Thread Safety at Low Precision on GPUs

A race-condition stress study across precision, architecture, and sync cost

fp64 → fp4 · Turing → Blackwell Ultra · CUDA + OpenMP

codeberg.org/srinathv/cuda-fp-race-stress · 2026-06-13

Why

Quantized model training & inference push work into **fp16 / bf16 / fp8 / fp4**. Lower precision changes *how race conditions manifest* — and whether you even notice them.

Questions:

- Do races behave differently across precision and GPU architecture?
- Do **atomics** always protect you? What do sync primitives **cost**?
- Can **compute-sanitizer** catch these on the newest Blackwell Ultra silicon?

Method

- **7 race patterns** (A–G) implemented as deliberate, minimal races + a safe twin
- **Precisions:** fp64, fp32, fp16, bf16, fp8 (e4m3, sw + native), fp4 (e2m1)
- **Architecture matrix:** Tesla **T4** (sm_75) · RTX **A6000** (sm_86) · **B300** Blackwell Ultra (sm_103)
- **CPU mirror:** OpenMP on AMD EPYC 9575F — same patterns, language with a barrier-rich model
- **Tooling:** cycle-cost microbenchmarks (`clock64`) + **compute-sanitizer** racecheck/synccheck/memcheck/initcheck

The pattern catalog

ID	Race	Observable corruption
A	Global RMW, no atomic	99.997% of updates lost
B	Shared-mem cross-warp read, no `__syncthreads`	**0.2%–24%** stale (arch-dependent)
C	Warp shuffle, wrong active-lane mask	none observable — sanitizer-only
D1/D3	fp8 byte RMW (sw + native `cuda_fp8.h`)	100% lost updates
D2	fp4 nibble-level byte RMW	200% nibble collision
E	Reduction missing `__syncthreads`	100% wrong blocks
F	Stream producer→consumer, no fence	structural (0 hits this run)
G	Tensor-core wmma split-K store	**100% wrong tiles** ★

Finding 1 — Race corruption is *structural*, not probabilistic

Races don't give "mostly right" answers with a little noise.

- Pattern A: survivors ≈ **SM count** (~33–48 of 1M), not a random fraction
- Pattern E: **100% of blocks** wrong, *every* run (fp16/bf16)
- Pattern D1/D3: **100%** lost updates; D2: **200%** nibble collisions

The corruption is determined by the **execution structure**, so it reproduces with eerie consistency — or hides with eerie consistency.

Finding 2 — Low precision *hides* races ⚠️

Pattern B stale-reads by precision (B300, N=1M):

Precision	Stale reads / run
fp64	477,517 (~46%)
fp32	260,861 (~25%)
fp16	**799**
bf16	**763**

Low-precision neighbours **collide to identical bytes**, so a stale read often returns the "right" value by accident. **The race is still there** — it's just invisible in output. Dangerous for quantized ML: the bug ships silently.

Finding 3 — Architecture dependence: Blackwell widens the window 119×

Pattern B fp32 stale-read rate, same kernel & N:

GPU	CC	fp32 stale rate
A6000	8.6	**0.2%**
B300	10.3	**~24%**

≈ **119×** more stale reads on Blackwell Ultra. The aggressive warp scheduler keeps more warps in flight, widening the gap between an un-synced write and a neighbour's read. *Same bug, two orders of magnitude more bite.*

Finding 4 — Tensor-core split-K accumulation race (Pattern G) ★

`C = A×B`, K split across 8 blocks; each `wmma::store_matrix_sync` writes the same C tile — **not atomic.**

```
racy (concurrent wmma store): wrong tiles = 32/32  (100%)
safe (workspace + atomicAdd): wrong tiles =  0/32
```

- Blocks **overwrite** instead of **sum** → C holds ≈16 instead of 128
- **No `atomicAdd` exists for matrix fragments**
- Correct split-K *must* use per-block workspace + reduction — exactly how cuBLAS / CUTLASS do it. Pattern G shows it's mandatory, not an optimization.

Cost — GPU sync primitives (B300, cycles/op)

Primitive	Cycles	vs barrier
`__syncthreads()`	32	1.0×
`atomicAdd` (shared, contended)	8	0.2×
`atomicAdd` (global, own addr)	103	3.2×
`__threadfence()`	281	8.8×
`atomicAdd` (global, contended)	**1968**	**61.5×**

A block barrier is nearly free; a **contended global atomic is ~62×** more.
Prefer a barrier + private accumulation over a shared atomic.

