Interprecision Transfers in Iterative Refinement

Making Half Precision on Desktops Less Painful

C. T. Kelley
NC State University
tim_kelley@ncsu.edu

Supported by DOE, NSF

XSDK-MULTIPRECISION, June 15, 2023

Outline

References

- CH18: E. Carson and N. J. Higham, Accelerating the solution of linear systems by iterative refinement in three precisions, SIAM Journal on Scientific Computing, 40 (2018), pp. A817–A847.
- H96: N. J. Higham, Accuracy and Stability of Numerical Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1996.
- HPZ19: N. J. Higham, S. Pranesh, and M. Zounon, Squeezing a matrix into half precision, with an application to solving linear systems, SIAM J. Sci. Comp., 41 (2019), pp. A2536–A2551.
- CTK22P: C. T. Kelley, Newton's method in mixed precision, SIAM Review, 64 (2022), pp. 191–211.
- CTK22B: C. T. Kelley, Solving Nonlinear Equations with Iterative Methods: Solvers and Examples in Julia, no. 20 in Fundamentals of Algorithms, SIAM, Philadelphia, 2022.

IR from textbooks

```
$r = b - Ax$
Factor $A = LU$ in low precision
**while** $\|r\|$ too large **do**
    $d = U^{-1}L^{-1}r$
    $x \leftarrow x + d$
    $r = b - Ax$
**end while**
```

Not clear what “factor in low precision” and $d = U^{-1}L^{-1}r$ mean

Interprecision transfers

- This is mostly a two-precision talk.
- $\mathcal{F}$ = set of floats, u = unit roundoff, fl rounding operator
- $\mathcal{F}^N$, $\mathcal{F}^{N \times N}$ vectors and matrices
- High (working and residual) precision = FP64, $\mathcal{F}_h$, u_h, fl_h
- Low (factorization precision) = FP32 or FP16, $\mathcal{F}_l$, u_l, fl_l
- I_s^t interprecision transfer from source s to target t
- We will explicitly put the interprecision transfers in the algorithms.

Interprecision transfer is more than rounding

- Memory allocation
- Data movement
- This matters even within registers because $u_l \in \mathcal{F}_l$, $a, b \in \mathcal{F}_h$ implies
  - $fl_h(u_l * a + b) = fl_h(I_l^h(u_l) * a + b)$ so
  - Promotion happens before binary operations.

Consider the triangular solve.

Begin with $\mathrm{A} \in \mathcal{F}_h^{N \times N}$, $\mathrm{b} \in \mathcal{F}_h^N$

- HPF: Factor $\mathrm{A} = \mathrm{LU}$
- LPF: Factor $I_h^l(\mathrm{A}) = \mathrm{A}_l = \mathrm{L}_l\mathrm{U}_l$
- Three triangular solves
  - HPS: $(\mathrm{LU})^{-1}\mathrm{b}$
  - LPS: $(\mathrm{L}_l\mathrm{U}_l)^{-1} I_h^l(\mathrm{b})$
  - MPS: $(\mathrm{L}_l\mathrm{U}_l)^{-1}\mathrm{b}$

    Don't forget $fl_h(u_l * a + b) = fl_h(I_l^h(u_l) * a + b)$

Julia 1.9.0, OpenBLAS, $A = rand(N, N)$

Timings, 2023 Mac Mini; M2 Pro; Double-Single

N	HPF	LPF	HPS	MPS	LPS
512	1.05e-03	9.77e-04	5.03e-05	1.00e-04	2.83e-05
1024	3.96e-03	2.98e-03	1.88e-04	4.31e-04	1.02e-04
2048	2.36e-02	1.46e-02	8.96e-04	3.70e-03	4.07e-04
4096	1.57e-01	8.61e-02	4.81e-03	1.47e-02	2.27e-03
8192	1.24e+00	6.13e-01	1.95e-02	5.88e-02	9.86e-03

Difference (MPS/LPS) a factor of 3–9. Performance problem in (CTK22B).

MPS is even more costly than high precision triangular solves.

This is not an issue in Matlab.

MPS vs LPS in IR

- Julia and LAPACK do MPS
  - will promote with each binary operation in MPS.
  - This is the pain point in the triangular solves.
- Fix for IR: Avoid $d = (L_l U_l)^{-1} r$ and use LPS
  - Scale and move r to the lower precision.
  - Do the solves and move back.
  - Remove the scaling.
  - So it's $d = \|r\| I_l^h ((L_l U_l)^{-1} I_h^l (r / \|r\|))$
- Matlab does $I_h^r(r)$ for you, so LPS is automatic.
- Most of you use LPS.

