Project #1: SIMD Advantage Profiling

Modern CPUs provide **SIMD** (Single Instruction, Multiple Data) vector units (e.g., SSE/AVX/AVX-512, NEON) that can process multiple data elements per instruction. This class-wide project quantifies the **real-world speedup** SIMD brings to simple numeric kernels and explains **when** and **why** vectorization helps (or doesn't), using principled measurement and analysis.

Learning Goals (What your experiments must reveal)

- **Baseline vs. SIMD speedup** for common kernels under realistic conditions.
- When kernels are **compute-bound** vs **memory-bound**, and how that limits SIMD gains.
- Effects of **data type**, **alignment**, **stride/tail handling**, and **working-set size** on performance.
- How to **verify vectorization** (compiler reports / disassembly) and relate results to a **roofline** model using measured memory bandwidth.

Tools You'll Use

- A modern **C/C++ compiler** (GCC or Clang) with optimization enabled (e.g., high-level auto-vectorization). Record exact versions and flags used (e.g., optimization level, target ISA, fast-math options, FTZ/DAZ settings).
- **Disassembly or compiler vectorization reports** to confirm SIMD (e.g., opt reports; check for vector instructions in the binary).
- **System timers and statistical scripts** to measure runtime, compute **GFLOP/s** and **cycles per element (CPE)**, and generate plots.
- (Recommended) OS performance counters (e.g., perf) to report instructions retired, vector instruction count, and memory traffic.

Scope: Keep experiments **single-threaded**. The goal is to isolate SIMD effects, not multithreading.

Kernel Set (choose at least three)

1. **SAXPY / AXPY**: $y \leftarrow a\,x + y$ (streaming FMA).
2. **Dot product / reduction**: $s \leftarrow \Sigma\, x_i\, y_i$ (reduction).
3. **Elementwise multiply**: $z_i \leftarrow x_i \cdot y_i$ (no reduction).
4. **1D 3-point stencil**: $y_i \leftarrow a\,x_{i-1} + b\,x_i + c\,x_{i+1}$ (neighbor access).

Experimental Knobs (orthogonal axes)

1. **Data type:** float32 vs float64 (and optionally int32).
2. **Alignment & tail:** aligned arrays vs deliberately misaligned; sizes that are multiples of the vector width vs sizes with remainders (tail handling).
3. **Stride / access pattern:** unit-stride (contiguous) vs strided (e.g., 2, 4, 8) or gather-like index patterns where applicable.
4. **Working-set size:** within **L1**, within **L2**, within **LLC**, and **DRAM-resident** (increase N across these regimes).
5. **Compiler / ISA flags:** scalar-only (vectorization disabled), default auto-vectorized, and a build targeted to your CPU's widest available vectors. Record any fast-math, FMA, FTZ/DAZ settings.

Required Experiments & Plots

1. **Baseline (scalar) vs auto-vectorized**
   Build a scalar-only baseline and an auto-vectorized version for each selected kernel. Measure runtime across sizes spanning L1→L2→LLC→DRAM. Report **speedup = scalar_time / simd_time** and achieved **GFLOP/s**.

2. **Locality (working-set) sweep**
   For one kernel, sweep N to cross cache levels. From the same runs, produce **GFLOP/s** (or GiB/s for purely streaming) and **CPE**; annotate cache transitions. Discuss where SIMD gains compress as the kernel becomes memory-bound.
3. **Alignment & tail handling**
   Compare aligned vs misaligned inputs and sizes with/without a vector tail. Quantify the throughput gap and explain (prologue/epilogue cost, unaligned loads, masking).
4. **Stride / gather effects**
   Evaluate unit-stride vs strided/gather-like patterns (where meaningful). Show the impact on effective bandwidth and SIMD efficiency; explain prefetcher and cache-line utilization effects.
5. **Data type comparison**
   Compare float32 vs float64 (and optionally int32). Report how vector width (lanes) and arithmetic intensity affect speedup and GFLOP/s.
6. **Vectorization verification**
   Provide evidence that SIMD occurred (compiler vectorization report and/or disassembly snippets identifying vector ops). Summarize—not full dumps.
7. **Roofline interpretation**
   For at least one kernel, compute arithmetic intensity (FLOPs per byte moved) and place your achieved GFLOP/s on a **roofline** using your **measured memory bandwidth** (from Project #2, if available) and an estimate of your CPU's peak FLOP rate. Explain whether you're **compute-bound** or **memory-bound** and how this predicts the observed SIMD speedup.

