Pruning the Unchangeable: Static Safety Proofs, Cardinality-Specialized Dispatch, and Self-Captured Data Relayout

for Bit-Exact Switch-Level Simulation of a Whole Console

Ying-Wei Li
Independent Researcher, Taiwan
baxermux@gmail.com

***Abstract*—Switch-level simulation executes a chip as its transistors — the only software model that reproduces pass-transistor buses, dynamic charge storage and precharge behaviour bit-for-bit, which is why it remains the reference tool for die-level preservation of 1980s NMOS parts. It is also notoriously slow: every event re-resolves a channel-connected component (CCC) through pointer-chasing graph walks, and a widespread engineering impression holds that managed languages are unsuited to such kernels. We present AprVisual, a C# whole-console simulator of the NES (2A03 + 2C02, ∼14.7K nodes / ∼26.8K NMOS transistors) whose resolution semantics are functionally MOSSIM II. On one Zen 2 core it sustains ∼136K master-clock half-cycles/s (an equivalent ∼68 kHz silicon master clock) while remaining bit-exact (full-state checksums). The gain is the method's, not the language's: our untuned C# baseline already matches the optimized C++ ancestor (MetalNES), and the techniques below add a further ∼2× — overall ∼2.5× that C++ ancestor and ∼5.6× a literal chipsim.js port. The speed comes from three families: (1) *provably-null event suppression* — load-time structural proofs (including a capacitance-dominance argument for charge-share tie-break immunity) that delete ∼21% of all re-evaluations before they are enqueued; (2) *cardinality-specialized dispatch* — runtime proofs that the active CCC has size $\leq$ 2, resolved inline; (3) *self-captured data relayout* — an in-process inspector-executor pass that re-derives the node id space from the production cascade's first-touch order at every load. A nine-stage same-day ablation with hardware counters quantifies each step and shows the engine is latency-chain-bound, not miss-count-bound: the relayout cuts L1d misses by 67% while wall-clock moves single digits, and L1i misses stay at 0.2–0.3 MPKI — the architectural reason interpretation beats compilation here. A companion ∼100-page monograph and the full source, benchmark package, and raw data are public [21].**

***Index Terms*—switch-level simulation, event-driven simulation, channel-connected components, event suppression, data layout, bit-exactness, hardware preservation.**

I. Introduction

RTL and behavioural models cannot represent what 1980s NMOS chips actually do: bidirectional pass-transistor buses, storage implemented as charge on a floating node, precharge/conditional-discharge logic, and bus contention resolved by drive strength. The 2C02 PPU's sprite memory is not a RAM macro — it is ∼2,300 floating capacitors. Switch-level simulation in Bryant's MOSSIM lineage [1] is the only software level at which these behaviours *emerge* rather than being hand-modelled, which makes it the reference model for die-shot-derived netlists [15], [16] and for archival preservation. The cost of that fidelity is the subject of this paper. This matters beyond one chip: die-derived switch-level models are the substrate for hardware preservation and for verifying reverse-engineered silicon, the load-time and structural techniques that raise our single-core throughput are portable to any event-driven switch-level workload, and the ceiling we reach is itself a result — real-time for a whole-system NMOS model remains out of reach on one core along this route. Bit-exactness is non-negotiable here: the floating tie-break *is* the chip's memory mechanism, and we show (§VII) that seemingly safe approximations fail catastrophically — black screens, program-counter derailment within <1000 half-cycles — not gracefully.

Event-driven switch-level simulation spends its time re-resolving channel-connected components: per event, a graph walk over the currently-conducting transistors, a flags-OR, and a priority/charge resolution. On a die-derived netlist the conducting groups are tiny — we measure a mean of 1.13–1.4 nodes — and each event is tens of nanoseconds, so there is no large, regular computation to batch, vectorize, or offload. The walk is a chain of dependent loads; the engine lives at the memory-latency floor. Two presumptions follow this workload around: that an interpreter must be left far behind by compilation, and that a managed language is disqualified outright. Our measurements contradict both.

This paper makes five contributions.

