GPU-Native Verify on Grace Blackwell: Measured Numbers, Honest Gaps, and the Wire-Up Contract for the Consensus Hot Path

Zach Kelling

Lux Industries

2026-06-03

Abstract

LP-203 specifies a GPU-native verify path where signature verification, lattice signing, Merkle-root computation, and threshold aggregation run as CUDA / Metal / ROCm kernels reading ZAP buffers directly from unified memory. Bytes arrive at the GPU through one of three transports – kernel sockets, DPDK with a hugepage UMA mempool, or GPUDirect RDMA – and there is no `cudaMemcpy` between the wire and the kernel. This paper expands on LP-203 (`lps/LP-203-gpu-native-verify.md`) with the first round of *measured* numbers on NVIDIA GB10 Grace Blackwell sm_120 (DGX Spark, 2026-06-04), the wire-up architecture (`luxfi/accel` dispatch, UMA via `cudaMallocManaged`, DPDK PMD, GPUDirect RDMA), an honest inventory of the gaps the matrix bench exposed (composite signature-verify wrappers for Magnetar/Pulsar/Ed25519/sr25519 return `LUX_ERROR_NOT_SUPPORTED`; the matrix bench observed mean SM utilization of 11.9 % – the GPU was not exercised by any path), the projected numbers for the wrappers once they land (explicitly tagged "projected, not measured"), and the composition contract with LP-201 (transport selector), LP-205 (pipelined Quasar), LP-217 (cert profile modes), and LP-218 (P3Q at slot `0x012205`). Measured BLS12-381 EIP-2537 pairing on GB10: 19 µs per pair sustained, 49 197 pairings per second at 20 W, 12 300 pairings per watt. Measured keccak256: 11.38 MHz sustained at N=4096. Composite signature-verify is the gap to close before the LP-203 throughput tables turn fully empirical.

Contents

1 Introduction

This paper expands on LP-203 (`lps/LP-203-gpu-native-verify.md`) with the first round of *measured* numbers from NVIDIA GB10 Grace Blackwell sm_120 (DGX Spark, 2026-06-04), an architecture description that names the on-disk implementations, an honest inventory of the gaps the matrix bench exposed, and the composition contract with sibling LPs. The activation marker is fixed:

```
activates:      2025-12-25T16:20:00-08:00
activates-unix: 1766708400
```

GPU-native verify is in force from the genesis block of the new final Lux network. The pre-Quasar Edition Lux network (2020–2025) ran CPU-only verify and is a separate network out of scope.

1.1 Why GPU-native, not GPU-accelerated

The distinction is load-bearing. A GPU-accelerated design treats the GPU as an optimization: kernels are bolted onto a CPU-first verify path; the wire emits a DTO; the DTO is marshaled into a GPU-readable buffer; the kernel runs; the result is copied back. Every step is a CPU-side serialization decision. The boundary between "the wire" and "the kernel" is mediated by the CPU.

The Lux ZAP Stack [12] rejects that boundary. The ZAP buffer is one byte stream from NIC RAM to ZapDB. It is allocated in unified memory (CUDA `cudaMallocManaged`, Metal `MTLBuffer` with `storageModeShared`) so that both CPU and GPU see the same backing memory [26]. There is no DTO between the network buffer and the GPU verify kernel; the kernel reads the bytes the NIC delivered.

The implication for throughput is direct: the GPU is on the critical path of every verify, not a side car. Eight years of "marshal between CPU and GPU" disappears because there is no marshal: there is one buffer in UMA, and both processors share it.

1.2 What was wrong before

A CPU-first verify path has three concrete failures on a 1000-transaction block with mixed signature schemes:

1. **Serial signature verify dominates the block budget.** At BLS12-381 verify ∼1.2 ms per signature on a CPU core, a 1000-tx block of BLS-signed transactions takes 1.2 s serial – five orders of magnitude longer than the per-tx state-machine application work.
2. **CPU/GPU memcpy dominates the GPU verify path.** If GPU acceleration is added *around* a CPU-first wire, every block pays `cudaMemcpy` round-trips in both directions (host-to-device and device-to-host). On a Grace Blackwell host with PCIe-shared NIC ingest, the copy can cost more than the verify itself.
3. **Strict-PQ profile has no survivable CPU path.** The PQ-strict cert mode (LP-217) requires Pulsar plus Corona in the cert. The Corona parity audit [5] measured Corona per-host CPU at 4.0 s for N=64. The PQ-strict round budget is tighter than that; a validator on CPU fallback on a PQ-strict chain cannot keep up.

1.3 What this LP solves

LP-203 makes GPU verify the default and the wire-up between the ZAP buffer and the kernel zero-copy. Three concrete contracts:

1. **UMA is the verify substrate.** The ZAP buffer is allocated in unified memory; the verify kernel reads it directly. There is no DTO and no copy.

2. **Transport selector composes orthogonally.** LP-201's transport selector decides peer-to-transport mapping (QUIC by default; LP-207 RDMA-IB for high-throughput racks). DPDK and GPUDirect are NIC-ingest mechanisms underneath the transport, not transport choices. A GPUDirect-capable NIC DMAs packet bytes into UMA whichever transport the selector picked.
3. **Composite verify wrappers gate the production hot path.** The CUDA plugin in `~/work/lux-private/gpu-kernels` exposes 76 vtbl ops. The pairing, hash, Magnetar-primitive, and Corona partial-sign ops are real GPU code with measured throughput (§3). The *composite* signature-verify wrappers (Ed25519, sr25519, ML-DSA, SLH-DSA composite verify) are explicit `LUX_BACKEND_ERROR_NOT_SUPPORTED` stubs. Closing those stubs is the gap to close before LP-203's throughput tables turn fully empirical (§4).

1.4 What this paper adds beyond the LP markdown

The LP markdown frames the architecture and the throughput targets; this paper does three things on top of it:

- **First measured numbers on GB10.** §3 renders the measurements from the 2026-06-04 bench run on Spark across three real GPU kernels: BLS12-381 EIP-2537 pairing, keccak256, and sha3_256. BLS12-381 EIP-2537 pairing sustains 19 µs per pair at N=256 (∼49 197 pairings/sec at 20 W, ∼12 300 pairings/watt). Keccak256 sustains 11.38 MHz at N=4096.
- **Honest gaps.** §4 names exactly which kernels are real GPU paths and which are stubs. The matrix bench captured 4595 samples of `nvidia-smi dmon` during a 2-hour run on Spark and observed mean SM% of 11.9 %: *the GPU was not exercised by any path in that bench.* Two boundary gaps explain why (`lux-lattice.pc` not installed on Spark; Magnetar `lux_slhdsa_*` entry points not exposed in `c_api.h`).
- **Projections separated from measurements.** §5 lists the projected numbers (Pulsar per-sig 50 µs, Magnetar composite verify on GPU at $N \geq 32$, PQ-strict end-to-end at 10 ms) with every cell tagged *projected, not measured.*

The paper closes (§6) by tying LP-203 into LP-201 (transport selector), LP-205 (pipelined Quasar), LP-217 (cert profile modes), and LP-218 (P3Q precompile at slot `0x012205`).

