



False Sharing

Private counters, shared cache lines | Practical 4

Ömer Gölcük — Sabancı University

Course lead: Prof. Dr. Özcan Öztürk

Count marked records

Each worker reads a disjoint slice of one input array.

Worker 0

1	0	1	1
---	---	---	---

Partial count: 3

Worker 1

0	1	0	1
---	---	---	---

Partial count: 2

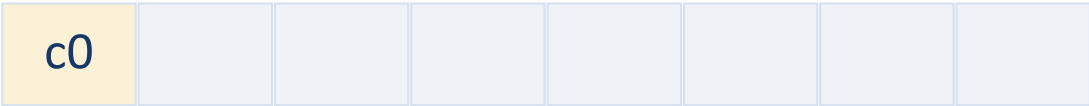
Final count: $3 + 2 = 5$

Each element is read exactly once, however the work is split.

Where each worker writes

The three atomic modes use the same update function.

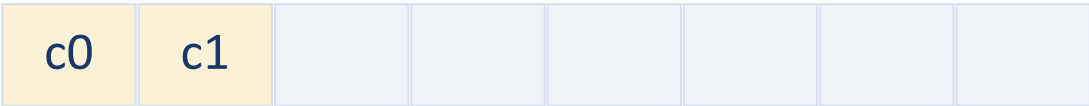
Shared



Same counter

T0 and T1 → c0 at byte offset 0

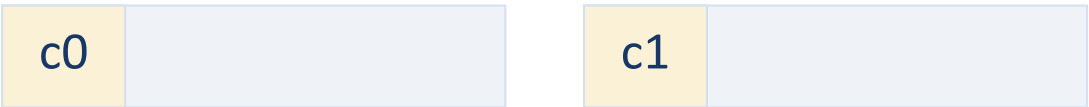
Packed



Same line

T0 → c0 at 0; T1 → c1 at 8

Separated



Different lines

T0 → c0 at 0; T1 → c1 at 64

[1, 2] Two workers; 8-byte counters; assumed 64-byte coherence lines. Code checks line-size reports and prints target addresses.

Why the sharing is false

Different variables, but one cache line is kept coherent.



Writable ownership can move even when the variables differ.

Separating counters removes this shared-line relationship.

[1] Simplified invalidation-based example. Ownership moves when a different core writes.

The two update kernels

count_atomic — shared, packed and separated

```
for (std::size_t i = 0; i < n; ++i)
    counter->fetch_add(flags[i], std::memory_order_relaxed);
```

Relaxed still means atomic; it does not order unrelated data.

count_local — sum first, publish once

```
std::uint64_t sum = 0;
for (std::size_t i = 0; i < n; ++i)
    sum += flags[i];
counter->store(sum, std::memory_order_relaxed);
```

[2] Exact kernel bodies from sharing.cpp. Counter is `std::atomic<std::uint64_t>`; even a zero flag issues an atomic update.

A controlled parallel comparison

Packed vs separated: same input, atomic kernel and worker CPUs.



Report launch skew; this interval excludes creation and joins.

Warm up each mode; randomize mode order; check every counter.

[3] External perf collection covers input setup, warm-up, thread management and verification.

Separating layout and algorithm effects

One worker

Packed and separated use the same address and atomic loop.

No competing core: compare any difference with run-to-run variation.

Separated counters

With two workers, two active lines replace one shared line.

The atomic update count is identical; only the active footprint changes.

Local accumulation

Each worker publishes once; the final sum is unchanged.

This also changes instruction mix and may enable vectorization.

The local version is not a replacement for an exact live progress counter.

[1, 2] Fixed 512-byte counter allocation in every mode. Local accumulation preserves the final result; intermediate values stay private.

Interpreting the live results

Timing plots	Compare parallel completion span and full trial time. Show repeated runs, medians and launch-skew warnings.
Paired speedup	Packed time ÷ separated time in the same program run. A local-accumulation speedup is not a padding-only effect.
Source + assembly	Locate <code>count_atomic</code> in both profiles; inspect the same loop. Default perf inheritance includes new workers; do not disable it.
Counter evidence	Whole-command counts include setup, warm-up and checks. Aggregate thread cycles are not elapsed wall-clock time.

[1–4] General cache events do not measure ownership transfers. Hardware-counter access and perf c2c support are host-dependent.

References

[\[1\] Linux kernel documentation — False Sharing](#)

Coherence granularity, cache-line contention and mitigation.

[\[2\] C++ working draft — Atomic ordering and data races](#)

Atomic read-modify-write operations; relaxed ordering.

[\[3\] Linux perf tutorial](#)

Counting, worker inheritance, sampling and source annotation.

[\[4\] perf-c2c\(1\) manual page — optional](#)

Cache-to-cache analysis; sampling support depends on the processor.

Teaching example: Ömer Gölcük. Original implementation and diagrams; complete commands and documentation links accompany the lecture.

Thank you!



**Co-funded by
the European Union**



EuroHPC
Joint Undertaking



EuroCC 3 has received funding from the [European High-Performance Computing Joint Undertaking \(JU\)](#) under Grant Agreement No. 101306701. The JU receives support from the [European Union](#)'s Digital Europe Programme and Germany, Albania, Austria, Belgium, Bosnia and Herzegovina, Bulgaria, Croatia, Cyprus, Czechia, Denmark, Estonia, Finland, France, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Montenegro, the Netherlands, North Macedonia, Norway, Poland, Portugal, Romania, Serbia, Slovakia, Slovenia, Spain, Sweden, Türkiye, and Kosovo.

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or EuroHPC Joint Undertaking. Neither the European Union nor the EuroHPC Joint Undertaking can be held responsible for them.