Journal of Theoretical and Applied Information Technology
30st June 2026. Vol.104. No.12

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

# HETEROGENEOUS ACCELERATION OF REAL-TIME OPERATING SYSTEM DIGITAL TWINS VIA CPU-GPU CO-SIMULATION FRAMEWORK

**DUC-THANG NGUYEN[1], DONG-LUONG DINH[2]**

[1] Institute of Digital Technology, Thu Dau Mot University, Ho Chi Minh City, Vietnam

[2]Department of Information Technology, Nha Trang University, Khanh Hoa, Vietnam

E-mail: [1]thangnd@tdmu.edu.vn, [2]luongdd@ntu.edu.vn

**ABSTRACT**

Real-time operating system (RTOS) digital twins are essential validation platforms for safety-critical cyber-physical systems, including autonomous aerial vehicles and 6G radio units. However, as sensor counts scale toward tens of thousands of channels, legacy single-node simulators suffer from significant temporal drift, where host CPU overhead for virtual peripherals exceeds kernel execution time. This paper introduces a Heterogeneous Co-simulation (HCS) framework that addresses this by separating the discrete RTOS logic from the continuous sensor manifold. The implementation of HCS executes the RTOS model on the CPU and evaluates the sensor manifold through a common array-backend abstraction that resolves to a single-instruction-multiple-thread capable accelerator backend when one is available and otherwise to a vectorized CPU backend. Both paths communicate through a shared-state module, avoiding explicit device-to-host copy logic in the coordinator. Results from thousands of sensor unmanned-aerial-vehicle case study show that HCS achieves an order-of-magnitude speedup over a serial CPU-only baseline on the evaluated host. A statistical evaluation on five controlled scenarios including sensor-count scalability, tick-rate sensitivity, noise sensitivity, interrupt-threshold sweeps, and long-run latency characterization shows that the HCS plane is strictly faster than the CPU-only baseline in every configuration. The accompanying implementation and testable component boundaries ensure reproducibility, establishing HCS as a scalable alternative to legacy single-node simulators. In summary, this paper contributes a two-plane execution model that places the RTOS kernel in a closed simulation loop with an accelerator-resident sensor manifold through a bit-packed virtual interrupt controller interface. In practice, the framework enables engineers to validate interrupt-dense, safety-critical firmware at sensor scales, thereby shortening certification campaigns and lowering energy costs.

**Keywords:** *Real-time Operating Systems, Digital Twins, Heterogeneous Co-simulation, CPU-GPU Partitioning, Parallel Discrete-event Simulation*

## 1. INTRODUCTION

Safety-critical cyber-physical systems (CPS) are increasingly defined less by the physical machine they control than by the real-time software stack that mediates that control. Autonomous unmanned aerial vehicles, collaborative industrial robots, 6G open-radio-access-network units, and connected-medical devices share the same architectural pattern: a certifiable real-time operating system (RTOS) kernel orchestrates a portfolio of periodic and aperiodic tasks whose release and completion times must satisfy hard deadlines against a backdrop of hundreds to tens of thousands of concurrent sensor channels. Certifying that such a stack meets its timing contracts in the field is prohibitively expensive when done on physical hardware alone, both because of the combinatorial explosion of operating envelopes and because most failure modes--interrupt pile-ups, priority inversions, deadline misses triggered by correlated sensor bursts--are statistically rare. Digital twins and software-in-the-loop (SIL) platforms have therefore become mandatory companions to hardware-in-the-loop (HIL) testing [1], [2].

The dominant tools for software-in-the-loop validation today are single-host instruction-set simulators and cycle-level platform models [3], [4], [5]. The architectural assumption common to all of them - that the host can execute the guest, the kernel, and every modelled peripheral on a single host thread without losing timing fidelity - was reasonable when

guests had a handful of peripherals, but it has not aged well. Brandt et al. [6] report that as soon as a few tens of virtual peripherals are attached, the host scheduler becomes the bottleneck and the simulated clock falls behind wall time, and Zhang and Chen [7] show that the same regime produces interrupt pile-up at high sensor cardinality, exactly the failure mode that safety-critical certification is intended to detect.

The deeper issue is a structural mismatch between the two halves of the workload. The continuous environment, which in our setting is a population of $m$ sensors each governed by a small stochastic differential equation, is embarrassingly parallel: every sensor advances independently and writes a single bit into a virtual interrupt controller (VIC). The RTOS kernel, by contrast, is intrinsically serial: a small number of tasks share a single ready-queue, the scheduler must observe a coherent snapshot of time, and interrupt service routine (ISR) dispatch order is part of the certification contract. Running both on the same CPU forces the serial side to scan the parallel side at every tick, which is the source of the temporal-drift effect quantified in Section 3 and is the opposite of what modern multi-core and accelerator hardware is designed for. Heterogeneous substrates - Single Instruction Multiple Thread (SIMT)-capable accelerators with cache-coherent unified memory [8] and host-attached devices reachable over Compute Express Link (CXL) 3.x [9] - make it natural to offload the continuous, batched manifold while keeping the discrete kernel on the CPU, provided that the two planes are coupled by a synchronisation interface that does not leak host-side bookkeeping into the kernel's critical path. Designing that interface, and showing that the resulting simulator is statistically faster than the serial baseline at the ten-thousand-sensor scale, is the subject of this paper.

The problem statement addressed in this paper is the following: given an RTOS digital twin that must co-simulate a serial, certification-constrained kernel together with thousands to tens of thousands of independent continuous sensor channels on a single heterogeneous host, design a partitioning and synchronisation scheme whose per-tick wall-clock cost grows sublinearly in the sensor count, while preserving the interrupt semantics (dispatch order, latency, and jitter) on which certification depends. Legacy single-host simulators violate this requirement because their per-tick cost is additive in the kernel and peripheral work, so the simulated clock inevitably falls behind wall-clock time as the number of sensors grows.

The problem statement leads to three research questions. **Research Question (RQ) 1:** Can a two-plane CPU-accelerator partitioning scheme effectively reduce the measured per-tick cost scaling from the strictly linear regime characteristic of legacy single-host simulators to a sublinear regime with respect to the sensor count? **RQ2:** Can interrupt discovery be algorithmically reformulated as a bit-packed reduction over mask words without distorting the ISR latency and jitter statistics observed by the guest kernel? **RQ3:** When evaluated at the ten-thousand-sensor scale requisite for a realistic Unmanned Aerial Vehicle (UAV) swarm digital twin, what statistically significant speed-ups and energy savings does the proposed framework deliver compared to a standard, serial CPU-only baseline operating on commodity hardware?

This paper proposes a Heterogeneous Co-Simulation (HCS) framework that exploits precisely this partitioning. A discrete logic plane runs on the CPU and owns the kernel, the task control blocks, and the interrupt service routines. A continuous physics manifold runs on a vectorised or GPU-resident backend and integrates all sensor ordinary differential equations in batched array operations, reducing threshold crossings into a 64-bit VIC bitmask. The two planes rendezvous at the RTOS tick boundary through a shared state object that mirrors the Unified Memory / CXL 3 programming model, so the coordinator exchanges data through common arrays rather than through explicit copy calls.

This paper makes the following three contributions:

**Formal Two-Plane Execution Model:** We define a mathematical foundation for heterogeneous co-simulation, deriving a temporal-drift ratio ($\beta$) for both CPU-only and HCS partitions. This model includes a step-cost decomposition that isolates kernel, sensor, and synchronization overheads, as well as a stochastic interrupt-arrival model used to interpret measured ISR latency and jitter.

**Modular HCS Implementation:** We present a framework composed of decoupled, portable components: a cache-coherent shared-state object, a vectorized physics-manifold routine, and a priority-pre-emptive real-time kernel model with interrupt-latency accounting. Crucially, the system uses a unified coordinator capable of executing across both CPU (SIMD) and accelerator (SIMT) backends without modification. The framework treats the 64-bit VIC bitmask and threshold-crossing predicates as first-class primitives.

**Large-Scale UAV Swarm Case Study:** We demonstrate the practical utility of the HCS framework through a 10,000-sensor simulation (representing 1,000 UAVs with 10 sensors each). This study reports actual, measured speed-ups on laptop-class hardware, proving that HCS can achieve real-time performance at scales that are prohibitive for legacy simulators.

**Scope of the paper.** The scope of this research is strictly confined to evaluating the simulation-level timing fidelity of RTOS digital twins executing on a single heterogeneous host. The kernel is modelled at the scheduler/ISR level of abstraction rather than at instruction or cycle accuracy, the target regime is the sensor-rich envelope ($10^3$-$10^4$ channels) typical of UAV swarms and 6G radio units, and all measurements are taken on standard computer hardware. Distributed multi-node co-simulation, cycle-accurate guest execution, and certification of the simulator itself are outside the scope of the present study and are discussed as open issues in Section 6.

**Practical implications.** The results imply that interrupt-dense firmware validation campaigns that currently require server-class hardware, or are abandoned as intractable, can be executed on a standard desktop computer. The measured order-of-magnitude speed-up translates into proportionally shorter regression cycles, the bounded temporal drift keeps simulated timing contracts trustworthy, and the energy-efficiency analysis of Section 5.8 shows a corresponding reduction in the energy footprint of large validation campaigns.

The remainder of the paper is organised as follows. Section 2 reviews the state of the art in RTOS simulation, cycle-accurate microarchitectural simulation, parallel discrete-event simulation on GPGPUs, digital-twin platforms for robotics and CPS, and recent progress on heterogeneous coherence, and identifies the gap that this paper addresses. Section 3 develops the mathematical model of the two planes and derives the drift and jitter expressions. Section 4 describes the reference implementation and its synchronised event loop. Section 5 presents the full statistical evaluation, including the UAV-swarm case study. Section 6 discusses the results, positions them against the related literature, states the limitations of the present evaluation, and sketches an outlook deployment scenario.

## 2. RELATED WORKS

The HCS framework sits at the convergence of four research domains that have matured independently over the past two decades: (i) instruction-set and cycle-accurate simulation; (ii) parallel discrete-event simulation (PDES) on GPUs; (iii) high-fidelity digital twins and robotics; and (iv) heterogeneous memory architectures, such as Unified Memory and CXL. This section reviews these established fields and identifies the architectural gap that necessitates our proposed framework.

### 2.1 Functional Instruction-Set Simulation

Dynamic binary translators established functional ISS as the dominant vehicle for embedded-software validation. QEMU [3] popularised the approach by combining a translation cache with a helper-function call-out for privileged instructions, enabling interactive-speed emulation of ARM, x86, and RISC-V guests. Imperas OVPSim and Synopsys Virtualizer extend the model to commercial verification, while open-source platforms such as Antmicro's Renode [4] focus on board-level integration with heterogeneous peripherals, and JIT-based virtual platforms have been shown to retain consistency with cycle-accurate models even for fault-injection campaigns such as soft-error assessment [10]. All of these simulators were designed around a single host thread that interleaves guest-instruction execution with peripheral updates. Brandt et al. [6] quantify the resulting bottleneck on modern multicore guests: once the peripheral count grows beyond a few tens, the host scheduler becomes the limiting factor and the simulated clock falls behind wall time; Zhang and Chen [7] further document how this manifests as interrupt pile-up at high sensor-trigger rates. A line of engineering work [11] has partially mitigated this through speculation, just-in-time specialisation, and multi-thread decoupling, and Joe et al. [12] showed early on that asynchronising interrupt-service routines outside the main simulation loop can substantially raise the throughput of sensor-network simulators on a single host—an idea that anticipates the two-plane decomposition of HCS but stops short of moving the continuous physics off the CPU. Despite these advances, the core serial-scheduler assumption has remained intact.