Reporting & Deliverables (commit everything **to GitHub)**

- **Source code and build scripts** (scalar & SIMD builds), run scripts, raw measurements, and plotting code (re-runnable).
- **Setup/methodology:** CPU model & ISA support, compiler & flags, OS, frequency policy (governor), SMT state, measurement method (timer, repetitions), and data initialization (to avoid trivial zeros/denormals).
- **Clearly labeled plots/tables** with units and error bars (≥3 runs when feasible). Speedup plots should show median with variability.
- **Analysis** tying results to cache locality, alignment, vector width, arithmetic intensity, and roofline predictions.
- **Limitations/anomalies** with hypotheses (e.g., denormals, thermal effects, frequency scaling, page faults, TLB behavior).

Grading Rubric (Total 110 pts)

1. **Baseline & correctness (10)**
   - (5) Scalar baseline established; results validated against a reference (relative error tolerance documented).
   - (5) Reproducible timing methodology with repetitions and error bars.
2. **Vectorization verification (15)**
   - (10) Clear evidence of SIMD via compiler reports or targeted disassembly.
   - (5) Correct interpretation (e.g., vector width, FMA usage).
3. **Locality sweep (15)**
   - (10) L1→L2→LLC→DRAM transitions identified and annotated.
   - (5) Discussion of where SIMD gains compress due to memory-bound behavior.
4. **Alignment & tail study (10)**
   - (10) Quantified impact of misalignment and tail handling with explanation.
5. **Stride / gather effects (10)**

    - (10) Impact of non-unit stride or gather-like access on SIMD efficiency analyzed.
6. **Data type comparison (10)**
    - (10) float32 vs float64 (and/or int32) results with lane-width reasoning.
7. **Speedup & throughput plots (10)**
    - (10) Clear scalar vs SIMD plots (speedup, GFLOP/s or GiB/s) with proper units/legends.
8. **Roofline analysis (20)**
    - (10) Correct arithmetic intensity and placement on a roofline using measured bandwidth and peak FLOPs.
    - (10) Sound conclusions on compute- vs memory-bound and expected SIMD gains.
9. **Reporting quality (10)**
    - (10) Clean repo structure, thorough methodology, labeled plots, and clear discussion of anomalies/limits.

Tips for Successful Execution

- **Fix CPU frequency** (performance governor) and **pin to a core** to reduce run-to-run variance; document SMT state.
- **Warm up** data to populate caches for "hot" runs; also test **cold** runs where relevant and state which you report.
- **Avoid denormals** and constant-zero paths; initialize with non-trivial data. Consider FTZ/DAZ if supported (and document).
- **Check vector width** available on your machine (e.g., 128-/256-/512-bit) and target that ISA with your compiler flags.
- **Verify alignment** of arrays; test deliberately misaligned variants to see the cost.
- **Measure more than time**: compute GFLOP/s, CPE, and memory traffic estimates; use multiple repetitions and report variability.
- **Keep it single-threaded** unless you explicitly evaluate threading; otherwise results conflate SIMD with parallelism.
- **Record everything**: compiler versions/flags, environment, thermal state; randomize run order to mitigate drift.

Project #2: Cache & Memory Performance Profiling

Modern CPUs have a deep memory hierarchy (L1/L2/L3 caches + DRAM). Each level differs by orders of magnitude in latency and bandwidth. Like storage, memory shows a throughput–latency trade-off: as concurrency (outstanding memory requests) rises, bandwidth approaches a limit while average latency increases due to queuing.

