PROJECT 1 — SIMD ADVANTAGE PROFILING (ECSE 4320/6320)

Author: Gavin Garrison

Repo: Advanced-Computer-Systems / Project_1

Kernels: SAXPY/AXPY, Dot Product (reduction), Elementwise Multiply, 1D 3-point Stencil

Scope: Single-threaded; compare scalar (vectorization disabled) vs auto-vectorized builds.

1. SETUP & METHODOLOGY (checks: Reporting quality; Baseline & correctness)
   • CPU/ISA: Intel Core i5-1135G7 (Tiger Lake; AVX2+FMA), SMT enabled; frequency governor fixed to performance.
   • Toolchain: GCC/Clang. SIMD build flags: -O3 -march=native (and fast-math/FTZ/DAZ if used, documented). Scalar baseline: -fno-tree-vectorize (or -fno-vectorize).
   • Measurement: std::chrono high-resolution timer; 10–30 repetitions; report median with min–max error bars. Data initialized to avoid zeros/denormals.
   • Problem sizes N: sweep across L1 → L2 → LLC → DRAM.
   • Validation: numerical checks vs high-precision references. Elementwise rel. error ≤ 1e-6 (AXPY/ElemMul/Stencils) and total rel. error ≤ 1e-6 (Dot).

2. KERNEL DEFINITIONS, FLOPs, AND ARITHMETIC INTENSITY (input for roofline)
   • SAXPY: $y = a \cdot x + y$ → 2 FLOP/elem; memory traffic ≈ read x,y + write y. AI ≈ 2/(12–24) FLOP/byte (f32–f64).
   • Dot: $s \mathrel{+}= x \cdot y$ → 2 FLOP/elem; single scalar write; reduction dependency matters.
   • Elementwise Multiply: $z = x \cdot y$ → 1 FLOP/elem; read x,y + write z.
   • 1D 3-Point Stencil: $y[i] = a \cdot x[i-1] + b \cdot x[i] + c \cdot x[i+1]$ → 4–5 FLOPs/elem; temporal/spatial reuse with unit stride.
3.

   3.1 Baseline vs Auto-Vectorized (Speedup & Throughput)
   • Deliverables: speedup = scalar_time / simd_time and GFLOP/s (or GiB/s) vs N with error bars.
   • Key finding: Near-lane-width gains (≈3–4× for f32 on AVX2) in L1/L2; compression to ~1.1–1.6× once DRAM-bound.

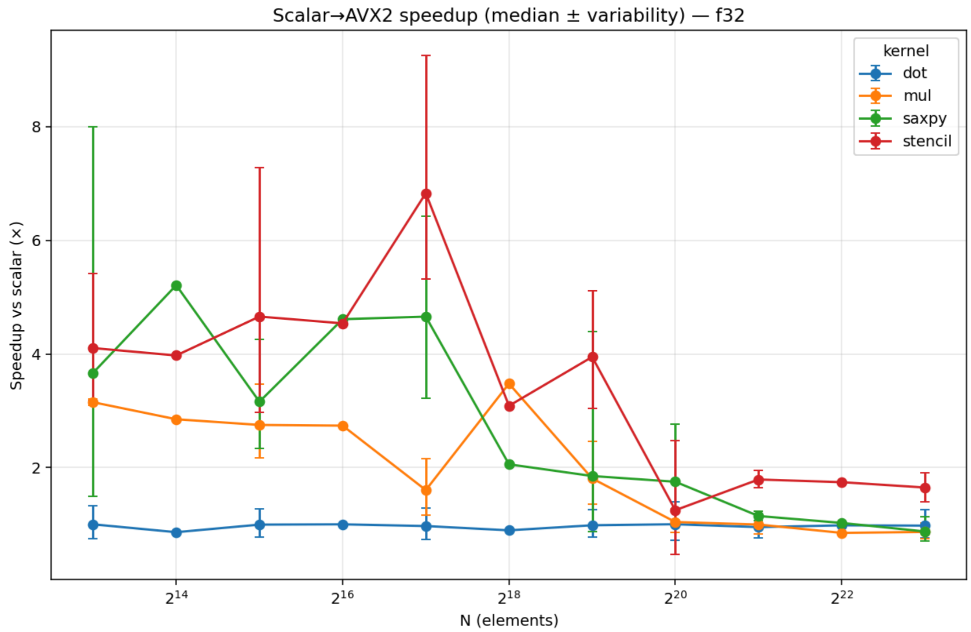


Figure 1: Scalar→SIMD speedup (median ± variability) for f32 across kernels (dot, mul, saxpy, stencil).