1) **A prune family with static safety proofs** (P-1…P-4): suppress an enqueue when the re-evaluation is provably a no-op, using only static structure plus the two endpoints' live states. It includes the paper's strongest claim, a **capacitance-dominance un-taint** — a node whose capacitance is strictly below all of its channel neighbours can never win a Bryant charge-share tie-break, so equal-state merges through it are provably null — and a **pre-queue isolation prune** for degree-1 driverless leaves. Together they delete ∼21% of all node re-evaluations (and 33% of retired instructions, §VI) bit-exactly.
2) **Cardinality-specialized dispatch**: runtime proofs that the active CCC is exactly $\{n\}$ (R-1) or exactly {seed,

neighbour} (B1, the pair path), resolved inline while replicating the walk's order-sensitive semantics byte-for-byte. We frame this honestly as an early-out execution filter over IRSIM-style dynamic sizing [3] — engineering, not new theory — but it is worth +18.7% (R-1) and a further +7.0% (B1) on the ablation ladder.

3) **Self-captured first-touch relayout**: a three-pass load that classifies nodes into prune classes, rebuilds the id space class-major so the hot loop's safety lookups become register range-compares (with ground-truth verification and a deoptimization-style safe fallback at every reset), and then re-captures the production cascade's true first-pop order through a cold instrumented copy of the settle loop and rebuilds with it. It uses no profile files, works on any ROM, and is immune to workload drift by construction. The empirically new finding is that the key's value is the *pruned cascade's order*, not cache-line density (§VI-D).
4) **A zero-configuration safety methodology** (offered as implementation, not as a claim): memory, bus, and storage nodes are identified by their physics — pull-up flags, channel-component union-find, capacitance comparison, driven-pin exclusion — never by name. Storage excludes itself, because large capacitance is what storage physically is.
5) **A falsifiable performance boundary map** for single-core switch-level simulation: a nine-stage same-day ablation with hardware counters (§VI), and measured negative results for every abstraction and parallelization route we tried (§VII).

Because a journal-length treatment must omit most proofs, full negative-result data, and engineering detail, a companion **∼100-page technical monograph**, together with the engine, the benchmark package, the public leaderboard, and all raw measurement data, is openly available [21].

II. Background and System Model

Charge sharing and the floating tie-break. In the Bryant switch-level model [1], a node's value is decided by the strongest driver reachable through currently-conducting transistors; when no driver is reachable, the node is electrically *floating* and retains charge. Our engine resolves a conducting group by OR-ing its members' flags and indexing a 256-entry priority lookup table (ForceCompute Gnd+Pwr cancel; then Gnd > Pwr > externally-driven-high > externally-driven-low > pull-up > hold-previous). A purely floating group resolves to the previous state of its largest-capacitance member (strict greater; first-seen wins ties). That last rule is the chip's entire storage mechanism: every dynamic latch, pipeline-register bit, and OAM cell stores its value through it. The model is functionally MOSSIM II minus the X (unknown) state, substituting a deterministic power-on settle for unknown initial charge.

The settle loop and its order. Time advances one master-clock half-cycle (hc) at a time. Each hc drains a chain of *settle waves* (we measure a mean of 12.06, max 45): a wave re-evaluates the currently-dirty nodes, each doing its group walk. Crucially, the within-wave evaluation order is Gauss-Seidel and is **observable semantics** — processing a node mutates the states that determine whether other nodes' channels conduct, so reordering changes the result. This sweep order is a software convention of the Visual 6502 lineage [15], not a physical law; reproducing it bit-for-bit is the *frozen reference contract* that die-level preservation and trace verification require — a one-cycle divergence in an RNG seed or a sprite-collision flag voids verification — not a claim about the silicon's continuous-time behaviour. This single property governs every optimization and every negative result below.

The family. The netlists descend from the Visual 6502 project's chipsim.js [15] applied to the NES chips [16]. A literal C++ port (VisualNes [19]) and an optimized 6502-only rewrite (perfect6502 [18]) followed; MetalNES [17] re-engineered the lineage into a whole-console composition with board TTL, behavioural board memories, and the 256-entry resolution table — and is our direct ancestor. AprVisual inherits MetalNES's composition and the bit-exact contract, and adds everything in §III–§V. The engine's hot state is unmanaged structure-of-arrays (1-byte states, a 16-byte node record with a 12-byte inline payload, ushort transistor lists) with double-buffered wave queues and a dedup hash; board memories (RAM/ROM) are behavioural callbacks, while every on-die node — including all storage — is switch-level. The engine code and the benchmark-orchestration scripts were developed with assistance from Anthropic's Claude; every accepted change is gated by the bit-exactness checksums, so the AI assistance never relaxes the correctness contract (see the Generative AI Disclosure before the references).

