrix values have to be initially mapped to the set of values existing in fp16. (Haidar et al., 2017; Higham, Pranesh, Zounon, 2019; Higham, Pranesh, 2021)
- Doing this, we may loose (will loose) initially some information.
- This problem is demonstrated using our matrix test set (next slide)
- Our squeezing into fp16 is based on l2 scaling of columns of the lower triangular part of A, scaling is a must, nothing more needed (in our case)

Test examples from SuiteSparse Matrix Collection.

Identifier	n	$nnz(A)$	$nnz(A, fp16)$	$normA$	$cond2(A)$
Boeing/msc01050	1050	1.51×10^{4}	4.63×10^{3}	2.58×10^{7}	4.58×10^{15}
HB/bcsstk11	1473	1.79×10^{4}	6.73×10^{3}	1.21×10^{10}	2.21×10^{8}
HB/bcsstk26	1922	1.61×10^{4}	6.69×10^{3}	1.68×10^{11}	1.66×10^{8}
HB/bcsstk24	3562	8.17×10^{4}	3.89×10^{4}	5.28×10^{14}	1.95×10^{11}
HB/bcsstk16	4884	1.48×10^{5}	5.25×10^{4}	4.12×10^{10}	4.94×10^{9}
Cylshell/s2rmt3m1	5489	1.13×10^{5}	5.09×10^{4}	9.84×10^{5}	2.50×10^{8}
Cylshell/s3rmt3m1	5489	1.13×10^{5}	5.08×10^{4}	1.01×10^{5}	2.48×10^{10}
Boeing/bcsstk38	8032	1.82×10^{5}	7.83×10^{4}	4.50×10^{11}	5.52×10^{16}
Boeing/msc10848	10848	6.20×10^{5}	3.02×10^{5}	4.58×10^{13}	9.97×10^{9}
Oberwolfach/t2dah_e	11445	9.38×10^{4}	4.88×10^{4}	2.20×10^{-5}	7.23×10^{8}
Boeing/ct20stif	52329	1.38×10^{6}	6.30×10^{5}	8.99×10^{11}	1.18×10^{12}
DNVS/shipsec8	114919	3.38×10^{6}	7.71×10^{5}	7.31×10^{12}	2.40×10^{13}
Um/2cubes_sphere	101492	8.74×10^{5}	4.57×10^{5}	3.43×10^{10}	2.59×10^{8}
GHS_psdef/hood	220542	5.49×10^{6}	2.66×10^{6}	2.23×10^{9}	5.35×10^{7}
Um/offshore	259789	2.25×10^{6}	1.17×10^{6}	1.44×10^{15}	4.26×10^{9}

Consequently: no direct method, only preconditioner

Problem 2: algebraic preconditioner, IC in our case, and growth

- The growth factor (early 60's, Wilkinson) for $A = (a_{ij})_{1 \le i \le n, 1 \le j \le n}$:

$$\rho_n = \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{\max_{i,j} |a_{ij}|} \ge 1$$

- Complete Cholesky factor does not grow (the growth factor ρ_n is equal to 1, e.g., Higham, 2002)
- But IC may have the growth. Even without fp16
- For example: (next slide)

Problem 2: growth in IC (continued)

- Example: incomplete factorization IC(0), well-conditioned matrix A.
-

$$A = \begin{pmatrix} 3 & -2 & 0 & 2 & 0 \\ -2 & 3 & -2 & 0 & 0 \\ 0 & -2 & 3 & -2 & 0 \\ 2 & 0 & -2 & 8+2\delta & 2 \\ 0 & 0 & 0 & 2 & 8 \end{pmatrix},$$

$$L = \begin{pmatrix} d_1 & & & & \\ -2/d_1 & d_2 & & & \\ 0 & -2/d_2 & d_3 & & \\ 2/d_1 & 0 & -2/d_3 & d_4 & \\ 0 & 0 & 0 & 2/d_4 & d_5 \end{pmatrix},$$

with $d_1^2 = 3$, $d_2^2 = 5/3$, $d_3^2 = 3/5$, $d_4^2 = 2\delta$, and $d_5^2 = 8 - 2/\delta$.

- The problem is incompleteness, not a specific IC choice ...