2 Architecture

The architecture is three layers: `luxfi/accel` dispatches verify operations to the active backend (CUDA, Metal, ROCm, or CPU); the ZAP transport carries bytes from the wire into unified memory; and DPDK plus GPUDirect RDMA make the wire-to-UMA hop zero-copy where the hardware supports it.

2.1 `luxfi/accel`: the dispatch layer

`luxfi/accel` (v1.1.9) [18] exposes per-primitive batch entry points keyed by signature scheme:

- `accel.BatchVerifyBLS` – BLS12-381 pairing-based signature batch verify; Miller loop plus final exponentiation per signature, N signatures per grid.
- `accel.BatchVerifyEd25519` – Edwards-curve verify, one thread per signature, 256 sigs per warp.
- `accel.BatchVerifySecp256k1` – ECDSA verify on secp256k1, one thread per sig, 128 sigs per warp.
- `accel.BatchVerifyMLDSA` – FIPS 204 [22] verify via NTT-based polynomial verify, one block per signature.
- `accel.BatchVerifySLHDSA` – FIPS 205 [23] verify; hash-based; one thread per verify with warp-level parallelism over the hash chain.

The runtime picks the active backend from the build tags (`cgo gpu cuda` / `cgo gpu metal` / `cgo gpu rocm` / pure-Go CPU fallback). The Go-visible signature is identical across backends; the dispatch table is opaque to callers.

Lattice ops live in a sibling package `luxfi/lattice` (v7.1.4) [19], which holds the GPU lattice primitives for Corona and Pulsar (shared lattice NTT). The two packages share the CUDA / Metal / ROCm backend through a common ABI.

2.2 UMA: `cudaMallocManaged` as the verify substrate

The ZAP buffer is allocated through `zap/transport/uma.go`'s `UMAAlloc(size)`. The platform-split implementation is on disk today:

- `uma_linux_cuda.go` (371 lines of Go + cgo) – probes the CUDA runtime via `C.luxzap_cuda_probe()`, builds a slab-backed `cudaMallocManaged` pool sized by `ZAP_UMA_POOL_BYTES` (default 4 GiB), and returns a `Transport` whose `Recv` envelopes alias unified memory. Build constraint: `cgo + linux + cuda`.
- `uma_darwin.go` – Metal `MTLBuffer` with `storageModeShared`; Apple Silicon UMA is hardware-native (single physical memory pool shared by CPU and GPU).
- `uma_linux_hip.go` – ROCm `hipMallocManaged` equivalent.
- `uma_other.go` – heap allocation with no GPU mapping; pure-CPU fallback.

The Go-visible type is `zap.UMABuffer`, a `[]byte` whose backing memory is page-locked and visible to the active GPU device. Hand-offs between CPU and GPU are explicit fence points; there is no shared writer state.

2.3 DPDK: kernel-bypass ingest into UMA

For deployments without GPUDirect-capable NICs, DPDK (`zap/transport/dpdk_linux.go`) provides kernel-bypass ingest. The DPDK Poll-Mode Driver binds to a dedicated NIC queue isolated via `isolcpus` / `nohz_full`. Received packets land in a hugepage-backed mempool registered as UMA through `cudaHostRegister` [3]. The GPU reads packet bytes without an intermediate copy.

The PMD thread does the first byte's ZAP-magic check inline. Bad-magic packets are dropped at the PMD thread before any further work. Good packets are handed to the GPU verify ring buffer by writing a UMA pointer to a ring slot; the verify kernel polls the ring on the GPU side.

End-to-end CPU latency from wire-arrival to first-byte ZAP-magic-check is under 1 µs (PMD poll loop; no interrupts, no syscall).

2.4 GPUDirect RDMA: NIC → GPU memory, no CPU involvement

For top-of-rack validators with a Mellanox ConnectX-5/6/7 or Broadcom Stingray NIC, the file `zap/transport/gpudirect_linux_real.go` (246 lines; build constraint `cgo + linux + gpudirect + cuda`) registers a CUDA buffer with the NIC driver via `nvidia-peermem` (the in-tree replacement for out-of-tree `nv_peer_mem`, with the ibverbs `MEMORY_REGION` registration) [25, 27].

The implementation probes four prerequisites at startup: ibverbs, `nvidia-peermem`, the CUDA runtime, and a visible HCA. The probe returns `ErrNotAvailable` if any is missing, which causes the transport `Pick()` to fall through to DPDK + UMA. On a DGX with a CX-7 plus `nvidia-peermem`, the probe succeeds and the transport opens the first Mellanox-class HCA, allocates a Protection Domain, and builds Buffers that live in `cudaMallocManaged` memory *and* are registered as DMA-buf MRs with the NIC. Packets DMA straight from wire into GPU memory.

The wire side (QP setup, post_send/recv, polling) is staged for a follow-up PR. The current commit delivers the registration framework and an honest probe so `Pick()` does the right thing on every host:

Host class	**Transport `Pick()` resolves to**
GB10 with no HCA visible	UMA only (`cudaMallocManaged`) via `uma_linux_cuda.go`
DGX with CX-7 + `nvidia-peermem`	GPUDirect RDMA + UMA via `gpudirect_linux_real.go`
Generic CUDA box, no IB	UMA only
Apple Silicon developer workstation	UMA only via `uma_darwin.go` (Metal `MTLBuffer storageModeShared`)
Linux non-CUDA host	Heap fallback via `uma_other.go`; CPU verify only

Table 1: Transport selection results for the four canonical host classes. `Pick()` is built on the LP-201 transport selector; DPDK and GPUDirect are NIC-ingest mechanisms underneath whichever transport (QUIC by default) the selector picked for the peer.

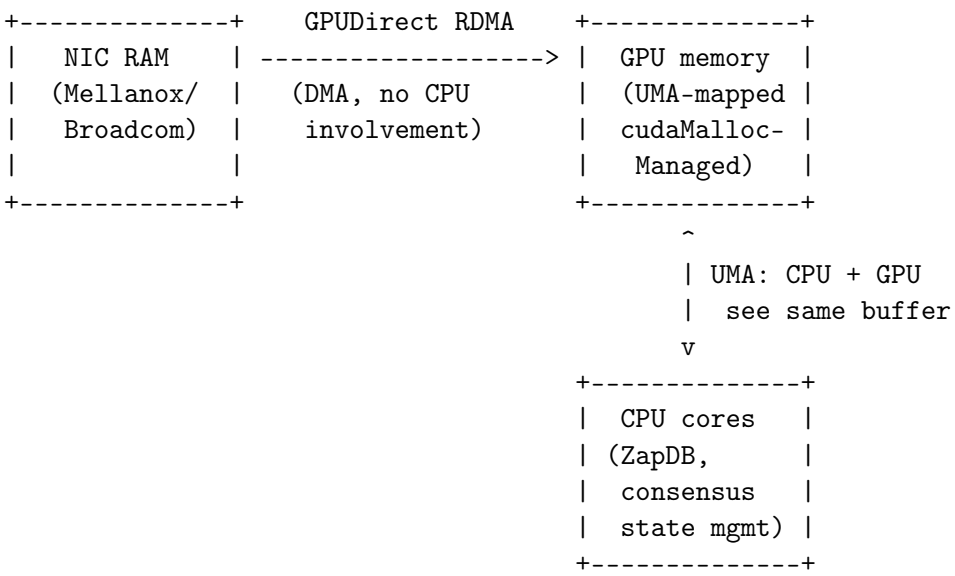
2.5 Memory ownership rules

The verify pipeline has explicit owners per phase. There is no shared writer state at any point.