Learning Goals (What your experiments must reveal)

- **Zero-queue latency** for **L1, L2, L3, and DRAM** (read & write where meaningful).
- **Maximum DRAM bandwidth** under different **access granularities/strides** (≈64B, ≈256B, ≈1024B) and **read/write mixes** (100%R, 100%W, 70/30, 50/50).
- The **throughput–latency trade-off** in DRAM as **access intensity** grows (MLC *loaded-latency* sweep).
- **Impact of cache-miss ratio** on the speed of a lightweight kernel (e.g., multiply or SAXPY).
- **Impact of TLB-miss ratio** on the same kernel's speed.

Tools You'll Use

- **Intel Memory Latency Checker (MLC)** for latency, bandwidth, and *loaded-latency* sweeps.
- **Linux perf** for cache/TLB miss measurement and correlation to performance.

Experimental Knobs

1. **Access pattern / granularity**: sequential vs. random; strides ≈64B / ≈256B / ≈1024B.
2. **Read/write ratio**: 100%R, 100%W, 70/30, 50/50.
3. **Access intensity (concurrency)**: multiple outstanding requests or threads; MLC's *loaded-latency* mode.

Required Experiments & Plots

1. **Zero-queue baselines**: Measure per-level latencies; table results.
2. **Pattern & granularity sweep**: Matrix covering pattern × stride; plot latency & bandwidth.
3. **Read/Write mix sweep**: Four ratios; discuss hardware effects.
4. **Intensity sweep**: ≥3 intensities; plot throughput vs latency; mark and explain "knee" using Little's Law.
5. **Working-set size sweep**: Show locality transitions; annotate regions.
6. **Cache-miss impact**: Use a light kernel; vary miss rate; correlate with performance.
7. **TLB-miss impact**: Vary page locality; test huge pages; correlate.

Reporting & Deliverables (post everything on Github)

- Scripts/commands, raw data, plotting code.
- System configuration and methodology.
- Clearly labeled plots/tables with units and error bars.
- Analysis tied to theory and counter data.
- Discussion of anomalies/limitations.

Grading Rubric (Total 230 pts)

1. **Zero-queue baselines (30 pts)**
    - (10 pts) Correct methods for isolating single-access latency.
    - (10 pts) Accurate per-level latency values with correct units.
    - (10 pts) Clear table format with CPU frequency conversions.
2. **Pattern & granularity sweep (40 pts)**
    - (15 pts) Complete coverage of required patterns/strides.
    - (15 pts) Plots showing both latency & bandwidth from same runs.
    - (10 pts) Insightful discussion of results, including prefetch and stride effects.

3. **Read/Write mix sweep (30 pts)**
    - (10 pts) Correct implementation of four R/W ratios.
    - (10 pts) Coherent explanation of observed performance differences.
    - (10 pts) Properly labeled and formatted plots.
4. **Intensity sweep (60 pts)**
    - (20 pts) ≥3 distinct intensity levels; single throughput–latency curve.
    - (15 pts) Clear identification and justification of "knee."
    - (15 pts) % of theoretical peak bandwidth with discussion of diminishing returns.
    - (10 pts) Logical tie-in to Little's Law.
5. **Working-set size sweep (20 pts)**
    - (10 pts) Annotated plots showing L1/L2/L3/DRAM transitions.
    - (10 pts) Correctly matched transition points to measured latencies.
6. **Cache-miss impact (25 pts)**
    - (10 pts) Proper control of miss rate through footprint/pattern changes.
    - (10 pts) Accurate correlation between perf counters and runtime.
    - (5 pts) Correct application of AMAT model to explain results.
7. **TLB-miss impact (25 pts)**
    - (10 pts) Methodologically sound variation of page locality and huge pages.
    - (10 pts) Accurate TLB miss measurement and correlation.
    - (5 pts) Discussion of DTLB reach and its effect on performance.