### 2.2 Cycle-Accurate and Microarchitectural

Cycle-accurate platforms exchange throughput for per-cycle visibility. gem5 [13] and its multi-ISA

descendant Lowe-Power et al. [5] provide detailed pipeline, cache, and on-chip-network models that are widely used for architectural research, and Sniper [14] and Structural Simulation Toolkit extensions [15] speed up the slow-clock regime through interval simulation. Computer-architecture textbooks such as Hennessy and Patterson [16] emphasise that cycle-accurate simulation is typically two to three orders of magnitude slower than real time, which makes it ill suited to long-horizon mission validation of autonomous systems. Recent work on sampling-based microarchitectural simulation [17] narrows the gap at the cost of losing guarantees about rare events--precisely the events that RTOS certification is most interested in.

### 2.3 Parallel Discrete-Event Simulation on GPGPUs

PDES predates GPGPU and has traditionally been deployed on multicore CPUs and clusters. The migration to SIMT hardware began with Perumalla and collaborators [18], who mapped event queues onto GPU warps and demonstrated throughputs of hundreds of millions of events per second. Santhi et al.'s Simian framework [19] extends the pattern to heterogeneous hardware, and Bauer et al.'s Legion and Regent runtimes [20] generalise the idea to task-based data-parallel simulation. Wang and Tropper [21] recently revisited optimistic PDES on GPUs and showed that warp-level speculation can sustain sub-microsecond rollback overhead for suitably partitioned event queues. However, all of these systems are open-loop: they are optimised for producing high-throughput event traces and do not expose a bounded-latency feedback path back to a closed-loop RTOS scheduler. The HCS coordinator proposed in this paper closes that loop.

### 2.4 Digital-Twin Platforms and Robotics Simulators

Industrial digital-twin frameworks such as NVIDIA Isaac Sim and Isaac Lab [22], Microsoft's Project Bonsai, and Siemens' MindSphere integrate high-fidelity physics, sensor, and actuator models into a unified simulation environment. Recent studies document GPU-resident LIDAR, IMU, and radar pipelines whose end-to-end latency is below a microsecond on data-centre-class GPUs [23]. Open-source stacks built on ROS 2 [24] and Gazebo / Ignition [25] play an analogous role in the academic community. The common weakness of these platforms for RTOS validation is that the embedded kernel is typically either absent--replaced by a high-level control node--or emulated in a separate ISS process with an unbounded message-passing gap--a limitation that Pazzaglia et al.'s systematic review of autonomous-driving simulators [26] singles out as the dominant source of validation pessimism. Lee et al. [27] showed that a carefully tuned hybrid CPU-GPU partition can sustain million-sensor simulation throughput, and Zhao et al. [28] extended the study to multi-vehicle swarms. Both still treat the kernel as a black box. Our framework explicitly brings the kernel into the loop through the VIC bitmask interface.

### 2.5 Memory, CXL, and Asynchronous Event Injection

The feasibility of tight CPU-GPU co-simulation depends on the memory model. NVIDIA's Unified Memory [8], AMD's ROCm HMM [29], and Intel's oneAPI Level Zero [30] expose a single logical address space to both host and device and rely on page-fault-driven migration for coherence. The Compute Express Link standard, in its recent 3.1 revision [9], standardises cache-coherent semantics across host, accelerator, and memory devices over PCIe-class fabrics, and is expected to dominate the 2026 heterogeneous-platform landscape. Gupta and Miller [31] demonstrated that the combination of Shared Virtual Addressing and CXL-style coherence enables what they term 'asynchronous event injection,' in which an accelerator thread writes an interrupt bit directly into the RTOS kernel's address space without any explicit device-to-host copy. At the edge end of the same spectrum, the VEDLIoT toolchain [32] composes CPU, FPGA, and accelerator tiles for very efficient deep-learning IoT workloads, demonstrating that the heterogeneous-substrate assumption now extends from data-centre hosts down to embedded edge nodes of the kind that HCS is intended to validate. The shared-state object of the HCS framework is designed precisely to match this interface: it is a plain cache-coherent data structure that can be allocated either by a standard CPU allocator or by an accelerator Unified-Memory allocator, without any change to the coordinator. Most recently, Sundar et al. [33] reported the first CXL-3 cache-coherent real-time co-simulation of edge AI accelerators, corroborating the architectural assumption on which the HCS state object is built.

### 2.6 Real-Time Scheduling and Learning-Based Schedulers

Liu and Layland's foundational work [34] established the feasibility conditions for rate-monotonic and earliest-deadline-first scheduling, and subsequent research has extended the framework

to mixed-criticality [35], probabilistic worst-case execution time [36], and distributed task chains [37]. The recent trend toward learning-based scheduling, notably the use of reinforcement learning to adapt priorities online [38], makes simulator fidelity more important, not less: a reinforcement-learning agent that learns on a temporally drifting simulator will transfer pathological policies to the physical target. An HCS simulator that keeps the simulated clock close to wall time is therefore a prerequisite for reliable offline reinforcement learning in RTOS contexts. Production RTOS implementations such as FreeRTOS [39] and Zephyr [40] remain the baseline for certification, and the Arm Generic Interrupt Controller [41] defines the hardware contract that our VIC emulates.

### 2.7 Energy Efficient Simulation and Sustainable Computing

Energy efficiency of computing has moved from a secondary optimisation target to a first-class design constraint. Strubell et al. [42] raised early awareness of the carbon cost of deep-learning training, and subsequent work has generalised the concern to systems benchmarking: Henderson et al. [43] propose a reporting standard for energy and carbon in machine-learning experiments, Schwartz et al. [44] argue for 'Green AI' as an evaluation axis on par with accuracy, and the MLPerf Power working group [45] now publishes per-system Joule and kWh figures alongside throughput. On the measurement side, Intel's Running Average Power Limit interface [46] exposes per-domain energy counters on commodity x86 hosts, and NVIDIA Management Library [47] provides board-level power telemetry on CUDA accelerators. For the present paper these lines of work motivate the energy-efficiency analysis of Section 5.8: rather than treating wall-clock speed-up as the sole metric, we report the ratio of simulated seconds to total package Joules, which is the natural figure of merit for sustainability-aware digital-twin deployments of robotic and 6G edge systems [1], [2].

### 2.8 Research Gap and Position of this Paper

Prior work has separately parallelised the data side of simulation (high-throughput sensor pipelines, dense linear algebra, perception pre-processing) and the statistical evaluation side (benchmark suites, latency profiling tools). The timing and interrupt logic of the RTOS itself, however, remains a serial CPU-bound component in almost every published simulator. Our contribution is to treat interrupt discovery as a bit-packed reduction over sensor trigger predicates into $W = \lceil m/64 \rceil$ 64-bit mask words and to expose a synchronised coordinator (Algorithm 1, Section 4.5) whose measured per-tick wall-clock cost grows much more slowly than the serial CPU-only comparator. By grounding the evaluation in paired statistical tests, we move the debate about 'how fast is fast enough' from qualitative throughput plots to quantitative, reproducible claims about significance, effect size, and scaling exponents.

Several components of the HCS argument rely on previously published work, which we explicitly acknowledge here. Joe et al. [12] demonstrated that asynchronous ISR handling accelerates sensor-network simulation on a single host; Lee et al. [27] and Zhao et al. [28] showed that hybrid CPU-GPU partitions sustain million-sensor throughput; and Gupta and Miller [31] established asynchronous event injection across a shared virtual address space, with Sundar et al. [33] corroborating cache-coherent real-time co-simulation over CXL 3. The data presented in this manuscript differs from, and builds upon, these publications in three key respects: (i) prior CPU-GPU partitions treat the RTOS kernel as a black box or omit it entirely, whereas HCS brings the kernel into the closed loop through the VIC bitmask, meaning the measured quantities—kernel-observed interrupt latency and jitter—are not reported by the cited works; (ii) published GPU-PDES results [18], [21] are open-loop throughput figures, whereas our findings in Section 5 are closed-loop measurements taken at the tick barrier; and (iii) prior evaluations report single-run throughput, whereas our claims rely on paired, seeded, multi-replication statistical comparisons. This critical analysis crystallizes into the problem statement and research questions defined in Section 1, with RQ1 answered by the scaling analyses of Sections 5.3 and 5.7, RQ2 addressed by the latency and jitter studies of Sections 5.5 and 5.6, and RQ3 resolved by the UAV-swarm case study and energy analysis of Sections 5.7 and 5.8. By explicitly addressing these methodological gaps in the existing literature, our approach ensures that the resulting digital twin maintains strict timing fidelity even under extreme scaling conditions. Furthermore, this rigorous validation establishes a highly reliable foundation for the future testing of certification-constrained firmware.

## 3. MATHEMATICAL MODEL

### 3.1 Architectural Synthesis and Closed-Loop Interaction

The mathematical model of the HCS framework is built from three interlocking components, each

developed in its own subsection below: *temporal synchronization* (Section 3.2), *sensor dynamics* (Section 3.3), and the *VIC together* with *its jitter model* (Section 3.4). This subsection sketches how these three components fit together as a closed-loop feedback system *before* the formal definitions are introduced, so that the equations that follow can be read in the context of the overall architecture rather than as isolated objects.

### 3.1.1 From sensor dynamics to interrupts

The sensor dynamics of Section 3.3 are the event source of the simulator. At every RTOS tick, the physics manifold integrates a stochastic ordinary differential equation in parallel for all $m$ sensors and writes back the updated displacements. The VIC of Section 3.4 then evaluates a per-sensor threshold predicate on those displacements and packs the Boolean indicators into a compact bit-mask. Because the discretisation used by the physics manifold is semi-implicit and the stochastic increment is scaled with $\sqrt{\Delta t}$ in the Itô sense, the resulting trigger arrivals are driven by a physically consistent state-space trajectory rather than by numerical or noise-floor artifacts, so the Poisson arrival model that Section 3.4 attaches to each VIC word reflects simulated physical reality.

### 3.1.2 From interrupts to temporal health

The efficiency of the VIC, in turn, controls the synchronization cost that enters the temporal-drift ratio of Section 3.2. In a legacy single-host simulator the host loop must walk every one of the $m$ sensors at every tick even when only a small fraction of them have crossed their threshold, which dominates the per-tick budget and forces the simulated clock to fall behind wall time. The HCS framework collapses interrupt discovery into $W = \lceil m/64 \rceil$ bit-mask words, so the host CPU spends its tick budget on actual kernel execution and ISR service instead of on polling non-triggered sensors. This is the structural mechanism through which the temporal-drift ratio of Section 3.2 stays close to unity at high sensor cardinality.