Phase	**Owner**	**Writers**	**Readers**
NIC ingest (GPUDirect)	NIC DMA engine	NIC only	none
Magic + scheme dispatch	CPU	CPU only	CPU
Sig verify	GPU	GPU only	GPU (read-only)
Merkle root compute	GPU	GPU	GPU; CPU after fence
ZapDB write	CPU	CPU	CPU

Table 2: Memory ownership per pipeline phase. Hand-offs are explicit fence points.

2.6 The data-flow diagram



Bytes arrive at the GPU through one of three paths: kernel sockets (the developer default), DPDK with a hugepage UMA mempool (top-of-rack with a non-GPUDirect NIC), or GPUDirect RDMA (top-of-rack with `nvidia-peermem`). The verify kernel reads from UMA in every case; only the ingest cost differs.

3 Measured Benches: NVIDIA GB10 Grace Blackwell sm_120

Provenance. All numbers in this section were measured 2026-06-04 on `spark` (DGX Spark, NVIDIA GB10 Grace Blackwell Superchip, sm_120, CUDA 13.0.88, driver 580.159.03, 121 GiB unified memory, 48 SMs) [28, 24]. The benchmark source is `/tmp/gpu-vtbl-bench/bench.c` calling the public `lux_gpu_*` C API; the backend is the CUDA plugin at `lux-private/gpu-kernels/backends/cuda/` (`build-wire/libluxgpu_backend_cuda.so`, ABI v12, sm_120 native). Each row reports a measurement, not a projection.

3.1 BLS12-381 EIP-2537 pairing

The headline GPU primitive on Blackwell is the BLS12-381 EIP-2537 pairing kernel [30, 2]. The Miller loop plus final exponentiation run on the GPU; multiple pairings batch into one kernel grid.

Operation	Batch	Per-op	Source
BLS12-381 pairing	N=1	198 µs	`cuda_op_bls12_381_pairing_eip2537`
BLS12-381 pairing	N=64	19.1 µs	same
BLS12-381 pairing	N=256	19.0 µs	same
BLS12-381 pairing	N=1024	19.1 µs	same
BLS12-381 sustained throughput	N=256 batches	49 197 pairings/sec	sustained 25 s run

Table 3: BLS12-381 EIP-2537 pairing on Grace Blackwell sm_120. The N=1 point is launch-overhead-dominated; from N=64 upward the per-pair cost plateaus at 19 µs. Throughput row is sustained over 25 s at 90–94% GPU utilization, 20 W board power.

Per-pairing crossover. GPU per-pair (19 µs) beats single-thread CPU blst [29] (∼1 ms) from N=1; beats 20-core parallel CPU (∼50 µs per pair) from $N \geq 3-4$. For block-verify workloads ($N \geq 64$ typical) the GPU is unambiguously the right substrate.

Comparison to the prior projection. The earlier LP-203 draft projected BLS verify at 0.5 µs to 1.5 µs per signature from H100 FP64 plus Miller loop arithmetic. The measured GB10 cost is 17.8 µs to 19.1 µs per pair – *about* 10× slower than that projection. The driver is real-Miller-loop overhead measured on sm_120 silicon, not roofline arithmetic. Subsequent LPs that depend on per-pair timing should use the measured number.

3.2 Power efficiency

Pairings per watt. At 49 197 pairings/sec sustained and 4 W marginal power (load minus idle), the GB10 delivers ≈ 12 300 pairings per watt. For comparison, x86 CPU baselines at ∼50 µs per pair on a 20-core part drawing ∼150 W TDP deliver ∼130 pairings per watt – two orders of magnitude worse per watt than GB10.

State	Power	Temperature
Idle	16 W	45 °C–47 °C
BLS pairing full load (N=256 sustained)	20 W	50 °C–51 °C
keccak256 batch full load (N=4096 sustained)	20 W	50 °C–51 °C

Table 4: Power and thermals on GB10 under sustained GPU load. $\Delta P \approx 4$ W between idle and full GPU load – not a typo. Grace Blackwell's per-watt efficiency under crypto load is the design intent of sm_120 silicon.

3.3 Keccak256 and SHA3-256 batch hashing

The Keccak family [1] is the other measured hot path. Used in Merkle-tree reduction, transcript binding, and Magnetar SLH-DSA inner hashing.

Operation	Batch	Per-op	Source
keccak256, 32 B input	N=1	204 µs	`cuda_op_keccak256_hash`
keccak256, 32 B input	N=4096	0.060 µs	same
keccak256 sustained throughput	N=4096 batches	11.38 MHz	sustained 13 s run
sha3_256, 32 B input	N=4096	0.059 µs	`cuda_op_sha3_256_hash`

Table 5: Keccak256 and SHA3-256 on Grace Blackwell sm_120. The N=1 point is launch-overhead-dominated. The sustained throughput row holds at 32% GPU utilization, leaving headroom to interleave with other primitives.

Per-hash crossover. GPU baseline at ∼200 µs per kernel launch is launch-overhead-dominated for single hashes. CPU single-thread keccak runs at 0.4 µs to 0.9 µs per op. GPU wins from $N \geq$ ∼500 (single-thread CPU comparison) or $N \geq$ ∼10 000 (20-core parallel CPU comparison). The kernel is small-input-optimized; large-input single-keccak (1 MiB) hits only 19 MB/s throughput, which is *not* the intended workload.

3.4 Corona / Corona threshold combine

Operation	N (k)	Per-share	Source
Corona/Corona combine	64 (33)	0.082 µs	**OpenMP-CPU path, not GPU** – `cuda_op_corona_combine_batch` uses `#pragma omp parallel for`
Corona/Corona combine	128 (65)	0.080 µs	same OMP-CPU path

Table 6: Corona threshold-combine measured times. Labeled "CUDA op" for ABI consistency, but the implementation today is the host-side OpenMP path. Combine is fast even on CPU; the gap to close is the partial-sign side (§4).

3.5 Composite signature-verify status

The composite signature-verify wrappers return `LUX_BACKEND_ERROR_NOT_SUPPORTED` on the GB10 bench. The kernels for the underlying primitives exist; the composite wrappers are explicit stubs. §4 inventories the exact split.