IR with LPS: explicit interprecision transfers

```
$r = b - Ax$
Factor $I_h^l(A) = A_l = L_l U_l$
**while** $\|r\|$ too large **do**
    (LPS) $d = \|r\| I_l^h((L_l U_l)^{-1} I_h^l(r/\|r\|))$
    $x \leftarrow x + d$
    $r = b - Ax$
**end while**
```

IR-LPS as a fixed point iteration

$$x \leftarrow G(x) \equiv x + \|r\| I_l^h (L_l U_l)^{-1} I_h^l (r/\|r\|)$$

where $r = b - Ax$.
G is not only nonlinear, it is not even continuous.
This makes IR a pain to analyze, but is a pedantic worry.

IR with LPS

If all the scaling does is avoid underflow, then

$$G(x) \approx x + (L_l U_l)^{-1} r + \delta_r$$

So it's almost the same as IR-MPS. Difference is

$$\delta_r = (L_l U_l)^{-1}(r - \|r\| I_h^l(r/\|r\|))$$

and $\|\delta_r\| \leq u_l \|(L_l U_l)^{-1}\| \|r\|$.

Classic (H96 + refs) Estimates for MPS

IR-MPS is a stationary iterative method

$$x \leftarrow x + U_l^{-1} L_l^{-1} (b - Ax)$$

with iteration matrix

$$M = I - U_l^{-1} L_l^{-1} A = U_l^{-1} L_l^{-1} (L_l U_l - A)$$

What is $\Delta A = A - L_l U_l$?

The classic estimates ignore the interprecision transfers to get

$$|\Delta A| \le \gamma^l_{3N} |L_l||U_l|$$

(see eq 7.1 of CH18)
But A_l is missing. Put it in to get

$$\begin{aligned} |\Delta A| &= |A - A_l + A_l - L_l U_l| \le |A - A_l| + |A_l - L_l U_l| \\ &\le u_l |A| + \gamma^l_{3N} |L_l||U_l| \end{aligned}$$

$u_l|A|$ is not likely to matter much, but it is there.

Estimate $\|M\|$

$$
\begin{aligned}
\|M\| &\le \| \, |U_l{}^{-1}| \, |L_l{}^{-1}| \, |\Delta A| \, \| \\
&\le u_l(\||U_l{}^{-1}||L_l{}^{-1}||A|\| + 3N\||U_l{}^{-1}| \, |L_l{}^{-1}| \, |L_l| \, |U_l|\|) \\
&\le u_l\|U_l^{-1}\|\|L_l^{-1}\|(\|A\| + \|L_l\|\|U_l\|)
\end{aligned}
$$

Effect on estimates in CH18

To get the convergence results from Section 7 in the case

- Factor in low precision
- do everything else in high

If

$$\phi_1 \equiv u_l \|U_l^{-1}\| \|L_l^{-1}\| (\|A\| + \|L_l\| \|U_l\|) << 1$$

then the bottom line from CH18 does not change.

Using $U_l^{-1} L_l^{-1}$ as a preconditioner

We just found that

$$\|U_l^{-1} L_l^{-1} A\| \le 1 + \phi_1$$

Also

$$\begin{aligned} \|A^{-1} L_l U_l\| &\le 1 + \|A^{-1}\| \|\Delta A\| \\ &\le 1 + \|A^{-1}\| (\|A\| + \|L_l\| \|U_l\|) \equiv 1 + \phi_2 \end{aligned}$$

So $\kappa(U_l^{-1} L_l^{-1} A) \le (1 + \phi_1)(1 + \phi_2)$.

Solving in high precision for preconditioning (CH)

- LPS won't do the job.
- Must one return to MPS: $U_l^{-1} L_l^{-1} r$?
  Yes, but you can reformulate and trade storage for time.

Remember the assumption: true for Intel and Apple Mx CPU

Assumption: If

- x_l is low precision,
- a and b are high precision

then computing $x_l * a + b$ returns

$$fl_h(I_l^h(x_l) * a + b)$$

So the low precision number is promoted before the operations begin.
Not true for the Apple Accelerator Framework on Mx chips.

Heavy IR: I

- Factor $I_h^l \mathrm{A} = \mathrm{A}_l = \mathrm{L}_l \mathrm{U}_l$ in low precision
- Promote the factors to high precision to get

$$\hat{\mathrm{L}} = I_l^h(\mathrm{L}_l) \text{ and } \hat{\mathrm{U}} = I_l^h(\mathrm{U}_l)$$

- solve the correction equation in high precision via

$$\mathrm{d} = (\hat{\mathrm{L}}\hat{\mathrm{U}})^{-1}\mathrm{r}$$

  with the promoted factors
- This is **equivalent to MPS**. Look at the loops to see.