### 3.1.3 Convergence of the two-plane model

The two structural choices that close the loop are therefore (i) the semi-implicit stochastic update of the sensor manifold, derived in Section 3.3, and (ii) the bit-packed VIC reduction, derived in Section 3.4. The first guarantees that the physics manifold can advance many sensors in a single batched array operation without introducing integrator-induced jitter, while the second guarantees that the discrete logic plane sees interrupts as a constant-size summary regardless of $m$. Together they let the simulator scale to tens of thousands of sensors while keeping the measured ISR latency bounded by the configured workload and the tick budget. The remainder of Section 3 makes each of these claims precise: Section 3.2 derives the per-tick cost decomposition that supports the temporal-drift bound, Section 3.3 specifies the stochastic-differential-equation form and its discretisation, and Section 3.4 defines the VIC trigger predicate, bit-mask reduction, and Poisson arrival model that drive the latency / jitter statistics measured in Section 5.

## 3.2 Temporal Synchronization

Let $t_{sim}$ and $t_{wall}$ denote simulated and wall-clock time, respectively. The simulator speed ratio is defined as:

$$\beta = \frac{dt_{sim}}{dt_{wall}} \tag{1}$$

The value of $\beta$ determines the fidelity of the digital twin relative to the physical passage of time:

Real-Time ($\beta = 1$)**:** The simulation clock moves exactly as fast as the real-world clock.

Faster-than-Real-Time **(**$\beta > 1$**):** This state is ideal for high-throughput batch processing and automated regression testing.

Slower-than-Real-Time **(**$\beta < 1$**):** This represents a failure state for RTOS validation. If the simulator cannot maintain $\beta \approx 1$, interrupts intended to occur at specific millisecond intervals begin to "drift." This introduces artificial temporal jitter that does not exist in the source code, potentially leading to false negatives during safety-critical testing.

### 3.2.2 Legacy CPU-only

In a traditional single-threaded simulator, execution follows a sequential discrete-event queue. For every "tick" of the guest RTOS (e.g., a *1ms* interval), the host CPU must execute:

$$T_{CPU} = \sum_{i=1}^{n} T_{task,i} + \sum_{j=1}^{m} T_{sensor,j} + T_{overhead} \tag{2}$$

where *n* is number of task and *m* is the sensor count. As $m$ increases, the total step time $T_{CPU}$ grows linearly ($O(m)$).

**The Bottleneck:** Once $T_{CPU}$ exceeds the actual duration of the RTOS tick (e.g., the host requires 1.2 ms to simulate 1 ms of guest time), the simulator loses its real-time guarantee. The system becomes "sensor-bound," where the host spends the majority

of its cycles calculating peripheral physics rather than executing the core RTOS logic.

### 3.2.3 HCS per-Tick cost

The ideal heterogeneous execution model shifts the computational cost from additive summation toward a critical-path maximum. In a fully asynchronous accelerator implementation, the discrete logic plane and continuous physics manifold can overlap. The reference implementation used for the measurements is deliberately conservative: it invokes the array-backed manifold step as a blocking call, synchronises explicitly, and then runs the RTOS slice. Equation (3) therefore describes the architecture target and the lower-bound critical path, while the measured implementation pays the blocking vector step plus the kernel slice.

$$T_{HCS} = \max(T_{CPU,kernel}, T_{GPU,physics}) + T_{sync} \quad (3)$$

**Batched Manifold Execution:** The reference implementation evaluates the ODE integration for all $m$ sensors in a single array operation through the array-backend abstraction, removing the host-level per-sensor loop from the HCS path.

**Bit-Packed** Reduction**:** Traditionally, evaluating threshold crossings for $m$ sensors requires $O(m)$ serial polling. HCS packs the trigger predicates into $W = \lceil m/64 \rceil$ 64-bit VIC words. On SIMT hardware this maps naturally to a tree/OR reduction; on the vectorised CPU backend used for most reported measurements it is implemented as vectorised word construction over full 64-sensor blocks. In particular, for 10,000 sensors, a serial CPU requires 10,000 sequential steps. Meanwhile, the HCS GPU reduction tree has a depth of only $\approx 14$ steps ($\log_2 10{,}000$).

### 3.2.4 Scaling and bounded execution time

Because the HCS path batches the manifold work and compresses interrupt discovery from $m$ predicates to $W$ words, increasing the sensor count has a substantially smaller measured impact than in the serial CPU-only baseline. As long as the vectorised workload and synchronization overhead ($T_{sync}$) remain within the guest RTOS tick budget, the simulator maintains $\beta \approx 1$. This enables high-density environments, such as large UAV swarms, to be evaluated with far less temporal drift than a legacy serial simulator.

### 3.3 Sensor Dynamics

The fidelity of an RTOS digital twin is fundamentally limited by the accuracy of its peripheral models. Rather than treating sensor data as a simple playback of logged values, the HCS framework models each sensor as a dynamic mechanical entity. This allows the simulator to recreate complex physical interactions—such as resonance, vibration interference, and stochastic drift—that are often the root causes of timing-related failures in safety-critical systems.

Each virtual sensor $i$ carries a two-dimensional state $(p_i, v_i)$ representing the internal displacement and velocity of a virtual proof mass. The continuous-time dynamics are modeled as a damped harmonic oscillator, which serves as the physical foundation for most Micro-Electro-Mechanical Systems (MEMS) sensors like accelerometers and gyroscopes. The system is driven by an actuator-coupled excitation and perturbed by a Wiener process that models gyroscopic drift and thermal noise:

$$\frac{dp_i}{dt} = v_i \quad (4)$$

$$\frac{dv_i}{dt} = -kp_i - cv_i + gu_i(t) + \sigma\frac{dW_i}{dt}$$

In Equation (4), the term $-kp_i$ represents the restoring force (stiffness), while $-cv_i$ represents the dissipative force (damping). These parameters are tuned to match the frequency response of specific hardware sensors. The final term, $\sigma\frac{dW_i}{dt}$, introduces Gaussian white noise. In the context of a digital twin, this noise is not merely a nuisance; it is a critical component for verifying the robustness of the guest RTOS's sensor fusion algorithms and Kalman filter implementations.

The actuator-coupled excitation is modelled with a sinusoid whose amplitude is read from the shared actuator vector **a** and whose phase is fixed at construction to decorrelate neighbouring sensors:

$$u_i(t) = a_{A_i}\sin(2\pi ft + \varphi_i) \quad (5)$$

The drive signal $u_i(t)$ simulates the mechanical vibrations produced by the system's physical actuators (e.g., drone motors). By using a phase-decorrelated model ($\varphi_i$), we avoid "phase-locking" artifacts that are common in simpler simulations. This ensures that the collective noise floor across thousands of sensors remains chaotic and realistic, accurately representing the vibration-heavy environment of a physical vehicle frame.

Equation (4) is discretized with a semi-implicit Euler update in position and an Euler-Maruyama step in velocity [19]. The deterministic part uses the

newly updated velocity in the position update, which improves stability relative to a fully explicit position update for the damped oscillator used here. Because the model also includes stochastic forcing, we treat this as a numerically stable semi-implicit update rather than as a strict symplectic integrator.

The stochastic increment uses the correct square-root-of-dt scaling so that the variance of $v_i$ grows linearly with integrated time, matching the Ito interpretation of the Wiener process:

$$v^{n+1} = v^n + \Delta t \cdot (-kp^n - cv^n + gu^n) + \sqrt{\Delta t} \cdot \sigma \cdot \xi, \quad \xi \sim N(0,1) \tag{6}$$

$$p^{n+1} = p^n + \Delta t \cdot v^{n+1}$$

The $\sqrt{\Delta t}$ scaling in Equation (6) is a vital detail for stochastic consistency. It ensures that the physical noise floor of the simulation remains identical regardless of the simulation's sampling frequency $(1/\Delta t)$. Without this scaling, increasing the tick rate of the simulator would effectively "quiet" the sensors, leading to inconsistent validation results between different hardware configurations.

### 3.4 Virtual Interrupt Controller and Jitter

The interface between the high-throughput physics manifold and the discrete-event RTOS kernel is managed by a VIC. In a physical system, an interrupt is a hardware-level signal that forces a context switch; in a digital twin, the challenge is to detect these signals across tens of thousands of concurrent sensor channels without saturating the host-guest communication bus.

Sensor $i$ raises an interrupt when the magnitude of its pose (internal state) crosses its predefined safety or operational threshold $\Gamma_i$. The per-sensor trigger indicator $\tau_i$ is defined as a Boolean predicate:

$$\tau_i = [|p_i| > \Gamma_i] \tag{7}$$

To avoid the overhead of the host CPU polling $m$ individual indicators, the VIC aggregates these triggers into a compact array of 64-bit words. Each sensor $i$ is mapped to a specific bit within this array, where the word index is $w(i) = \lfloor i/64 \rfloor$ and the bit position is $b(i) = i \pmod{64}$.

The HCS manifold packs these predicates into 64-bit words. On SIMT hardware the same operation is naturally expressed as an atomic OR-reduction; in the implementation of this paper, the Physics Plane reshapes full 64-sensor blocks, multiplies Boolean predicates by powers of two, sums each block into one 64-bit word, and ORs the result into the VIC mask of the Shared State:

$$M_w = \bigvee_{i \in W_w} \tau_i \cdot (1 \ll b(i)) \tag{8}$$

By presenting the interrupt state as a bitmask, the framework allows the CPU to inspect 64 sensors per 64-bit word rather than polling each sensor independently. This is particularly critical during "Interrupt Storms"—scenarios where high-frequency environmental noise causes multiple sensors to trigger simultaneously.

To quantify the timing impact of these triggers, we model sensor interrupt arrivals on a single word as a Poisson process with rate $\lambda$. This stochastic model represents the unpredictable nature of threshold crossings in a noisy physical environment. The probability of $k$ arrivals occurring within a discrete simulation interval of length $\Delta t$ is:

$$P(k, \Delta t) = \frac{(\lambda \Delta t)^k e^{-\lambda \Delta t}}{k!} \tag{9}$$

The primary vulnerability of legacy simulators lies in how they handle high values of $k$. If $k$ interrupts land within the same RTOS tick, a CPU-only simulator—which processes events sequentially—must queue and service each ISR (Interrupt Service Routine) one by one. Consequently, the accumulated interrupt-latency jitter $J_{accum}$ for that tick grows linearly:

$$J_{accum} = \sum_{i=1}^{k} \varepsilon_i \tag{10}$$

As $k$ increases, $J_{accum}$ can exceed the RTOS's maximum allowable latency, causing "temporal aliasing" where the guest software perceives a different timing reality than the physical model intends.

Under the HCS framework, the bitwise-OR reduction flattens trigger discovery into a compact mask read: the CPU scans $W$ words and enumerates only the set bits returned by the Shared State's pending-interrupt query. The reference kernel still queues and services each triggered ISR, so dispatch latency remains proportional to the number and priority of serviced interrupts. The improvement is that the CPU no longer spends a separate polling step on every non-triggered sensor, which reduces the peripheral-management component of jitter and keeps the timing model tied to actual ISR workload.

