Ports as Instruments: CSV Load Timing in Zig and n0is C++

Tyler Garrett / 2026-11-09 / Working paper 0.2

Abstract

The same CSV input and output contract can yield different performance rankings depending on how many loads a process has already run. We compare a Zig port with the n0is C++ CSV loader on one Apple M5 Max in two sessions, with 20 and 30 paired blocks per condition. In the later session, Zig was faster in 23 of 30 first-load pairs at one million rows, and n0is C++ was faster in all 30 warmed pairs. At ten million rows, n0is C++ again won all warmed pairs, while first-load wins split 15 each and the adjusted interval spanned equal time. The second session reproduced the one-million-row crossover and the warmed advantage at both sizes. The first-load lead at ten million rows depended on the session. Descriptive splits by execution order and a leave-one-pair-out check show where the comparisons were sensitive.

1. Research question

The companion Swift study used a port to expose differences in work and timer placement. Here the question is whether the ordering of two implementations reverses between first and warmed loads when both follow the same timed output contract. A workflow may load a table once in a newly launched worker or repeatedly in a long-lived process. A load-only measurement gives a view of each case, while user-visible latency may also include launch and file acquisition.

A November 8 comparison of five builds first showed a Zig lead on the initial load and a C++ lead after repeated loads at both tested sizes. Two subsequent sessions ran only the C++ and Zig executables in pairs. They used the same input bytes and executables as the five-build comparison. The pairwise sessions ran the two observations consecutively within each block and balanced which executable ran first. The five-build schedule used different preceding workloads and a different family of intervals. We analyze each pairwise session on its own.

2. Workload and clocks

2.1 Input and output contract

The fixtures contain one million and ten million rows with the same six-column schema and occupy about 46.3 MB and 472.6 MB. Each timed load parses all six columns into a materialized text table, converts `amount` to float64, and builds one dictionary for `region`. The harness disables the C++ path's two optional dictionaries so both sides perform the same requested materialization. Parallel delimiter parsing is already well studied [2]. This comparison measures a larger unit of work that includes typed conversion and dictionary construction.

Separate preflight and postflight checks match both builds' parsed-cell fingerprints against the archived reference for each input. The reference covers table dimensions and the text of all parsed cells in row order. This can catch a change in row counts, fields, or ordering that would invalidate the comparison. Zig uses its serial path for the

fingerprint check. The check leaves derived numeric values, dictionary IDs, and the output of each timed parallel load unverified. Every timing record separately reports the expected row count and the requested participant count.

2.2 Clock boundaries

- **First load.** The first timed load in a fresh process. Both builds read the input into memory before timing. Worker setup also starts before the load clock, with build-specific readiness. The OS file cache is uncontrolled.
- **Warmed load.** The median of three timed loads after three warmup loads in the same process. A prior output remains live during the next load. The harness uses the individual load clocks to compute one median per process.

The first-load label refers to a process state. Since input acquisition precedes the clock, repeated access to the same fixture can benefit from the operating system file cache without that access being timed. In warmed mode the earlier in-process loads can also change allocator, worker, and memory state before the measured work begins. The design compares those bundled conditions. It does not identify the contribution of each state change.

The harness also records command wall time from subprocess invocation through exit and output capture. In the second session, the 10M first-load command medians were 112.692 ms for C++ and 99.663 ms for Zig, compared with internal load medians of 46.267 ms and 45.998 ms. The command clock combines launch, acquisition, runtime setup, the load, and teardown. Its gap shows how the selected measurement boundary can change the picture. It cannot assign the extra time to any one stage. The warmed commands include all six loads and therefore have a different unit from the internal per-load statistic.

3. Paired design and analysis

3.1 Collection

The November 8 pairwise session used 20 pairs per condition, for 80 pairs and 160 processes. The November 9 session used 30 pairs per condition, for 120 pairs and 240 processes. Both ran on the same Apple M5 Max with macOS 26.4.1, on battery. The harness requested 16 participants for both builds and set a common utility scheduling priority ceiling. Pool readiness, scheduling, allocation, and chunking remained implementation choices. The frozen plan recorded binary and fixture digests, which the harness rechecked after collection.