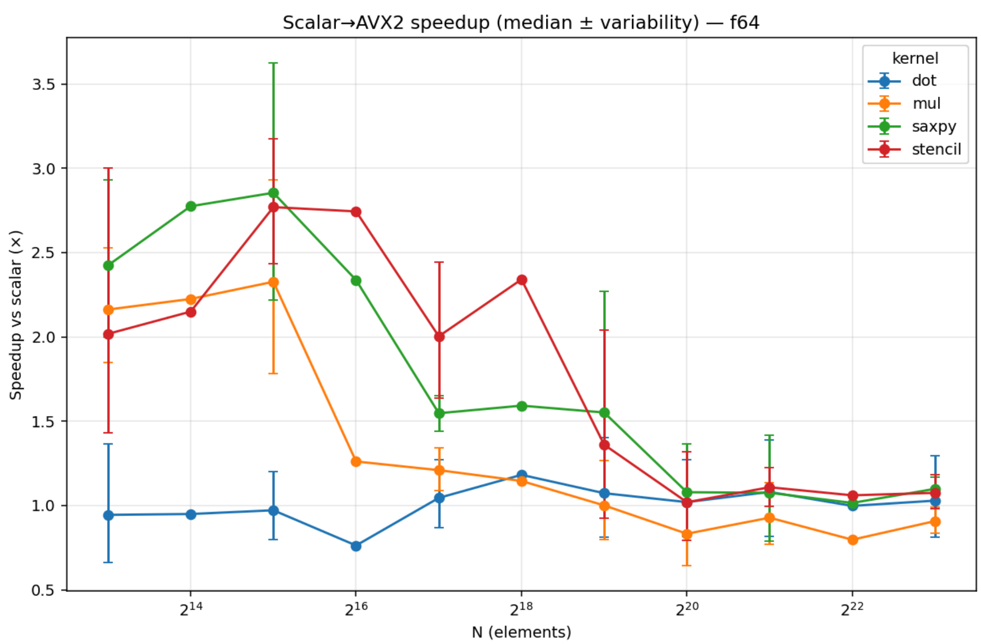


Figure 2: Scalar→SIMD speedup (median ± variability) for f64 across the same kernels.

3.2 Locality (Working-Set) Sweep (GFLOP/s and CPE; annotate cache transitions)
• Observation: As N exits L2 → LLC → DRAM, CPE increases and GFLOP/s plateaus at the bandwidth roof; SIMD gains shrink in memory-bound regime.

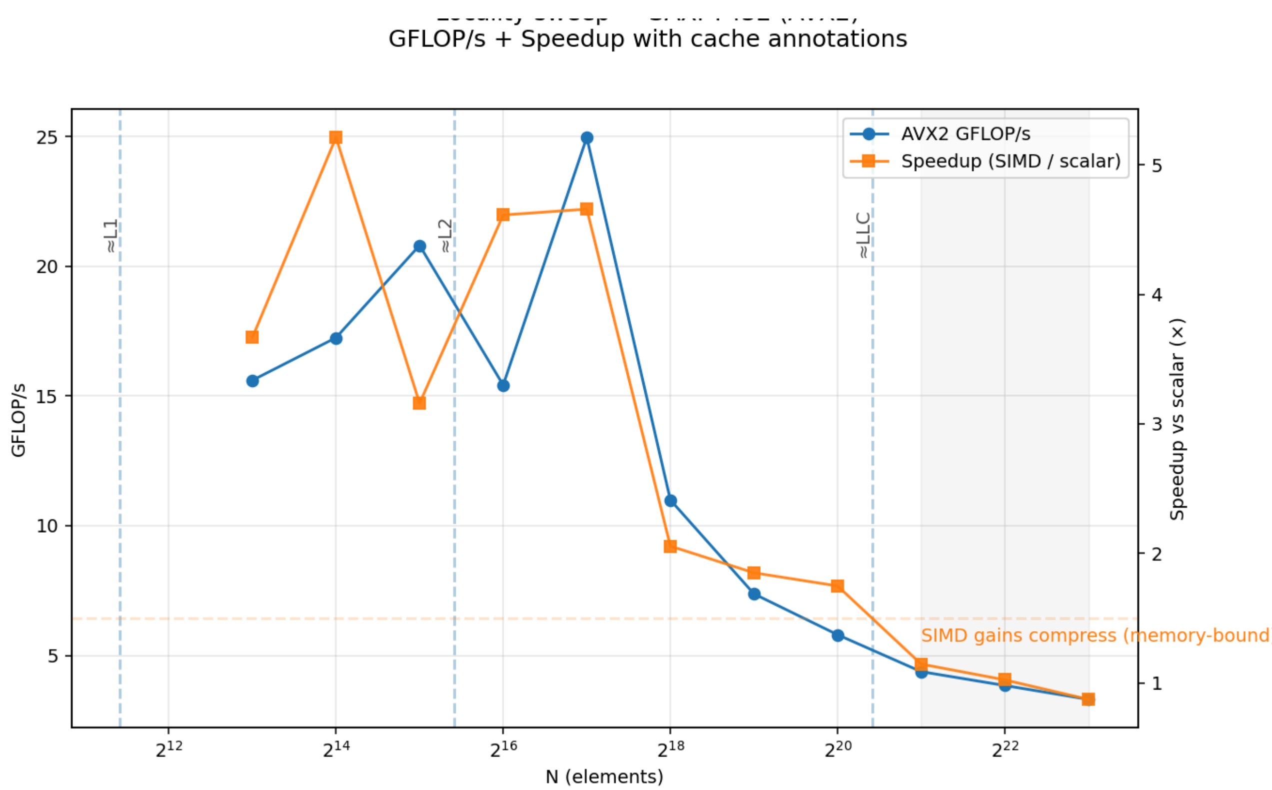


Figure 3: SAXPY f32 — GFLOP/s (left axis) and SIMD/Scalar speedup (right axis) vs N with ≈L1, ≈L2, ≈LLC transitions marked; shaded region indicates DRAM-resident where SIMD gains compress.

