The Journal of Open Source Software

MultiPrecisionArrays.jl: A Julia package for iterative refinement

C. T. Kelley [1]

1 North Carolina State University, Raleigh NC, USA

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

MultiPrecisionArrays.jl, (Kelley, 2024b) provides data structures and solvers for several variations of iterative refinement (IR). IR can speed up an LU matrix factorization for solving linear systems of equations by factoring a low precision copy of the matrix and using that low precision factorization in an iteration to solve the system. For example, if high precision is double and low precision is single, then the factorization time is cut in half. The additional storage cost is the low precision copy, so IR is at time vs storage trade off. IR has a long history and a good account of the classical theory is in (Higham, 1996).

Statement of need

The solution of linear systems of equations is a ubiquitous task in computational science and engineering. A common method for dense systems is Gaussian elimination done via an LU factorization, (Higham, 1996). Iterative refinement is a way to reduce the factorization time at the cost of additional storage. MultiPrecisionArrays.jl enables IR with a simple interface in Julia (Bezanson et al., 2017) with an IR factorization object that one uses in the same way as the one for LU. The package offers several variants of IR, both classical (Higham, 1996; Wilkinson, 1948), and some from the recent literature (Amestoy et al., 2024; Carson & Higham, 2017).

Algorithm

This package will make solving dense systems of linear equations faster by using the LU factorization and IR. While other factorizations can be used in IR, the package is limited to LU for now. A very generic description of this for solving a linear system $Ax = b$ in a high (working) precision is

IR(A, b)

- $x = 0$
- $r = b$
- Factor $A = LU$ in a lower precision
- While $\|r\|$ is too large
  - $d = (LU)^{-1}r$
  - $x = x + d$
  - $r = b - Ax$
- end

- end

In Julia, a code to do this would solve the linear system $Ax = b$ in the working precision, say double, by using a factorization in a lower (factorization) precision, say single, within a residual correction iteration. This means that one would need to allocate storage for a copy of A in the factorization precision and factor that copy.

The multiprecision factorization `mplu` makes the low precision copy of the matrix, factors that copy, and allocates some storage for the iteration. The original matrix and the low precision factorization are stored in a factorization object that you can use with `\`.

IR is a perfect example of a storage/time trade off. To solve a linear system $Ax = b$ in R^N with IR, one incurs the storage penalty of making a low precision copy of A and reaps the benefit of only having to factor the low precision copy.

Installation

The standard way to install a package is to type `import.Pkg; Pkg.add("MultiPrecisionArrays")` at the Julia prompt. One can run the unit tests with `Pkg.test("MultiPrecisionArrays")`. After installation, type `using MultiPrecisionArrays` when you want to use the functions in the package.

There are only two direct dependencies outside of the Julia standard libraries. The factorization in half precision (Float16) uses OhMyThreads.jl. The GMRES and Bi-CGSTAB solvers for Krylov-IR methods are taken from SIAMFANL.jl (Kelley, 2022c).

A Few Subtleties

Within the algorithm one has to determine what the line $d = (LU)^{-1}r$ means. Does one cast r into the lower precision before the solve or not? If one casts r into the lower precision, then the solve is done entirely in the factorization precision. If, however, r remains in the working precision, then the LU factors are promoted to the working precision on the fly. This makes little difference if TW is double and TF is single and there is a modest performance benefit to downcasting r into single. Therefore that is the default behavior in that case. If TF is half precision, `Float16`, then it is best to do the interprecision transfers on the fly and if one is using one of the Krylov-IR algorithms (Amestoy et al., 2024) then one must do the interprecision transfers on the fly and not downcast r.

There are two half precision (16 bit) formats. Julia has native support for IEEE 16 bit floats (Float16). A second format (BFloat16) has a larger exponent field and a smaller significand (mantissa), thereby trading precision for range. In fact, the exponent field in BFloat is the same size (8 bits) as that for single precision (Float32). The significand, however, is only 8 bits. Compare this to the size of the exponent fields for Float16 (11 bits) and single (24 bits). The size of the significand means that you can get in real trouble with half precision in either format and that IR is more likely to fail to converge. GMRES-IR can mitigate the convergence problems (Amestoy et al., 2024) by using the low-precision solve as a preconditioner. We support both GMRES (Saad & Schultz, 1986) and BiCGSTAB (Vorst, 1992) as solvers for Krylov-IR methods. One should also know that LAPACK and the BLAS do not yet support half precision arrays, so working in Float16 will be slower than using Float64.