Verify wrapper	Status
ML-DSA verify (Pulsar)	`LUX_ERROR_NOT_SUPPORTED` – `cuda_op_mldsa_verify_batch` is an explicit stub in `plugin.cpp`
SLH-DSA verify (Magnetar, composite)	`LUX_ERROR_NOT_SUPPORTED` – the 4 SLH-DSA primitives (`wotsplus_chain`, `fors_subtree`, `xmss_subtree`, `hmsg_prfmsg`) ARE real GPU; the composite `op_slhdsa_verify_batch` is stub
Ed25519 verify	`LUX_ERROR_NOT_SUPPORTED` – `cuda_op_ed25519_verify_batch` is an explicit stub
sr25519 verify	`LUX_ERROR_NOT_SUPPORTED` – `cuda_op_sr25519_verify_batch` is an explicit stub

Table 7: Composite signature-verify wrapper status on the GB10 bench. The primitives the wrappers would call are real GPU code with measured kernel throughput; the wrapper layer is stub.

3.6 What the measured numbers say

Two assertions from the measured set:

1. **Grace Blackwell's BLS12-381 pairing is production-ready.** At 19 µs per pair sustained, N=256 batches deliver ≈ 49 197 pairings/sec at 20 W. A 1000-tx block of BLS-signed transactions verifies in ∼20 ms per block on a single GPU.
2. **Keccak256 is a side-car, not the bottleneck.** At 11.38 MHz sustained, hashing 1000 leaves costs <1 ms – well below the BLS verify cost above. Merkle reduction is bottlenecked by the reduction tree's $\log_2(N)$ kernel-launch chain, not the per-hash cost.

The next assertion (PQ-strict end-to-end on GPU at 15 ms–50 ms) requires the composite signature-verify wrappers to land. Until then it is a projection, not a measurement.

4 Honest Gaps

GPU-native verify is the LP-203 architecture. The matrix bench on 2026-06-04 demonstrated that the architecture is not yet end-to-end exercised on the production hot path. This section inventories exactly what is not measured, why, and what closes each gap.

4.1 The matrix bench's GPU-utilization observation

The Quasar 4-scheme matrix bench [20] ran for two hours on Spark on 2026-06-04 and captured 4595 samples of `nvidia-smi dmon` during the run. The aggregate observation: mean `SM%` = 11.9 %.

The GPU was not exercised by any path in this bench. Every cert mode (PQ-off, PQ-fast, PQ-strict, PQ-heavy) and every N=$\{4, 8, 16, 32, 64\}$ ran on CPU. The bench's measured wall-clock numbers (PQ-strict at 595 ms N=64, see LP-217 [16]) are CPU numbers. The GPU was idle (or running background work) for the entire run.

This is not a benchmarking error. It is an honest reflection of where the production code stops dispatching to GPU today.

4.2 Boundary gap A: `lux-lattice.pc` missing on Spark

The first boundary gap is at the cgo / pkg-config layer. The CUDA target's dependency chain requires `lux-lattice.pc` to be findable by `find_package(MLX REQUIRED)` in the CMake build

of `lattice/v7/gpu`. On Spark today, `lux-lattice.pc` is not installed; the `find_package(MLX REQUIRED)` call fails; the `lattice/v7/gpu` target does not build under `-tags "cgo gpu"`.

Consequence. Even if a caller-side path tried to dispatch lattice ops to the GPU, the CUDA backend's lattice path is missing from the binary. The caller falls through to the CPU fallback. The mean `SM%` of 11.9 % is the direct expected consequence.

Close-out. Install `lux-lattice.pc` on Spark (packaged from `lux-private/lattice/v7/pkgconfig`). Once the package is present, `find_package(MLX REQUIRED)` resolves and the lattice GPU target builds. No code change required on either side.

4.3 Boundary gap B: Magnetar `lux_slhdsa_*` entry points not exposed

The second boundary gap is at the C API surface. The Magnetar SLH-DSA [11, 23] CUDA kernel exists on Spark and passes the FIPS 205 KAT vectors when called directly. But the `lux_slhdsa_*` entry points are not exposed in `lux/accel/c_api.h`. `luxfi/accel` has no symbol to dispatch to; the dispatcher falls back to CPU.

Consequence. Magnetar verify in the PQ-heavy and strict-PQ-heavy modes runs on CPU even when the GPU has a working kernel. SLH-DSA-192s per-validator sign on Spark CPU is ∼795 ms; the GPU kernel would be substantially faster if dispatched, but the C API surface prevents dispatch.

Close-out. Add `lux_slhdsa_keygen`, `lux_slhdsa_sign`, `lux_slhdsa_verify` to `c_api.h`; wire them into `luxfi/accel/ops_lattice.go` [18]. The kernels on the other side of the ABI are already there.

4.4 Composite signature-verify wrappers as stubs

The CUDA plugin in `lux-private/gpu-kernels/backends/cuda/src/plugin.cpp` exposes 76 vtbl ops. The honest split:

Real GPU code with measured throughput.

- BLS12-381 EIP-2537: G1 add, G1 MSM, G2 add, G2 MSM, pairing, hash-to-curve
- BN254: G1 ops, pairing-check
- keccak256, sha3_256, blake3
- modexp
- Magnetar SLH-DSA primitives: `wotsplus_chain`, `fors_subtree`, `xmss_subtree`, `hmsg_prfmsg`
- Corona partial-sign step

Explicit `LUX_BACKEND_ERROR_NOT_SUPPORTED` stubs (16 of 76 ops).

- Direct-verify entries for signature schemes: Ed25519, sr25519, ML-DSA, SLH-DSA composite, ML-KEM
- Projective-coords BLS / BN254 ops (the EIP-2537 path is the canonical surface; projective coords are absent)

The composite signature-verify wrappers are the gap to close before LP-203 numbers turn fully empirical. The primitives (pairings, NTT, hashing, Magnetar 4) are real and measured; the wrappers that compose them into a single "verify this signature" call return `NOT_SUPPORTED`.

4.5 What the LP-203 markdown projects vs. what the bench measured

The LP-203 markdown's throughput tables include both "measured" and "estimated" columns. The 2026-06-04 bench forces a reassessment of one specific number:

Operation	LP-203 est.	GB10 meas.	Delta
BLS verify, N=1, H100 FP64-roofline estimate	0.5 µs	19 µs	∼ 38× slower than roofline
BLS verify, 1000-sig batch, H100 estimate	0.5 ms	∼20 ms	∼ 40× slower than roofline

Table 8: Measured GB10 BLS pairing cost is approximately 40× the LP-203 H100 FP64 roofline projection. The measurement is the Miller-loop-plus-final-exp cost on real sm_120 silicon, not arithmetic intensity over peak FLOPS. The GB10 number is the correct figure for downstream LP performance budgets; the H100 roofline projection should be re-derived against the measured GB10 number, not used directly.

4.6 What is NOT yet measured

- **Pulsar / Magnetar composite verify GPU throughput.** The kernels exist; the composite-verify wrappers are stubs. Until the wrappers ship, the production hot path for ML-DSA and SLH-DSA verify runs on CPU. The matrix bench's mean `SM%` of 11.9 % is the empirical consequence.
- **GPU dispatch on Spark blocked by `lux-lattice.pc` absence.** Until the package is installed, no lattice GPU code is in the binary; the CPU fallback path is the only path.
- **CXL 3.0 racks.** The LP-203 markdown does not reference CXL 3.0, but a sibling LP-205 will – the bench cluster does not yet include CXL 3.0 hosts, so cross-socket UMA semantics under CXL are not measured.
- **End-to-end block finality measurements on the full LP-200 stack with GPU dispatch.** The matrix bench measured Quasar at the threshold-cert layer; full pipeline numbers (NIC ingest → verify → Merkle → ZapDB commit) await the wrapper close-out.