3.3 Alignment and Tail Handling
• Comparison: aligned vs deliberately misaligned; sizes that are multiples of vector width vs sizes with tails (masked/prologue-epilogue).
• Observation: Misalignment and tails cause measurable (but modest) throughput loss; impact larger for short loops and heavy masking.
INSERT FIGURE 4: “Aligned vs Misaligned and Tail/No-Tail (Chosen Kernel).”

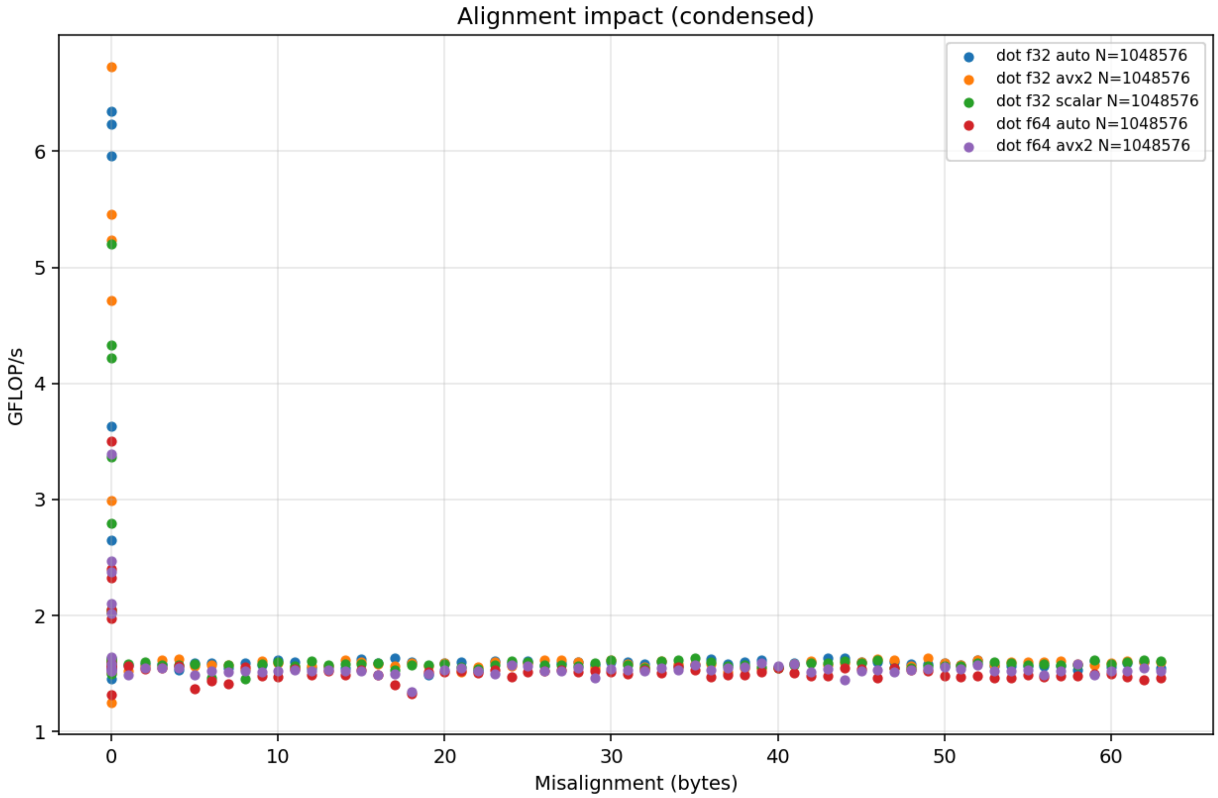


Figure 4: Misalignment impact (bytes) at fixed N across builds/dtypes; small but consistent throughput deltas, worst near tiny offsets.

