




# faer: A linear algebra library for the Rust programming language

Sarah El Kazdadi <sup>1</sup>

<sup>1</sup> Independent Researcher, France

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

## Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Jed Brown](#)  

## Reviewers:

- [@fcl](#)
- [@mhoemmen](#)

Submitted: 14 November 2023

Published: unpublished

## License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

## Summary

faer is a high performance dense linear algebra library written in Rust. The library offers a convenient high level API for performing matrix decompositions and solving linear systems. This API is built on top of a lower level API that gives the user more control over the memory allocation and multithreading settings.

The library provides a Mat type, allowing for quick and simple construction and manipulation of matrices, as well as lightweight view types MatRef and MatMut for building memory views over existing data.

Multiple scalar types are supported, and the library code is generic over the data type. Native floating point types f32, f64, c32, and c64 are supported out of the box, as well as any user-defined types that satisfy the requested interface.

## Statement of need

Rust was chosen as a language for the library since it allows full control over the memory layout of data and exposes low level CPU intrinsics for SIMD computations. Additionally, its memory safety features make it a perfect candidate for writing efficient and parallel code, since the compiler statically checks for errors that are common in other low level languages, such as data races and fatal use-after-free errors.

Rust also allows compatibility with the C ABI, allowing for simple interoperability with C, and most other languages by extension.

Aside from faer, the Rust ecosystem lacks high performance matrix factorization libraries that aren't C library wrappers, which presents a distribution challenge and can impede generic programming.

## Features

faer exposes a central Entity trait that allows users to describe how their data should be laid out in memory. For example, native floating point types are laid out contiguously in memory to make use of SIMD features, while complex types have the option of either being laid out contiguously or in a split format. The latter is also called a zomplex data type in CHOLMOD (Chen et al. (2008)). An example of a type that benefits immensely from this is the double-double type, which is composed of two f64 components, stored in separate containers. This separate storage scheme allows us to load each chunk individually to a SIMD register, opening new avenues for generic vectorization.

The library generically implements algorithms for matrix multiplication, based on the approach of Van Zee & van de Geijn (2015). For native types, faer uses explicit SIMD depending on

the detected CPU features. The library then uses matrix multiplication as a building block to implement commonly used matrix decompositions, based on state of the art algorithms in order to guarantee numerical robustness:

- Cholesky (LLT, LDLT and Bunch-Kaufman LDLT),
- QR (with and without column pivoting),
- LU (with partial and full pivoting),
- SVD (with or without singular vectors, thin or full),
- eigenvalue decomposition (with or without eigenvectors).

For algorithms that are memory-bound and don't make much use of matrix multiplication, faer uses optimized fused kernels. This can immensely improve the performance of the QR decomposition with column pivoting, the LU decomposition with full pivoting, as well as the reduction to condensed form to prepare matrices for the SVD or eigenvalue decomposition, as described by Van Zee et al. (2012).

State of the art algorithms are used for each decomposition, allowing performance that matches or even surpasses other low level libraries such as OpenBLAS (Wang et al. (2013)), LAPACK (Anderson et al. (1999)), and Eigen (Guennebaud et al. (2010)).

To achieve high performance parallelism, faer uses the Rayon library (Rayon developers (2015)) as a backend, and has shown to be competitive with other frameworks such as OpenMP (Chandra et al. (2001)) and Intel Thread Building Blocks (Pheatt (2008)).

## Performance

Here we present the benchmarks for a representative subset of operations that showcase our improvements over the current state of the art.

The benchmarks were run on an 11th Gen Intel(R) Core(TM) i5-11400 @ 2.60GHz with 12 threads. Eigen is compiled with the `-fopenmp` flag to enable parallelism.