Problem 2: growth in IC (continued)

- Our matrices and IC(3) (level-based IC):

Detected growth in our experiments. IC(0), fp64.

Identifier	n	detected $growth\ factor$
Boeing/msc01050	1050	$9.43{\times}10^{1}$
HB/bcsstk11	1473	$1.98{\times}10^{3}$
HB/bcsstk24	3562	$5.10{\times}10^{2}$
Boeing/bcsstk38	8032	$4.52{\times}10^{47}$
Boeing/msc10848	10848	$9.59{\times}10^{16}$
Boeing/ct20stif	52329	$1.14{\times}10^{6}$

- Different for different IC. May be (approximately) put into the context of the amount of (inexact updates) inside IC, but not always
- fp16: typically worse tendency. But in some cases no growth detected.
- This needs to be solved. The values in fp16 must not grow so much!

Problem 3: overflows and breakdowns

- Growth means facing dangers of overflows! This is another serious problem of fp16 in our computations.
- Problem of overflows can be solved similarly as the problem of getting high quality IC without breakdowns.
- Nemely, using diagonal shifts introduced to monitori sizes of diagonal entries
  - Whenever a diagonal entry is small: $A \to A + \alpha I$ (Manteuffel (1980), seminal implementation Lin, Moré (1999))
- Using diagonal shifts to avoid breakdowns COINCIDES with the effort to find efficient preconditioners. But in fp16 still the sole monitoring of diagonal entries cannot prohibit overflows.

Overflows must be treated as specific (potential) breakdowns

Problem 3: overflows and breakdowns

- Growth means facing dangers of overflows! This is another serious problem of fp16 in our computations.
- Problem of overflows can be solved similarly as the problem of getting high quality IC without breakdowns.
- Nemely, using diagonal shifts introduced to monitori sizes of diagonal entries
  - Whenever a diagonal entry is small: $A \to A + \alpha I$ (Manteuffel (1980), seminal implementation Lin, Moré (1999))
- Using diagonal shifts to avoid breakdowns COINCIDES with the effort to find efficient preconditioners. But in fp16 still the sole monitoring of diagonal entries cannot prohibit overflows.

Overflows must be treated as specific (potential) breakdowns

Problem 3: overflows and breakdowns

- Growth means facing dangers of overflows! This is another serious problem of fp16 in our computations.
- Problem of overflows can be solved similarly as the problem of getting high quality IC without breakdowns.
- Nemely, using diagonal shifts introduced to monitori sizes of diagonal entries
  - Whenever a diagonal entry is small: $A \rightarrow A + \alpha I$ (Manteuffel (1980), seminal implementation Lin, Moré (1999))
- Using diagonal shifts to avoid breakdowns COINCIDES with the effort to find efficient preconditioners. But in fp16 still the sole monitoring of diagonal entries cannot prohibit overflows.

Overflows must be treated as specific (potential) breakdowns

Problem 3: overflows and breakdowns

- Growth means facing dangers of overflows! This is another serious problem of fp16 in our computations.
- Problem of overflows can be solved similarly as the problem of getting high quality IC without breakdowns.
- Nemely, using diagonal shifts introduced to monitori sizes of diagonal entries
  - Whenever a diagonal entry is small: $A \rightarrow A + \alpha I$ (Manteuffel (1980), seminal implementation Lin, Moré (1999))
- Using diagonal shifts to avoid breakdowns COINCIDES with the effort to find efficient preconditioners. But in fp16 still the sole monitoring of diagonal entries cannot prohibit overflows.

Overflows must be treated as specific (potential) breakdowns

Problem 3: overflows and breakdowns

- Growth means facing dangers of overflows! This is another serious problem of fp16 in our computations.
- Problem of overflows can be solved similarly as the problem of getting high quality IC without breakdowns.
- Nemely, using diagonal shifts introduced to monitori sizes of diagonal entries
  - Whenever a diagonal entry is small: $A \to A + \alpha I$ (Manteuffel (1980), seminal implementation Lin, Moré (1999))
- Using diagonal shifts to avoid breakdowns COINCIDES with the effort to find efficient preconditioners. But in fp16 still the sole monitoring of diagonal entries cannot prohibit overflows.