III. Provably-Null Event Suppression

Profiling the steady state shows that ∼80% of node re-evaluations change nothing: the node is recomputed and resolves to the value it already held. These are not random — they are structural. The prune family deletes the provably-null subset of them *before they are enqueued*, so the saving is the whole enqueue→pop→walk→resolve chain, not merely a fast resolve.

A. P-1: the same-state turn-on prune

When a gate turns on, it merges two groups. If the two channel endpoints already hold the same state, and the merged group resolves through the monotone driven-priority LUT, the merge cannot change any value, so the enqueue is suppressed. Soundness requires a safety taint: nodes with no pull-up (whose value is the floating hold-previous, where an equal-state merge could still flip a tie-break) and ForceCompute channel-components are *unsafe* and always enqueue. The taint is computed once at load by union-find over channels.

B. P-2: the turn-off isolation prune

When a gate turns off, a channel endpoint that is degree-1 and driverless becomes an isolated, undriven singleton the instant its only channel opens. It therefore floats and *holds* its previous value — re-evaluating it is a guaranteed no-op. This is a purely static structural property, and suppressing it pre-queue is a claimed (secondary) original contribution.

C. P-3/P-4: the capacitance-dominance un-taint

> ***Property 1 (capacitance dominance).*** Let node x have a connection-count (capacitance proxy) strictly smaller than that of every one of its channel neighbours. Then x can never be the largest-capacitance member of any group it joins, so it can never determine a floating tie-break; consequently an equal-state merge through x cannot change any resolved value, and x may be un-tainted for P-1 even though it lacks a pull-up.

This is an exact algorithmic deduction rather than a physical approximation: the metric the resolver actually executes to settle a (rare) purely-floating group *is* the structural connection count — a topological proxy for parasitic capacitance — so Property 1 reasons about the exact integer the engine compares, not an estimate of it. The point is constructive: heavy storage nodes are exactly the dominant-capacitance participants, so a dominated node is provably *not* storage and is safe to prune. This recovers most of the no-pull-up mass that P-1's conservative taint would otherwise forbid. The nearest neighbour in the literature is IRSIM's `maxnode` size-coarsening [3], but that is a manual knob spent on *resolution* efficiency; ours is a derived inequality spent on *event suppression*, and it is exact rather than heuristic.

All three proofs are evaluated once at load. The classification pass is a union-find over channels plus capacitance comparisons and driven-pin exclusion; the hot loop only tests a precomputed bit. Together (ladder stages S2–S3, §VI) the prunes are worth +26.4% then +7.2%, delete ∼21% of re-evaluations, and cut retired instructions by 33% across the full ladder. What they cannot reach is the residual ∼80% no-change mass (pull-up and supply recomputations): three independent analyses confirm it has no static subset, so it is the structural floor.

IV. Cardinality-Specialized Dispatch

The group-size distribution is extreme: 77% of group walks build a group of exactly two nodes, and the BFS depth averages 1.13. Two runtime cardinality proofs exploit this, each resolving inline without touching the general BFS scratch.

A. R-1: the dynamic singleton

A node classified as a dynamic-singleton candidate has pass channels but, whenever *all* of its gates are off in the current half-cycle, its conducting group is exactly $\{n\}$. A one-pass scan of the gate states proves this at runtime; when it holds, the group resolves in O(1), bit-identically to the general walk. This is the single largest ablation step (+18.7%): it converts the most common case into a short dependent-load chain rather than a BFS.

B. B1: the pair path

When exactly one gate is on and the neighbour's own on-channels all lead back to the seed, the group is provably exactly {seed, neighbour}. It is resolved inline, replicating the walk's order-sensitive behaviour exactly: the member's dedup-hash is cleared, both nodes' supply lists are OR-ed, the floating tie-break uses the same strict-greater/seed-wins form, and the two writes happen in the walk's order so the next wave's enqueue order is preserved. Every bail-out (a second on-gate, an overflow or callback/ForceCompute neighbour, any on-channel of the neighbour reaching a third node) is checked before any mutation, so the fallback to the general walk is exact. B1 is worth +7.0% on the ladder. Both R-1 and B1 are honest variants of dynamic active-subnetwork sizing, distinguished from prior art only by being early-out execution filters with a bit-exactness obligation rather than resolution simplifications.