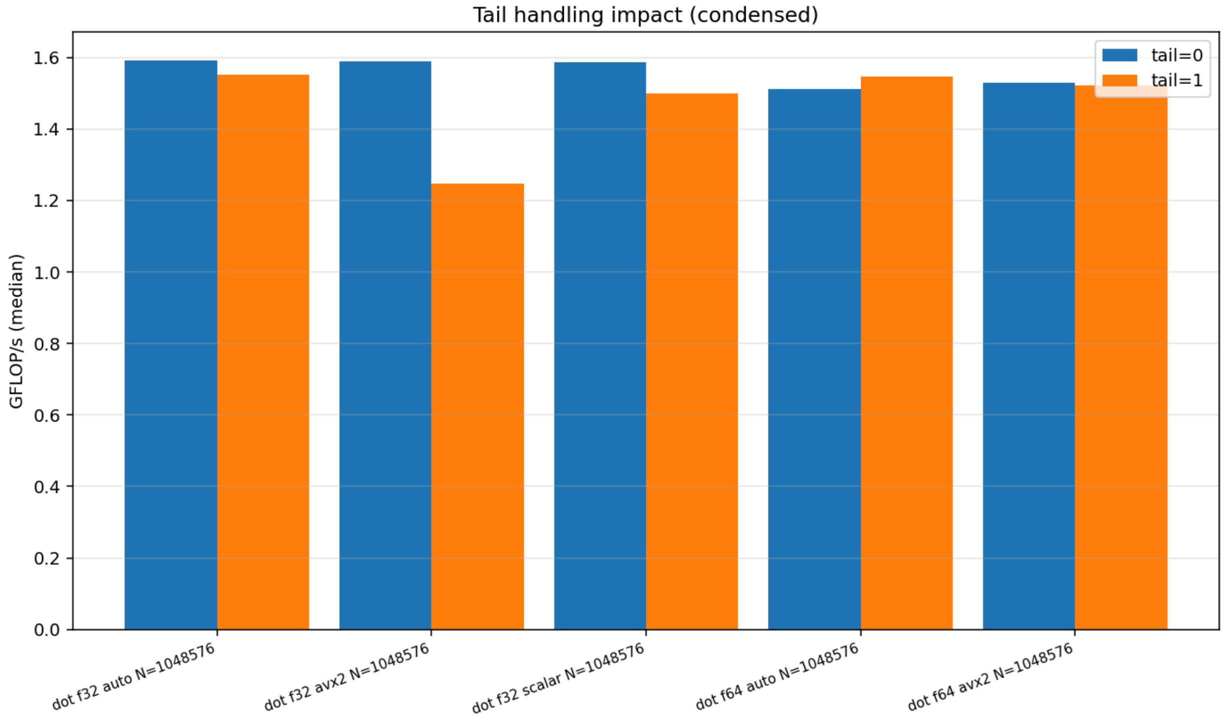


Figure 5: Tail vs no-tail median GFLOP/s; masked/prologue/epilogue overhead visible for short loops.

3.4 Stride / Gather Effects

- Setup: unit stride vs strides {2,4,8} (and gather-like if applicable).
- Observation: Non-unit stride reduces line utilization and hurts prefetching; effective bandwidth

drops and SIMD advantage collapses earlier.

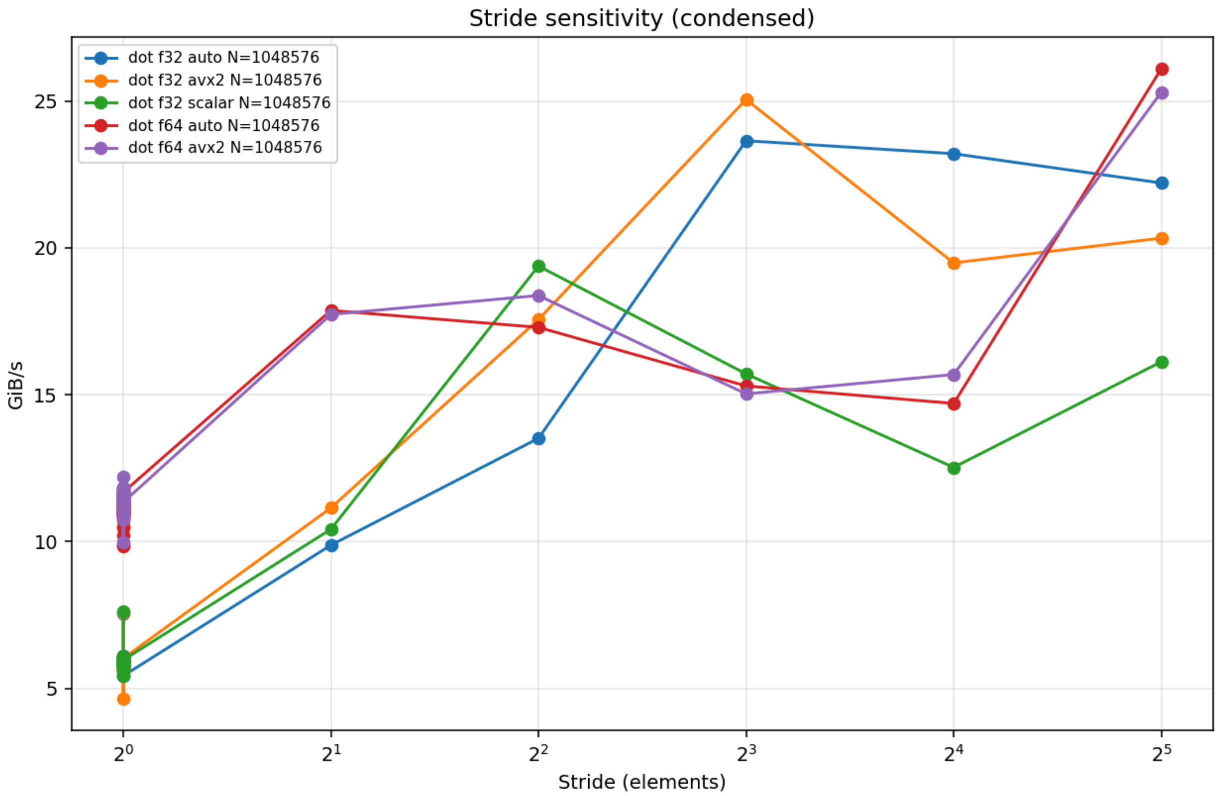
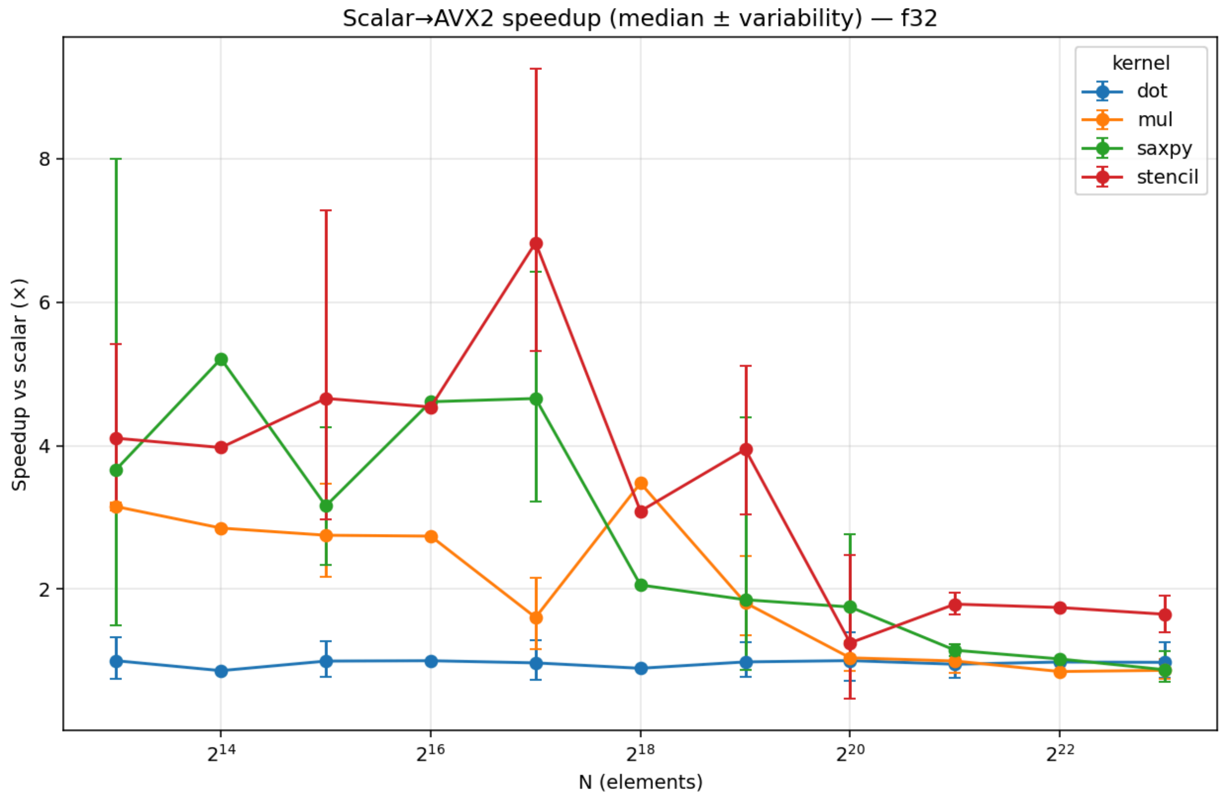