The C++ binary was built for arm64 with Apple clang 17.0.0, C++20, and `-O3`. The archived Zig build procedure uses `ReleaseFast` and `-mcpu=native`. The two sessions fixed the resulting executables so rebuilds and changes in optimization settings would be outside the comparison.

Each round visited all four conditions in a seeded randomized order. A pair ran the two executables consecutively in separate processes, with one process finishing before the next began. Each build occupied the first position 10 times per condition on November 8 and 15 times on November 9. Balancing order gives equal weight to the two execution sequences. The second process still runs on the host state left by the first. The fixed schedule required valid clocks and intact inputs. We retained all timed processes.

The harness recorded power state and background activity just before and after each pair. It flagged a process using at least half a CPU core at the boundaries of two pairs on November 8 and four on November 9. These samples describe the host at the boundaries. They do not continuously measure competing activity during the load.

3.2 Estimation

Each process contributes one value. In warmed mode that value is the median of the three timed loads, so three loads in the same process do not count as three independent observations. We divide the C++ time by the Zig time within each pair and exponentiate the mean of the log ratios [1]. A ratio above 1 favors Zig. A ratio below 1 favors n0is C++. The two arm medians provide times in milliseconds. Their quotient and the geometric mean of paired ratios summarize the data differently.

For each condition, 50,000 seeded resamples draw complete pairs with replacement and recompute the geometric mean. The percentile endpoints use 0.625% in each tail, a Bonferroni adjustment across four contrasts within a session. This targets approximate 95% simultaneous coverage if pairs are exchangeable within each condition. The prespecified directional criterion calls for both first-load intervals above 1 and both warmed intervals below 1. An interval that contains 1 leaves the direction unresolved for that session and does not establish equivalence.

4. Results

4.1 November 9 load clocks

Three of four conditions met the directional criterion. At 1M rows, the paired estimates put Zig about 15% lower on first-load time and n0is C++ about 37% lower on warmed time. At 10M rows, n0is C++ was about 23% lower on warmed time. First-load wins split evenly and the interval spanned 1.

CSV load	C++ (ms)	Zig (ms)	Paired ratio (interval)	C++ wins	Zig wins
1M, first load	5.714	5.122	1.174 (1.081-1.276)	7	23
1M, warmed	2.864	4.702	0.629 (0.606-0.661)	30	0
10M, first load	46.267	45.998	0.993 (0.952-1.016)	15	15
10M, warmed	32.337	41.617	0.775 (0.765-0.787)	30	0

Table 1. Times are medians across 30 processes per build and condition. The ratio and adjusted interval use within-pair times. Win counts describe the observed pairs. There were no ties.

At 10M the first-load interquartile ranges nearly coincide, while in warmed mode the slowest C++ observation, 35.340 ms, is still below the fastest Zig observation, 40.457 ms. The 1M warmed observations also have disjoint ranges. Figure 1 shows the corresponding spread within pairs.

CSV load	C++ middle 50% (ms)	Zig middle 50% (ms)	C++ min to max (ms)	Zig min to max (ms)
1M, first load	5.387-6.444	4.286-5.615	5.267-7.131	4.018-6.811
1M, warmed	2.816-2.919	4.570-4.790	2.756-3.948	4.095-5.004
10M, first load	45.602-46.632	45.515-46.623	44.617-49.419	44.107-65.982
10M, warmed	31.736-32.796	41.262-41.966	30.666-35.340	40.457-43.688

Table 2. The middle 50% spans the 25th through 75th percentiles. Ranges show the observed extrema.