Overflows must be treated as specific (potential) breakdowns

Problem 3: overflows and breakdowns

- Let us distinguish three kinds of potential breakdowns (up to now without showing an IC algorithm) ☺
- B1: The computed diagonal entry l_{kk} (termed the pivot at step k) may be unacceptably small (close to zero or negative).
  - This could cause a breakdown in a later step of the factorization
  - Its treatment goes hand in hand with the effort to compute an efficient preconditioner
- B2: The scaling of column entries $l_{ik} \leftarrow l_{ik}/l_{kk}$ may overflow.
- B3: The update $l_{ij} \leftarrow l_{ij} - l_{ik}l_{jk}$ may overflow.

Problem 3: overflows and breakdowns

- Let us distinguish three kinds of potential breakdowns (up to now without showing an IC algorithm) ☺
- B1: The computed diagonal entry l_{kk} (termed the pivot at step k) may be unacceptably small (close to zero or negative).
  - This could cause a breakdown in a later step of the factorization
  - Its treatment goes hand in hand with the effort to compute an efficient preconditioner
- B2: The scaling of column entries $l_{ik} \leftarrow l_{ik}/l_{kk}$ may overflow.
- B3: The update $l_{ij} \leftarrow l_{ij} - l_{ik}l_{jk}$ may overflow.

Problem 3: overflows and breakdowns

- Let us distinguish three kinds of potential breakdowns (up to now without showing an IC algorithm) ☺
- B1: The computed diagonal entry l_{kk} (termed the pivot at step k) may be unacceptably small (close to zero or negative).
  - This could cause a breakdown in a later step of the factorization
  - Its treatment goes hand in hand with the effort to compute an efficient preconditioner
- B2: The scaling of column entries $l_{ik} \leftarrow l_{ik}/l_{kk}$ may overflow.
- B3: The update $l_{ij} \leftarrow l_{ij} - l_{ik}l_{jk}$ may overflow.

Problem 3: overflows and breakdowns

- Let us distinguish three kinds of potential breakdowns (up to now without showing an IC algorithm) ☺
- B1: The computed diagonal entry l_{kk} (termed the pivot at step k) may be unacceptably small (close to zero or negative).
  - This could cause a breakdown in a later step of the factorization
  - Its treatment goes hand in hand with the effort to compute an efficient preconditioner
- B2: The scaling of column entries $l_{ik} \leftarrow l_{ik}/l_{kk}$ may overflow.
- B3: The update $l_{ij} \leftarrow l_{ij} - l_{ik}l_{jk}$ may overflow.

Detecting breakdowns

Problem 3: overflows and breakdowns

- Two differentto treat breakdowns (apriori/aposteriori detection):
  - Detect them safely inside the code
  - Use IEEE exceptions (if enabled) and test them aposteriori

Detecting breakdowns

Problem 3: overflows and breakdowns

- Safe detections inside the code:
- Straighforward to detect directly B1 (small diagonal entries)
- B2 (breakdown in scaling) also straightforward: no overflow if

$$l_{kk} \geq 1 \quad \text{or} \quad l_{kk} \geq l_{ik}/x_{max}$$

  Otherwise, breakdown B2 detected
- B3 breakdown: needed to avoid overflow in any of the partial results in $l_{ij} \leftarrow l_{ij} - l_{ik}l_{jk}$
- Safe detections can be fast if applied to rows/columns using maximum magnitudes inside them.

Problem 3: overflows and breakdowns

- Safe detections inside the code:
- Straighforward to detect directly B1 (small diagonal entries)
- B2 (breakdown in scaling) also straightforward: no overflow if

$$l_{kk} \geq 1 \quad \text{or} \quad l_{kk} \geq l_{ik}/x_{max}$$

  Otherwise, breakdown B2 detected
- B3 breakdown: needed to avoid overflow in any of the partial results in $l_{ij} \leftarrow l_{ij} - l_{ik}l_{jk}$
- Safe detections can be fast if applied to rows/columns using maximum magnitudes inside them.