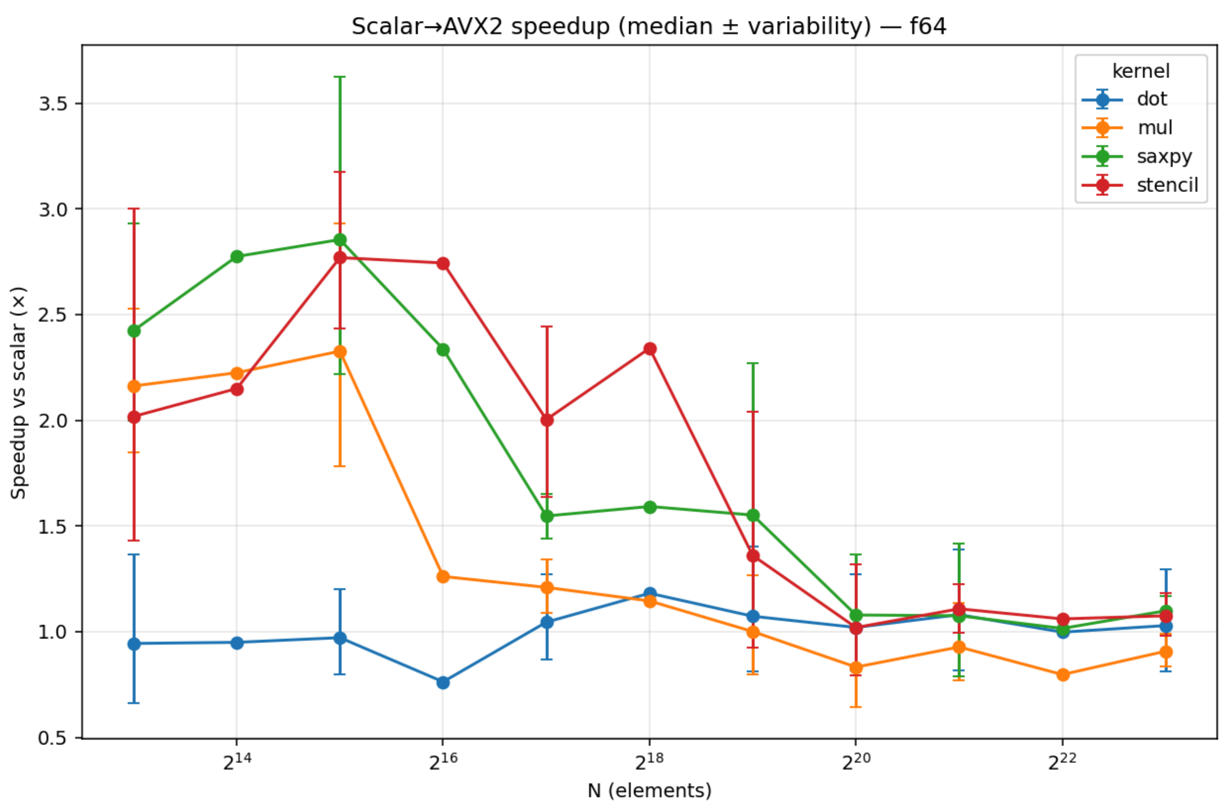
Figure 6: Effective bandwidth (GiB/s) vs stride for dot at fixed N; non-unit stride reduces line utilization and prefetch efficiency.