Tips for Successful Execution

- **Pin and isolate cores** using taskset or numactl to avoid interference from other processes.
- **NUMA awareness**: Keep threads and memory on the same NUMA node unless measuring remote access.
- **Disable frequency scaling**: Set CPU governor to performance for consistent results.
- **Warm-up runs**: Perform initial iterations to stabilize caches and clock frequency.
- **Randomize experiment order**: Helps avoid systematic bias from thermal or frequency drift.
- **Use error bars**: Repeat each test ≥3 times and report mean ± standard deviation.
- **Document everything**: SMT state, prefetcher configuration, and exact tool versions.
- **Check tool flags**: MLC and perf event names can vary by version—record what you used.
- **Be mindful of prefetchers**: Large strides or random patterns reduce their effectiveness.
- **Think about bottlenecks**: Past the knee, latency rises sharply with little throughput gain.

Project #3: SSD Performance Profiling

Modern SSDs (especially NVMe) can deliver massive IOPS and GB/s when requests are issued concurrently. Like memory, storage exhibits a classic throughput–latency trade-off governed by queuing theory: increasing queue depth improves utilization and throughput up to a saturation **knee**, after which **latency** rises sharply with little additional throughput.

⚠ **Data-loss warning**

Never benchmark against a partition that contains valuable data. Create a dedicated, empty partition or use a raw device reserved for tests. If you must test via a file, ensure direct I/O (to bypass the page cache) and use a large file on an otherwise empty filesystem. You are responsible for preventing data loss.

Learning Goals (What your experiments must reveal)

- **Zero-queue (QD=1) latency** for reads and writes under both **random 4 KiB** and **sequential 128 KiB** patterns.
- **Maximum small-IOPS** (4 KiB) and **maximum large-block throughput** (≥128 KiB, reported in MB/s or GB/s).
- The **throughput–latency trade-off** as **queue depth / parallelism** increases, and identification of the **knee**.
- **Block-size and access-pattern effects** (sequential vs. random) on latency, IOPS, and bandwidth.
- **Read/Write mix effects** (read-only, write-only, 70/30, 50/50) at fixed conditions.
- **Working-set / LBA-range effects** and the difference between **burst** and **steady-state** behavior (e.g., SLC cache exhaustion, thermal throttling).
- **Tail latency** characterization (p95/p99) and its relationship to queueing and device state.

Tools You'll Use

- **FIO (Flexible I/O Tester)** for generating controlled storage workloads (reads/writes, block sizes, queue depth, parallel jobs, random/sequential access, read/write mixes, time-based runs, percentiles).
- (Optional) **Basic observability**: OS disk stats and SSD SMART/health tools to note temperature, throttling indicators, and total writes.

Experimental Knobs (orthogonal axes)

1. **Block size**: 4 KiB, 16 KiB, 32 KiB, 64 KiB, 128 KiB, 256 KiB (and optionally 512 KiB/1 MiB for sequential).
2. **Access pattern**: sequential vs. random; 4 KiB alignment for random; contiguous LBA for sequential.
3. **Read/Write ratio**: 100%R, 100%W, 70/30, 50/50 (hold other knobs fixed when sweeping mix).
4. **Queue depth & parallelism**: vary **iodepth** and/or **numjobs/threads** (e.g., QD 1→2→4→8→16→32→64→128→256 when device supports it).
5. **Data pattern**: incompressible vs. compressible payload (impacts some consumer SSDs); keep consistent and document.

Required Experiments & Plots

1. **Zero-queue baselines**
   Measure QD=1 **latency** for: (a) 4 KiB random reads & writes, (b) 128 KiB sequential reads & writes. Report average and percentiles (p95/p99). Provide a clearly labeled table.
2. **Block-size sweep (pattern fixed)**
   Hold pattern fixed (do both **random** and **sequential** in separate runs). Sweep 4 KiB→256 KiB (and optionally 512 KiB/1 MiB). From the **same runs**, produce **IOPS**/**MB/s** and **average latency** (two panels or dual axis). Mark where reporting naturally shifts from IOPS (≤64 KiB) to MB/s (≥128 KiB).
3. **Read/Write mix sweep (knobs fixed)**
   With block size and pattern fixed (e.g., 4 KiB random), run **100%R, 100%W, 70/30, 50/50**. Plot throughput and latency from the same runs and discuss differences.