Problem 3: overflows and breakdowns

- Safe detections inside the code:
- Straighforward to detect directly B1 (small diagonal entries)
- B2 (breakdown in scaling) also straightforward: no overflow if

$$l_{kk} \geq 1 \quad \text{or} \quad l_{kk} \geq l_{ik}/x_{max}$$

  Otherwise, breakdown B2 detected
- B3 breakdown: needed to avoid overflow in any of the partial results in $l_{ij} \leftarrow l_{ij} - l_{ik} l_{jk}$
- Safe detections can be fast if applied to rows/columns using maximum magnitudes inside them.

Problem 3: overflows and breakdowns

- Safe detections inside the code:
- Straighforward to detect directly B1 (small diagonal entries)
- B2 (breakdown in scaling) also straightforward: no overflow if

$$l_{kk} \geq 1 \quad \text{or} \quad l_{kk} \geq l_{ik}/x_{max}$$

  Otherwise, breakdown B2 detected
- B3 breakdown: needed to avoid overflow in any of the partial results in $l_{ij} \leftarrow l_{ij} - l_{ik}l_{jk}$
- Safe detections can be fast if applied to rows/columns using maximum magnitudes inside them.

Detecting breakdowns

Algorithm

Basic IC factorization with safe checks for breakdown

Initialize $l_{ij} = a_{ij}$ *for all* $(i,j) \in \mathcal{S}\{L\}$
Set $flag = 0$
for $k = 1 : n$ ***do*** ▷ *Start of* k*-th major step*
  If $l_{kk} < \tau$ **then** *Set* $flag = -1$ *and* **return** ▷ *B1 breakdown*
  $l_{kk} \leftarrow (l_{kk})^{1/2}$
  $a = \max_{i=k+1:n}\{|l_{ik}| : (i,k) \in \mathcal{S}\{L\}\}$
  If $l_{kk} \geq 1$ *or* $l_{kk} \geq a/x_{max}$ **then** ▷ *If* $l_{kk} \geq 1$, a *does not need to be computed*
  for $i = k+1 : n$ *such that* $(i,k) \in \mathcal{S}\{L\}$ ***do***
    $l_{ik} \leftarrow l_{ik}/l_{kk}$ ▷ *Perform safe scaling*
  end for ▷ *Column* k *of* L *has been computed*
  Else Set $flag = -2$ *and* **return** ▷ *B2 breakdown*
  for $j = k+1 : n$ *such that* $(j,k) \in \mathcal{S}\{L\}$ ***do***
    Test safe update If not_safe **return** ▷ *B3 breakdown*
    for $i = j : n$ *such that* $(i,j) \in \mathcal{S}\{L\}$ ***do***
      $l_{ij} \leftarrow l_{ij} - l_{ik}l_{jk}$ ▷ *Perform safe update operation*
    end for
  end for ▷ *Column* j *of* L *has been updated*
end for

Detecting breakdowns

- Experimental results uses
  - Safe detection of breakdowns
  - Ill-conditioned matrices shown above
    (also better-conditioned ones in Scott, T., 2024)
- Our work should answer two questions:
  - Are we able to use fp16 in such environment?
  - Are we able to achieve high accuracy when using fp16 preconditioner?

Detecting breakdowns

- Experimental results uses
  - Safe detection of breakdowns
  - Ill-conditioned matrices shown above
    (also better-conditioned ones in Scott, T., 2024)
- Our work should answer two questions:
  - Are we able to use fp16 in such environment?
  - Are we able to achieve high accuracy when using fp16 preconditioner?

Experiments

- NAG Fortran implementation with the full IEEE fp16 computation (storage and operations) (internally simulated)
- Any detected breakdown solved by the shift
- To achieve high accuracy of the solution: GMRES iterative refinement (IR) that may employ more precisions. (CG results in parentheses.)
- IR in full precision vectors. Otherwise, possibility of breakdowns in substitution steps: scaling not necessarily helps.
- A lot of previous work on achieving the high (double precision) accuracy. See Carson, Higham, 2017; but see also the efforts in Arioli, Duff, 2009; Buttari, 2008; Buttari, 2007; more citations in Scott, T., 2024

