RingKernel

GPU-Native Persistent Actor Model for Rust

Lock-Free Message Passing • Hybrid Logical Clocks • Multi-Backend

Rust core • Python wrapper • CUDA • WebGPU • Metal • CPU

Executive Overview

Version 0.4.0

A framework for treating GPU compute units as persistent actors
with causal ordering, zero-copy messaging, and enterprise-grade tooling.

Built with Rust & Python • 20+ modular crates • 950+ tests

Open-source (Apache-2.0) • Published on crates.io & PyPI

Contact: michael.ivertowski@ch.ey.com

Contents

1 Executive Summary

RingKernel is a GPU-native persistent actor model framework for Rust. It enables GPU-accelerated actor systems where compute kernels persist for the lifetime of a simulation, exchanging messages through lock-free ring buffers and maintaining causal ordering via hybrid logical clocks (HLC). Written entirely in Rust for memory safety and performance, RingKernel provides a unified programming model across CUDA, WebGPU, Metal, and CPU backends.

Why RingKernel?

- **Persistent GPU actors** — kernels remain resident on the GPU, eliminating per-command launch overhead (11,327× faster command injection vs. traditional kernel dispatch).
- **Zero-copy message passing** — rkyv-based serialization enables host↔GPU and kernel-to-kernel (K2K) communication without intermediate copies.
- **Causal ordering** — hybrid logical clocks provide happened-before semantics across heterogeneous compute units.
- **Multi-backend** — write once, run on CUDA, WebGPU, Metal, or CPU. Rust DSL transpilers generate native GPU code (CUDA C, WGSL).
- **Enterprise-ready** — built-in observability, circuit breakers, multi-GPU coordination, TLS, RBAC, and rate limiting.

1.1 At a Glance

93B elem/sec CUDA throughput	**11,327×** faster command injection	**280×** GPU stencil speedup
20+ modular Rust crates	**950+** unit & integration tests	**4** GPU backends

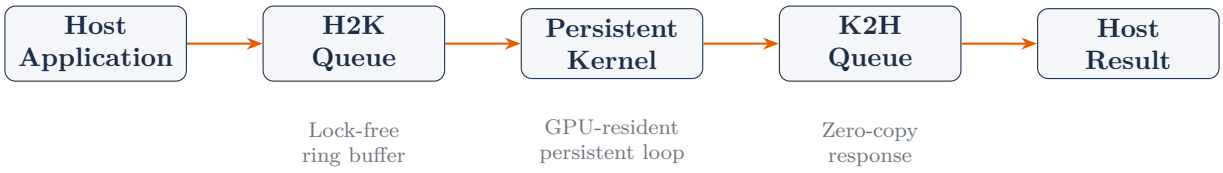
2 Core Abstractions

2.1 Persistent Actor Model

Traditional GPU programming launches kernels, waits for results, and repeats. RingKernel keeps actors *resident* on the GPU, processing messages in a persistent loop. This eliminates kernel launch overhead and enables real-time interactive workloads.

Abstraction	Description
RingMessage	Trait for GPU-transferable messages using rkyv zero-copy serialization with automatic type ID generation and correlation tracking.
MessageQueue	Lock-free ring buffer (power-of-2 capacity) for host↔GPU and kernel-to-kernel message passing with atomic head/tail pointers.
ControlBlock	128-byte GPU-resident structure managing kernel lifecycle state, activation, and termination signals.
HlcTimestamp	Hybrid logical clock combining physical time with logical counters for causal ordering across heterogeneous compute units.
K2K Broker	Kernel-to-kernel direct messaging broker with endpoint registration and topic-based publish/subscribe with wildcard matching.
RingContext	GPU intrinsics facade passed to kernel handlers, providing thread indexing, synchronization, shared memory, and warp operations.

2.2 Message Flow



3 GPU Backends & Code Generation

3.1 Multi-Backend Architecture

RingKernel supports four backends through the `RingKernelRuntime` trait. Selection is automatic or explicit via feature flags:

Backend	Feature	Capabilities
CUDA	`cuda`	Full persistent actors, cooperative groups, multi-stream execution, PTX caching, GPU memory pool, NVTX profiling.
WebGPU	`wgpu`	Cross-platform (Vulkan, Metal, DX12), host-driven dispatch loop, compute shader generation via WGSL transpiler.
Metal	`metal`	macOS/iOS scaffold with runtime, buffer management, and pipeline infrastructure.
CPU	`cpu`	Always available. Tokio-based async runtime for testing, development, and fallback execution.

3.2 Rust-to-GPU Transpilers

Write GPU kernels in a Rust DSL and transpile to native GPU code. The transpilers support four kernel types:

Kernel Type	Targets	Description
Global Kernels	CUDA, WGSL	Generic GPU compute (SAXPY, reductions, transforms).
Stencil Kernels	CUDA, WGSL	Grid-based operations with `GridPos` abstraction for 2D/3D neighbor access, shared memory tiling, and halo exchange.
Ring Kernels	CUDA	Persistent actor kernels with envelope-based messaging, HLC clocks, K2K routing, and ControlBlock lifecycle management.
Persistent FDTD	CUDA	Truly persistent 3D simulation kernels with cooperative groups, H2K/K2H queues, and tile-based halo exchange.