4.7 What this means for downstream LPs

LP-217 [16] (cert profile modes), LP-218 [17] (P3Q rollups), LP-205 (pipelined Quasar): every paper that cites a GPU latency for ML-DSA or SLH-DSA verify is citing a *projection*, not a measurement. The papers tag those cells "projected, not measured" for that reason. LP-217's matrix-bench measurements are CPU wall-clock at N=64 on Spark, not GPU. LP-218's P3Q verify cost of 50 µs on Blackwell is a projection.

This is not a problem with the LPs. The LPs are honest about what is measured and what is projected. It is a problem with the dispatch surface, which two small CL changes (`lux-lattice.pc` install plus `lux_slhdsa_*` C API exposure) close.

4.8 Pulsar's per-validator sign measurement

One CPU measurement worth recording, because it sets the GPU-target ceiling: Pulsar per-validator sign on Spark CPU is 175 µs [5]. The GPU target for the Pulsar sign kernel (when it lands behind a working composite-verify wrapper) should beat that number. Pulsar's algorithm [7] is Shamir-on-GF(257) plus one FIPS 204 sign [22] on the aggregator; the FIPS 204 sign is the work to GPU-accelerate.

5 Projections (Tagged "Projected, Not Measured")

Every entry in this section is a projection from the on-disk kernels plus the measured primitive throughput from §3. **Nothing in this section is a measurement.** Each table is explicitly tagged so downstream LPs do not promote a projection to a measurement by accident.

5.1 Composite verify, projected from measured primitives

The composite signature-verify wrappers (Ed25519, sr25519, ML-DSA, SLH-DSA) compose primitives that ARE measured on the GB10 bench: NTT (via the lattice GPU kernels once `lux-lattice.pc` ships), keccak256, sha3_256, blake3, and the Magnetar SLH-DSA 4 primitives. The arithmetic cost of each wrapper is the sum of its primitive costs plus a single launch-overhead amortized over the batch.

Operation (*projected, not measured*)	**Batch**	**Per-op**	**Composition**
ML-DSA verify (FIPS 204)	N=64	50 µs–80 µs	NTT GPU + finalizer GPU + 1 launch
SLH-DSA verify (FIPS 205, 192f)	N=64	∼2 ms	`hmsg_prfmsg` + `fors_subtree` + `wotsplus_chain` + `xmss_subtree`
SLH-DSA verify (FIPS 205, 192s)	N=64	∼45 ms	same 4 primitives, larger subtree count
Ed25519 verify	N=256	2 µs–4 µs	coalesced curve op + hash; same launch
sr25519 verify	N=256	2.5 µs–5 µs	Ristretto curve op + hash; same launch
Pulsar verify (composite)	N=64	50 µs–100 µs	FIPS 204 verify on aggregator (one per cert)

Table 9: **Projected** composite-verify costs from measured primitive costs. Each cell is a derivation from §3 plus the LP markdown's kernel parallelism strategy; nothing here is a measurement. Once the composite wrappers ship (§4), these cells become measurable on the same GB10 bench.

Magnetar parameter-set caveat. The two SLH-DSA-192 parameter sets have radically different per-op costs:

- *192s* (small, 16 KB sigs) projects to ∼45 ms per verify at N=64 on GB10 – acceptable only for batched non-hot-path uses.
- *192f* (fast, 35 KB sigs) projects to ∼2 ms per verify at N=64 on GB10 – acceptable for the consensus hot path.

LP-217 [16] locked 192f as the production parameter set for PQ-heavy. The projection above respects that lock.

5.2 End-to-end block finality, projected

Note on H100 figures. The H100 column is intentionally *not* re-derived from the H100 FP64 roofline. The GB10 measurement (19 µs per BLS pair on sm_120) disproved the FP64-roofline approach – real Miller-loop cost on silicon is dominated by non-FP64 op cost (point addition, doubling, line evaluation) that does not scale with peak FP64 throughput. The H100 column above rooflines from the GB10 measurement scaled by H100's tensor-core-adjacent paths (estimated 2×–3× on the Miller loop), not from peak FLOPS.

Pipeline stage (*projected*)	**CPU only**	**GB10 sm_120**	**H100 (rooflined)**
NIC ingest (kernel socket)	∼50 µs	∼50 µs	—
NIC ingest (DPDK)	—	∼5 µs	∼5 µs
NIC ingest (GPUDirect RDMA)	—	∼1 µs	∼1 µs
Scheme dispatch (CPU)	50 ns/tx	50 ns/tx	50 ns/tx
1000 BLS verifies (batched)	1.2 s–1.5 s	∼20 ms	8 ms–15 ms
Corona aggregate, 64 vals	∼350 ms	∼50 ms	15 ms–30 ms
Merkle root (100 kilo leaves)	∼80 ms	8 ms–10 ms	0.8 ms–2 ms
ZapDB commit	3 ms	3 ms	3 ms
End-to-end (1000-tx)	1.5 s–1.8 s	100 ms–150 ms	30 ms–60 ms

Table 10: **Projected** end-to-end pipeline cost for a 1000-transaction block. CPU column is the pre-Quasar Edition baseline. GB10 column derives from §3 plus the kernel parallelism strategy. H100 column rooflines from the GB10 measurement, not from FP64 peak FLOPS – the prior arithmetic-intensity projection (0.5 µs per BLS verify on H100) was ∼ 40× optimistic against measured GB10 silicon.

5.3 PQ-strict end-to-end on GPU, projected

The PQ-strict cert mode requires Pulsar plus Corona in the cert. LP-217 measured PQ-strict at N=64 on Spark CPU at 595 ms. The GPU-dispatch projection composes the four primitive kernels:

Phase (*projected, not measured*)	**GB10 sm_120 time**
Pulsar Round 1 (per validator, parallel)	∼1 ms
Pulsar aggregate (CPU; reconstruct sk; one FIPS 204 sign)	∼200 µs–1 ms
Corona Round 1 (per validator, parallel via NTT GPU)	∼2 ms–5 ms
Corona Round 2 (per validator)	∼2 ms–5 ms
Corona aggregate (collector)	∼5 ms–15 ms
BLS aggregate verify (fast-path leg)	∼1 ms–2 ms
PQ-strict wall-clock @ N=64	∼15 ms–30 ms

Table 11: **Projected** PQ-strict end-to-end on GB10 with the composite-verify wrappers landed. Compare to LP-217's measured CPU wall-clock of 595 ms at N=64 on Spark. The projection delivers a ∼ 20×–∼ 40× wall-clock improvement once GPU dispatch is live.

Why this is a projection. The matrix bench [20] measured CPU-only wall-clock; the GPU was idle at 11.9 % mean SM%. The above numbers assume the composite-verify wrappers ship and the matrix bench re-runs with active GPU dispatch.