Experiments

- NAG Fortran implementation with the full IEEE fp16 computation (storage and operations) (internally simulated)
- Any detected breakdown solved by the shift
- To achieve high accuracy of the solution: GMRES iterative refinement (IR) that may employ more precisions. (CG results in parentheses.)
- IR in full precision vectors. Otherwise, possibility of breakdowns in substitution steps: scaling not necessarily helps.
- A lot of previous work on achieving the high (double precision) accuracy. See Carson, Higham, 2017; but see also the efforts in Arioli, Duff, 2009; Buttari, 2008; Buttari, 2007; more citations in Scott, T., 2024

Experiments

- NAG Fortran implementation with the full IEEE fp16 computation (storage and operations) (internally simulated)
- Any detected breakdown solved by the shift
- To achieve high accuracy of the solution: GMRES iterative refinement (IR) that may employ more precisions. (CG results in parentheses.)
- IR in full precision vectors. Otherwise, possibility of breakdowns in substitution steps: scaling not necessarily helps.
- A lot of previous work on achieving the high (double precision) accuracy. See Carson, Higham, 2017; but see also the efforts in Arioli, Duff, 2009; Buttari, 2008; Buttari, 2007; more citations in Scott, T., 2024

Experiments

- NAG Fortran implementation with the full IEEE fp16 computation (storage and operations) (internally simulated)
- Any detected breakdown solved by the shift
- To achieve high accuracy of the solution: GMRES iterative refinement (IR) that may employ more precisions. (CG results in parentheses.)
- IR in full precision vectors. Otherwise, possibility of breakdowns in substitution steps: scaling not necessarily helps.
- A lot of previous work on achieving the high (double precision) accuracy. See Carson, Higham, 2017; but see also the efforts in Arioli, Duff, 2009; Buttari, 2008; Buttari, 2007; more citations in Scott, T., 2024

Iterative refinement

Algorithm

Krylov-based iterative refinement with an incomplete factorization preconditioner using five precisions (IC-Krylov-IR)

Input: *SPD matrix A and vector b in precision u, a Krylov subspace method, and five precisions u_r, u_g, u_p, u and u_ℓ*

Output: *Computed solution of the system $Ax = b$ in precision u*

1: *Compute an incomplete Cholesky factorization of A in precision u_ℓ*
2: *Initialize $x_1 = 0$*
3: ***for*** *i = 1 : itmax or until converged* ***do***
4: *Compute $r_i = b - Ax_i$ in precision u_r; store r_i in precision u*
5: *Solve $Ad_i = r_i$ using the preconditioned Krylov method in precision u_g, preconditioning and products with A performed in precision u_p; d_i stored in precision u* ▷ *Computed factors used as the preconditioner*
6: *Compute $x_{i+1} \leftarrow x_i + d_i$ in precision u*
7: ***end for***

IC(3) preconditioning in fp16 + iterative refinement

	Preconditioner fp16-$IC(3)$					
Identifier	$resfinal$	$nnz(L)$	o_{its}	t_{its}	$nmod$	$nofl$
Boeing/msc01050	6.47×10^{-14}	3.74×10^{4}	3	125 (64)	2	0
HB/bcsstk11	1.89×10^{-13}	4.18×10^{4}	3	265 (184)	0	2
HB/bcsstk26	1.51×10^{-13}	3.43×10^{4}	3	80 (70)	2	0
HB/bcsstk24	1.77×10^{-13}	2.27×10^{5}	3	437 (260)	0	2
HB/bcsstk16	6.64×10^{-15}	4.89×10^{5}	3	17 (17)	0	0
Cylshell/s2rmt3m1	4.11×10^{-15}	2.60×10^{5}	3	83 (117)	0	0
Cylshell/s3rmt3m1	9.01×10^{-15}	2.60×10^{5}	3	**386** (504)	1	1
Boeing/bcsstk38	1.43×10^{-15}	5.64×10^{5}	4	1004 (282)	2	0
Boeing/msc10848	1.13×10^{-14}	2.51×10^{6}	3	138 (89)	0	0
Oberwolfach/t2dah_e	1.88×10^{-16}	3.29×10^{5}	3	6 (6)	0	0
Boeing/ct20stif	1.63×10^{-9}	6.70×10^{6}	3	$> 10^3$ ($> 10^3$)	2	0
DNVS/shipsec8	1.26×10^{-16}	1.22×10^{7}	4	2067 (1399)	2	0
Um/2cubes_sphere	1.63×10^{-16}	8.70×10^{6}	3	6 (6)	0	0
GHS_psdef/hood	5.01×10^{-17}	2.78×10^{7}	4	**444** (**406**)	0	4
Um/offshore	1.38×10^{-13}	2.08×10^{7}	3	**103** (**40**)	0	4