The DSL provides 120+ GPU intrinsics across 13 categories (synchronization, atomics, math, trigonometry, warp operations, shared memory, and more). The `ringkernel-ir` crate provides a unified intermediate representation that lowers to CUDA, WGSL, or MSL.

4 Performance

4.1 Throughput Benchmarks (RTX Ada)

Benchmark	Throughput	Batch Time	vs. CPU
CUDA Codegen (1M floats)	93B elem/sec	0.5 μs	12,378×
CUDA SAXPY PTX	77B elem/sec	0.6 μs	10,258×
GPU Stencil (64^3)	78,046 Mcells/s	—	280.6×
GPU Persistent	18.2 Mcells/s	—	1.2×
CPU Baseline (Rayon)	278 Mcells/s	—	1.0×

4.2 Persistent vs. Traditional Kernels

The persistent actor model excels at **interactive command latency**:

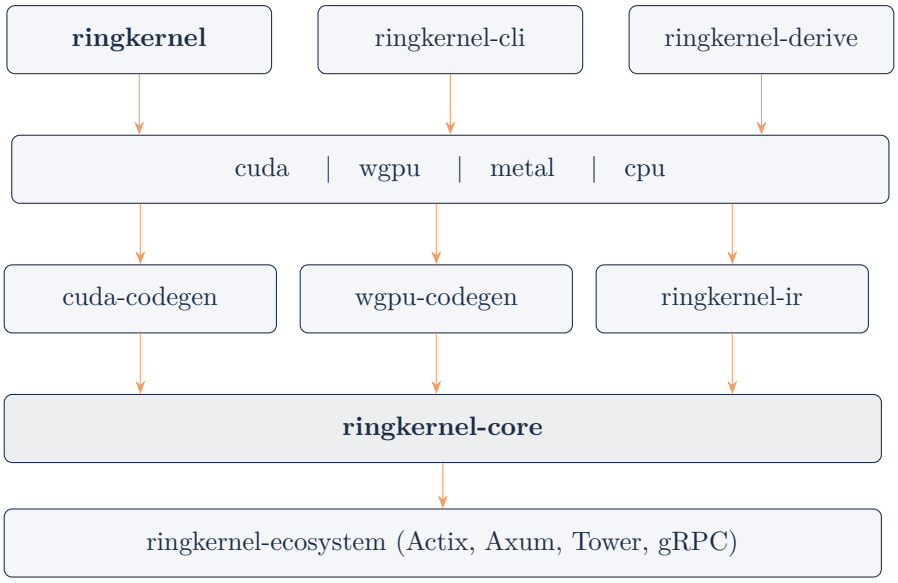
Operation	Traditional	Persistent	Winner
Inject (send command)	317 μs	0.03 μs	**Persistent 11,327×**
Query (read state)	0.01 μs	0.01 μs	Tie
Single step (compute)	3.2 μs	163 μs	Traditional 51×
Mixed workload	40.5 ms	15.3 ms	**Persistent 2.7×**

When to Use Each Approach

- **Batch compute** (1000s of steps) — Traditional kernels for maximum per-step throughput.
- **Interactive commands** — Persistent actors for 11,327× faster command injection via mapped memory.
- **Real-time GUI** (60 FPS) — Persistent actors deliver 2.7× more operations per 16 ms frame budget.
- **Dynamic topology** — Persistent actors avoid kernel relaunch when the actor graph changes.

5 Architecture

5.1 Layered Crate Architecture



5.2 Enterprise Infrastructure

Capability	Details
Health & Resilience	Liveness/readiness probes, circuit breakers with automatic recovery, graceful degradation (Normal → Critical), kernel watchdog with heartbeat monitoring.
Observability	Prometheus metrics export, OpenTelemetry OTLP tracing (Jaeger, Datadog), structured logging with trace correlation, alert routing with deduplication.
Security	AES-256-GCM and ChaCha20-Poly1305 encryption, K2K message encryption with forward secrecy, TLS/mTLS via rustls, API key and JWT authentication, RBAC with deny-by-default policy.
Multi-GPU	Device selection with load balancing, live kernel migration between GPUs via checkpoints, NVLink/PCIe topology discovery.
Resource Control	Memory limit enforcement with RAII reservations, per-tenant quotas, token-bucket and sliding-window rate limiting, adaptive CPU/GPU workload routing.
Lifecycle	Full state machine (Initializing → Running → Draining → Stopped), graceful shutdown with statistics report, operation timeouts with deadline propagation.

