How Many Cycles to Access L1 Cache and Shared Memory on NVIDIA GPUs?

Wenbo Han

Master in High Performance Computing
University of Luxembourg

9 July 2026

Introduction

Today's reality

GPUs run LLMs, HPC simulations, scientific computing. Almost all workloads run on **NVIDIA GPUs**.

Unfortunately

Memory-intensive applications constantly hit the **L1 cache** and **shared memory** — yet we don't know what it costs!

If we know it...

- We can **quantify** optimizations — know exactly what a conflict costs, and whether it's worth avoiding
- We can **simulate GPUs accurately** — and design better hardware

This work: we measure the exact cost — and find it follows a simple, hidden rule.

UNIVERSITÉ DU LUXEMBOURG

Two Memory Spaces: Shared and Global

GPUs have two very common memory spaces:

- **Shared memory:** fast, small
- **Global memory:** large, slow — but data can be cached in the **L1 cache**
- Shared memory and the L1 cache **share the same hardware**

Global → L1 Cache → register

```
__global__ void kernel(int* g_data){
  int r = g_data[threadIdx.x];
  // global -> register
}
```

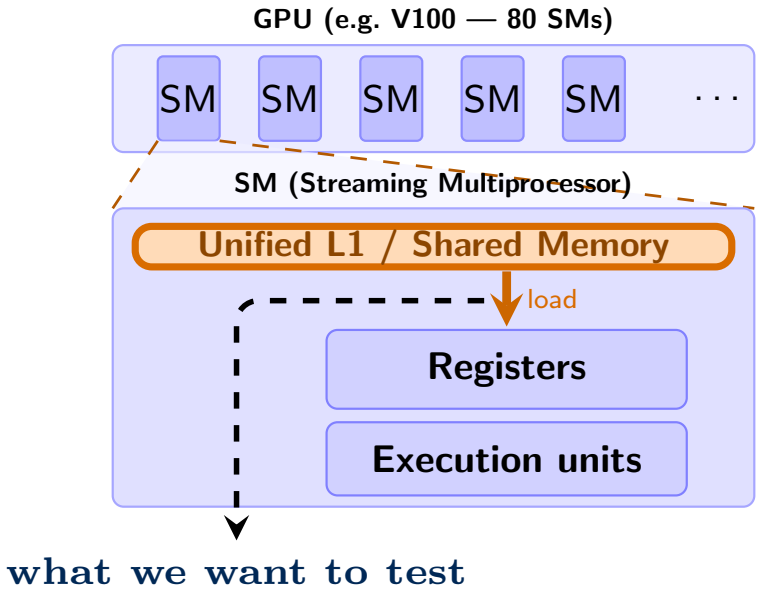
Shared → register

```
__global__ void kernel(){
  __shared__ int s[256];
  int r = s[threadIdx.x];
  // shared -> register
}
```

UNIVERSITÉ DU LUXEMBOURG

GPU Background

- A GPU has many **SMs** (Streaming Multiprocessors)
- Each SM runs **warps** of 32 threads and one instruction drives all 32 threads in lockstep
- A load moves data from **Unified L1 / Shared Memory** into **registers**
- ⇒ **one instruction can generate up to 32 memory accesses!**

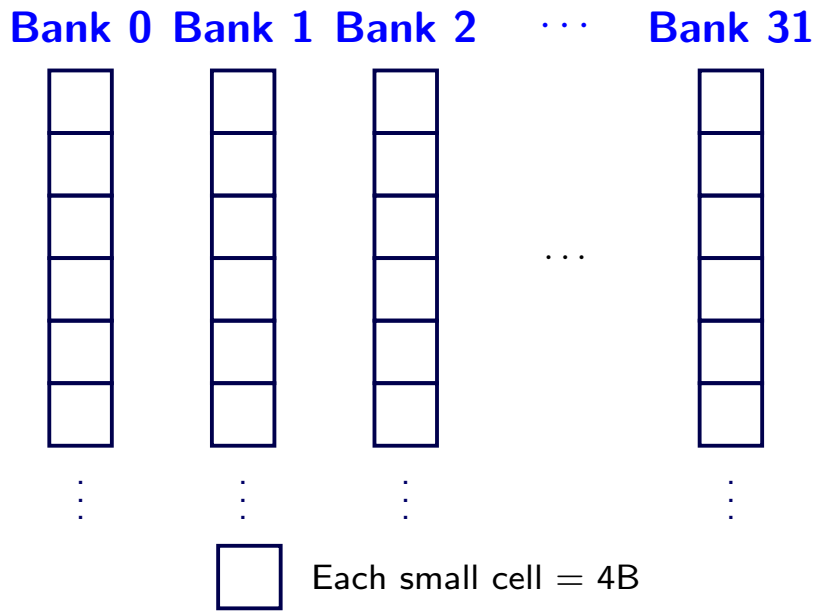


UNIVERSITÉ DU LUXEMBOURG

Memory Banks

The actual storage is **32 banks** — the *same* 32 banks serve as both **L1 cache** *and* **shared memory** (*unified*).

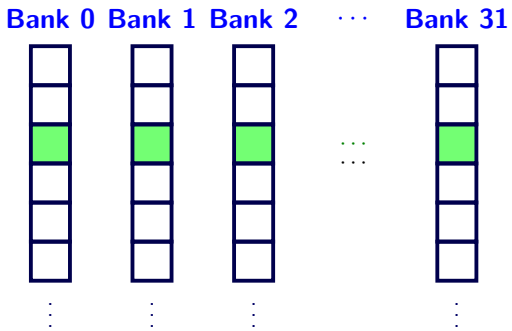
- **32 banks** can be seen as 32 columns, each has many **4B units**.
- Basically, a **warp (32 threads)** accessing memory is just accessing **banks**.