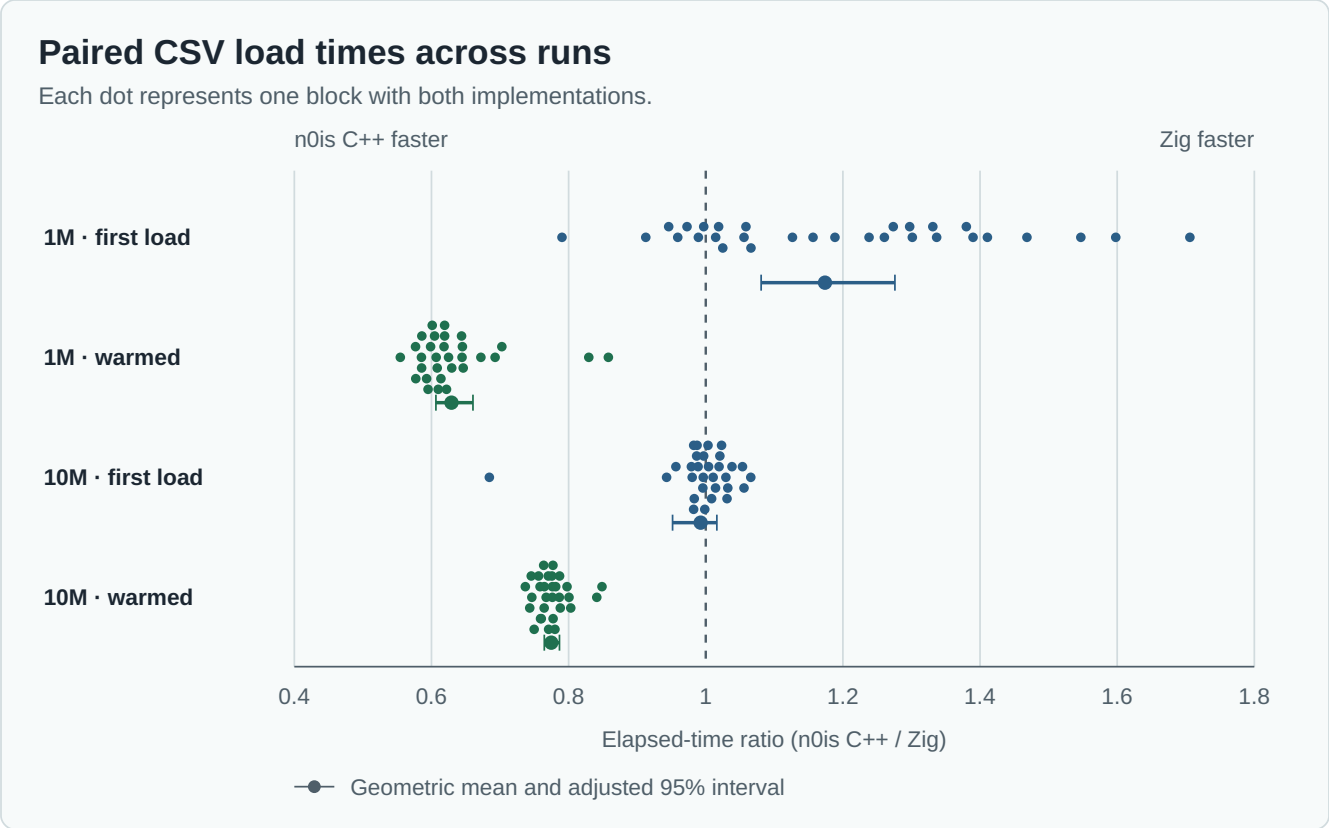


Figure 1. Each dot is one pair's C++ time divided by Zig time. Vertical offsets separate nearby observations. Whiskers show adjusted intervals around the paired geometric means. The line at 1 marks equal elapsed time.