Cost — GPU vs CPU asymmetry

	GPU (B300)	CPU (EPYC, 8T)
Cheapest barrier/reduce	`__syncthreads` 32 cyc	`omp reduction` 11 µs
Contended single-loc update	atomicAdd **1968 cyc**	`omp atomic` / mutex **~3100 µs**
contended ÷ barrier	**~62×**	**~275×**

The "privatize + merge once" lesson holds on both — but the penalty for a contended shared lock is **~4× larger on the CPU**. (CPU lock-free CAS loop was the *slowest* mechanism at 365×: lock-free ≠ free.)

Atomics are *not* always enough

Failure	The atomic is correct…	…but
ABA	`atomicCAS` swaps fine	value restored between load-next & CAS
Composability	`atomicSub` decrements fine	`load` + `sub` isn't atomic → counter hit **−64,448** (64.5M negative transitions)
Ordering	counter value correct	guarded data invisible without `__threadfence`
Sub-word	—	no `atomicAdd` for fp8/fp4

✅ Atomics protect a **single-variable RMW**. ❌ They don't compose, order, or cover sub-word types.

Mutexes — when you *do* need a critical section

A mutex buys what atomics can't: **region atomicity** + **acquire/release ordering**. At a price (~270× a reduction) and with hazards:

- **deadlock** (lock ordering / ABBA) · **livelock** · **starvation** · **priority inversion** · **convoying**
- **contention** — our own incident: 60 threads on 30 cores drove **load-avg 45**, starved `sshd`, wedged the host
- **GPU anti-pattern:** no warp fairness → a descheduled lock-holder deadlocks; use **barriers + atomics + workspace reduction** instead

Full review: `docs/mutex_concepts_and_issues.md`

compute-sanitizer on B300 — the toolkit story

Toolkit	Sanitizer	B300 (CC 10.3)
CUDA 12.8	2024.x	`Device not supported`
CUDA 12.9.2	2025.2.1	**still** `Device not supported`
CUDA 13.3	**2026.2.0**	✅ **works**

12.9 added sm_103 *compiler* support — **not** sanitizer *device* support.
Lesson: verify sanitizer device support empirically; release-note "Blackwell support" ≠ every tool covers CC 10.3.

compute-sanitizer — hazard *exists* everywhere; *bite* is local

racecheck errors (Pattern B), `--n 65536`:

T4 (7.5)	A6000 (8.6)	B300 (10.3)
2,097,160	2,097,160	**2,097,160**

Identical across architectures — the count comes from the *static access pattern*, not timing. Contrast Finding 3 (observable bite swings 119×).

- **synccheck:** 64 Pattern C errors — *no observable corruption on any GPU*
- Global races (A, D, F, **G**) → **0 sanitizer hits** (racecheck = shared-mem only)

> A race that "never reproduces" on Turing is the *same 2.1M-hazard bug* that corrupts 24% of reads on Blackwell.

Takeaways

1. **Race corruption is structural** — it reproduces (or hides) with consistency.
2. **Low precision hides races** — collided bytes mask real hazards. *Most dangerous for quantized ML.*
3. **Architecture matters** — Blackwell's scheduler widens windows up to **119×**.
4. **Atomics aren't enough** — ABA, composability, ordering, sub-word types.
5. **Prefer barrier + privatize** over contended atomics (62× GPU / 275× CPU).
6. **Tooling lags hardware** — sanitizer needed **CUDA 13.x** for B300; and it only sees *shared-memory* races.

Repo & artifacts

codeberg.org/srinathv/cuda-fp-race-stress

`stress_race.cu` (A–G) · `stress_race_cost.cu` · OpenMP mirrors

`results/` — per-arch runs + sanitizer reports

`docs/` — mutex review · sanitizer runbook · these slides

Hardware: T4 sm_75 · A6000 sm_86 · B300 sm_103 (Blackwell Ultra, CUDA 13.3)

Next: Pattern G2 — tcgen05 tensor-memory guardrails (`-Xptxas -g-tmem-access-check`)