UNIVERSITÉ DU LUXEMBOURG

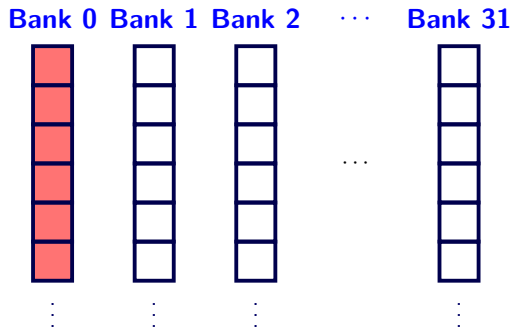
Bank Conflict

- **Mechanism:** for each bank, at any moment, **only one unit** can be accessed; to read multiple units, they must **queue up**.
- **Bank conflict:** multiple threads in a warp access **different units of the same bank**.
- **N-way bank conflict:** the number of different units accessed in the busiest bank is N



Best: same row — 1 per bank, **no conflict**

We call this **1-way bank conflict**



Worst: same column, **all 32 collide**

We call this **32-way bank conflict**

UNIVERSITÉ DU LUXEMBOURG

How We Measure the Cost

A controlled experiment — one clean data point per conflict degree:

1. **Generate:** 32 kernels — kernel N forces an exact N-**way bank conflict** $(N = 1 \ldots 32)$
2. **Isolate:** keep only the **one conflicting load**, wrapped in two clock reads
3. **Measure:** `CLK` before & after = the **exact cycles** of that one instruction

```
A = CLK()
LOAD INSTRUCTION
B = CLK()

COST = B - A
```

UNIVERSITÉ DU LUXEMBOURG

Cost of a 4-Byte Shared Load

- **No conflict (1-way):** base cost = **24 cycles** on A100
- **Each extra conflicting way adds exactly $+2$ cycles**

conflict	1-way	2-way	3-way	$\cdots$	n-way
cycles	24	26	28	$\cdots$	$24 + 2(n-1)$

$$\text{cycles} = 24 + 2\,(n-1)$$

n = n-way conflict in the busiest bank

8-Byte Loads: Split Into Two Batches

Key finding: an 8-byte load is served in **two batches** of 16 threads:

batch 0 = threads 0–15 **batch 1** = threads 16–31

- Bank conflicts happen **only within a batch** — never across
- Even if threads 0–15 all hit **bank 0** and threads 16–31 all hit **bank 1** (different banks!), the two batches **still fully serialize**
- The two batches' costs simply **add up**

$$\text{cycles} = 28 + 2(n_0 - 1) + 2(n_1 - 1)$$

n_0, n_1 = conflicts number of busiest-bank in each batch

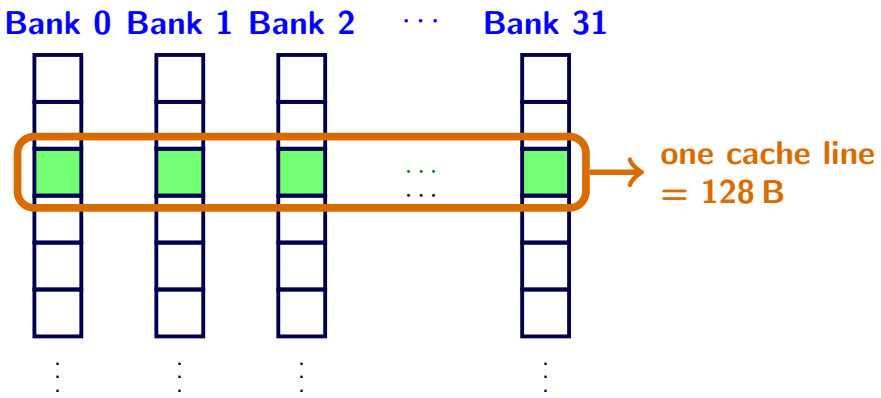
Likewise, a **16-byte** load splits into **4 batches** — the cost is a sum of **four** terms.

UNIVERSITÉ DU LUXEMBOURG

Global Loads Go Through the L1 Cache

A **global load** is served by the **L1 cache** (assume it **hits**):

- Data is organized in **cache lines** — 128 B each
- Each line stores its **high-order address bits** as a **tag**
- Checking tags **costs cycles**: the L1 does **4 tags / cycle**



So touching L distinct cache lines adds an extra $\left\lfloor \frac{L-1}{4} \right\rfloor$ cycles

— one extra cycle for every 4 lines.

Global Loads: Bank Conflicts Identical to Shared

A global load's **bank-conflict cost** is **exactly the same as shared memory** — same $+2$ per way, and the **same batching** by width (4B→1, 8B→2, 16B→4 batches).

The only extra is the **cache-line tag term**:

$$\textbf{4 B}: \quad 32 + \left\lfloor \frac{L-1}{4} \right\rfloor + 2(n-1)$$

$$\textbf{8 B}: \quad 36 + \left\lfloor \frac{L-1}{4} \right\rfloor + 2(n_0-1) + 2(n_1-1)$$

$$\textbf{16 B}: \quad 44 + \left\lfloor \frac{L-1}{4} \right\rfloor + 2\sum_{i=0}^{3}(n_i-1)$$

Conclusion & Future Work

1. Base cost depends on the **memory space** and the **access width**.
2. Each bank conflict $\rightarrow$ **+2 cycles**; for global, every **4 cache lines** $\rightarrow$ **+1 cycle**.
3. Wide accesses are **split into batches**.

Future work: other architectures · reverse-engineer more mechanisms · build them into a **simulator**.

UNIVERSITÉ DU LUXEMBOURG