V. Self-Captured Data Relayout

Classical reverse-Cuthill–McKee renumbering [22] measured useless here: the hot working set (∼15 KB) is already L1-resident, so densifying it by 45% bought ±0. The objective had to change from *locality* to *deleting dependent loads on the carried chain*.

A. Range-prune (class-major renumbering)

The prune classes are made contiguous id blocks by a class-major renumber (boundaries A = 460, S = 1275, B = 7532 on this netlist), so the hot loop's safety lookups become register compares against constants instead of dependent memory loads (turn-off skip $\Leftrightarrow$ `c < S`; turn-on unsafe $\Leftrightarrow$ `c < A || c ≥ B`). The mask is recomputed from structure at every reset as ground truth and the ids are verified against it; a mismatch falls back to a safe-degenerate layout (prunes off, still correct) — the deoptimization-guard pattern [13]. This step is worth +4.2%.

B. The self-captured first-touch key

A three-pass load follows: (i) classify nodes; (ii) rebuild class-major, warm the engine past reset, and record 32,768 hc of the production cascade's *true* first-pop order through a cold instrumented copy of the settle loop; (iii) rebuild a final time using (class, captured-first-touch-order) as the sort key. Total load cost is ∼1.3 s, needs no profile file, and is re-derived on every load, so it is immune to workload drift. Correctness is layout-independent by construction: the power-on sweep and the state checksum iterate the original id order through the permutation, so all golden checksums are unchanged on any layout. This step is worth +5.0%.

C. Order, not density

The decisive control is a four-key experiment measuring per-half-cycle NodeInfo cache-line footprint against speed: a blind-BFS key touches 118.4 lines/hc; the same capture taken with the prunes *disabled* reaches 110 lines/hc yet yields ±0.0%; the capture taken with the prunes *enabled* reaches 105.9 lines/hc and yields +6.17%. Equal density, different lineage, opposite outcomes — the active ingredient is the order in which the *pruned production cascade* touches memory, not line packing. The hardware counters of §VI corroborate this directly.

VI. Evaluation

A. Environment and methodology

All measurements are on one machine: AMD Ryzen 7 3700X (Zen 2, 8C/16T, 32 KB L1d + 32 KB L1i per core), Windows 11, .NET 10/11 as noted per stage. The workload is `full_palette.nes` (a community open-source NES test ROM displaying the full palette), 400,000 master half-cycles per run (1M for counter runs), one benchmark process at a time. The clock is unlocked; variance is controlled by protocol — the ablation stages are real historical commits rebuilt in isolated worktrees and interleaved round-robin per round so thermal drift hits every stage equally, and we report per-stage medians. Every run is gated on the full-state FNV-1a checksum (`0x9174E19D961CB6E5` at 400k hc); all 72 ablation runs passed. Per-stage variation is small: the coefficient of variation across the nine rounds is 1.1–2.3% (median 1.7%), and a paired sign test over the rounds finds every adopted step significant — 9/9 rounds positive ($p = 0.004$) for every step but the +2.8% S4 (clause reorder + supply-skip), which holds in 8/9 ($p = 0.039$) — so even the smallest reported step clears the noise floor.

B. The ablation ladder

The cumulative gain S0→S7 is +94.6% (median), same machine, same day, all bit-exact. The two interposed neutral commits were tri-interleaved separately to confirm the S6→S7 attribution (+8.9% on fully separated distributions). The ladder is additive by construction — each stage is measured on top of the previous one in a fixed order — so it reports marginal contributions along that path, not a full-factorial isolation of interactions; stages that touch the same bottleneck (e.g. dispatch and the prunes) could redistribute credit under a different ordering. The *Nature* column states each step's standing against prior art (detailed in §III–V): *Engineering* = standard implementation, no novelty claimed; *Variant* = a known technique with a specific twist (the dispatch early-outs carry a bit-exactness obligation; the prunes/relayout are load-time/structural forms); *Original* = our two claimed contributions — the capacitance-dominance un-taint (P-3/P-4) and the self-captured first-touch relayout.