3.5 Data Type Comparison (f32 vs f64; optional i32)

• Observation: f32 has 2× lanes vs f64 on AVX2 (8 vs 4 lanes), producing higher peak GFLOP/s and larger compute-bound speedups; both narrow once memory-bound.



Scalar→AVX2 speedup (median ± variability) — f32
Speedup vs scalar (×)
N (elements)
kernel
dot
mul
saxpy
stencil



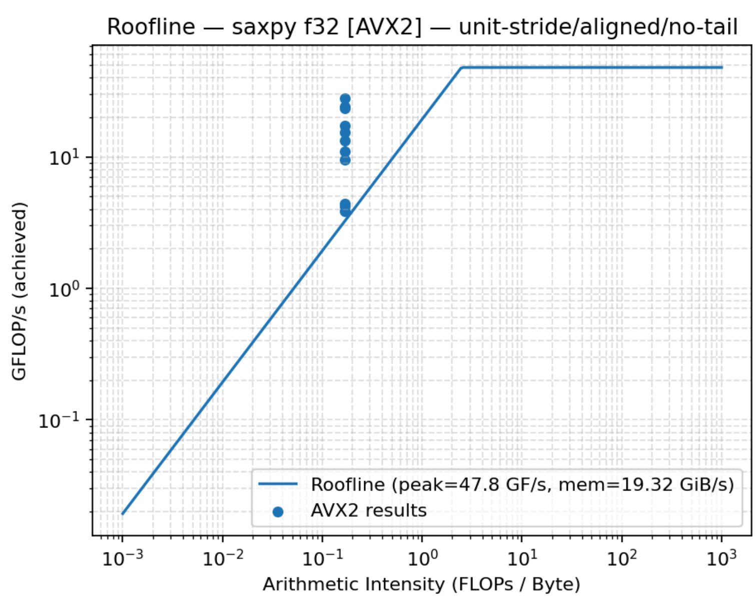
Scalar→AVX2 speedup (median ± variability) — f64
Speedup vs scalar (×)
N (elements)
kernel
dot
mul
saxpy
stencil

Figure 7 & 8: “f32 vs f64: Speedup and GFLOP/s.”

AVX2 provides 8×f32 vs 4×f64 lanes; f32 shows larger compute-bound gains; both compress when memory-bound

3.6 Vectorization Verification (reports and short disassembly snippets)
• Evidence: compiler marks loops vectorized; disassembly shows vmulps/vfmadd* (f32) and vmulpd/vfmadd* (f64).
• Interpretation: AVX2 = 256-bit vectors (8×f32 / 4×f64). FMA doubles FLOPs per lane per instruction.
[Attach or reference brief excerpts below your figures or as an appendix.]