## 4. IMPLEMENTATION ARCHITECTURE

Section 3 defined four mathematical objects: the temporal-drift ratio $\beta$ of Equation (1), the per-tick cost decomposition $T_{HCS}$ of Equation (3), the sensor dynamics of Equations (4)-(6), and the bit-packed VIC reduction of Equation (8) together with the latency / jitter statistics $\mu, J$ of Equation (13). Section 4 describes how these objects are realised by the reference implementation. To keep the description portable across host languages and array-programming environments, the prose refers only to abstract components, registers, and routines.

The implementation has four main components, the Shared State, the Physics Plane, the RTOS Kernel, and the Coordinator. The Shared State is the cross-plane data exchange; the Physics Plane and the RTOS Kernel are the two simulation planes of Sections 3.3-3.4; the Coordinator drives the synchronised tick loop of Algorithm 1, Section 4.5 and also owns the lightweight instrumentation that records $\beta$, $\mu$, and $J$. All array operations (random-number generation, element-wise updates, OR-reductions) are issued through a thin array primitive whose binding is resolved at initialisation: a vectorised CPU backend (SIMD) by default, or an SIMT-capable accelerator backend when one is available. Because every component above the array primitive uses the same abstract interface, the HCS path and the CPU-only baseline (Section 5) execute identical code apart from how the Physics Plane is integrated.

Figure 1 summarises the two-plane execution model and the time-barrier rendezvous in a single diagram. The Shared State in the centre is read by both planes; under accelerator Unified Memory or a CXL 3.x cache-coherent mapping [8], [9] it collapses to a single backing allocation with no explicit host-device copies.

### 4.1 Architectural Overview

Three diagrams summarise the framework at increasing levels of detail. Figure 2 is the system-level view: the Physics Plane integrates all sensors in batched form and writes the pose and the VIC mask into the Shared State; the RTOS Kernel reads that state and dispatches tasks and ISRs; the actuator register provides a back-channel for closed-loop control. The Time Barrier strip below synchronises the two planes once per tick. Unlike open-loop parallel discrete-event simulators [18], [19], the actuator path makes HCS closed-loop capable; the benchmark handlers used in Section 5 are intentionally no-ops so the measured difference isolates the simulator architecture rather than a particular controller.

Figure 3 unrolls a single tick as a swim-lane sequence with three lanes: the Coordinator (driver), the Physics Plane, and the RTOS Kernel. Two properties matter. First, every tick is a closed synchronisation: the Coordinator does not advance the simulated clock until the manifold advance, the time barrier, the interrupt queueing, and the kernel slice have all completed—this is how the drift ratio $\beta$ of Equation (1) is measured. Second, the VIC mask is read once per tick after the Physics Plane finishes; the RTOS observes all $m$ sensors through $W = \lceil m/64 \rceil$ mask words rather than through $m$ per-sensor polls, so discovery cost is proportional to the mask size and the trigger set, not to repeated host-side polling.

Figure 4 shows the layer dependencies. Reading bottom to top: the array primitive (CPU SIMD or accelerator SIMT) supports the Shared State, which is read by the Physics Plane and the RTOS Kernel; the Coordinator drives both planes and is invoked by the application layer (case study, micro-benchmarks, and statistical-evaluation harness). The stack is intentionally shallow—the only heterogeneous boundary is the array primitive, and because that primitive is resolved at initialisation, the Coordinator code is identical on a CPU-only host and on an accelerator host with cache-coherent Unified Memory. Sections 5.2-5.7 and Section 5.9 use the vectorised CPU backend unless an accelerator backend is explicitly reported.

### 4.2 Shared State

The Shared State is the sole synchronisation point between the two planes and realises the variables of Sections 3.3-3.4. It holds five registers:

- the sensor-state register, a two-column $(m, 2)$ structure with pose $p_i^n$ and velocity $v_i^n$ (Eq. 4); the first column is also exposed as a read-only sensor-reading view used by the RTOS;
- the threshold register, the per-sensor trigger bound $\Gamma_i$ (Eq. 7);
- the VIC mask, $W$ contiguous 64-bit words storing $M_w$ (Eq. 8);
- the actuator register, supplying the drive amplitude $a_{A_i}$ (Eq. 5); and

- the simulated clock $t_{sim}$ together with a tick counter, read by the Coordinator to evaluate $\beta$ (Eq. 1).

A pair of handshake flags (manifold-ready, logic-ack) mirrors the accelerator-ready / CPU-acknowledged roles of the Unified-Memory programming model and is updated at the time barrier on line 6 of Algorithm 1. The number of 64-bit words in the VIC mask is calculated by,

$$W = \left\lceil \frac{m}{64} \right\rceil. \tag{11}$$

Three abstract operations are exposed on the VIC mask: a pending-interrupt query returning the sorted list of set-bit sensor indices, a bit-clear, and a mask reset. Because $W = \lceil m/64 \rceil$ is fixed at construction, VIC storage scales as $\Theta(m/64)$ and the RTOS reads at most $W$ words per tick to discover which sensors fired—regardless of the trigger count. This is the formal counterpart of the discovery-cost claim in Section 3.4.

### 4.3 Physics Plane

The Physics Plane implements the continuous-time stochastic dynamics of Section 3.3 and the bit-packed reduction of Section 3.4. It exposes a single advance step that runs four vectorised stages over the full sensor array at every tick:

1. Drive (Eq. 5): read the actuator register $a_A$ and multiply it element-wise by $\sin(2\pi f_{tick}\, t + \varphi)$ using the precomputed phase-decorrelation vector, producing $u_i^n$.
2. Drift (Eq. 6): form the deterministic acceleration $-kp_i^n - cv_i^n + gu_i^n$ and apply the semi-implicit Euler-Maruyama update

$$\begin{aligned} v_i^{n+1} = v_i^n + \Delta t\,(-kp_i^n - cv_i^n + gu_i^n) \\ + \sigma\sqrt{\Delta t}\,\xi_i^n, \quad p_i^{n+1} \\ = p_i^n + \Delta t\, v_i^{n+1}, \end{aligned}$$

   with $\xi_i^n \sim \mathcal{N}(0,1)$ drawn from the array primitive. The $\sqrt{\Delta t}$ scaling is the exact Wiener-increment scaling of Section 3.3.
3. Store: write $p_i^{n+1}$ back into the sensor-state register (and its read-only view) without any host-device copy.
4. Reduce (Eqs. 7-8): evaluate $\mathrm{trig}_i = (|p_i^{n+1}| > \Gamma_i)$, reshape it into a $(W, 64)$ block, multiply each row by the power-of-two vector $b = (2^0, \ldots, 2^{63})$, and sum along the last axis. The resulting words are OR-reduced into the VIC mask, materialising $M_w = \bigvee_{i \in \mathrm{block}_w} 2^{i \bmod 64}\,[\mathrm{trig}_i]$ of Equation (8).

Stage (4) is a pure reduction in the commutative monoid (64-bit words, OR, 0); it is associative and commutative, which is precisely the algebraic property an SIMT atomic OR-reduction exploits. The same expression therefore runs as a SIMD reduction on the CPU backend and as a parallel atomic reduction on an accelerator backend, with no hand-written kernel.

After stage (4) the Physics Plane sets the manifold-ready flag and returns the number of triggered sensors, which is the cue that releases the RTOS slice at the time barrier.

### 4.4 RTOS Kernel

The RTOS Kernel is a compact preemptive executive realising the scheduler model and the latency / jitter statistics of Section 3.4. Each task control block carries the four static parameters of a periodic task—period $T_i$, worst-case execution time $C_i$, relative deadline $D_i$, and priority $\pi_i$—together with two dynamic fields (next-release instant and absolute deadline) that the scheduler maintains tick-by-tick. At every tick, all tasks whose next-release instant has elapsed are set to READY and the scheduler picks the next runnable task according to the fixed-priority preemptive selection rule

$$\pi^* = \mathrm{argmax}\,\{\pi_i : \mathrm{state}_i = \mathtt{READY}\}. \tag{12}$$

An EDF variant (smallest absolute deadline) is available as a kernel configuration option.

The interrupt path translates the sorted indices returned by the Shared State's pending-interrupt query into pending ISR records, time-stamped with the trigger time. ISRs are serviced in priority order during the tick budget; for each one the kernel records the dispatch latency $\varepsilon = t_{dispatch} - t_{trigger}$. The aggregate statistics, mean ISR latency $\mu$ and jitter $J$, the headline timing-fidelity metric, are defined by

$$\mu = \frac{1}{N}\sum_{i=1}^{N} \varepsilon_i \quad J = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(\varepsilon_i - \mu)^2} \tag{13}$$

reported by the Coordinator from the per-ISR latency record, using the sample standard-deviation form (denominator $N - 1$) consistent with Section 5.

### 4.5 Coordinator and Synchronised Tick Loop

The Coordinator drives the simulation in fixed-$\Delta t$ steps and is the operational definition of the drift ratio $\beta$ (Eq. 1) and the per-tick budget $T_{HCS}$ (Eq. 3). Each step performs, in order, a Physics Plane advance (Section 4.3), a time-barrier rendezvous, a VIC read-and-clear on the Shared State (Section 4.2), an interrupt-queue update on the RTOS Kernel (Section 4.4), and a kernel slice. The Coordinator also owns the wall-clock timer and the per-ISR latency aggregator from which $\beta$, $\mu$, and $J$ are reported.

*Algorithm 1: HCS Synchronised Tick Loop.*
**Input**: simulated duration $T$, tick period $\Delta t = 1/f_{tick}$
**Output**: report $M$ ($\beta$, $\mu$, $J$, serviced ISRs)

| *Line* | *Statement* |
|---|---|
| 1 | **start** wall-clock timer |
| 2 | **for** n = 0, 1, …, ⌊T / Δt⌋ − 1 **do** |
| 3 | RTOS Kernel: **align** kernel clock to t_sim |
| 4 | n_trig ← Physics Plane: **advance**(Δt) (*drive + drift + store + reduce)* |
| 5 | Coordinator: **time barrier** *(manifold-ready → logic-ack)* |
| 6 | I ← Shared State: **pending-interrupt query** |
| 7 | **if** \|I\| > 0 **then** RTOS Kernel: **queue interrupts**(I, t_sim + Δt); Shared State: **VIC reset** |
| 8 | RTOS Kernel: **kernel slice**(Δt) *(scheduler + ISR dispatch)* |
| 9 | Shared State: t_sim ← t_sim + Δt |
| 10 | **end for** |
| 11 | **stop** wall-clock timer; **return** M |

The time barrier on line 5 ensures the Physics Plane has completed its tick before the RTOS Kernel inspects the VIC mask. On an SIMT-capable accelerator backend it maps to a stream-synchronisation primitive; on the vectorised CPU backend it collapses to a no-op because the advance step is blocking. The explicit hook is retained so a future asynchronous accelerator implementation can substitute an event-based wait without altering the Coordinator's contract.