C. Hardware-counter profile

Three findings drive the paper's thesis. *(1) L1i is a non-issue in the interpreter* — 0.17–0.30 MPKI (≈0.6% of fetches) at every stage, because the fully-inlined settle loop fits the 32 KB L1i. This is the counter-level reason every compiled/AOT variant lost (§VII): code footprint explodes exactly where the interpreter pays nothing. *(2) The relayout cuts L1d misses by ∼52% per instruction (−67% absolute, 27.3→13.2 MPKI), yet wall-clock moved only +4.2%*, and the later self-capture key (+5.0%) barely moves the miss counters at all — direct evidence that the engine is dependent-load-*latency*-bound, not miss-*count*-bound, and that the locality key's value is order, not packing. *(3) The prunes delete instructions, not just events* — retired instructions fall 33% (155.6B→104.4B per 1M hc) while IPC stays ≈2.1–2.4. (This machine exposes no functioning last-level-cache counter; the ∼15 KB hot set is L1-resident, so L3 traffic is expected negligible.)

D. The family comparison

On the same machine, ROM, and unit, a literal chipsim.js port (VisualNes) sustains ∼24K hc/s and our direct ancestor MetalNES ∼54K; AprVisual reaches ∼136K — ≈2.5× the ancestor and ≈5.6× the literal port. (perfect6502 ∼29K is 6502-only on a different unit and not directly rankable.) NES real time is 42.95M hc/s, so the current single-core gap is ≈316×. Two comparisons matter here, and they are distinct. *Across languages*, our untuned C# baseline (S0, ∼68K) already matches the optimized C++ ancestor (∼54K), so the managed runtime is not the deciding factor — the language-viability claim rests on this baseline, not on the headline number. *Across methods*, the ≈2.5× from 54K to 136K is the algorithmic work of this paper (S0→S7), which is portable and not language-specific. (We did not re-tune MetalNES's build; the cross-language point is therefore made conservatively, baseline-to-ancestor.)

E. Generalization across the NMOS era

To test whether the engine is overfitted to the NES 2A03, we ran it unchanged on three other standalone Visual 6502 netlists — the MOS 6502, Motorola 6800, and Zilog Z80 — driven at the pin boundary by an "infinite NOP sled" (the data bus is forced to each chip's NOP opcode on every read, so no test ROM is needed). The only new code is a raw-netlist loader and a per-chip clock/bus driver; *the simulation core is untouched*, and the full method (lowering, P-1–P-4, class-major renumber, self-captured relayout, unmanaged hot path) applies as-is. All three power on and execute — the address bus advances as the program counter increments — refuting a netlist-specific implementation.

This generalization also lets us state the contribution precisely and defend it against the obvious objection — that a rewrite of a JavaScript simulator is fast merely because it left the browser. As a *calibration*, we transliterated the reference algorithm (chipsim.js's recursive group-walk) node-for-node into *both* C# and C++ (no prune, fast-path, or SoA; transistor counts identical to the JavaScript) and ran every engine on the identical workload (Table III). The two compiled naive baselines land within ∼1.4× of each other, so the ∼70–85× step from the JavaScript original is a one-time *language dividend* (interpreter→compiled), not part of our contribution. Factoring it out, our methods add a further **∼5–8× over a naive switch-level simulator in the same language** — consistent with the ∼2.5× over the optimized C++ ancestor of §VI-D (a tuned baseline is itself several× a naive one). The per-node checksum is identical across all configurations, confirming bit-exactness. We do not pursue a managed-vs-native comparison: at the same algorithm the two compiled languages sit within ∼1.4×, and the contribution is the algorithm, not the runtime. Finally, the prunes (P-1–P-4) and the capacitance-dominance tie-break rest on fundamental NMOS pass-transistor physics rather than NES-specific topology — which is why they carry to the 6502, 6800, and Z80 unchanged. Expressed as a clock,

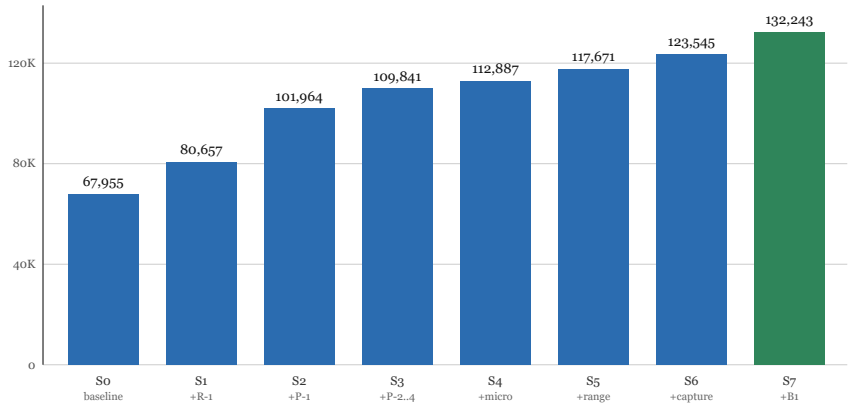

Fig. 1. Nine-stage same-day ablation ladder (median hc/s, 400k hc/run, all bit-exact). S0 baseline → S7 with the full method: +94.6% cumulative on one core.