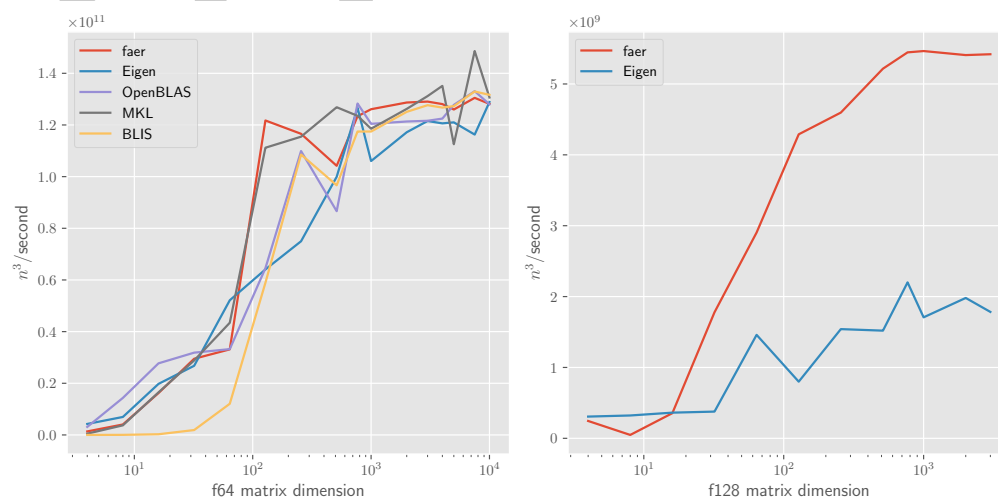


Figure 1:  $n^3$  over run time of matrix multiplication. Higher is better

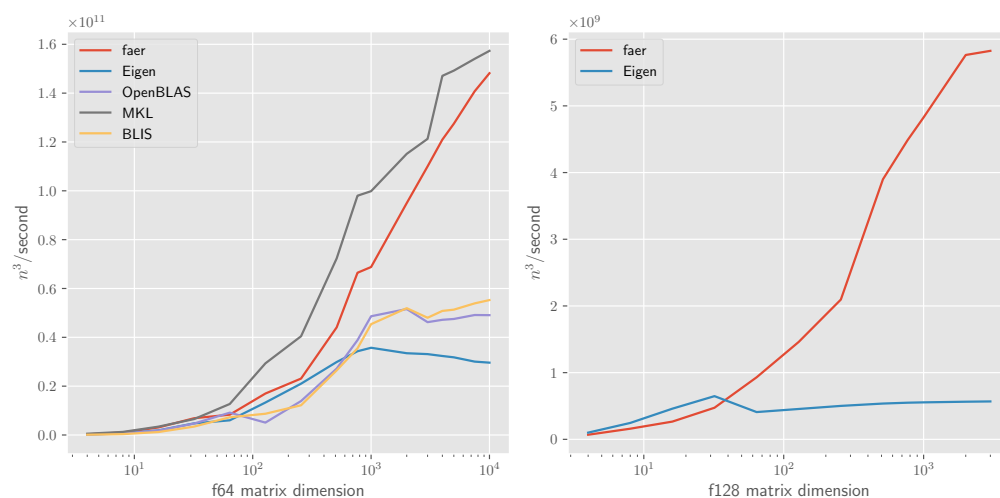


Figure 2:  $n^3$  over run time of QR decomposition. Higher is better

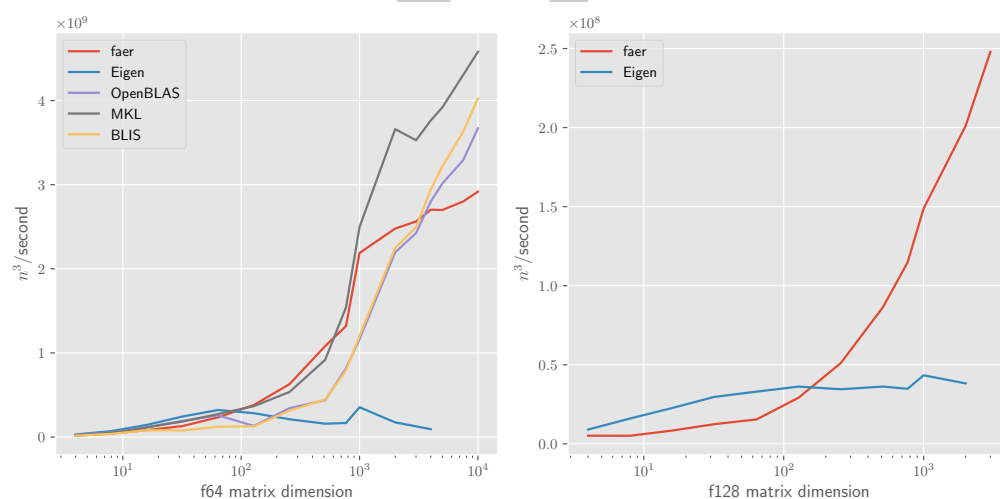


Figure 3:  $n^3$  over run time of eigenvalue decomposition. Higher is better

## Future work

We have so far focused mainly on dense matrix algorithms, which will eventually form the foundation of supernodal sparse decompositions. Sparse algorithm implementations are still a work in progress and will be showcased in a future paper.

## References

- Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., & Sorensen, D. (1999). *LAPACK users' guide* (Third). Society for Industrial; Applied Mathematics.
- Chandra, R., Dagum, L., Kohr, D., Menon, R., Maydan, D., & McDonald, J. (2001). *Parallel programming in OpenMP*. Morgan kaufmann.

- 72 Chen, Y., Davis, T. A., Hager, W. W., & Rajamanickam, S. (2008). Algorithm 887:  
73 CHOLMOD, supernodal sparse cholesky factorization and update/downdate. *ACM Trans.*  
74 *Math. Softw.*, 35(3). <https://doi.org/10.1145/1391989.1391995>
- 75 Guennebaud, G., Jacob, B., & others. (2010). *Eigen v3*. <http://eigen.tuxfamily.org>.
- 76 Pheatt, C. (2008). Intel® threading building blocks. *J. Comput. Sci. Coll.*, 23(4), 298.  
77 <https://doi.org/10.1016/b978-0-12-803761-4.00011-3>
- 78 Rayon developers. (2015). *Rayon*. <https://github.com/rayon-rs/rayon>.
- 79 Van Zee, F. G., & van de Geijn, R. A. (2015). BLIS: A framework for rapidly instantiating  
80 BLAS functionality. *ACM Transactions on Mathematical Software*, 41(3), 14:1–14:33.  
81 <https://doi.acm.org/10.1145/2764454>
- 82 Van Zee, F. G., van de Geijn, R. A., Quintana-Ortí, G., & Elizondo, G. J. (2012). Families of  
83 algorithms for reducing a matrix to condensed form. *ACM Trans. Math. Softw.*, 39(1).  
84 <https://doi.org/10.1145/2382585.2382587>
- 85 Wang, Q., Zhang, X., Zhang, Y., & Yi, Q. (2013). AUGEM: Automatically generate high  
86 performance dense linear algebra kernels on X86 CPUs. *Proceedings of the International*  
87 *Conference on High Performance Computing, Networking, Storage and Analysis*. <https://doi.org/10.1145/2503210.2503219>  
88

DRAFT