

# cuxray: Optimize Kernels Without a GPU

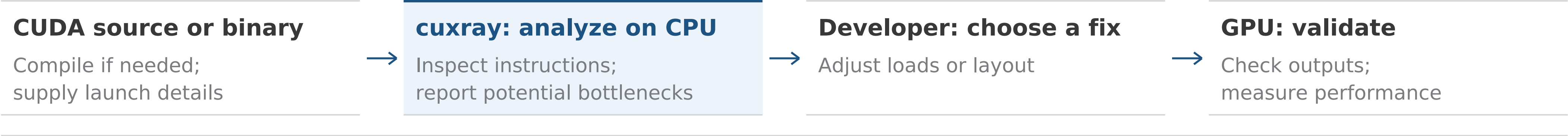
Kareem Fareed



[Code + methods](#)

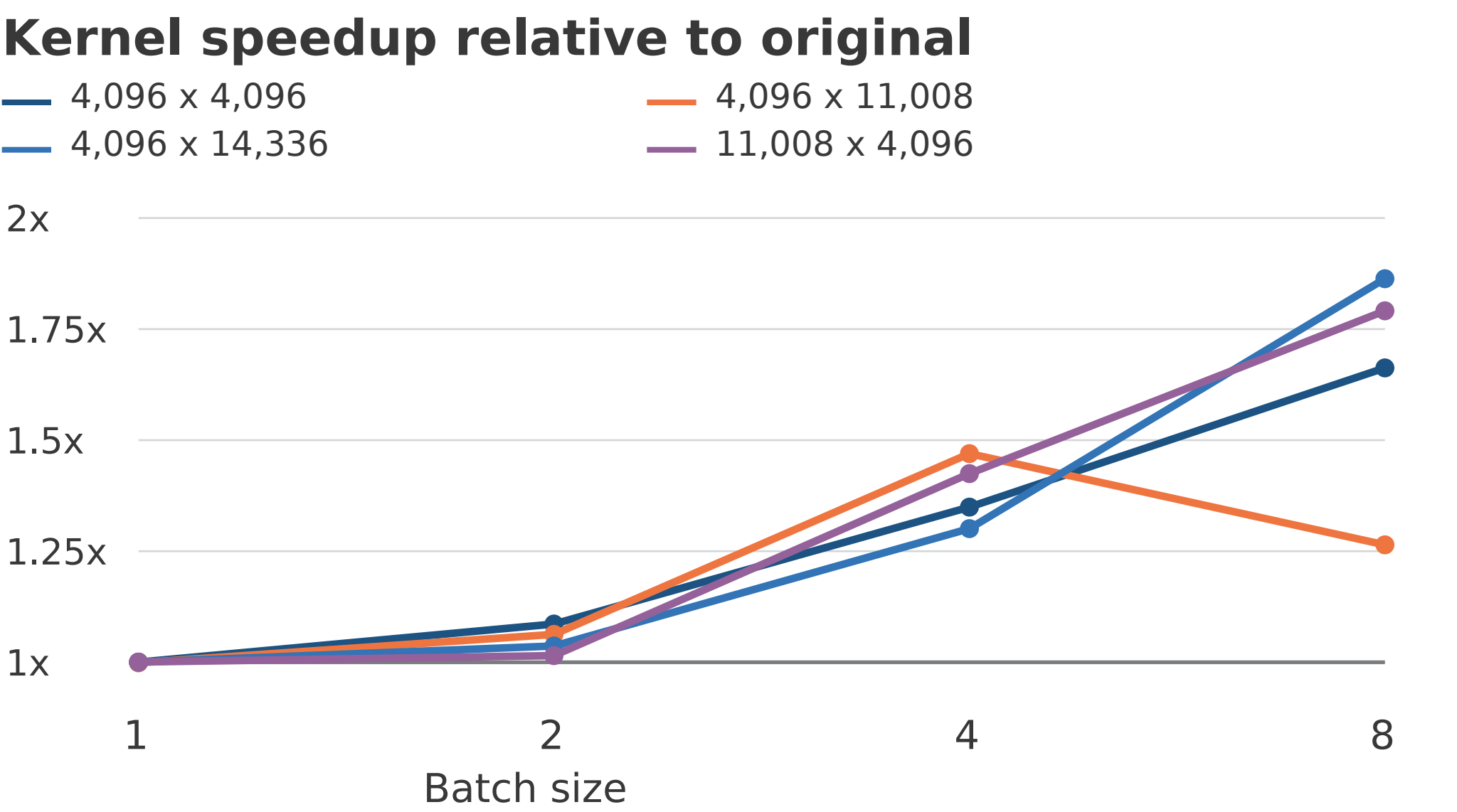
## HOW TO USE CUXRAY

Source code does not always match the instructions emitted by the CUDA compiler. `cuxray` inspects compiled binaries on a CPU to find potential bottlenecks before spending GPU time.



## 01 Binary-guided optimization of W4A8 decode GEMV

ptxas emitted narrower shared-memory loads than the source intended. Vector loads followed by a solved swizzle removed the measured bank overhead, yielding up to **1.86x kernel speedup** versus the original implementation.



**Shared-memory load wavefronts**  
Normalized to the conflict-free variant

|                  | B2  | B4  | B8  |
|------------------|-----|-----|-----|
| Original         | 2.0 | 4.0 | 4.0 |
| Vector loads     | 1.0 | 2.0 | 2.0 |
| + Solved swizzle | 1.0 | 1.0 | 1.0 |

Static bank patterns matched all 12 Nsight profiles.

**Figure 1.** Four matrix shapes and batches 1/2/4/8 on A100-SXM4-40GB; 15 randomized timing blocks per point with streaming weights. All 16 tested outputs were bitwise identical across three variants. Separately, a `cuxray`-guided A100 rework of our W4A16 GEMV ran **1.15x faster than vLLM's Marlin W4A16** at 4096×4096, batch 1, on the same GPU; Marlin was faster at larger shapes and batches.

## 02 Find spills before you run

\$ `cuxray demo` Try this bundled example without a GPU

**SOURCE**

**40 accumulators**  
compiled with a 32-register cap

```
#define ACC 40
float acc[ACC];
```

`spill.cu:14`  
cuxray attributes the spill finding to a source location.

**CUXRAY FINDING**

**Register spills**

154

155

→ store instructions

load instructions

Registers

Local memory

store

load

Static instruction counts; execution frequency is not measured.

**NEXT EXPERIMENT**

**Relax the cap**

32 → 64

→ Candidate register cap

**Recompile + compare**

- Fewer spills?
- Occupancy tradeoff?
- Registers used?

THEN USE GPU TIME      Check correctness      →      Measure both versions