# GEMM Benchmarks

devald@stanford.edu

06 June 2026

From the last note it's clear that LAK does well for "shorter" $C$ ($m < 64$), but lags behind OpenBLAS for "taller" $C$. This is because LAK did not block $C$ by `MC`, but worked on full contiguous column panels. As matrices got taller, full column-panels of $C$ were not able to be kept close in cache for the next `KC` block of $A$ and $B$ to accumulate into.

I wrote a second GEMM implementation, meant for taller matrices that does block by `MC`. Internally, `gemm` uses the shorter matrix variant until $m$ exceeds some preset constant threshold. Then it switches to using the taller matrix variant.

Below are the GEMM benchmarks for the common no-transpose $\times$ no-transpose case. I made benchmarks to both highlight LAK's behavior for small $m$ matrices and overall performance as $n$ approaches 1792. I use the colorblind-safe Okabe-ito pallete.

These benchmarks may change once I better tune blocking constants for the taller matrix implementation.

---

The benchmarks were run on Apple M4, 16GB unified memory.

All benchmarks are single-threaded and were run on square $n \times n$ matrices. To compare against faer I set $\beta = 0$ for the BLAS libraries, including `lak`, and $\beta =$`Accum::Replace` for faer. This makes the benchmark only time the operation $C \leftarrow \alpha AB$.

Each plot shows:

- LAK - (portable-simd, safe Rust)
- OpenBLAS armv8
- BLIS
- faer

## Small Matrices

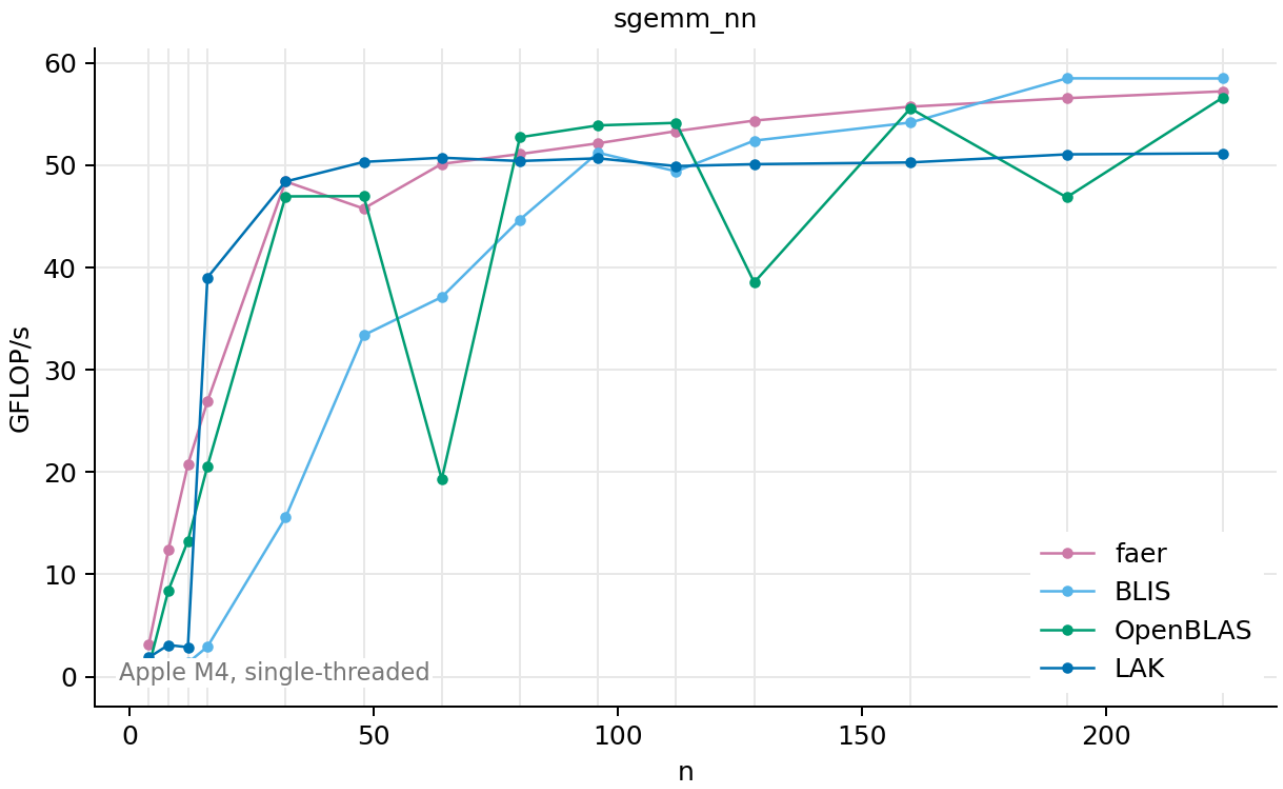


Figure 1: Single-precision GEMM benchmark for small $m$ matrices.

