SIMD-Optimized Numerical Library Project Plan

0) Project Quick Pitch

Goal: A small, clean C++23 or Rust library that delivers measurable 2–10× CPU speedups on core numeric kernels (matmul, dot, conv/fft optional), with clear APIs, correctness tests, and reproducible benchmarks.

Targets: x86-64 (AVX2, AVX-512 if available) and ARM64 (NEON). Falls back to scalar code when SIMD not available.

1) Repo Layout

```
simd-lib/
 ├── include/ or src/      # public headers (C++) / lib modules (Rust)
 ├── simd/                 # arch-specific intrinsics & dispatch
 │   ├── x86/
 │   └── arm/
 ├── kernels/              # matmul, dot, conv, reductions
 ├── bench/                # micro & macro benchmarks
 ├── tests/                # unit & property tests
 ├── data/                 # tiny fixtures for tests
 ├── cmake.toml/cargo.toml # build config
 └── docs/                 # design notes, results, plots
```

2) Initial Scope (MVP)

Level-1 kernels (easiest wins)
 - dot, axpy (y := a*x + y), sum, mean, variance

Level-2 kernels
 - gemv (matrix–vector), row/col reductions, sliding-window ops

Level-3 kernels (headline demo)
 - gemm (matrix–matrix, blocked + vectorized)

Stretch: 1D conv, FFT (radix-2), or softmax.

3) Public API (C++ or Rust)

C++ Example:
void dot_f32(const float* x, const float* y, size_t n, float* out);
void gemm_f32_dispatch(const MatrixView& A, const MatrixView& B, MatrixView C);

Rust Example:
pub fn dot_f32(x: &[f32], y: &[f32]) -> f32;
pub fn gemm_f32_dispatch(a: &Matrix, b: &Matrix) -> Matrix;

4) Architecture + Dispatch

- Compile-time feature sets: SCALAR, SSE2, AVX2, AVX512F, NEON.
- Runtime CPU detection: cpuid/getauxval → function pointer table.
- Alignment: 32/64-byte alignment; safe tail handling.
- Layout: row-major with stride; blocked GEMM.

5) Kernel Design Notes

Dot/Reductions: unroll 4–8 lanes; accumulate in multiple registers; horizontal add at end.

GEMM: blocking, packing, micro-kernel (8x8 AVX2), prefetching, edge handling.

Numerics: FMA where available; optional Kahan summation.

6) Portability Strategy

Write scalar reference first, then add intrinsics (immintrin.h for x86, arm_neon.h for ARM).

7) Benchmarking Plan

Micro-benchmarks: dot, axpy, reduce for n = 1e3 … 1e8.
Macro-benchmarks: GEMM for 256x256x256, 1024x1024x1024, tall/skinny matrices.
Metrics: wall-time, GB/s, GFLOP/s, cache misses (perf/VTune).
Expected speedups: dot/axpy 3–8×, GEMM 5–10×.

8) Correctness & Tests

Unit tests vs scalar within tolerance. Randomized property tests. Sanitizer runs.

9) Developer Tooling

Profiling: perf, VTune. CI: scalar tests + SIMD gating. Docs with charts & CPU info.

10) Milestone Plan (4 Weeks)

Week 1: Scalar dot/axpy; AVX2 dot; benchmark baseline.
Week 2: SIMD axpy/reduce; runtime dispatch; tolerance tests.
Week 3: Scalar blocked GEMM; AVX2 micro-kernel; first benchmarks.
Week 4: Edge handling; docs & charts; optional stretch kernel.

11) Stretch Goals

FFT, 1D conv, multithreading, aligned memory pool.

12) Show Your Work Checklist

Flamegraphs, benchmark plots, blog-style write-up, reproducible bench scripts.

13) Starter Tasks

Set up project; implement scalar dot_f32; add AVX2 dot; runtime dispatch; benchmark speedup.