The classic algorithm from (Wilkinson, 1948) and its recent extension (Carson & Higham, 2017) evaluate the residual in a higher precision that the working precision. This can give improved accuracy for ill-conditioned problems at a cost of the interprecision transfers in the residual computation. This needs to be implemented with some care and (Demmel et al., 2006) has an excellent account of the details.

MultiPrecisionArrays.jl provides infrastructure to manage these things and we refer the reader to (Kelley, 2024b) for the details.

Projects using MultiPrecisionArrays.jl.

This package was motivated by the use of low-precision factorizations in Newton's method Kelley (2022c) and the interface between a preliminary version of this package and the solvers from Kelley (2022b) was reported in (Kelley, 2023). That paper used a three precision form of IR (TF=half, TW=single, nonlinear residual computed in double) and required direct use of multiprecision constructors that we do not export in **MultiPrecisionArrays.jl**. We will fully support the application to nonlinear solvers in a future version. We give a detailed account of interprecision transfers in (Kelley, 2024a) and use **MultiPrecisionArrays.jl** to generate the table in that paper.

Other Julia Packages for IR

The package IterativeRefinement.jl is an implementation of the IR method from (J.Dongarra et al., 1983). It has not been updated in four years.

The unregistered package Itref.jl implements IR and the GMRES-IR method from (Amestoy et al., 2024) and was used to obtain the numerical results in that paper. It does not provide the data structures for preallocation that we do and does not seem to have been updated lately.

Acknowledgements

This work was partially supported by Department of Energy grant DE-NA003967. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the United States Department of Energy.

References

Amestoy, P., Buttari, A., Higham, N. J., L'Excellent, J.-Y., Mary, T., & Vieublé, B. (2024). Five-precision GMRES-based iterative refinement. *SIAM Journal on Matrix Analysis and Applications*, *45*(1), 529–552. https://doi.org/10.1137/23M1549079

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, *59*, 65–98. https://doi.org/10.1137/141000671

Carson, E., & Higham, N. J. (2017). A new analysis of iterative refinement and its application of accurate solution of ill-conditioned sparse linear systems. *SIAM Journal on Scientific Computing*, *39*(6), A2834–A2856. https://doi.org/10.1137/17M1122918

Demmel, J., Hida, Y., & Kahan, W. (2006). Error bounds from extra-precise iterative refinement. *ACM Trans. Math. Soft.*, 325–351. https://doi.org/10.1145/1141885.1141894

Higham, N. J. (1996). *Accuracy and stability of numerical algorithms* (p. xxviii+688). Society for Industrial; Applied Mathematics. https://doi.org/10.1137/1.9780898718027

J.Dongarra, J., B.Moler, C., & H.Wilkinson, J. (1983). Improving the accuracy of computed eigenvalues and eigenvectors. *SIAM Journal on Numerical Analysis*, *20*, 23–45. https://doi.org/10.1137/0720002

Kelley, C. T. (2022a). Newton's method in mixed precision. *SIAM Review*, *64*, 191–211. https://doi.org/10.1137/20M1342902

Kelley, C. T. (2022b). *SIAMFANLEquations.jl*. https://github.com/ctkelley/SIAMFANLEquations.jl. https://doi.org/10.5281/zenodo.4284807

Kelley, C. T. (2022c). *Solving Nonlinear Equations with Iterative Methods: Solvers and Examples in Julia*. SIAM, Philadelphia. https://doi.org/10.1137/1.9781611977271

Kelley, C. T. (2023). *Newton's method in three precisions*. https://doi.org/10.48550/arXiv.2307.16051

Kelley, C. T. (2024a). *Interprecision transfers in iterative refinement*. https://arxiv.org/abs/2407.00827

Kelley, C. T. (2024b). *Using MultiPrecisionArrays.jl: Iterative refinement in Julia*. https://doi.org/10.48550/arXiv.2311.14616

Saad, Y., & Schultz, M. H. (1986). GMRES a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comp.*, *7*, 856–869. https://doi.org/10.1137/0907058

Vorst, H. A. van der. (1992). Bi-CGSTAB: A fast and smoothly converging variant to Bi-CG for the solution of nonsymmetric systems. *SIAM J. Sci. Stat. Comp.*, *13*, 631–644. https://doi.org/10.1137/0913035

Wilkinson, J. H. (1948). *Progress report on the automatic computing engine* (No. MA/17/1024). Mathematics Division, Department of Scientific; Industrial Research, National Physical Laboratory, Teddington, UK. http://www.alanturing.net/turing_archive/archive/l/l10/l10.php