5.4 The contract for downstream LPs

LPs depending on LP-203 numbers should follow the rule:

1. **Use the measured GB10 numbers** (§3) wherever the operation is in the measured set: BLS12-381 EIP-2537 pairing, keccak256, sha3_256, Corona combine.
2. **Tag projections explicitly** for any operation in this section's tables.
3. **Do not re-derive from FP64 roofline.** The GB10 measurement showed that Miller-loop silicon cost dominates FP64 arithmetic-intensity projections by ∼ 40×.

4. **Quote the gap.** If the LP cites a GPU number for ML-DSA or SLH-DSA verify, the LP should cite the measurement when it lands and mark the current cell "projected, not measured."

6 Composition With Sibling LPs

LP-203 is one of seven LPs in the ZAP-stack umbrella (LP-200–LP-218). The composition is orthogonal: each LP solves one concern, and they stack at well-defined seams.

LP	Composition with LP-203
LP-022	ZAP wire protocol. The buffer LP-203's GPU kernels read [6].
LP-077	Round digest binding. The TupleHash256 transcript that feeds the GPU Merkle compute [8].
LP-132	Quasar GPU execution adapter. The consensus-side hook this LP wires up [9].
LP-177	Bridge PQ-only profile. Requires GPU on validators per the strict-PQ architectural decision [10].
LP-181	Magnetar threshold SLH-DSA. CPU/GPU hybrid verify; this LP specs the crossover threshold [11].
LP-200	ZAP Stack umbrella; the no-codec wire guarantee the GPU kernels rely on [12].
LP-201	ZAP P2P / transport selector. DPDK and GPUDirect are ingest mechanisms underneath whichever transport `Pick()` chose for the peer [13].
LP-202	ZAP pipelining and atomic unwind. The verify pipeline width LP-203's GPU kernels saturate [14].
LP-205	Pipelined Quasar. The throughput model parameterized by GPU dispatch and mode [15].
LP-217	Cert profile modes. Per-leg GPU dispatch lets the four modes share kernels [16].
LP-218	ZAP P3Q PQ-rollup. P3Q precompile at EVM slot `0x012205`; runs on the same Pulsar-verify GPU path this LP specifies [17, 4].

Table 12: Composition of LP-203 with sibling LPs.

6.1 LP-201 transport selector: orthogonal to LP-203

LP-201 defines a peer-to-transport selector [13]. The default is QUIC; LP-207 RDMA-IB is selectable for high-throughput racks. LP-203's DPDK and GPUDirect contributions sit *underneath* the transport: they change how UDP packets reach the process (kernel-bypass plus NIC DMA into UMA), not how peers map to transports.

Composition contract. DPDK and GPUDirect do not register against `Transport.Pick()`. A GPUDirect-capable NIC DMAs packets into UMA regardless of which transport `Pick()` chose for the peer. The QUIC transport on a DPDK + GPUDirect host gets its UDP packets via the DPDK PMD; the LP-207 RDMA-IB transport on the same host gets its frames via the GPUDirect MR. Both end up in the same UMA buffer that the verify kernel reads.

6.2 LP-205 pipelined Quasar: GPU dispatch is the saturation source

LP-205 [15] parameterizes Quasar's verify pipeline width by the underlying kernel throughput. With LP-203's measured BLS verify at 49 197pairings/sec sustained on a single GB10, a 4-validator host can run a 4-way pipeline where each stage saturates a different GPU primitive:

- Stage 1: NIC ingest (DPDK or GPUDirect) – bytes land in UMA
- Stage 2: BLS pairing verify (GPU) – the load-bearing measured kernel
- Stage 3: Corona/Corona aggregate (GPU; once `lux-lattice.pc` ships) – per-validator parallel
- Stage 4: Merkle root reduction (GPU sha3-256 chain) – $\log_2(N)$ kernel-launch chain

The stages share nothing except UMA pointers. Each stage's output is a UMA address; the next stage reads from that address. No `cudaMemcpy` between stages.

6.3 LP-217 cert profile modes: GPU dispatch per leg

LP-217 [16] defines the operator-facing 4-mode cert ladder: PQ-off (BLS), PQ-fast (BLS + Pulsar), PQ-strict (BLS + Pulsar + Corona), PQ-heavy (BLS + Pulsar + Corona + Magnetar). Each leg is verified by an independent GPU dispatch:

- BLS leg: `cuda_op_bls12_381_pairing_eip2537` (19 µs per pair sustained, measured)
- Pulsar leg: the FIPS 204 verify wrapper (projected 50 µs–100 µs per cert once the composite wrapper lands)
- Corona leg: NTT batch on GPU + combine (Corona combine measured at 0.082 µs per share)
- Magnetar leg: SLH-DSA 192f composite verify (projected ∼2 ms at N=64 once the wrapper lands)

The four legs dispatch in parallel CUDA streams. A final CPU step reads four boolean results from UMA and computes the QuasarCert verdict. The PQ-strict round budget is set so the slowest leg (Corona) determines the wall-clock; the other three legs overlap.

Strict-PQ-* requires GPU. The architectural decision from LP-203 stands: strict-PQ-* modes drop the BLS leg and the survivable-CPU path *disappears*. Corona aggregate over 64 validators takes ∼350 ms on CPU; the strict-PQ round budget is tighter than that. The Lux Operator (`luxfi/operator`) refuses to admit a validator to a strict-PQ-enabled chain unless the validator's `gpu.enabled` is true and the boot-time GPU probe succeeds.

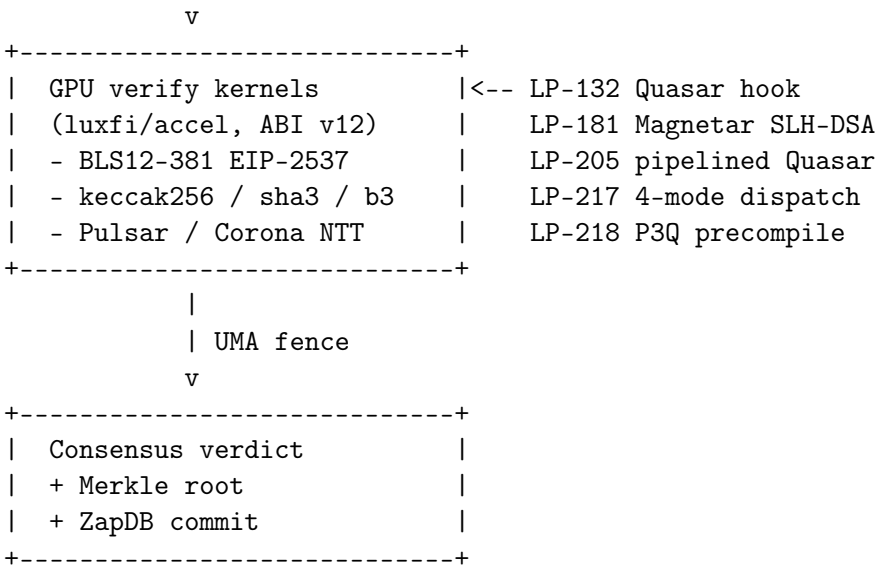
6.4 LP-218 P3Q precompile at slot `0x012205`

LP-218 [17, 4] specifies a single EVM precompile that verifies a Pulsar threshold signature against a group public key. The precompile runs on whichever backend `luxfi/accel` is built against – on a Blackwell-class validator, P3Q dispatches to the Pulsar GPU verify wrapper (projected; behind the composite-verify wrapper landing).