4.2 Repeatability across sessions

The November 8 pairwise session met the directional criterion in all four conditions. Zig won all 20 pairs on the first 10M load. On November 9 the estimate for that condition moved from 1.072 to 0.993 and its adjusted interval included 1. The 1M first-load estimates, by comparison, remained near 1.18 in both sessions. The warmed estimates stayed below 1 at both sizes even as their magnitudes changed. C++ won every warmed pair in each session. These observations support three directional findings across the two sessions and leave the first 10M load sensitive to the collection.

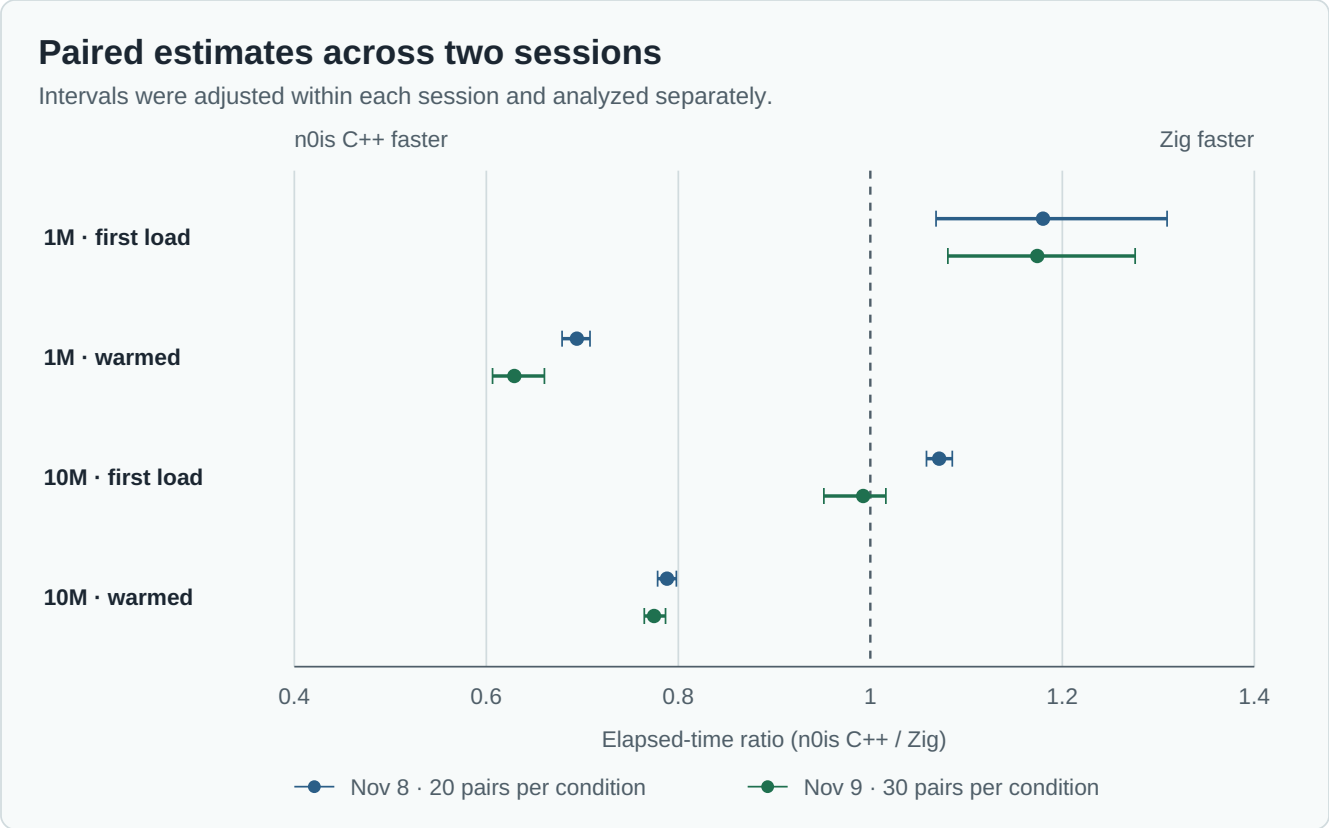


Figure 2. Paired geometric means and adjusted intervals for the November 8 and November 9 sessions. Each session uses its own four-contrast family for approximate 95% simultaneous coverage. The two sessions are analyzed separately.