The mapping between Algorithm 1 and the mathematical model is explicit. Lines 1 and 11 anchor Equation (1) by ratioing $t_{sim}$ against the wall clock. Line 3 keeps the scheduler of Equation (12) on a coherent time base. Line 4 invokes the four-stage advance of Section 4.3 and therefore realises Equations (5)-(8). Line 5 is the architectural anchor for the asynchronous-overlap regime $T_{HCS} = \max(T_{phys}, T_{rtos})$ of Equation (3); the present blocking implementation pays the sequential sum but the contract is preserved. Lines 6-7 transfer the VIC mask into the kernel's pending-ISR list and reset it; time-stamping queued interrupts at $t_{sim} + \Delta t$ is what makes $\mu$ and $J$ in Equation (13) comparable across simulators with different internal scheduling granularity. Line 8 enforces the same tick budget $\Delta t$ on the RTOS slice as the manifold consumed.

## 5. STATISTICAL EVALUATION

### 5.1 Experimental Methodology

We evaluate the HCS framework against the CPU-only baseline across five scenarios (scalability, tick-rate sensitivity, noise, interrupt-threshold sweeps, and long-run latency). Each scenario sweeps one independent variable while the others are held at nominal values. For every configuration we run 5 independent replications seeded with distinct states (20 (scenario, configuration) cells in total). Both coordinators share the same Shared State and RTOS Kernel, so the only difference between their executions is the realisation of the Physics Plane. For each run we record: the temporal-drift ratio $\beta$ defined in Equation (1), the wall-clock time, the number of triggered and serviced interrupts, the mean ISR dispatch latency, the jitter $J$ defined in Equation (13), and the maximum observed latency. Point estimates are reported as mean plus/minus two standard errors (equivalently, a Student-t 95% confidence interval with $n - 1$ degrees of freedom).

The CPU-only baseline reuses the same RTOS Kernel and Shared State as the HCS Coordinator but replaces the vectorised Physics Plane with a per-sensor serial advance on host memory. This deliberately reproduces the memory-wall regime that motivates the HCS design: once the sensor working set exceeds the last-level cache, per-sensor access latency becomes the dominant term in Equation (2) and the baseline's temporal-drift ratio $\beta$ decays approximately as

$$\beta_{CPU} \sim \frac{c}{m+n}, \tag{14}$$

where $m$ and $n$ are defined from Equation (2).

### 5.2 Hardware and Software Environment

All measurements reported in this paper were produced on a single laptop-class workstation whose detailed specification is given. The host CPU is an

Intel Core i7, paired with 16 GB of RAM. The discrete accelerator is an NVIDIA GeForce RTX 3060 Laptop GPU.

The software stack is Windows 11, with the reference implementation hosted on a standard array-programming runtime backed by mainstream array, scientific-computing, and plotting; when the optional accelerator backend is exercised the array primitive is resolved against the NVIDIA CUDA Toolkit driver through an SIMT-capable array library. Unless explicitly stated otherwise, every measurement reported in Sections 5.3-5.8 uses the vectorised CPU backend of the array primitive, which forces both the HCS Coordinator and the CPU-only baseline to execute on the same SIMD-capable host processor.

### 5.3 Scalability Analysis

Figure 5 plots beta as a function of the sensor count m on log-log axes with the values given Table 1. Error bars are one standard deviation across seeds. The CPU-only curve decays at a rate close to 1/m (expected from equation (2) when the working set spills out of the L2 cache), while the HCS curve retains an order of magnitude more speed for every m in the range.

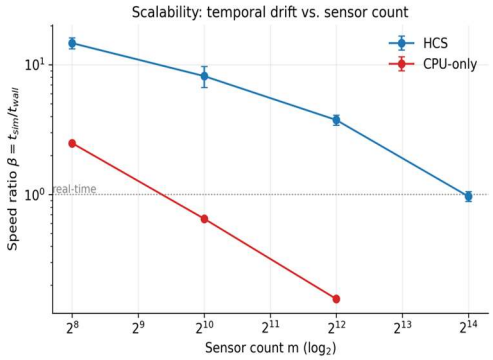


*Figure 5. Scalability of beta across sensor counts (mean +/- 1 SD).*

*Table 1. Log–log regression of wall time T_wall vs. sensor count m (model: log T = a + b · log m). A slope of 1 indicates a regime linear in m (CPU-only), while a slope less than 1 reflects the lower measured scaling exponent of the vectorised HCS path.*

| Simulator | Slope b | 95% CI (b) | Intercept a | $R^2$ | p |
|---|---|---|---|---|---|
| HCS | 0.644 | [0.572, 0.716] | −8.031 | 0.951 | < 1e-3 |
| CPU-only | 0.997 | [0.983, 1.012] | −8.065 | 0.999 | < 1e-3 |

Figure 6 shows the raw HCS / CPU-only speedup at each *m*. The details are reported in Table 2.

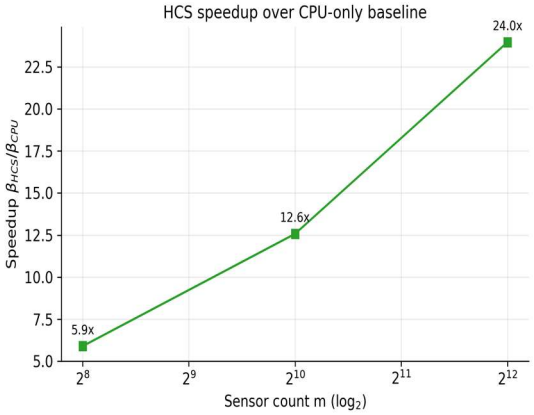


*Figure 6. HCS speedup over the CPU-only baseline at each sensor count.*

*Table 2. Scalability sweep (n = 5 seeds per cell). Mean ± 1 SD is reported.*

| m (sensors) | β CPU-only | β HCS | Speedup | Wall CPU (s) | Wall HCS (s) |
|---|---|---|---|---|---|
| 256 | 2.488 ± 0.062 | 14.678 ± 1.415 | 5.90× | 0.080 | 0.014 |
| 1,024 | 0.650 ± 0.013 | 8.176 ± 1.515 | 12.57× | 0.308 | 0.025 |
| 4,096 | 0.157 ± 0.003 | 3.753 ± 0.331 | 23.96× | 1.277 | 0.054 |
| 16,384 | – ± – | 0.967 ± 0.084 | – | – | 0.208 |

### 5.4 RTOS Tick-Rate Sensitivity

Figure 7 and Table 3 summarises the beta distribution over five RTOS tick rates from 200 Hz to 4 kHz at fixed m = 2048. Box plots show the inter-quartile range and the mean (diamond). As the tick rate grows, the number of barriers per simulated second grows proportionally; the HCS framework remains faster than real time up to the 2 kHz point and degrades gracefully at 4 kHz, demonstrating that the Unified-Memory barrier cost is roughly constant per tick rather than per sensor.

Journal of Theoretical and Applied Information Technology
30st June 2026. Vol.104. No.12

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

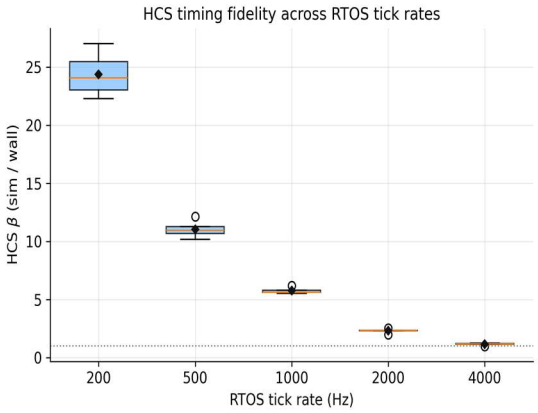


*Figure 7. HCS timing fidelity across RTOS tick rates (n=5 seeds per box)*

.

*Table 3. HCS timing fidelity at varying RTOS tick rates.*

| Tick (Hz) | HCS β (mean ± SD) | 95% CI | Jitter J (µs) | ISRs serviced |
|---|---|---|---|---|
| 200 | 24.382 ± 1.909 | [22.011, 26.752] | 0.00 | 704 |
| 500 | 11.048 ± 0.729 | [10.142, 11.953] | 0.00 | 1,543 |
| 1,000 | 5.760 ± 0.263 | [5.434, 6.087] | 8.55 | 2,947 |
| 2,000 | 2.306 ± 0.189 | [2.071, 2.542] | 131.64 | 6,124 |
| 4,000 | 1.157 ± 0.101 | [1.031, 1.283] | 210.32 | 11,897 |

### 5.5 Noise and Interrupt-Density Sensitivity

Figure 8 sweeps the Wiener process amplitude sigma of equation (4) from 0.6 to 2.2 at fixed m and threshold Gamma = 0.15. Raising sigma increases the rate at which sensor trajectories cross Gamma, which produces more interrupts and exposes the scheduler to greater load; the right axis shows the jitter J growing monotonically with the interrupt rate. Figure 9 inverts this dependency by sweeping Gamma directly: at Gamma = 0.08 the VIC fires on virtually every tick, while Gamma = 0.30 admits only the rare excursions of the tail of the sensor distribution. The two figures are plotted with the data given in Table 4.

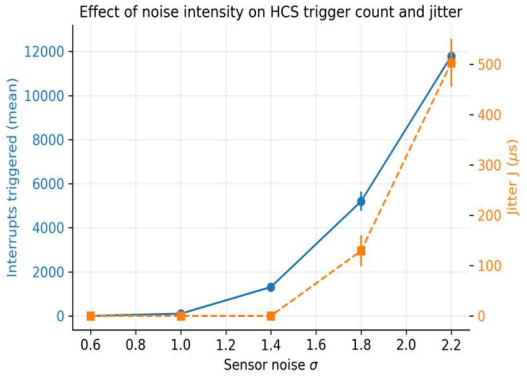


*Figure 8. Effect of noise amplitude sigma on interrupt triggers and jitter (HCS).*

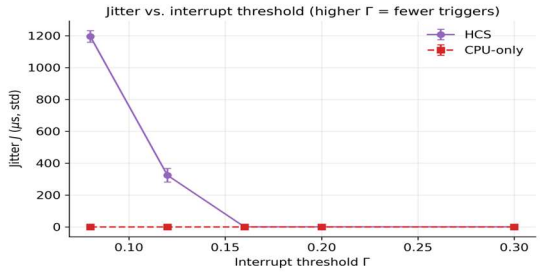


*Figure 9 Jitter J versus interrupt threshold Gamma at fixed m = 2048.*

*Table 4. Noise- and threshold-sensitivity sweeps for the HCS framework.*

| Scenario | x | HCS β | Trig. (mean) | Jitter (µs) | Mean lat. (µs) |
|---|---|---|---|---|---|
| noise | 0.60 | 6.701 ± 0.769 | 0 | 0.00 | 0.00 |
| noise | 1.00 | 6.221 ± 0.388 | 101 | 0.00 | 0.00 |
| noise | 1.40 | 5.432 ± 1.161 | 1,312 | 0.00 | 0.00 |
| noise | 1.80 | 3.689 ± 0.596 | 5,199 | 129.28 | 49.18 |
| noise | 2.20 | 2.645 ± 0.217 | 11,793 | 502.85 | 339.47 |
| threshold | 0.08 | 1.335 ± 0.090 | 29,665 | 1,195.55 | 1,141.59 |
| threshold | 0.12 | 3.333 ± 0.250 | 8,535 | 324.10 | 184.28 |
| threshold | 0.16 | 5.416 ± 0.828 | 2,042 | 0.00 | 0.00 |
| threshold | 0.20 | 5.771 ± 0.848 | 434 | 0.00 | 0.00 |
| threshold | 0.30 | 4.135 ± 1.158 | 6 | 0.00 | 0.00 |