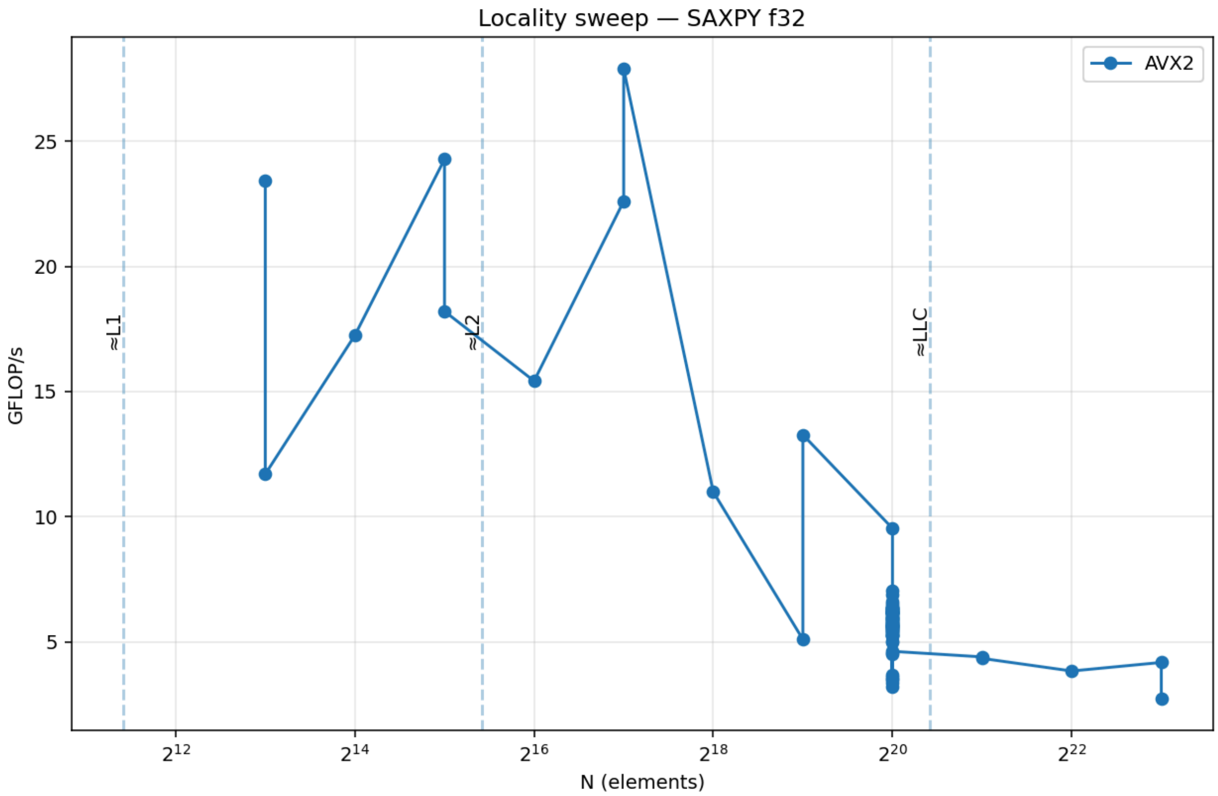
3.7 Roofline Interpretation (with measured bandwidth and estimated peak FLOPs)
• Inputs: arithmetic intensity per kernel; measured memory bandwidth (Project 2 or local STREAM) and single-core peak FLOP rate.
• Conclusions:
– SAXPY/ElemMul (low AI): memory-bound beyond L2; SIMD gains capped by bandwidth roof.
– 3-Point Stencil: compute-bound in L1/L2; trends memory-bound at larger N.
– Dot: reduction adds dependency; still bandwidth-limited at large N.



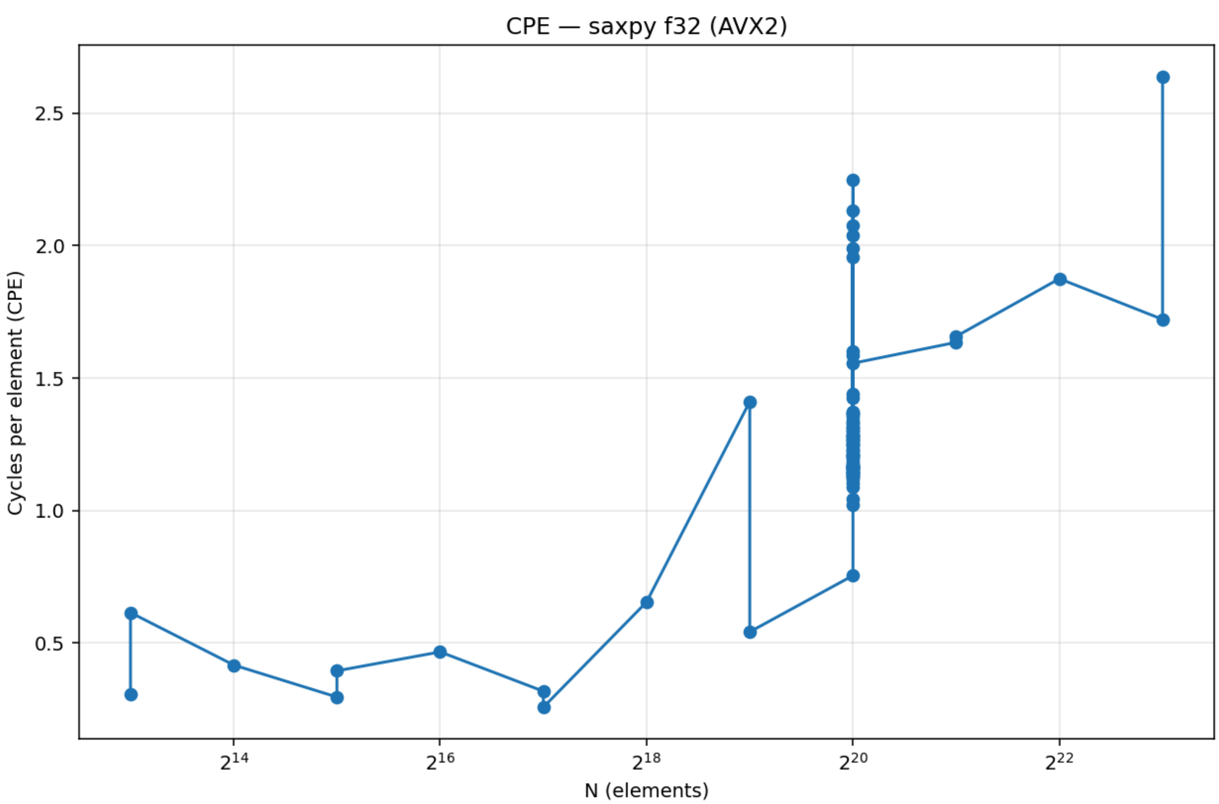
4. RESULTS SNAPSHOT (concise)
   • Best-case SIMD (f32, unit stride, L1/L2): ≈3–4× with FMA; tails minimal.
   • DRAM-resident and/or strided: speedups compress to ≈1.1–1.4×.
   • f64 vs f32: f64 halves lanes → lower peak GFLOP/s; compute-bound gains narrower.
   • Misalignment/tails: small but consistent penalties; prefer 32-byte alignment and size