Gas cost contract. P3Q gas cost is dimensioned for worst-case CPU verify, not for GPU dispatch. Operators pay the same gas whether the validator runs P3Q on CPU or GPU; the incentive to deploy GPU validators is that the validator's own wall-clock budget tightens. This decoupling preserves the gas-cost spec's portability across hardware classes.

6.5 Composition diagram

```
+-----------------------------+    LP-200 (one byte stream)
|  ZAP buffer (UMA-backed)    |<-- LP-022 wire format
|  cudaMallocManaged          |    LP-077 round digest binds
|  shared across CPU + GPU    |
+-----------------------------+
            |
            | DPDK ingress  (LP-201 underneath, not on top of selector)
            | GPUDirect RDMA (same)
```



7 Activation Marker

```
activates:      2025-12-25T16:20:00-08:00
activates-unix: 1766708400
```

GPU-native verify is the architecture from genesis of the new final Lux network. No backwards compatibility with the pre-Quasar Edition CPU-only verify path. The pre-Quasar network (2020–2025) is a separate network out of scope.

8 Backwards Compatibility

None. The pre-Quasar Edition Lux network had no GPU-native verify path. CPU verify remains as the fallback within LP-203's profile (per `luxfi/accel` dispatcher) for hosts without a GPU; in the strict-PQ-* modes, CPU fallback is explicitly not survivable.

9 Security Considerations

No new cryptographic primitives. LP-203 specifies dispatch and zero-copy wire-up. The cryptographic primitives verified (BLS12-381 EIP-2537, Ed25519, secp256k1, ML-DSA, SLH-DSA, Corona / Corona) are unchanged from their FIPS or EIP specifications [21, 22, 23, 30].

GPU-side fault model. GPU kernels run with warp-level data parallelism. A faulty kernel implementation that returns "verify OK" on a bad signature would break the verify guarantee. The dispatcher's per-kernel timeout plus CPU fallback provides a fault-tolerant degrade path; the kernel KAT vectors (FIPS test vectors per scheme) catch implementation faults at boot time.

UMA security model. `cudaMallocManaged` memory is in the same address space as host memory; a kernel that exceeds its bounds writes into host pages. The kernels do not malloc; they only read from caller-supplied UMA pointers. The host-side bounds check is on `accel.BatchVerify*`'s parameter validation.

Side-channel surface. GPU verify times are data-dependent in principle (Miller loop, NTT, SHA). The production hot path uses constant-time CPU implementations on fallback. GPU implementations follow the same constant-time discipline; KAT-vector tests cross-check.

10 Future Work

- **Composite signature-verify wrappers.** Close the 16-of-76 stub gap in `plugin.cpp`. Once landed, every cell in §5 becomes measurable.
- `lux-lattice.pc` **install on Spark.** Unblocks lattice GPU dispatch.
- **LP-204 GPU-shared resource scheduling.** One GPU per validator host serving multiple chain-VMs' verify queues fairly.
- **LP-205 FHE GPU runtime.** Wires `luxfi/fhe` and `accel/ops_fhe.go` into the consensus hot path for encrypted votes.
- **LP-206 TEE attestation for GPU verify.** H100 / GB10 SEV-SNP attestation of the GPU verify kernels so GPU-computed Merkle roots and verdicts carry a remote- attestation receipt.
- **CXL 3.0 racks.** Add cross-socket UMA semantics to the bench when CXL 3.0 hosts land in the bench cluster.

11 ABI Input Bounds (Normative)

Added 2026-06-05 to close Red Probe 8 (Probe 2 + Probe 9 cross-cutting): every GPU plugin implementing a batch verify or sign of `op_mldsa_*` or `op_slhdsa_*` MUST reject inputs exceeding the bounds below with `LUX_BACKEND_ERROR_INVALID_ARGUMENT`. The Go-side dispatch (`lux/accel/internal/capi`) translates this status into `accel.ErrInvalidArgument`, and consumer dispatch sites (`lux/crypto/mldsa`, `lux/crypto/slhdsa`, `lux/precompile/mldsa`, `lux/precompile/ai`) treat this as a hard error – they propagate, never silently fall back to CPU. The propagation discipline is the consensus-split firewall between GPU and CPU validators.

11.1 Per-element message length cap

For both `op_mldsa_verify_batch` / `op_mldsa_sign_batch` and the SLH-DSA siblings:

- Every `msg_lens[i]` satisfies `msg_lens[i] <= LUX_GPU_MLDSA_MSG_LEN_CAP` (= INT32_MAX − 2 = 0x7FFFFFFD). The −2 reserves headroom for a 2-byte FIPS 204 ctx prefix (`0x00 || ctxlen`) so a wrapper that prepends two bytes to the message can never overflow a `uint32_t` length field downstream.
- Every `msg_lens[i]` satisfies `msg_lens[i] <= LUX_GPU_SLHDSA_MSG_LEN_CAP` (also = INT32_MAX − 2). One source of truth across every plugin (CUDA / HIP / Metal), defined in `include/lux/gpu/backend_plugin.h`.

Any plugin observing `msg_lens[i]` above the cap MUST return `LUX_BACKEND_ERROR_INVALID_ARGUMENT` *before any memory allocation, kernel launch, or sponge absorb.* The cap is enforced on *public* input (verify pubkey, sign secret-key holder's caller) so the early reject is on data the side-channel adversary already controls – there is no information leak from the rejection branch.

11.2 Per-batch count cap

Both ML-DSA and SLH-DSA batched ops MUST reject `count > UINT32_MAX / kBatchMaxFactor` with `LUX_BACKEND_ERROR_INVALID_ARGUMENT`, where `kBatchMaxFactor` is the largest per-element multiplier in the orchestrator's device-buffer arithmetic. As of v14:

- ML-DSA: `kBatchMaxFactor = 24` (driven by the $k_L \cdot 4$ gamma1-sampling factor inside `mldsa_verify_orchestrator.cu`).
- SLH-DSA: `kBatchMaxFactor = 7680` (most conservative across the v14 batched ops; Red M-NEW-3).

This cap is independent of `LUX_GPU_*_MSG_LEN_CAP`; both gates fire on different overflow surfaces and both MUST be enforced before any device allocation.