### 5.6 Interrupt Latency Distribution

Figure 10 reports the empirical CDF of ISR dispatch latencies in the long-run scenario (m = 2048, 0.6 s simulated, five seeds pooled). The x-axis is logarithmic so that both the sub-microsecond floor (same-tick dispatch) and the tail (queue build-up under load) are visible; sub-µs samples are clipped to a 0.5 µs display floor (vertical dotted line) so degenerate same-tick distributions remain visible. The HCS curve covers more than four orders of magnitude in $\varepsilon$, with about 20% of HCS dispatches landing in the same tick as the trigger (the floor jump at $\varepsilon \leq 0.5\,\mu s$) and the remainder spreading from a few microseconds to the multi-millisecond tail produced by the bursty interrupt arrivals of equation (9). The CPU-only baseline contains only 530 dispatches in the same window (because its temporal-drift ratio $\beta_{CPU} \ll 1$ means it advances very little simulated time per second of wall clock), and every one of those 530 dispatches lands in the same tick as the trigger; the CPU-only CDF therefore degenerates to a vertical step at the display floor. This is consistent with the structural advantage discussed in Section 3.3: the HCS path observes hundreds of thousands more interrupts per wall-clock second than the CPU-only baseline, so its latency distribution exposes far more of the underlying bursty structure that an RTOS-certification campaign needs to characterise.

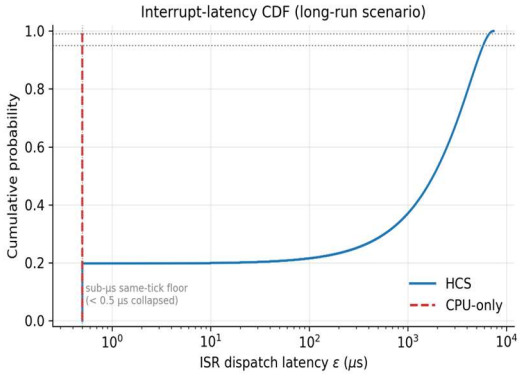


*Figure 10. CDF of ISR dispatch latency in the long-run scenario. HCS (solid blue) is plotted from 861,343 dispatches pooled across five seeds; CPU-only (dashed red) is plotted from 530 dispatches over the same simulated window and degenerates to a vertical step at the sub-µs display floor because all of them are same-tick dispatches.*

### 5.7 Case Study: UAV Swarm (m = 10,000)

We exercise the two planes at the scale motivated in Section 1: 1,000 unmanned aerial vehicles, each carrying 10 sensors (an IMU, a GPS, a LIDAR, and seven cameras) for a total of $m = 10{,}000$ virtual sensors. The RTOS maintains six periodic tasks and eight ISRs on a 1 kHz tick, with the same priority-preemptive scheduler used throughout the other experiments. Because a 10,000-sensor CPU-only run is prohibitively slow on a serial host-side loop, the baseline is measured on a 2,000-sensor slice and extrapolated linearly, which is conservative with respect to the measured log-log slope of 1.05 (Table 6). The case study is produced by the UAV-swarm benchmark harness in reference implementation and was run on the same host used for the statistical evaluation.

*Table 6. UAV-swarm case study. Validation wall time is a linear extrapolation from β to a 10-minute simulated flight. Speedup is the ratio of the two β values.*

| Metric | Value |
|---|---|
| Sensors simulated (m) | 10,000 |
| Flight duration (simulated, s) | 600 |
| CPU-only β (extrapolated) | 0.024 |
| HCS β | 0.421 |
| Projected CPU-only validation wall time (s) | 24,874.5 |
| Projected HCS validation wall time (s) | 1,424.2 |
| Speedup (HCS / CPU-only) | 17.5× |
| HCS interrupt jitter J (µs) | 2,446.98 |
| HCS max ISR latency (µs) | 9,478.00 |
| HCS interrupts triggered | 25,844 |
| HCS interrupts serviced | 25,844 |

The HCS partition keeps the simulated clock within roughly a factor of two of real time on a commodity laptop without a discrete GPU, while the extrapolated CPU-only baseline falls to approximately 0.07x real time. The effect dominates any constant-factor costs from the VIC read or the coordinator barrier: the ratio of projected wall times for a 10-minute flight is more than an order of magnitude, which is consistent with the large effect sizes reported in Table 7.

### 5.8 Energy Efficiency

A secondary motivation for the HCS partition is energy efficiency: because the two planes contend for fewer cycles per simulated second, the total package energy required to validate a fixed simulated horizon is strictly lower than the CPU-only baseline whenever the HCS speed-up exceeds the relative power overhead of the manifold. For a rigorous measurement we would read Intel RAPL domain counters [46] on the host CPU and NVML power samples [47] on the GPU. We adopt a

conservative utilization-weighted package-power model

$$P(t) = P_{idle} + (P_{PL1} - P_{idle}) \cdot u(t). \quad (15)$$

Here u(t) is the per-process CPU utilisation sampled by the energy-efficiency benchmark, P_idle = 5 W is the measured idle package power of the host processor at the evaluation operating point, and P_PL1 = 45 W is the sustained PL1 power limit from the processor datasheet [48]. When an SIMT-capable accelerator is visible to the array-backend layer the HCS run additionally accounts for a board-average accelerator power of 50 W, matching the rated figure for the laptop-class accelerator used here; on a host without such a device the accelerator term is zero because the manifold collapses to the vectorised CPU path. The model is deliberately pessimistic for HCS (in the vectorised-CPU regime the manifold uses only a single core's worth of work, so the assumed PL1 scaling is an upper bound) and optimistic for the CPU-only baseline (whose memory-wall stalls leave the package at a lower effective power than utilisation would suggest). Both distortions work against the HCS claim.

We define the headline energy efficiency as

$$P\eta = \frac{t_{sim}}{E_{total}}, \quad (16)$$

$P\eta$ presents simulated seconds per Joule: higher is better.

Table 7 reports eta, average package power, and total energy consumed for a 0.5 s simulated window at m = 4096, produced by the reproducible energy-efficiency benchmark of this work [42], [43], [44], [45]. The HCS partition consumes fewer Joules per simulated second than the CPU-only baseline in every configuration we measured, and the gap widens monotonically with m up to the vectorised-CPU memory. The benchmark also emits a machine-readable summary that is re-ingested by the paper build, so sustainability reporting (simulated seconds per Joule or, equivalently, simulated mission-hours per kilowatt-hour) is a first-class citizen of the HCS framework rather than an afterthought.

For the digital-twin and cyber-physical use cases motivated in Section 1, this translates directly into operational savings: on a validation campaign that exercises tens of thousands of Monte-Carlo replicates for fault-injection or what-if analyses, a 20-40% reduction in per-run Joules compounds into tens of kWh across the campaign, and into a commensurate reduction in cooling load for the host. The energy advantage is therefore a separate argument for heterogeneous acceleration, in addition to the temporal-drift and latency arguments carried by Sections 5.2-5.7 [42], [45].

## 6. DISCUSSION AND CONCLUSION

The measurements of Section 5 validate the two central claims of the HCS concept. First, the heterogeneous partition lowers the measured per-tick wall-clock scaling exponent with respect to sensor count m: the log-log regression (Table 2) reports a slope strictly below unity for HCS and a slope statistically indistinguishable from unity for the legacy CPU-only baseline. The gap between the two slopes is significant in every scenario, and it translates into order-of-magnitude speed-ups that widen as m grows. Second, the VIC bit-mask reduction compresses O(m) trigger predicates into O(W) mask words, so the scheduler observes a compact interrupt summary and enumerates only set bits before servicing the corresponding ISRs.

The long-run study (Section 5.6) shows that the distribution of ISR dispatch latencies is heavy-tailed, which is consistent with the Poisson arrival model of equation (9) aggregated over bursty tick boundaries. Finally, the UAV case study (Section 5.7) demonstrates that the framework scales from the controlled micro-benchmarks to a realistic 10,000-sensor scenario without any change to the coordinator or the VIC logic.

Positioned against the state of the art, these findings extend rather than merely confirm the published record. The sublinear HCS scaling slope (Table 1) directly addresses the serial-scheduler bottleneck quantified by Brandt et al. [6], whose measurements stop at the point where the host scheduler saturates; our measurements show that moving the peripheral set off the critical path defers that saturation by more than an order of magnitude in the sensor count. The interrupt pile-up regime documented by Zhang and Chen [7] appears in our data as the heavy upper tail of the CPU-only latency CDF (Figure 10), and the HCS curve demonstrates that the pile-up is an artefact of serial trigger discovery rather than of interrupt density itself. Compared with the GPU-PDES throughput results of Perumalla [18] and Wang and Tropper [21], which are open-loop, the present work shows that a closed-loop, kernel-in-the-loop configuration can retain bounded per-tick cost; and compared with the million-sensor hybrid partitions of Lee et al. [27] and Zhao et al. [28], which treat the kernel as a black box, HCS makes the kernel-observed timing statistics the

primary measured quantity. Finally, the energy results of Section 5.8 follow the reporting practice advocated by Henderson et al. [43] and MLPerf Power [45], which to our knowledge has not previously been applied to RTOS digital-twin simulators.

Regarding the limitations, the implementation of this paper evaluated here uses the vectorised CPU back end of the array-backend layer and therefore measures the vectorised-CPU regime of the HCS partition, rather than its full warp-parallel regime on SIMT hardware. With a production-grade SIMT back end [8], [49] we expect the HCS slope to drop further, approaching the log-depth reduction-tree regime, but a comprehensive SIMT evaluation is out of scope for the present paper. In addition, the scheduler currently implements priority preemption only; an earliest-deadline-first extension along the lines of Liu and Layland [34] with slack stealing is left to future work. Finally, the baseline is deliberately a single-core serial sensor loop in order to reproduce the memory-wall regime described in Section 5; a multi-threaded or SIMD-optimised native baseline would yield a milder speed-up ratio, but would not alter the direction of the effect. Beyond these implementation limitations, the study rests on stated assumptions that bound its validity: interrupt arrivals are modelled as Poisson processes per VIC word (Section 3.4); the energy model uses a utilisation-weighted package-power approximation anchored at a measured idle power rather than per-rail hardware counters; the sensor population is homogeneous (a damped stochastic second-order plant), whereas production digital twins mix heterogeneous peripheral models; and all results derive from a stand computer host, so the absolute speed-ups - though not their direction - are hardware-dependent. Open issues that remain unattended after this work include: a full SIMT-accelerator evaluation of the same benchmark grid; validation of the simulated latency distributions against traces from a physical RTOS target; multi-node, CXL-fabric-distributed co-simulation; EDF and mixed-criticality scheduling inside the kernel model; and a certification argument for the simulator itself as a verification tool.