4. **Queue-depth/parallelism sweep (trade-off curve)**
   Using 4 KiB random (and optionally 128 KiB sequential), increase queue depth/numjobs across ≥5 points. Produce a single **throughput vs. latency** trade-off curve. **Identify the knee** and relate it to Little's Law (Throughput ≈ Concurrency / Latency).
5. **Tail-latency characterization**
   For at least one workload (e.g., 4 KiB random read at mid-QD and near-knee QD), report **p50/p95/p99/p99.9** latency and discuss queueing impact and SLA implications.

Reporting & Deliverables (commit everything to GitHub)

- **Scripts/configs**, raw results, and plotting code (re-runnable).
- **Setup/methodology:** SSD model, interface (PCIe gen/lanes or SATA), capacity used, system CPU/OS, filesystem vs raw device, direct I/O usage.
- **Clearly labeled plots/tables** with units and error bars (≥3 runs when feasible); indicate which knobs are fixed vs varied.
- **Analysis** grounded in queuing theory and device behavior.
- **Limitations/anomalies** with hypotheses (e.g., background GC, thermal events, host-side cache interference).

Grading Rubric (Total 170 pts)

1. **Zero-queue baselines (30)**
   - (10) Correct isolation of QD=1 latency (page cache bypassed; alignment documented).
   - (10) Accurate average & percentile latencies for 4 KiB random and 128 KiB sequential (R/W).
   - (10) Clear tabular presentation with units.
2. **Block-size & pattern sweep (40)**
   - (15) Complete coverage of required sizes for both patterns using one coherent matrix.
   - (10) Plots show IOPS/MB/s **and** latency from the same runs (proper axes/legends).
   - (15) Insightful discussion of prefetching/queue coalescing, controller limits, and cross-over from IOPS- to bandwidth-dominated regimes.
3. **Read/Write mix sweep (30)**
   - (10) Correct implementation of four mixes under fixed conditions.
   - (10) Coherent explanation of differences (write buffering, WA, flushes).
   - (10) Proper labeling/units and comparison on matched axes.
4. **Queue-depth/parallelism sweep (40)**
   - (15) ≥5 QD points; single **throughput–latency** curve with error bars.
   - (10) Clear identification and justification of the **knee** via Little's Law.
   - (10) % of interface or vendor-spec peak; discussion of diminishing returns.
   - (5) Tail-latency note at or near the knee (p95/p99) and implications.
5. **Synthesis & reporting quality (30)**
   - (15) Full reproducibility (configs, versions, environment) and data hygiene.
   - (15) Thoughtful anomalies/limitations section with plausible hypotheses.

Tips for Successful Execution

- **Use direct I/O and 4 KiB alignment** to avoid the page cache and partial-block penalties. Document filesystem vs raw device.
- **Precondition for steady-state** when testing random writes (e.g., fill target range with random data first). Note any TRIM/discard.
- **Control device temperature** (consistent airflow); record temperature and watch for throttling.
- **Randomize trial order** and **repeat** runs to capture variance; report mean ± stdev and latency percentiles.

- **Keep data patterns consistent** (compressible vs incompressible) to avoid misleading results on consumer SSDs.
- **Isolate the host** (CPU governor fixed, background tasks minimized) to reduce host-side noise.
- **Note interface ceilings** (e.g., SATA ~550 MB/s; PCIe 3.0×4 ≈ 3.5 GB/s; PCIe 4.0×4 ≈ 7.5–7.8 GB/s) when interpreting limits.

Vendor reference (for discussion)

The Intel Data Center NVMe SSD **D7-P5600 (1.6 TB)** lists **≈130K 4 KiB random write IOPS**. Compare your results to this enterprise spec and explain discrepancies (e.g., SLC caching, data pattern compressibility, controller/firmware policy, interface limits, host CPU effects). Provide reasoned analysis in your report.