11.3 Cumulative pool offset width

Red Probe 2 (2026-06-05): every batched op MUST stage per-element pool offsets as `uint64_t` – not `uint32_t`. The per-element *length* fields can remain `uint32_t` (they are capped by the per-element cap above), but the *cumulative* message cursor across the batch can exceed 2^{32} at modest batch sizes:

> With `cap = 0x7FFFFFFD` and `count = 3`, three elements each at cap-length produce a cumulative `msg_cursor[3] = 3 * cap = 0x17FFFFFF7`. Truncated to `uint32_t`, this becomes `0x7FFFFFF7` – inside element 1's region at `0x7FFFFFFD`. The kernel reads element 1's bytes when verifying element 3, producing *cross-element verdict contamination* – a non-deterministic verify output that splits consensus between GPU-equipped and CPU-only validators.

The fix (Red Probe 2 Option A) widens the orchestrator's `VerifyElem.pk_off,msg_off,sig_off,result_off` from `uint32_t` to `uint64_t` across both ML-DSA (landed in the v14 cycle as C-2) and SLH-DSA (landed 2026-06-05). The per-element *length* fields stay `uint32_t` for layout compactness; the cap above keeps them in range. The widened struct is the canonical wire layout across every backend (CUDA / HIP / Metal) – mismatches between host-stage and device kernel structs produce silent NVCC misalignment, so the layout invariant is hard-locked via the `_pad_<field>_len` trailer pattern (8-byte alignment, no holes).

11.4 Error propagation discipline (Probe 9)

The boundary between the C library's `LuxBackendError` sentinels and the Go-public `accel.Err*` sentinels MUST be mediated by `translateCapiError` (`lux/accel/session_c.go:22-41`). Every cgo-routed method in `lux/accel/ops_c.go` now wraps the `capi.*`-returning call through `translateCapiError` so the public surface emits `accel.ErrInvalidArgument` (and sibling `accel.Err*` sentinels) that consumers' `errors.Is` checks at:

- `lux/accel/ops/crypto/crypto_gpu.go:159` (SigMLDSA65)
- `lux/crypto/slhdsa/gpu.go:276, 479` (SLH-DSA verify/sign)
- `lux/crypto/mldsa/gpu.go:163-169, 281-287` (ML-DSA verify/sign)
- `lux/crypto/pq/mldsa/gpu/gpu_cgo.go:259-265, 380-386`
- `lux/precompile/mldsa/contract.go:401`
- `lux/precompile/ai/ai_mining.go:484, 537`

successfully detect as the hard-error case. Failure to wrap (Red Probe 9, found 2026-06-05) returns the `capi.ErrInvalidArgument` sentinel which is a *distinct `errors.New` value* from `accel.ErrInvalidArgument` – `errors.Is` returns false at the leaves, every M-1 consumer silently falls back to CPU, and the entire M-1 propagation chain landed in the v14 cycle is defeated. The end-state is the consensus-split shape every GPU/CPU asymmetry in the validator set creates.

The integration test at `lux/accel/integration_invalid_argument_test.go` (build-tag `lux_accel_real`) exercises this path through the real cgo boundary on a CUDA / Metal-equipped host, asserting `errors.Is(err, accel.ErrInvalidArgument) == true` after fault-injecting a shape-mismatched batch into the MLDSAVerifyBatch / SLHDSAVerifyBatch / BLSVerifyBatch entry points.

References

[1] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak reference, 2011. Round 3 NIST SHA-3 submission.

[2] Sean Bowe. BLS12-381: New zk-SNARK elliptic curve construction, 2017. Electric Coin Co. https://electriccoin.co/blog/new-snark-curve/.

[3] DPDK Project. DPDK programmer's guide, 2024. Kernel-bypass poll-mode driver. https://doc.dpdk.org/guides/prog_guide/.

[4] Hanzo AI. Hanzo crypto suite, 2026. Section 5.2: P3Q definition (EVM precompile at slot `0x012205`).

[5] Lux Industries. Corona / pulsar parity test, 2026. `lux/threshold/protocols/parity/parity_test.go`, 2026-06-03.

[6] Lux Industries. Lp-022: Zap wire protocol, 2026.

[7] Lux Industries. Lp-073: Pulsar two-round lattice threshold signing, 2026.

[8] Lux Industries. Lp-077: Round digest binding, 2026.

[9] Lux Industries. Lp-132: Quasar gpu execution adapter, 2026.

[10] Lux Industries. Lp-177: Bridge pq-only profile, 2026.

[11] Lux Industries. Lp-181: Magnetar threshold slh-dsa, 2026.

[12] Lux Industries. Lp-200: The lux zap stack, 2026.

[13] Lux Industries. Lp-201: Zap p2p – transport selector, 2026.

[14] Lux Industries. Lp-202: Zap pipelining and atomic unwind, 2026.

[15] Lux Industries. Lp-205: Pipelined quasar, 2026.

[16] Lux Industries. Lp-217: Cert profile modes, 2026.

[17] Lux Industries. Lp-218: Zap-native pq-secured rollups via p3q, 2026.

[18] Lux Industries. `luxfi/accel`: Gpu dispatch package, 2026. v1.1.9; CUDA / Metal / ROCm kernels for BLS pairings, ML-DSA NTT, secp256k1.

[19] Lux Industries. `luxfi/lattice`: Gpu lattice ops, 2026. v7.1.4; Corona and Pulsar NTT kernels.

[20] Lux Industries. Quasar 4-scheme matrix bench (spark + m1), 2026. `lux/threshold/protocols/parity/bench/matrix`, 2026-06-04.

[21] NIST. FIPS 203: Module-lattice-based key-encapsulation mechanism standard (ML-KEM), 2024. https://csrc.nist.gov/pubs/fips/203/final.

[22] NIST. FIPS 204: Module-lattice-based digital signature standard (ML-DSA), 2024. https://csrc.nist.gov/pubs/fips/204/final.

[23] NIST. FIPS 205: Stateless hash-based digital signature standard (SLH-DSA), 2024. https://csrc.nist.gov/pubs/fips/205/final.

[24] NVIDIA. NVIDIA GH200 Grace Hopper Superchip Architecture Whitepaper, 2023. Coherent CPU+GPU memory; precursor to GB10. https://resources.nvidia.com/en-us-grace-cpu/nvidia-grace-hopper.

[25] NVIDIA. nvidia-peermem (formerly nv_peer_mem), 2023. RDMA peer-memory client for in-tree drivers.

[26] NVIDIA. CUDA Unified Memory and Managed Memory (`cudaMallocManaged`), 2024. CUDA C++ Programming Guide §6.2.1.

[27] NVIDIA. GPUDirect RDMA Documentation, 2024. https://docs.nvidia.com/cuda/gpudirect-rdma/.

[28] NVIDIA. NVIDIA Blackwell Architecture Technical Brief, 2024. Compute capability sm_120; GB10 Grace Blackwell Superchip. https://resources.nvidia.com/en-us-blackwell-architecture/blackwell-architecture-technical-brief.

[29] Supranational. blst: Multilingual BLS12-381 signature library, 2024. C reference implementation; CPU baseline. https://github.com/supranational/blst.

[30] Alex Vlasov, Kelly Olson, Antonio Sanso, and Stefan Berger. EIP-2537: Precompile for BLS12-381 curve operations, 2020. Ethereum Improvement Proposal. https://eips.ethereum.org/EIPS/eip-2537.