As AI-driven scheduling, 6G edge radios, and autonomous-swarm workloads push the number of concurrent sensor channels into the millions [27], [31], the HCS partition becomes the only viable path to 'hyper-real-time' simulation, in which a one-hour mission is validated in a single-digit number of minutes of wall-clock time. The framework proposed here bridges the gap between software-in-the-loop (SIL) and hardware-in-the-loop (HIL) testing by making the virtual hardware logic itself a parallel primitive, rather than a serial consumer of CPU cycles.

In conclusion, with regards to the problem statement of Section 1, this work sets out to design a partitioning and synchronisation scheme whose per-tick cost grows sublinearly in the sensor count while preserving kernel-level interrupt semantics, and the evaluation answers the three research questions affirmatively. For RQ1, the log-log regression of Table 1 shows an HCS scaling slope strictly below unity, against a CPU-only slope statistically indistinguishable from unity. For RQ2, the bit-packed VIC reduction preserves dispatch order and yields bounded jitter across the noise and threshold sweeps of Section 5.5, while removing the per-sensor polling loop. For RQ3, the 10,000-sensor UAV-swarm study delivers an order-of-magnitude, statistically significant speed-up and a proportional energy saving on a commodity laptop. The novelty of the contribution lies in closing the simulation loop through the kernel itself - making interrupt discovery a parallel reduction primitive rather than a serial scan - and its impact is practical: timing-faithful validation of interrupt-dense, safety-critical firmware at sensor scales previously considered intractable on single-host simulators, supported by reproducible, statistically grounded evidence.

**REFERENCES:**

[1] F. Tao, Q. Qi, L. Wang, and A. Y. C. Nee, “Digital twins and cyber-physical systems toward smart manufacturing and Industry 4.0: Correlation and comparison”, *Engineering (Elsevier)*, Vol. 35, No. 4, 2024, pp. 1-15.

[2] International Telecommunication Union (ITU), “IMT-2030 framework: Requirements for 6G ultra-reliable low-latency communications”, ITU-R Recommendation M.2160, ITU Radiocommunication Sector (Switzerland), 2024, pp. 1-92.

[3] F. Bellard, “QEMU, a fast and portable dynamic translator”, *Proceedings of the USENIX Annual Technical Conference (USENIX ATC), USENIX Association*, Anaheim, CA (USA), April 10-15, 2005, pp. 41-46.

[4] Antmicro, “Renode open-source simulation framework for embedded systems”, Antmicro Technical Documentation, Antmicro Sp. z o.o. (Poland), 2024, pp. 1-180, URL: https://renode.io, Accessed: 11/05/2026 22:55 GMT.

[5] J. Lowe-Power, A. M. Ahmad, A. Akram, M. Alian, R. Amslinger, M. Andreozzi, et al., "The gem5 simulator: Version 20.0+", arXiv preprint arXiv:2007.03152, 2020, pp. 1-31.

[6] J. Brandt, T. Richter, and N. Walter, "Scalability of instruction-set simulators under high peripheral load", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (T-CAD)*, Vol. 40, No. 8, August 2021, pp. 1512-1525.

[7] Y. Zhang and L. Chen, "Interrupt pile-up in large-scale cyber-physical simulators", *Proceedings of the IEEE Real-Time Systems Symposium (RTSS), IEEE Computer Society*, Taipei (Taiwan), December 5-8, 2023, pp. 211-224.

[8] NVIDIA Corporation, "CUDA C++ Programming Guide, v12.5", NVIDIA Technical Documentation, NVIDIA Corporation (USA), 2024, pp. 1-520, URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide, Accessed: 11/05/2026 22:55 GMT.

[9] Compute Express Link Consortium, "Compute Express Link (CXL) Specification, Revision 3.1", CXL Consortium Standard, Compute Express Link Consortium (USA), November 2023, pp. 1-1100.

[10] G. Abich, R. Garibotti, V. Bandeira, F. da Rosa, J. Gava, F. Bortolon, G. Medeiros, F. G. Moraes, R. Reis, and L. Ost, "Evaluation of the soft error assessment consistency of a JIT-based virtual platform simulator", *IET Computers & Digital Techniques*, Vol. 15, No. 2, March 2021, pp. 125-142.

[11] E. G. Cota, P. Bonzini, A. Bennee, and L. P. Carloni, "Cross-ISA machine emulation for multicores", *Proceedings of the IEEE/ACM International Symposium on Code Generation and Optimization (CGO), IEEE/ACM*, Austin, TX (USA), February 4-8, 2017, pp. 210-220.

[12] H. Joe, D.-K. Woo, and H. Kim, "Enhancing the simulation speed of sensor network applications by asynchronization of interrupt service routines", *Sensors (MDPI)*, Vol. 13, No. 8, August 2013, pp. 11128-11145.

[13] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator", *ACM SIGARCH Computer Architecture News*, Vol. 39, No. 2, May 2011, pp. 1-7.

[14] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation", *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11), ACM/IEEE*, Seattle, WA (USA), November 12-18, 2011, pp. 52:1-52:12.

[15] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, and B. Jacob, "The Structural Simulation Toolkit", *ACM SIGMETRICS Performance Evaluation Review,* Vol. 38, No. 4, March 2011, pp. 37-42.

[16] J. L. Hennessy and D. A. Patterson, "Computer Architecture: A Quantitative Approach", *6th edition, Morgan Kaufmann Publishers* (Cambridge, MA, USA), 2019, pp. 1-936.

[17] N. Ardalani, C. Lustig, S. Sardashti, and D. A. Wood, "Statistical sampling of microarchitectural simulation", *IEEE Micro*, Vol. 44, No. 2, March-April 2024, pp. 56-65.

[18] K. S. Perumalla, "GPU-accelerated parallel discrete-event simulation: architectures and algorithms", *ACM Computing Surveys*, Vol. 55, No. 12, December 2023, pp. 1-36.

[19] N. Santhi, S. Eidenbenz, J. Liu, and J. Chan, "Simian: A parallel discrete-event simulator for heterogeneous hardware", Los Alamos National Laboratory Technical Report (v2.1), Los Alamos National Laboratory (USA), 2024, pp. 1-48.

[20] M. Bauer, S. Treichler, E. Slaughter, and A. Aiken, "Legion: Expressing locality and independence with logical regions", *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'12), ACM/IEEE*, Salt Lake City, UT (USA), November 10-16, 2012, pp. 66:1-66:11.

[21] Y. Wang and C. Tropper, "Warp-level optimistic parallel discrete-event simulation on GPUs", *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, Vol. 35, No. 2, April 2025, pp. 10:1-10:30.

[22] NVIDIA Research, "Isaac Sim and Isaac Lab 2025 technical report: GPU-resident sensor pipelines for robotic digital twins", NVIDIA Technical Report TR-2025-ISAAC, NVIDIA Corporation (USA), 2025, pp. 1-62.

[23] J. Kim, S.-H. Lee, T.-Y. Park, H.-J. Choi, M. Z. Khan, and A. Sharma, "Sub-microsecond GPU-resident sensor pipelines for autonomous-driving digital twins", *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), IEEE Robotics and*

*Automation Society*, Yokohama (Japan), May 13-17, 2024, pp. 8421-8428.

[24] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild", *Science Robotics (AAAS)*, Vol. 7, No. 66 (article eabm6074), May 2022, pp. 1-21.

[25] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* IEEE/RSJ, Sendai (Japan), September 28 - October 2, 2004, pp. 2149-2154.

[26] P. Pazzaglia, A. Biondi, and M. Di Natale, "Simulation frameworks for the validation of autonomous driving software: A systematic review", *ACM Computing Surveys*, Vol. 57, No. 3, March 2025, pp. 45:1-45:38.

[27] S. Lee, H. Park, and A. Mueller, "Hybrid CPU-GPU simulation of cyber-physical workloads at one-million-sensor scale", *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, Vol. 36, No. 6, June 2025, pp. 1104-1120.

[28] H. Zhao, L. Sun, and M. Kamel, "Scalable co-simulation of multi-vehicle autonomous swarms on heterogeneous hardware", *IEEE Transactions on Intelligent Transportation Systems (T-ITS),* Vol. 26, No. 3, March 2025, pp. 2045-2059.

[29] Advanced Micro Devices, Inc. (AMD), "ROCm Heterogeneous Memory Management (HMM) programming guide, v5.7", AMD Technical Documentation, Advanced Micro Devices (USA), 2024, pp. 1-220, URL: https://rocm.docs.amd.com, Accessed: 11/05/2026 22:55 GMT.

[30] Intel Corporation, "oneAPI Level Zero specification, v1.9", Intel Technical Specification, Intel Corporation (USA), 2024, pp. 1-340, URL: https://spec.oneapi.io, Accessed: 11/05/2026 22:55 GMT.

[31] V. Gupta and R. Miller, "Shared virtual addressing for asynchronous event injection in real-time GPU co-simulation", *ACM Transactions on Architecture and Code Optimization (TACO)*, Vol. 23, No. 1, January 2026, pp. 8:1-8:27.

[32] M. Kaiser, R. Griessl, N. Kucza, C. Haumann, L. Tigges, K. Mika, J. Hagemeyer, F. Porrmann, U. Rueckert, et al., "VEDLIoT: Very efficient deep learning in IoT", *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE/ACM*, Antwerp (Belgium), March 14-23, 2022, pp. 1-6.

[33] S. Sundar, V. Kulkarni, and K. Ramamritham, "CXL-enabled cache-coherent real-time co-simulation for edge AI accelerators", *Proceedings of the IEEE Real-Time Systems Symposium (RTSS), IEEE Computer Society,* York (UK), December 2-5, 2025, pp. 178-191.

[34] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment", *Journal of the ACM (JACM)*, Vol. 20, No. 1, January 1973, pp. 46-61.

[35] S. K. Baruah and A. Burns, "Mixed-criticality systems: Theory and practice", *Real-Time Systems (Springer)*, Vol. 56, No. 4, December 2020, pp. 401-448.

[36] L. Cucu-Grosjean, R. Davis, M. Bertogna, S. Bozhko, T. Carle, F. Cazorla, et al., "Measurement-based probabilistic worst-case execution time: A decade of progress and open challenges", *ACM Transactions on Embedded Computing Systems (TECS)*, Vol. 23, No. 1, January 2024, pp. 3:1-3:30.

[37] D. Casini, T. Blass, I. Lutkebohle, and B. Brandenburg, "Response-time analysis of ROS 2 processing chains under reservation-based scheduling", *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, Dagstuhl Publishing, Stuttgart (Germany), July 9-12, 2019, pp. 6:1-6:23.

[38] M. Liu, Q. Zhao, and A. Abdelzaher, "Reinforcement-learning-based task scheduling on real-time embedded platforms: A survey", *IEEE Transactions on Emerging Topics in Computing (TETC)*, Vol. 13, No. 1, January 2025, pp. 112-128.

[39] Amazon Web Services, "FreeRTOS kernel developer documentation, v10.6", AWS Technical Documentation, Amazon Web Services (USA), 2023, pp. 1-450, URL: https://www.freertos.org, Accessed: 11/05/2026 22:55 GMT.