IC(3) preconditioning in fp64 + iterative refinement

	Preconditioner fp64-$IC(3)$				
Identifier	$resfinal$	$nnz(L)$	o_{its}	t_{its}	
Boeing/msc01050	5.80×10^{-14}	3.74×10^{4}	3	38	(25)
HB/bcsstk11	6.96×10^{-14}	4.18×10^{4}	3	29	(29)
HB/bcsstk26	1.68×10^{-13}	3.43×10^{4}	3	60	(62)
HB/bcsstk24	1.01×10^{-13}	2.27×10^{5}	3	71	(67)
HB/bcsstk16	3.13×10^{-15}	4.89×10^{5}	3	15	(14)
Cylshell/s2rmt3m1	8.74×10^{-15}	2.60×10^{5}	3	71	(105)
Cylshell/s3rmt3m1	6.12×10^{-5}	2.60×10^{5}	1	$> 10^{3}$	$(> 10^{3})$
Boeing/bcsstk38	8.34×10^{-14}	5.64×10^{5}	3	154	115)
Boeing/msc10848	2.26×10^{-16}	2.51×10^{6}	3	47	(46)
Oberwolfach/t2dah_e	6.54×10^{-16}	3.29×10^{5}	3	5	(5)
Boeing/ct20stif	2.02×10^{-11}	6.70×10^{6}	3	$> 10^{3}$	$(> 10^{3})$
DNVS/shipsec8	1.11×10^{-16}	1.22×10^{7}	4	313	(181)
Um/2cubes_sphere	1.63×10^{-16}	8.70×10^{6}	3	5	(5)
GHS_psdef/hood	‡	‡	‡	‡	‡
Um/offshore	‡	‡	‡	‡	‡

Implications of the double precision preconditioning

- Possibly MORE than just detecting breakdowns needed
- What about?: bounding off-diagonals in sparse modified Cholesky. Motivated by Gill, Murray, 1974; Gill, Murray, Wright, 2019 (2nd edition); see Fang, O'Leary, 2008.
- Not achievable by checking IEEE exceptions
- The principle:

$$l_{kk} = \max\left\{l_{kk},\ \left(\frac{l_{kmax}}{\beta}\right)^2\right\}, \quad l_{kmax} = \max_{i>k}\{|l_{ik}| : (i,k) \in \mathcal{S}\{L\}\}.$$

- Detected troubles with this GMW(β) treated also as global shifts

Lemma

Let the matrix A be sparse and SPD. Assume that, using the GMW(β) strategy, columns 1 to $j-1$ columns of the IC factor L have been successfully computed in fp16 arithmetic. For $i \ge j$ let $nz(i)$ denote the number of nonzero entries in $L_{i,1:j-1}$. If

$$|a_{ij}| + \min(nz(i), nz(j))\beta^2 \le x_{max} \quad \textit{for all} \quad (i,j) \in \mathcal{S}\{L\},$$

then B3 breakdown cannot occur in the j-th step.

Implications of the double precision preconditioning

- Possibly MORE than just detecting breakdowns needed
- What about?: bounding off-diagonals in sparse modified Cholesky. Motivated by Gill, Murray, 1974; Gill, Murray, Wright, 2019 (2nd edition); see Fang, O'Leary, 2008.
- Not achievable by checking IEEE exceptions
- The principle:

$$l_{kk} = \max\left\{l_{kk},\ \left(\frac{l_{kmax}}{\beta}\right)^2\right\}, \quad l_{kmax} = \max_{i>k}\{|l_{ik}| : (i,k) \in \mathcal{S}\{L\}\}.$$

- Detected troubles with this GMW(β) treated also as global shifts

Lemma

Let the matrix A be sparse and SPD. Assume that, using the GMW(β) strategy, columns 1 to $j-1$ columns of the IC factor L have been successfully computed in fp16 arithmetic. For $i \geq j$ let $nz(i)$ denote the number of nonzero entries in $L_{i,1:j-1}$. If

$$|a_{ij}| + \min(nz(i), nz(j))\beta^2 \leq x_{max} \quad \textit{for all} \quad (i,j) \in \mathcal{S}\{L\},$$

then B3 breakdown cannot occur in the j-th step.