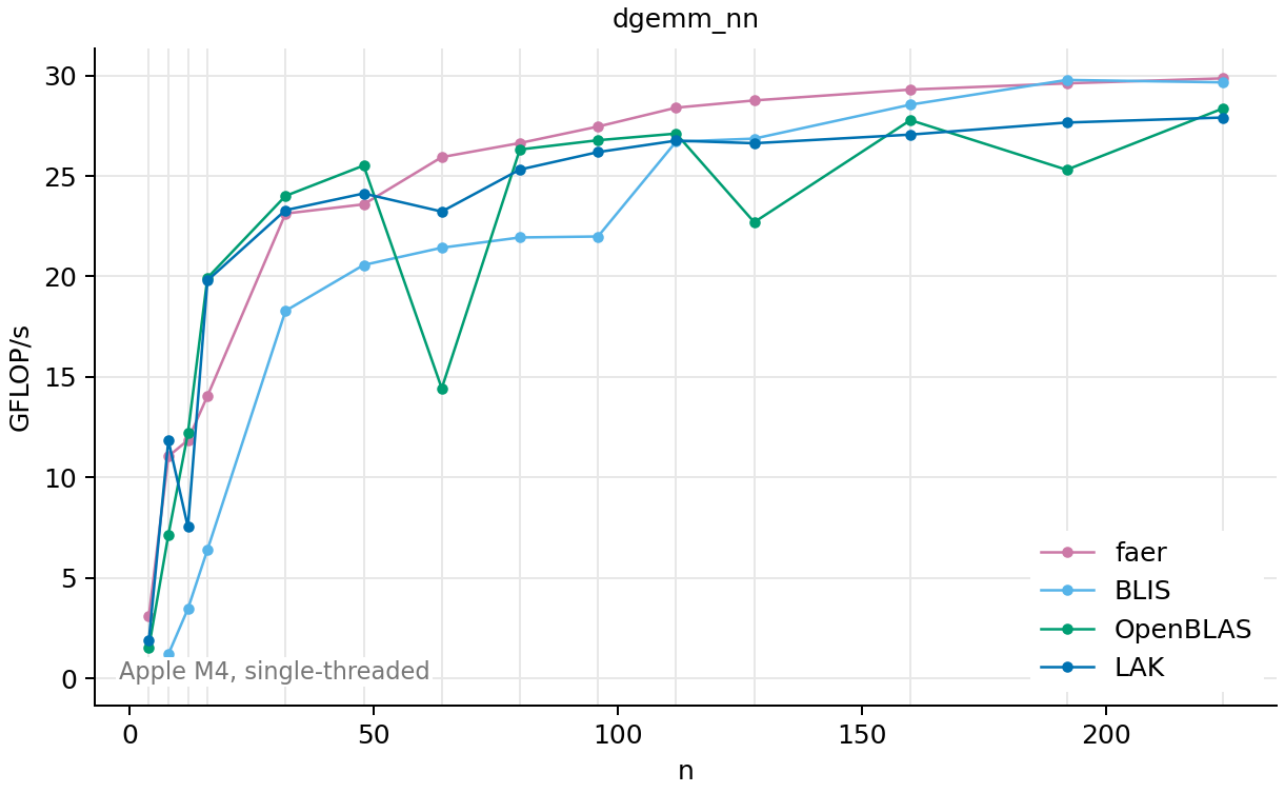


Figure 2: Double-precision GEMM benchmark for small $m$ matrices.