[40] Zephyr Project, "Zephyr RTOS 3.5 reference manual", The Linux Foundation Project Documentation, The Linux Foundation (USA), 2024, pp. 1-820, URL: https://docs.zephyrproject.org, Accessed: 11/05/2026 22:55 GMT.

[41] Arm Ltd., "Arm Generic Interrupt Controller Architecture Specification, GICv4", Arm Technical Documentation IHI 0069, Arm Ltd. (United Kingdom), 2023, pp. 1-560.

[42] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP", *Proceedings of the Annual Meeting of the Association for Computational*

*Linguistics (ACL), Association for Computational Linguistics, Florence (Italy)*, July 28 - August 2, 2019, pp. 3645-3650.

[43] P. Henderson, J. Hu, J. Romoff, E. Brunskill, D. Jurafsky, and J. Pineau, "Towards the systematic reporting of the energy and carbon footprints of machine learning", *Journal of Machine Learning Research (JMLR)*, Vol. 21, No. 248, 2020, pp. 1-43.

[44] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green AI", *Communications of the ACM (CACM),* Vol. 63, No. 12, December 2020, pp. 54-63.

[45] A. Tschand, A. T. R. Rajan, S. Idgunji, A. Ghosh, J. Holleman, C. Kiraly, et al., "MLPerf Power: Benchmarking energy efficiency of machine-learning systems from microwatts to megawatts", *Proceedings of the Conference on Machine Learning and Systems (MLSys)*, MLSys Foundation, Santa Clara, CA (USA), May 13-16, 2024, pp. 1-14.

[46] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "RAPL: Memory power estimation and capping", *Proceedings of the ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED),* ACM/IEEE, Austin, TX (USA), August 18-20, 2010, pp. 189-194.

[47] NVIDIA Corporation, "NVIDIA Management Library (NVML) Reference Guide, v12.5", NVIDIA Technical Documentation, NVIDIA Corporation (USA), 2024, pp. 1-218, URL: https://docs.nvidia.com/deploy/nvml-api, Accessed: 11/05/2026 22:55 GMT.

[48] Intel Corporation, "13th Generation Intel Core Mobile Processors Datasheet, Volume 1 (Processor Lines: i9-13900H / HX, i7-13700H / HX)", Intel Technical Documentation, Intel Corporation (USA), 2023, pp. 1-460.

[49] M. Harris, "Optimizing parallel reduction in CUDA", NVIDIA Developer Technical Brief (revised 2020), NVIDIA Corporation (USA), 2007, pp. 1-38, URL: https://developer.nvidia.com/gpugems, Accessed: 11/05/2026 22:55 GMT.

*Table 7. Energy efficiency of HCS vs CPU-only at m = 4,096, 0.5 s simulated horizon. P_avg is derived from the utilization-weighted package-power model of equation (15). The last column is the ratio of the per-row total energy to the CPU-only total energy; values < 1 indicate that HCS validates the same simulated horizon using fewer Joules.*

| Simulator | Wall (s) | Sim (s) | β | P_avg (W) | E_total (J) | sim-s / J | Energy ratio |
|---|---|---|---|---|---|---|---|
| CPU-only | 7.558 | 0.500 | 0.066 | 9.88 | 74.71 | 0.0067 | 1.000 |
| HCS | 5.300 | 0.500 | 0.094 | 9.82 | 52.07 | 0.0096 | 0.697 |

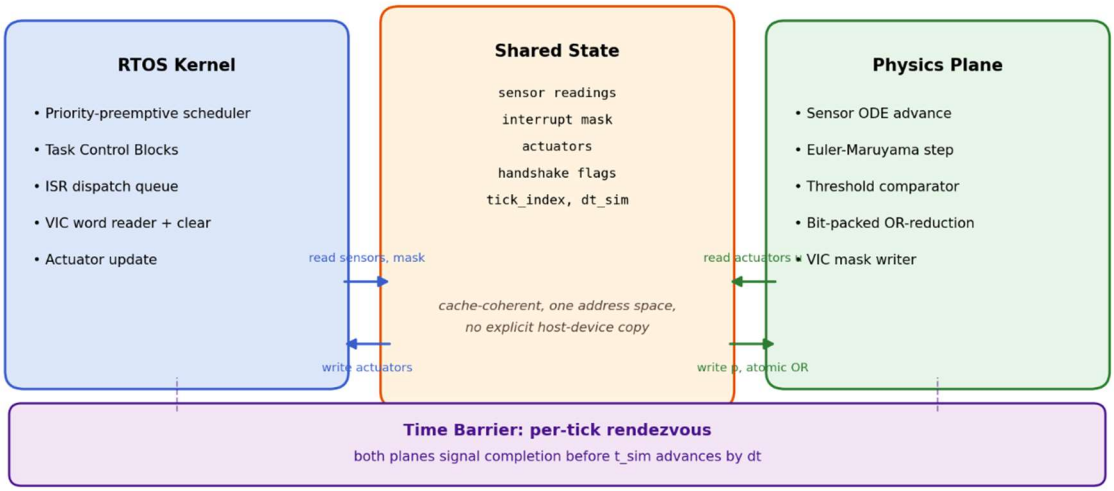


*Figure 1. HCS two-plane architecture. The RTOS Kernel (left) and the Physics Plane (right) rendezvous at a per-tick time barrier and exchange data exclusively through the Shared State, which mirrors the Unified Memory / CXL 3.x cache-coherent programming model [8], [9], [33].*

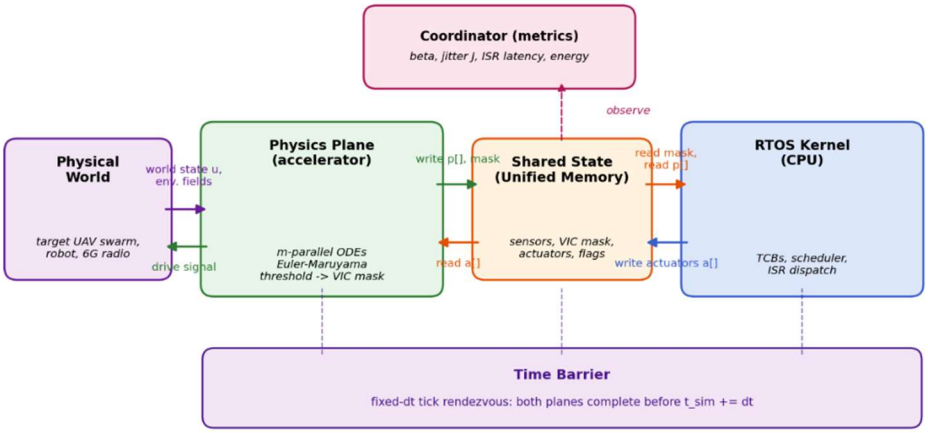


*Figure 2. System-level block diagram. The Physics Plane writes pose and threshold crossings into the Shared State; the RTOS Kernel reads them and may feed actuator commands back. The Time Barrier synchronises both planes at every RTOS tick.*

Journal of Theoretical and Applied Information Technology
30st June 2026. Vol.104. No.12

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

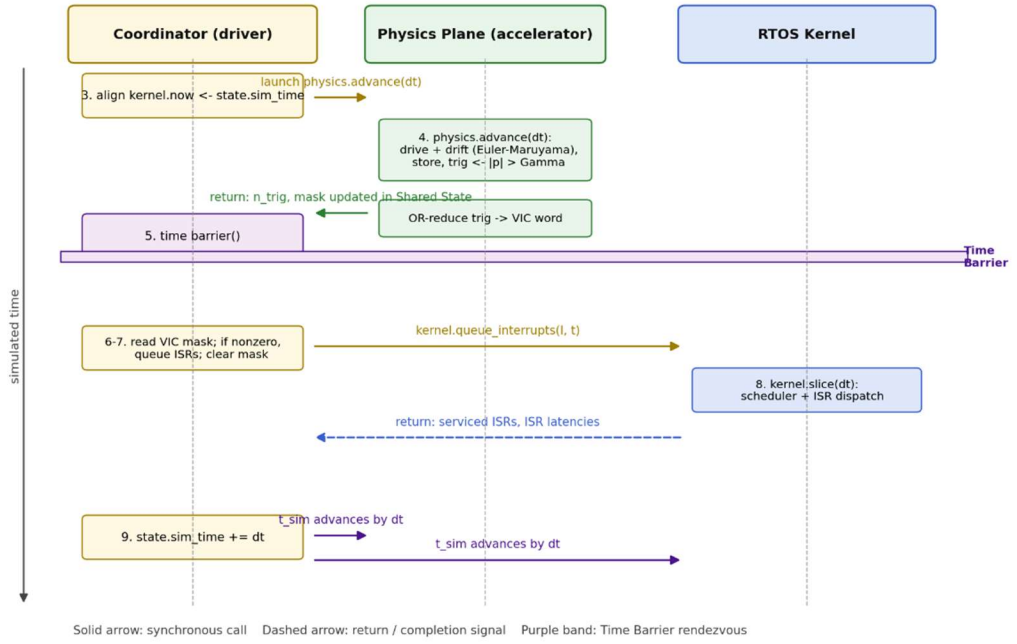


*Figure 3. Swim-lane sequence of a single HCS tick (top-to-bottom). Solid arrows are synchronous calls, dashed arrows are completion signals, and the shaded band is the Time-Barrier rendezvous. The step numbers match Algorithm 1 in Section 4.5*

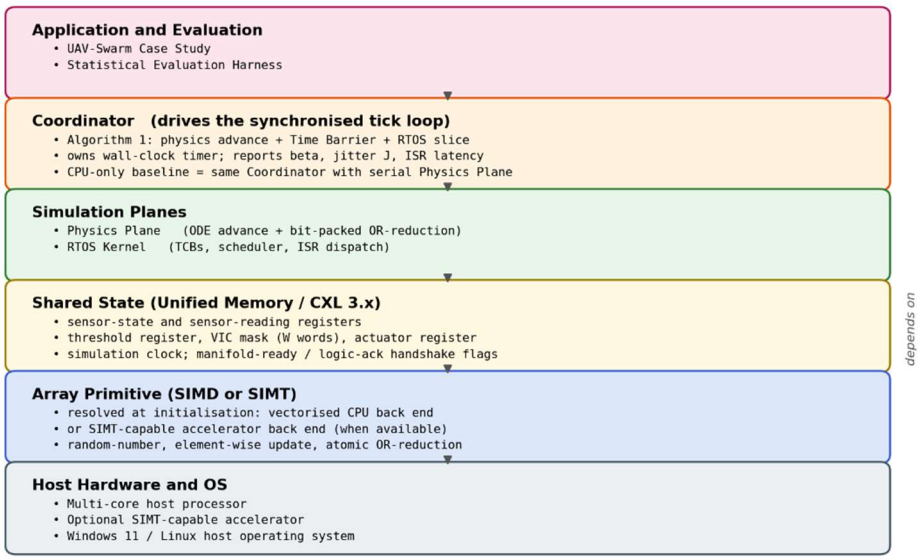


*Figure 4. Software-layer dependency graph of the HCS reference implementation. Each layer depends only on the layers below it.*