Implications of the double precision preconditioning

- Possibly MORE than just detecting breakdowns needed
- What about?: bounding off-diagonals in sparse modified Cholesky. Motivated by Gill, Murray, 1974; Gill, Murray, Wright, 2019 (2nd edition); see Fang, O'Leary, 2008.
- Not achievable by checking IEEE exceptions
- The principle:

$$l_{kk} = \max\left\{l_{kk},\ \left(\frac{l_{kmax}}{\beta}\right)^2\right\}, \quad l_{kmax} = \max_{i>k}\left\{|l_{ik}| : (i,k) \in \mathcal{S}\{L\}\right\}.$$

- Detected troubles with this GMW(β) treated also as global shifts

Lemma

Let the matrix A be sparse and SPD. Assume that, using the GMW(β) strategy, columns 1 to $j-1$ columns of the IC factor L have been successfully computed in fp16 arithmetic. For $i \ge j$ let $nz(i)$ denote the number of nonzero entries in $L_{i,1:j-1}$. If

$$|a_{ij}| + \min(nz(i), nz(j))\beta^2 \le x_{max} \quad \textit{for all} \quad (i,j) \in \mathcal{S}\{L\},$$

then B3 breakdown cannot occur in the j-th step.

GMW + IC(3) with look-ahead in fp64 + iterative refinement

Identifier	GMW(0.5) *its* ($n2$)	GMW(10) *its* ($n2$)	GMW(100) *its* ($n1, n2$)	IC(3)-LA *its* ($n1$)
Boeing/msc01050	78 (60)	24 (0)	24 (4, 0)	24 (0)
HB/bcsstk11	1087 (476)	201 (0)	201 (0, 0)	232 (0)
HB/bcsstk26	775 (476)	79 (0)	79 (0, 0)	79 (0)
HB/bcsstk24	913 (409)	89 (0)	89 (0, 0)	89 (0)
HB/bcsstk16	41 (26)	22 (0)	22 (0, 0)	22 (0)
Cylshell/s2rmt3m1	792 (585)	146 (0)	146 (0, 0)	146 (0)
Cylshell/s3rmt3m1	2901 (710)	102 (0)	102 (0, 0)	102 (0)
Boeing/bcsstk38	1301 (943)	141 (0)	141 (0, 0)	141 (0)
Boeing/msc10848	790 (600)	68 (0)	68 (0, 0)	68 (0)
Oberwolfach/t2dah_e	14 (6)	6 (0)	6 (0, 0)	6 (0)
Boeing/ct20stif	2122 (4847)	2036 (40)	>1000 (0, 0)	1940 (2)
DNVS/shipsec8	701 (4658)	354 (0)	354 (0, 55)	354 (0)
GHS_psdef/hood	2480 (25054)	>1000 (2013)	>1000 (0, 11998)	568 (5)
Um/offshore	>1000 (4094)	>1000 (6327)	‡(4, 4730)	128 (5)

Apparently a possibility, look-ahead means more tests for B1, $n2$ are local modifications