TABLE I
ABLATION LADDER: PER-STAGE MEDIAN THROUGHPUT (HC/S), STEP GAIN, AND THE CONTRIBUTION'S NATURE RELATIVE TO PRIOR ART (§III–V).

Stage	Commit	Adds	median hc/s	step	Nature
S0	09565de	baseline fork (static fast path)	67,955	—	Engineering (baseline; known static-CCC dispatch)
S1	a80dab4	+ R-1 dynamic singleton	80,657	+18.7%	Variant (early-out over IRSIM dynamic sizing)
S2	ed8c457	+ P-1 same-state prune	101,964	+26.4%	Variant (load-time event suppression)
S3	6bdc25b	+ P-2/P-3/P-4	109,841	+7.2%	**Original** (capacitance-dominance proof — strongest claim; P-2 secondary)
S4	00ab4fa	+ clause reorder + supply-skip fold	112,887	+2.8%	Engineering (micro-optimization)
S5	51e046d	+ range-prune (.NET 11)	117,671	+4.2%	Variant (inspector-executor data reorg)
S6	3e4a571	+ self-captured first-touch key	123,545	+5.0%	**Original** (self-captured trace-driven relayout; order-not-density)
S7	2749105	+ B1 pair path	132,243	+7.0%	Variant (runtime cardinality template match)

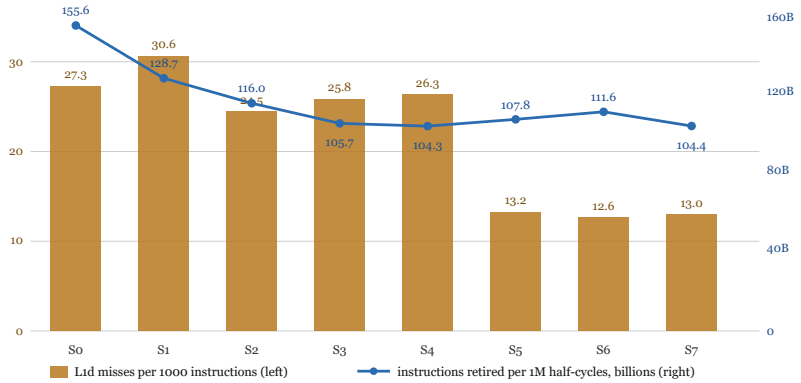

Fig. 2. Per-stage hardware counters (ETW PMC sampling, 1M hc/stage): L1d misses per 1000 instructions (bars) and retired instructions per 1M hc (line). The relayout (S5) halves L1d MPKI; instructions fall 33% across the ladder; L1i stays negligible throughout.

TABLE II
PER-STAGE PMC PROFILE (1M HC/RUN; EVENTS ≈ SAMPLES × INTERVAL).

Stage	IPC	L1i MPKI	L1d MPKI	Br-mispr MPKI	instr (B/1M hc)
S0	2.11	0.169	27.3	3.27	155.6
S2	2.38	0.187	24.5	2.74	116.0
S5	2.25	0.284	13.2	2.87	107.8
S7	2.34	0.292	13.0	2.68	104.4

these bare CPUs simulate at ∼31–73 kHz on one core (no NES master/12 divide), within ∼14–128× of their 1980s home-computer hosts — far closer to interactive than the whole NES (∼316×).

To verify these algorithmic gains are not artifacts of a specific microarchitecture, we replicated the entire evaluation on an ARMv8.2 platform (Broadcom BCM2712, Cortex-A76). The engine maintains strict cross-ISA bit-exactness — identical full-state checksums on both hosts — and the same-language algorithmic speedup over the naive baseline

TABLE III

GENERALIZATION TO THREE NMOS CPUS, AND THE LANGUAGE CALIBRATION: A NODE-FOR-NODE NAIVE PORT IN JAVASCRIPT / C# / C++ ISOLATES THE LANGUAGE DIVIDEND, LEAVING THE ALGORITHM COLUMN (NAIVE→APRVISUAL, SAME C#) AS THE CONTRIBUTION. HC/S, SAME MACHINE, NOP-SLED, PINNED; ALL BIT-EXACT.