## Complete Benchmark

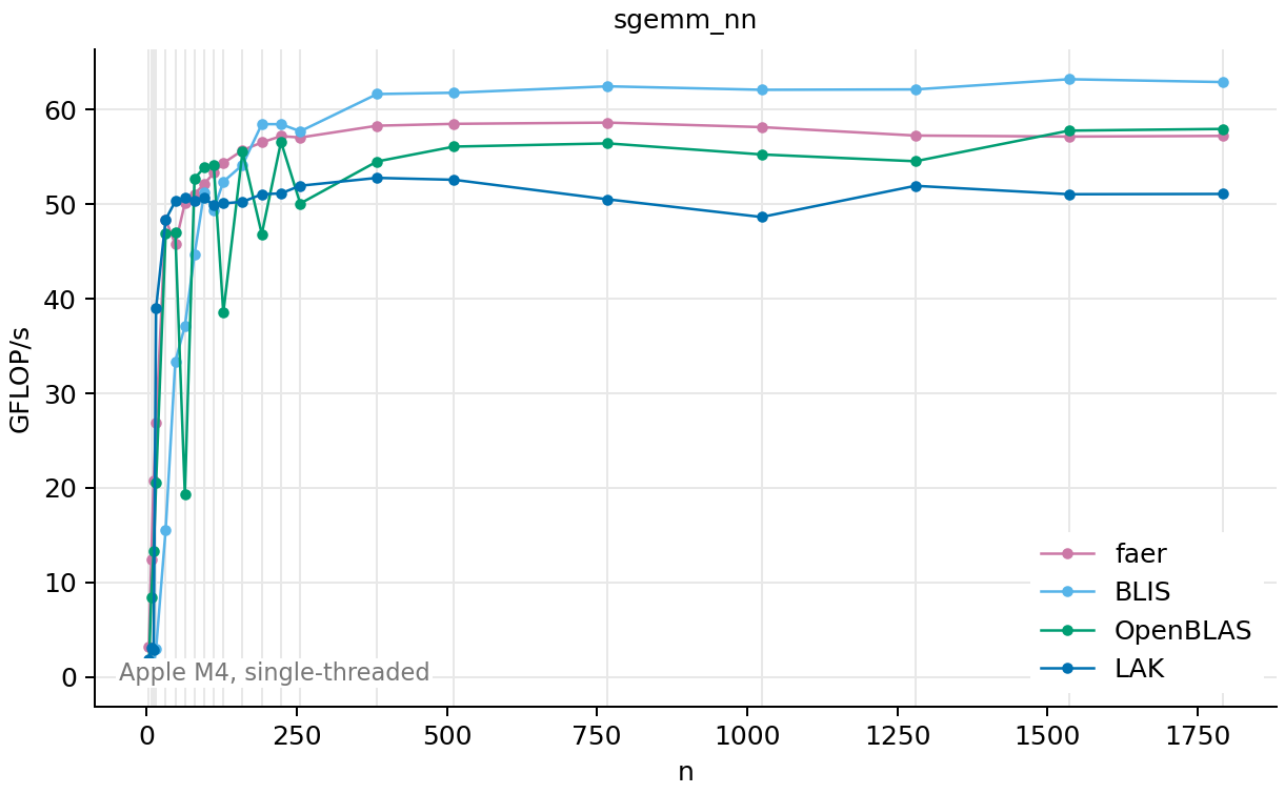


Figure 3: Single-precision GEMM benchmark.

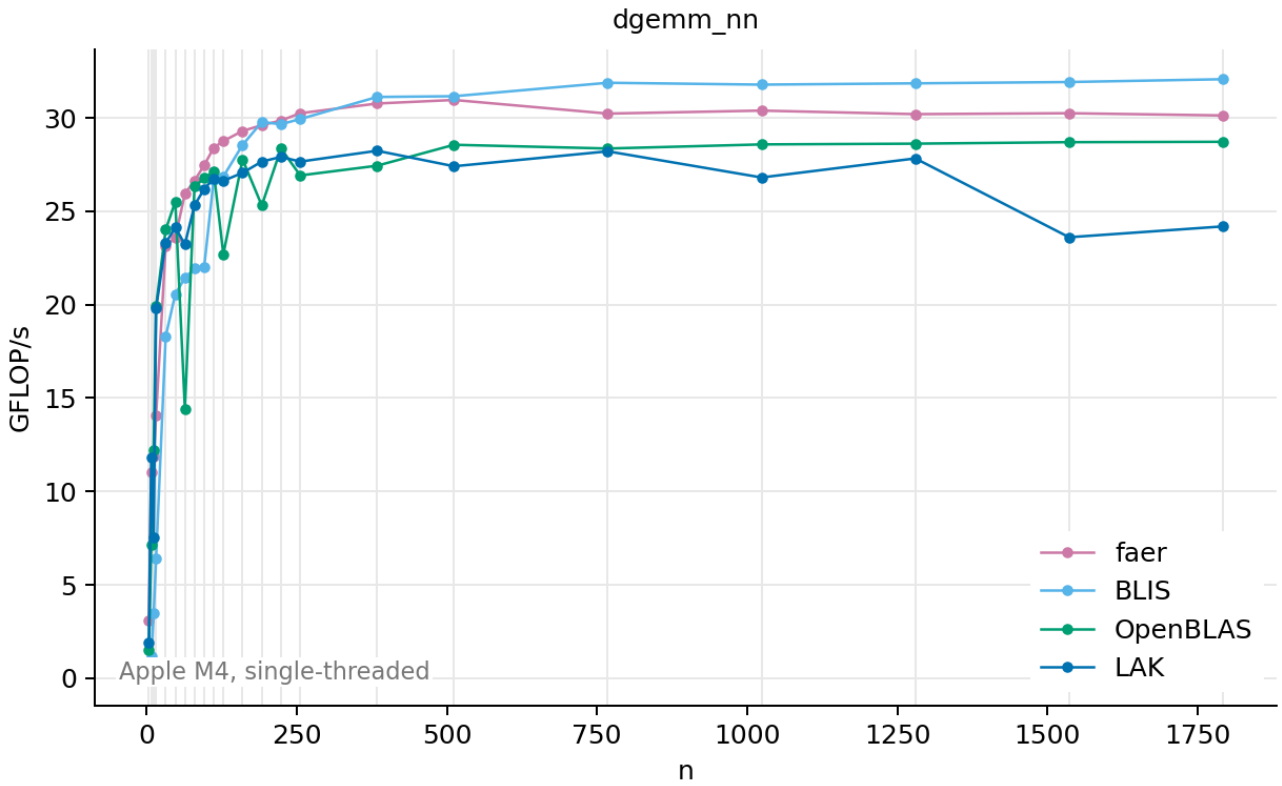


Figure 4: Double-precision GEMM benchmark.