6 Applications & Ecosystem

6.1 Showcase Applications

RingKernel ships with full showcase applications demonstrating the persistent actor model in practice:

Application	Description
WaveSim 2D	Acoustic wave simulation with GPU-accelerated FDTD, tile-based ring kernel actors, and educational modes (sequential, vector, actor-based).
WaveSim 3D	3D acoustic simulation with binaural audio, persistent GPU actors using cooperative groups, volumetric ray marching, and K2K halo exchange.
TxMon	GPU-accelerated transaction monitoring with real-time fraud detection GUI, stencil kernels for batch processing, and ring kernel backends.
ProcInt	Process intelligence with GPU-accelerated DFG mining, pattern detection, conformance checking, and partial order analysis.
AccNet	Accounting network visualization with chart-of-accounts generation, industry templates, and GAAP compliance checking.
Audio FFT	GPU-accelerated audio FFT processing with resampling support.
Graph Library	CSR matrix, BFS, SCC (Tarjan/Kosaraju), Union-Find, SpMV, power iteration.
Monte Carlo	Philox RNG, antithetic variates, control variates, importance sampling.

6.2 Web Framework Integrations

RingKernel integrates with popular Rust web frameworks, exposing persistent GPU actors as network services:

Framework	Integration
Actix	`GpuPersistentActor` wraps a persistent kernel handle as an Actix actor with typed message handlers.
Axum	`PersistentGpuState` provides REST and Server-Sent Events (SSE) endpoints with shared GPU state.
Tower	`PersistentKernelService` exposes persistent kernels as Tower services with middleware support.
gRPC	Tonic-based streaming RPCs for remote kernel control and observation.
ML Bridges	Integration points for machine learning frameworks via the ecosystem crate's bridge traits.

6.3 Python Wrapper

The `ringkernel` Python package (PyO3 + maturin) provides the full runtime, messaging, and GPU management API to Python 3.8+ with both async and synchronous interfaces:

Module	Capabilities
`ringkernel.core`	`RingKernel` runtime, `KernelHandle` lifecycle, `LaunchOptions` configuration. Async and sync APIs.
`ringkernel.hlc`	`HlcTimestamp` and `HlcClock` for distributed causal ordering.
`ringkernel.k2k`	`K2KBroker` and `K2KConfig` for kernel-to-kernel messaging.
`ringkernel.hybrid`	`HybridDispatcher` for adaptive CPU/GPU workload routing.
`ringkernel.cuda`	CUDA device enumeration and profiling (feature-gated).
`ringkernel.benchmark`	Benchmarking suite with regression detection (feature-gated).

7 Use Cases

Primary Use Cases

- **Real-Time Simulation** — Physics simulations (FDTD, fluid dynamics) with persistent GPU actors that maintain state across timesteps without kernel relaunch.
- **Interactive GPU Workloads** — GUI-driven applications where sub-microsecond command injection enables 60 FPS+ interactive control of GPU computations.
- **Graph Analytics** — GPU-accelerated BFS, SCC, PageRank, and SpMV with stratified memory pools and multi-phase kernel execution.
- **Financial Computation** — Monte Carlo pricing, transaction monitoring, and fraud detection with GPU-native actor pipelines.
- **Digital Twins** — Persistent actors as long-running digital twin processes with bi-temporal state management and causal event ordering.
- **Process Mining** — GPU-accelerated directly-follows graph construction, conformance checking, and pattern detection at scale.
- **Edge & WebGPU Deployment** — Cross-platform execution via WebGPU for browser-based and embedded GPU workloads using the same Rust DSL.

8 Getting Started

8.1 Quick Start (Cargo)

```
# Add to Cargo.toml
cargo add ringkernel -features cpu

# Or with CUDA backend
cargo add ringkernel -features cuda

# Build entire workspace
cargo build -workspace
```

8.2 Quick Start (Python)

```
pip install ringkernel

from ringkernel import RingKernel, LaunchOptions
runtime = RingKernel.create_sync()
kernel = runtime.launch_sync("my_actor", LaunchOptions())
kernel.send_sync(b"hello from Python")
```

8.3 CLI Tool

```
# Scaffold a new project
ringkernel new my-app -template persistent-actor

# Generate GPU kernel code from Rust DSL
ringkernel codegen src/kernels/mod.rs -backend cuda,wgsl

# Check backend compatibility
ringkernel check -backends all
```

8.4 Run Examples

```
# Basic persistent actor
cargo run -p ringkernel -example basic_hello_kernel

# Kernel-to-kernel messaging
cargo run -p ringkernel -example kernel_to_kernel

# Transaction monitoring GUI (CUDA)
cargo run -p ringkernel-txmon -release -features cuda-codegen

# 3D wave simulation benchmark
cargo run -p ringkernel-wavesim3d -bin wavesim3d-benchmark
-release -features cuda-codegen
```

Links & Resources

Repository	https://github.com/mivertowski/RustCompute
Documentation	https://mivertowski.github.io/RustCompute/
Crates.io	https://crates.io/crates/ringkernel
PyPI	https://pypi.org/project/ringkernel/
Academic Paper	*The GPU-Native Persistent Actor Model* (arXiv, 2026)

Contact: michael.ivertowski@ch.ey.com