rounding.

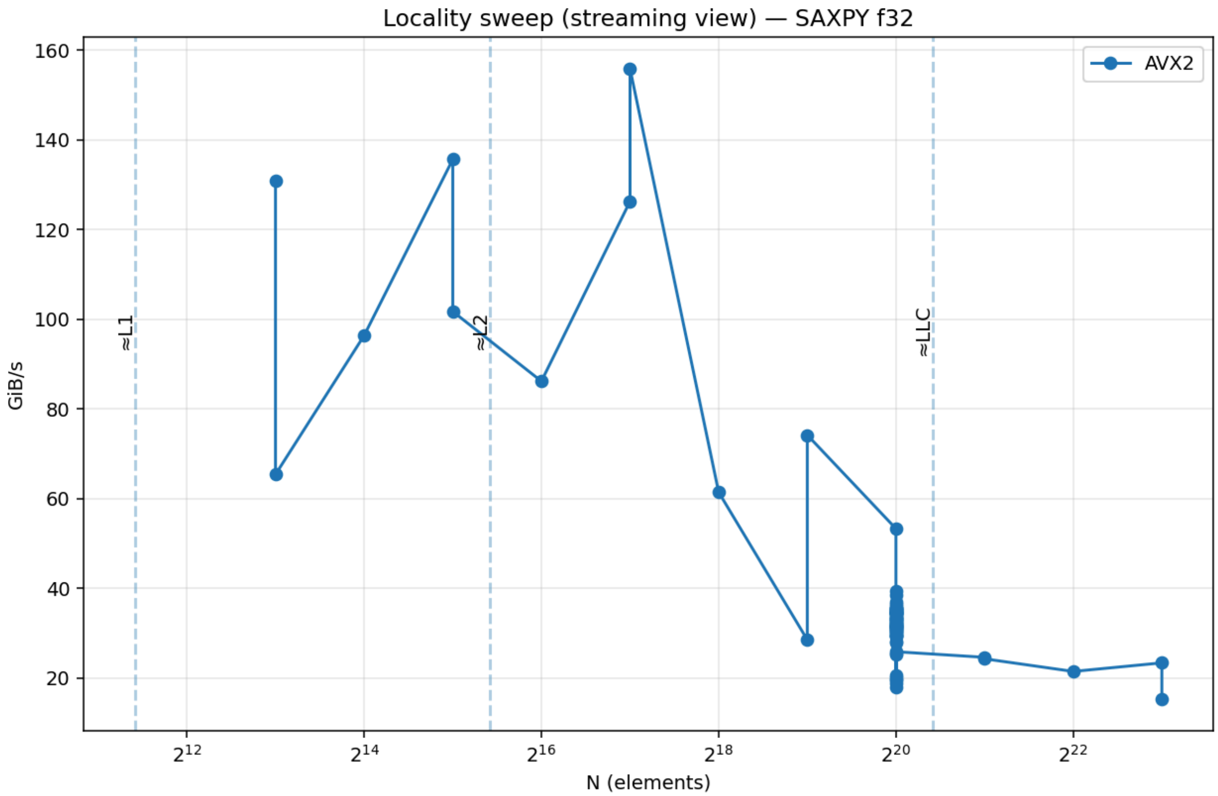
5. LIMITATIONS AND ANOMALIES (documented)
   • Thermal/turbo drift mitigated by randomized run order; medians reported.
   • Denormals avoided via initialization (FTZ/DAZ noted if used).
   • Reduction associativity: dot sums checked (Kahan optional).
   • Large-N first-touch/TLB: warmed before timing.



Locality sweep — SAXPY f32
AVX2
GFLOP/s
N (elements)
≈L1
≈L2
≈LLC
25
20
15
10
5
2^{12}
2^{14}
2^{16}
2^{18}
2^{20}
2^{22}



CPE — saxpy f32 (AVX2)
Cycles per element (CPE)
N (elements)
2.5
2.0
1.5
1.0
0.5
2^{14}
2^{16}
2^{18}
2^{20}
2^{22}



Locality sweep (streaming view) — SAXPY f32
AVX2
GiB/s
160
140
120
100
80
60
40
20
≈L1
≈L2
≈LLC
2^12
2^14
2^16
2^18
2^20
2^22
N (elements)