Heavy IR: II. New(?) version of MPS

```
$r = b - Ax$
Factor $I_h^l A$ to obtain $L_l$ and $U_l$
Promote the factors to obtain $\hat{L}$ and $\hat{U}$.
**while** $\|r\|$ too large **do**
    $d = (\hat{L}\hat{U})^{-1} r$
    $x \leftarrow x + d$
    $r = b - Ax$
**end while**
```

Why do this?

- Bad
  - $\hat{A} = \hat{L}\hat{U}$ costs the same as A to store, so the storage burden is heavy
  - Triangular solves are in high precision
- Good
  - Faster to do MPS this way for GMRES-IR.
    Avoid interprecision transfers within the iteration.
  - Makes half precision experiments on desktops less painful
    eg: Use factorization for several nonlinear iterations
  - Can reuse space for A_l for Krylov vectors ...

Simple model problem

Composite midpoint discretization of

$$(\mathcal{A}u)(x) \equiv u(x) - \alpha \int_0^1 g(x, y)u(y)\,dy = f(x)$$

where g is the discretization of the Greens function for the negative Laplacian with homogeneous Dirichlet boundary conditions.

$$g(x, y) = \begin{cases} y(1-x) & \text{if } x > y \\ x(1-y) & \text{if } y \geq x \end{cases}$$

$\mathcal{A}$ is self-adjoint and positive definite if $\alpha < \pi^2$.

Experiments with $\alpha = 800$

- $\mathcal{A}$ is singular if $\alpha = 9^2\pi^2 \approx 799.4$
  and hence ill-conditioned for $\alpha = 800$
- For u_h = FP64 and u_l = FP32 and FP16 we tabulate
  - Norm of iteration matrix $\mathrm{I} - \hat{\mathrm{A}}^{-1}\mathrm{A}$
  - Norm of $\hat{\mathrm{A}}^{-1}(\mathrm{A} - \mathrm{A}_l)$ to see if it matters
  - Condition number of $\hat{\mathrm{A}}^{-1}\mathrm{A}$

$u_l = FP32,\ u_h = FP64,\ \alpha = 800$

N	$\kappa(A)$	$\|\hat{A}^{-1}(A - A_l)\|_2$	$\|I - \hat{A}^{-1}A\|_2$	$\kappa(\hat{A}^{-1}A)$
2048	1.11e+05	7.45e-05	1.39e-03	1.00e+00
4096	1.13e+05	5.81e-05	9.95e-03	1.01e+00
8192	1.14e+05	3.90e-05	3.17e-03	1.00e+00

$u_l = FP16, \ u_h = FP64, \ \alpha = 800$

N	$\kappa(A)$	$\|\hat{A}^{-1}(A - A_l)\|_2$	$\|I - \hat{A}^{-1}A\|_2$	$\kappa(\hat{A}^{-1}A)$
2048	1.11e+05	8.19e-03	1.27e+00	1.34e+02
4096	1.13e+05	2.65e-03	1.51e+00	3.89e+02
8192	1.14e+05	2.89e-03	3.92e+00	1.54e+03

And so . . .

- $\hat{A}^{-1}(A - A_l)$ is negligible
  and was in every other experiment we did
- Conditioning for u_l = FP16 looks bad.
  - GMRES-IR does well anyway if you
    - make the GMRES tolerance very tight (10^{-6}) and
    - allocate lots of room for Krylov vectors
  - There are many problem eigenvalues and they need many GMRES iterations.
- We tried scaling A and got no change in the results.

Half precision is slow, but getting better

LU timings: 8192x8192 Random

CPU	Double	Single	Half	T_{16}/T_{64}
A	1.37e+00	6.07e-01	3.92e+02	287
B	1.10e+00	5.94e-01	1.16e+02	106
C	1.17e+00	6.10e-01	6.46e+01	55

A: 2019 8 core Intel iMac, Julia 1.8.5

B: 2023 M2 Pro, Julia 1.8.5

C: 2023 M2 Pro, Julia 1.9.1

BLAS and LAPACK not there, but . . .

- Julia 1.9.1 using 8 threads
- 2023 M2 Pro, 8 performance cores
- Brute force $A^T * B$ matrix multiply. N=8000.

Timings in seconds $\quad T_{64} = 26 \quad T_{32} = 10 \quad T_{16} = 4.5$

Summary

- Interprecision transfers are costly.
  - Avoid on-the-fly transfers by synchronizing precision before matrix operations
  - You knew this already.
- Some consequences
  - Heavy IR
  - Heavy GMRES-IR