Chip	Nodes	JS naive	C# naive	C++ naive	AprVisual	Algorithm
MOS 6502	1704	249	17,914	26,239	145,337	∼8.1×
Motorola 6800	2012	149	12,582	17,137	89,233	∼7.1×
Zilog Z80	3595	166	12,255	18,452	62,491	∼5.1×

TABLE IV

MEASURED NEGATIVE RESULTS (SINGLE INSTANCE, BIT-EXACT).

Route	Result
IR interpreter	−2.5%
AOT / codegen (C# emit, LLVM)	3–6× slower
GPU (single instance, D3D11)	∼10.7× slower
Bit-parallel BFS (Ligra-style [23])	∼156× slower
Two-thread CPU‖PPU split (per-wave)	0.68–0.89× (slower)
Maintained runtime facts (P-5)	+13.8% gross, −6.8% best net
Under-settling (truncate deep waves)	catastrophic divergence <1000 hc

reproduces (∼4–7× on ARM vs ∼5–8× on x64), confirming the optimizations are invariant to the underlying hardware.

VII. LIMITATIONS AND BOUNDARIES

We treat negative results as first-class: the project's documented outcome is a falsifiable boundary map, not only an engine. Every route below was implemented and measured; each row is one bounded implementation on this workload, not a proof that the whole class is unviable — e.g. a compiled route that grouped isomorphic CCC topologies into reusable functions (in the spirit of COSMOS [2]) rather than flattening the netlist could fare better, and is left as future work.

The structural reasons are consistent: the mean conducting group is 1.13–1.4 nodes (no batchable regularity, which kills SIMD and GPU); ∼94% of the dependency graph is one bidirectional strongly-connected component (no static partition); and the within-wave Gauss-Seidel order is observable semantics (no reordering). The latter is also why multi-threading fails even with a clean partition. We built a real bit-exact two-thread CPU‖PPU split [24] (the two chips are channel-disjoint apart from 14 coupling wires, so no lock is needed — only a barrier); it is bit-exact but parallelizes 0% of waves, because the shared clock distribution is present in ∼92% of waves and gates both sides, and a relaxed variant that fires 53% of waves runs at 0.68× because the ∼134 ns cross-core barrier swamps the ∼0.6 μs of per-wave work. The honest conclusion is that more cores help this workload only as throughput across many independent instances, which is outside this paper's single-instance, bit-exact scope. The compiled-route losses have a direct counter-level explanation (§VI-C): an interpreter that fits L1i and is bound by dependent-load latency leaves a compiler nothing to remove and a larger footprint to pay for. One scope caveat: bit-exactness here is defined against the Visual 6502 / MetalNES reference lineage [15]–[17], not against logic-analyzer captures of physical 2A03/2C02 silicon; the contribution is faithful reproduction of the established reference model, and validation against real dies is out of scope. The headline measurements are on one Zen 2 core; §VI-E shows the engine and its full method carry over unchanged to the 6502, 6800, and Z80 netlists — and, bit-exactly, to a second microarchitecture (ARM Cortex-A76) — so the result is not specific to the NES or to one host CPU. The non-NES chips were profiled with a synthetic pin-driven NOP sled (a conservative upper-bound stress test that keeps the engine CPU-bound, free of external-memory or I/O modeling); end-to-end test-ROM validation of those bare dies, and CMOS chips (whose complementary pathways the NMOS prunes do not yet model), are bounded future work.

VIII. CONCLUSION

A managed-language, bit-exact, whole-console switch-level simulator can reach ∼0.3% of silicon speed on one core — ≈2.5× its optimized C++ ancestor — by deleting provably-null events, specializing tiny-cardinality resolutions, and letting the engine re-derive its own memory layout from its own execution trace at load. The hardware counters locate the remaining time precisely: dependent-load latency on an L1-resident working set, with instruction fetch effectively free — the regime in which interpretation beats compilation and the floor moves only with the hardware (memory latency, per-core IPC). The strongest single claim — the capacitance-dominance un-taint, spent on pre-queue event suppression rather than IRSIM-style resolution coarsening [3] — and the order-not-density finding are, to our knowledge, new in this setting; the relayout itself is a zero-configuration *application* of inspector-executor and trace-driven layout [7]–[12], [20] to switch-level EDA graphs, and the dispatch specializations are honest variants of known techniques [2], [3]. Bounded future work, set by our own negative results, is the gate-level analysis view built on the same extraction machinery (value beyond speed) and revisiting the boundary map on future hardware, since every ceiling we have published fell to a measurement rather than to an argument.