4.3 Execution order and influential pairs

The 1M first-load comparison varied substantially with execution order. In both sessions Zig had a larger lead when it ran second. The estimate was close to equal time when Zig ran first. Each descriptive split uses half the pairs from its session. The primary estimate describes a schedule with equal counts of both sequences. The split does not isolate a host mechanism or establish that running second has the same effect on both programs.

Session	First process in pair	Pairs	Paired C++/Zig ratio	Zig wins
November 8	C++	10	1.384	10
November 8	Zig	10	1.006	7
November 9	C++	15	1.364	15
November 9	Zig	15	1.010	8

Table 3. Exploratory order splits for the 1M first load. The ratios use the same paired geometric mean as the primary analysis.

For the 10M first load on November 9, Zig's longest observation was 65.982 ms against 45.158 ms for C++ in the same pair. Both activity snapshots were below the logging threshold. Recomputing the geometric mean after leaving out each of the 30 pairs in turn gives estimates from 0.990 to 1.005. The upper endpoint omits that

longest Zig observation. The sign can therefore change by omitting a single pair, while all of these descriptive point estimates remain near 1. The reported interval and win counts retain all 30 pairs.

The first-load timing records also identify where to look for differences. For the 10M fixture, Zig's median reported parse phase was 38.635 ms on November 8 and 40.256 ms on November 9. The C++ medians were 37.140 ms and 36.596 ms. Zig's longest observation recorded 55.060 ms in its parse phase. These phase fields locate variation within the measured load, while separate phase medians cannot be summed to recover a median total. The records do not explain what caused the phase timing to move.

5. Interpretation and replication

Benchmarking work recommends specifying the work and checking outputs before comparing systems [3]. In this study, keeping those contracts fixed and separating the two load clocks exposes a reliable warmed ordering alongside a less stable first-load comparison. For an interactive workflow that reuses a running process, the warmed results give a stronger reason to investigate the C++ path. For a workflow that launches a process per request, the internal first-load clock answers a narrower question than launch-to-result latency. The 10M results demonstrate why both clocks should be reported if the application cares about both.

5.1 Scope and follow-up

The synthetic fixtures share one schema, and the sessions reuse the same builds on one host. The study therefore cannot distinguish effects of implementation, compiler, or scheduling policy. A follow-up could record launch, input acquisition, pool readiness, parsing, and materialization boundaries under the same workload, then change one condition at a time. In particular, an explicit readiness checkpoint would test whether the first-load results change when both pools are ready before timing. A replication should also compare canonical typed and dictionary outputs from the timed paths, then extend to a second host and CSVs with different row widths and quoting patterns.

The research archive retains every per-pair timing receipt, the frozen schedules and artifact digests, the correctness receipts, and the seeded analysis. The public plot lists the November 9 pairs so readers can inspect the spread and reconstruct the point estimates. Repeating acquisition requires the private n0is executable and the matching port and fixtures.

Further reading

- Comparing Swift with the n0is C++ engine
- Parallel evaluation throughput and latency in n0is
- Tips for Improving the Performance of Your ETL Processes from Dev3lop

References

1. Kalibera, T., and Jones, R. Quantifying Performance Changes with Effect Size Confidence Intervals. arXiv, 2020.
2. Stehle, E., and Jacobsen, H.-A. ParPaRaw: Massively Parallel Parsing of Delimiter-Separated Raw Data. PVLDB, 2020.
3. Raasveldt, M., Holanda, P., Gubner, T., and Mühleisen, H. Fair Benchmarking Considered Difficult: Common Pitfalls in Database Performance Testing. DBTest, 2018.