GMW + IC(3) with look-ahead in fp16 + iterative refinement

Identifier	GMW(0.5) *its* ($n1, n2$)	GMW(10) *its* ($n1, n2$)	GMW(100) *its* ($n1, n2$)	IC(3)-LA *its* ($n1$)
Boeing/msc01050	96 (0, 60)	65 (1, 0)	65 (1, 0)	84 (4)
HB/bcsstk11	1092 (0, 476)	> 1000 (0, 310)	205^* (0, 0)	205 (1)
HB/bcsstk26	786 (0, 476)	111 (1, 0)	111 (1, 0)	87 (1)
HB/bcsstk24	1018 (0, 446)	> 1000 (0, 428)	$428^\sharp$ (0, 0)	428 (1)
HB/bcsstk16	41 (0, 26)	23 (0, 0)	23 (0, 0)	23 (0)
Cylshell/s2rmt3m1	787 (0, 584)	155 (0, 0)	155 (0, 0)	155 (0)
Cylshell/s3rmt3m1	2017 (1, 710)	630 (2, 0)	630 (2, 0)	630 (2)
Boeing/bcsstk38	1335 (1, 914)	313 (1, 0)	$313^\sharp$ (0, 0)	313 (1)
Boeing/msc10848	684 (0, 591)	81 (0, 0)	81 (0, 0)	81 (0)
Oberwolfach/t2dah_e	11 (0, 6)	7 (0, 0)	7 (0, 0)	7 (0)
Boeing/ct20stif	2139 (0, 4827)	1900 (2, 0)	1900 (2, 0)	1900 (2)
DNVS/shipsec8	2569 (1, 1)	2390 (1, 0)	2390 (1, 0)	1492 (1)
GHS_psdef/hood	2459 (0, 25074)	581^* (0, 0)	581 (5, 0)	581 (5)
Um/offshore	> 1000 (0, 3846)	> 1000 (0, 5838)	2013 (4, 2)	129 (5)

Also fp16 works well with GMW

Conclusions

- fp16 IC preconditioning possible. All breakdowns including overflows can be safely detected.
- The way to convert them to global shifts is viable.
- Extensions to other IC, ILU possible.
- Monitoring growth (as done in GMW) can solve related problems, like breakdowns in fp64.
- But note that development of more adaptive preconditioner should be done hand in hand with the output of reproducible software
- Heretic questions (in Cathar coutry): Do we need such (double precision) accuracy? What do we have to pay for the IR? (Suspicion is that that the price is not small) ☺

Conclusions

- fp16 IC preconditioning possible. All breakdowns including overflows can be safely detected.
- The way to convert them to global shifts is viable.
- Extensions to other IC, ILU possible.
- Monitoring growth (as done in GMW) can solve related problems, like breakdowns in fp64.
- But note that development of more adaptive preconditioner should be done hand in hand with the output of reproducible software
- Heretic questions (in Cathar coutry): Do we need such (double precision) accuracy? What do we have to pay for the IR? (Suspicion is that that the price is not small) ☺

Conclusions

- fp16 IC preconditioning possible. All breakdowns including overflows can be safely detected.
- The way to convert them to global shifts is viable.
- Extensions to other IC, ILU possible.
- Monitoring growth (as done in GMW) can solve related problems, like breakdowns in fp64.
- But note that development of more adaptive preconditioner should be done hand in hand with the output of reproducible software
- Heretic questions (in Cathar coutry): Do we need such (double precision) accuracy? What do we have to pay for the IR? (Suspicion is that that the price is not small) ☺

Conclusions

- fp16 IC preconditioning possible. All breakdowns including overflows can be safely detected.
- The way to convert them to global shifts is viable.
- Extensions to other IC, ILU possible.
- Monitoring growth (as done in GMW) can solve related problems, like breakdowns in fp64.
- But note that development of more adaptive preconditioner should be done hand in hand with the output of reproducible software
- Heretic questions (in Cathar coutry): Do we need such (double precision) accuracy? What do we have to pay for the IR? (Suspicion is that that the price is not small) ☺

Conclusions

- fp16 IC preconditioning possible. All breakdowns including overflows can be safely detected.
- The way to convert them to global shifts is viable.
- Extensions to other IC, ILU possible.
- Monitoring growth (as done in GMW) can solve related problems, like breakdowns in fp64.
- But note that development of more adaptive preconditioner should be done hand in hand with the output of reproducible software
- Heretic questions (in Cathar coutry): Do we need such (double precision) accuracy? What do we have to pay for the IR? (Suspicion is that that the price is not small) ☺

Conclusions

- fp16 IC preconditioning possible. All breakdowns including overflows can be safely detected.
- The way to convert them to global shifts is viable.
- Extensions to other IC, ILU possible.
- Monitoring growth (as done in GMW) can solve related problems, like breakdowns in fp64.
- But note that development of more adaptive preconditioner should be done hand in hand with the output of reproducible software
- Heretic questions (in Cathar coutry): Do we need such (double precision) accuracy? What do we have to pay for the IR? (Suspicion is that that the price is not small) ☺

Last but not least

Thank you for your attention!