GENERATIVE AI DISCLOSURE

The author acknowledges the use of Anthropic's Claude (accessed via the Claude Code CLI) to assist in drafting and polishing the manuscript, surveying related work, writing and refactoring the simulator engine code, orchestrating the benchmark and hardware-counter runs, and parsing experimental results. All AI-generated content and code were thoroughly reviewed, tested (via the bit-exactness checksums and the preserved hardware-counter measurements reported here), and verified by the human author, who takes full responsibility for the originality, accuracy, and integrity of this work. No AI system is listed as an author or co-author. This statement is

intended to satisfy arXiv's generative-AI policy, IEEE PSPB guidance, and the ACM Policy on Authorship.

References

[1] R. E. Bryant, "A Switch-Level Model and Simulator for MOS Digital Systems," *IEEE Trans. Computers*, C-33(2), 1984.

[2] R. E. Bryant et al., "COSMOS: A Compiled Simulator for MOS Circuits," *Proc. DAC*, 1987.

[3] A. Salz and M. Horowitz, "IRSIM: An Incremental MOS Switch-Level Simulator," *Proc. DAC*, 1989.

[4] E. G. Ulrich, "Exclusive Simulation of Activity in Digital Networks," *Commun. ACM* 12(2), 1969.

[5] M. Boehner, "LOGEX — an Automatic Logic Extractor from Transistor to Gate Level for CMOS," *Proc. DAC*, 1988.

[6] M. Ohlrich, C. Ebeling, E. Ginting, L. Sather, "SubGemini: Identifying SubCircuits using a Fast Subgraph Isomorphism Algorithm," *Proc. DAC*, 1993.

[7] J. Saltz et al., "A Manual for the CHAOS Runtime Library," Univ. of Maryland TR, 1995.

[8] C. Ding and K. Kennedy, "Improving Cache Performance in Dynamic Applications through Data and Computation Reorganization at Run Time," *Proc. PLDI*, 1999.

[9] M. M. Strout, L. Carter, J. Ferrante, "Compile-time Composition of Run-time Data and Iteration Reorderings," *Proc. PLDI*, 2003.

[10] T. M. Chilimbi, M. D. Hill, J. R. Larus, "Cache-Conscious Structure Layout," *Proc. PLDI*, 1999.

[11] T. M. Chilimbi, "Efficient Representations and Abstractions for Quantifying and Exploiting Data Reference Locality," *Proc. PLDI*, 2001.

[12] B. Calder et al., "Cache-Conscious Data Placement," *Proc. ASPLOS*, 1998.

[13] U. Hölzle, C. Chambers, D. Ungar, "Debugging Optimized Code with Dynamic Deoptimization," *Proc. PLDI*, 1992.

[14] S. Somogyi et al., "Spatial Memory Streaming," *Proc. ISCA*, 2006.

[15] The Visual 6502 project — chipsim.js and the die-shot-derived 6502 netlist. visual6502.org.

[16] Quietust, Visual 2A03 and Visual 2C02 die-shot-derived netlists.

[17] iaddis, MetalNES — transistor-level NES-001 simulation. github.com/iaddis/metalnes.

[18] M. Steil et al., perfect6502. github.com/mist64/perfect6502.

[19] VisualNes — C++ port of the Visual 2A03/2C02 simulators.

[20] M. R. Haghighat and D. C. Sehr, "Profile-guided data layout," U.S. Patent 7,143,404 B2, 2006.

[21] AprVisual — engine, benchmark package, leaderboard, raw data, and the companion ∼100-page monograph: github.com/erspicu/AprVisual; erspicu.github.io/AprVisual.

[22] E. Cuthill and J. McKee, "Reducing the Bandwidth of Sparse Symmetric Matrices," *Proc. 24th Nat. Conf. ACM*, 1969.

[23] J. Shun and G. E. Blelloch, "Ligra: A Lightweight Graph Processing Framework for Shared Memory," *Proc. PPoPP*, 2013.

[24] R. M. Fujimoto, "Parallel Discrete Event Simulation," *Commun. ACM* 33(10), 1990.