CS554 Final Project Report

Manh Tien Nguyen

December 6, 2025

1 Introduction

This project aims to study Nvidia PTX memory model by implementing classical litmus tests as inline PTX inside CUDA kernels. The focus was on three standard patterns:
- Store Buffering (SB)
- Message Passing (MP)
- Load Buffering (LB)

For each test, I want to observe weak behaviour across different setting, specifically semantic strength(weak, relaxed, rel/acq,...), scope(CTA, GPU, SYS), thread placement,...

The goal was:
- To see which weak behavior actually occurs on real hardware
- Compare experimental results with the current PTX ISA document and formal analyses of the PTX memory model
- Understand why certain settings could dramatically change the observed outcomes.

2 Litmus test implementation

The code is located at: *https://github.com/manhntm3/CS554FinalProject.git*

I implemented three canonical litmus test patterns as three separate CUDA kernels. To isolate hardware behavior and avoid compiler-induced instruction reordering, all memory operations were executed using **volatile** inline PTX assembly. The implementation includes options for inter-block and intra-block configuration, as Nvidia forces memory instruction based on scope (.gpu and .sys).

WEAK

```
// P0: Store X, Load Y
st.global.u32 [x], 1;
r0 = ld.global.u32 [y];

// P1: Store Y, Load X
st.global.u32 [y], 1;
r1 = ld.global.u32 [x];
```

RELAXED GPU

```
// P0: Store X, Load Y
st.global.relaxed.gpu.u32 [x], 1;
r0 = ld.global.relaxed.gpu.u32 [y];

// P1: Store Y, Load X
st.global.relaxed.gpu.u32 [y], 1;
r1 = ld.global.relaxed.gpu.u32 [x];
```

Figure 1: Store Buffering (SB) Litmus Test PTX: Weak vs Relaxed GPU-scope

Variants testing atomic operations and fence-based ordering were also included to validate the Nvidia PTX memory consistency model.

RELEASE/ACQUIRE GPU

```
// P0 (Producer)
st.global.relaxed.gpu.u32 [data], 1;
st.global.release.gpu.u32 [flag], 1;

// P1 (Consumer)
r_flag = ld.global.acquire.gpu.u32 [flag];
r_data = ld.global.relaxed.gpu.u32 [data];
```

FENCE SC GPU

```
// P0 (Producer)
st.global.relaxed.gpu.u32 [data], 1;
fence.sc.gpu;
st.global.relaxed.gpu.u32 [flag], 1;

// P1 (Consumer)
r_flag = ld.global.relaxed.gpu.u32 [flag];
fence.sc.gpu;
r_data = ld.global.relaxed.gpu.u32 [data];
```

Figure 2: Message Passing (MP) Litmus Test: Release/Acquire vs Fence SC

3 Experimental Results

I ran the program for 5M iterations and one or two active threads per grid. Hardware: Nvidia Ada (CUDA 12.9, sm 89, PTX 8.9).

3.1 First try (scalar version)

Table 1: Litmus Test Results (5M Iterations)

Variant	**Message Passing (MP) Raw**		**Store Buffer (SB) Raw**	
	Weak Count	Weak %	Weak Count	Weak %
Inter-Block WEAK	0	0.00%	0	0.00%
Inter-Block RELAXED (CTA)	–	–	0	0.00%
Inter-Block RELAXED (GPU)	14,669	0.29%	1,260,147	25.20%
Inter-Block RELAXED (SYS)	9,248	0.19%	1,258,053	25.16%
Inter-Block ACQ/REL (GPU)	726,303	14.53%	1,165,585	23.31%
Inter-Block ACQ/REL (SYS)	312,353	6.25%	–	–
Inter-Block FENCE SC (GPU)	973,503	19.47%	821,159	16.42%
Inter-Block FENCE SC (SYS)	0	0.00%	189	0.00%

In this initial implementation, a scalar reuse strategy was used. Specifically, I've tried to reuse the variable in every iteration(i.e: litmus variables x and y were reset to zero every time). This turns out to be wrong and I observed several weak behaviors that are even prohibited in the documentation. I've been stuck in this implementation for a while, trying to implement a barrier that synchronize thread execution. However this type of barrier fails to address L1/L2 cache visibility. Consequently, the reuse of the memory location magnifies the impact of caching and of any residual state, causing stale data from iteration i to be read in iteration i+1, resulting in false weak memory behavior.

3.2 Second try (array version)

SB has been implemented as array-strided version, which guarantee variable initialization in global memory before every iteration. For SB array version, it's well aligned with the documentation. Some weak behaviours are definitely stronger in the experiment; however the setup include Default(Weak) and Relaxed CTA-scope variants still exhibited high rates of weak behavior, confirm that without explicit fencing, the hardware aggressively buffers stores and reorders memory operations

3.3 Store Buffering litmus test

The Store Buffering test results strictly adhered to the architectural expectations defined in the PTX ISA. Significant rates of weak behavior (r0=0, r1=0) were observed only in the WEAK (Default) and RELAXED_CTA variants.

3.4 Message Passing litmus test

No weak behaviors (r= 1, r_data = 0) were observed across any of the implemented variants, including the explicitly weak and relaxed config.

3.5 Load Buffering litmus test

Nvidia document mentioned "no thin-air" result from the test[1, 2]. Behaviour x=0,y=0 dominates the test and no weak behaviour was observed.

Table 2: Memory Model Litmus Test Results (5M Iterations)

Variant	**SB Array**		**MP**	
	Count	%	Count	%
Inter-Block WEAK	4,374,988	87.50%	0	0.00%
Inter-Block RELAXED (GPU)	0	0.00%	0	0.00%
Inter-Block RELAXED (SYS)	0	0.00%	0	0.00%
Inter-Block RELAXED (CTA)	4,375,000	87.50%	0	0.00%
Inter-Block ACQ/REL (GPU)	0	0.00%	0	0.00%
Inter-Block FENCE SC (GPU)	0	0.00%	0	0.00%
Inter-Block FENCE SC (SYS)	0	0.00%	0	0.00%

3.6 Challenges and lesson learned

- Nvidia Hardware is stronger than the spec. Nvidia could enforce implicit ordering stronger than the PTX specification required.
- Fences do work; however, barriers don't(or very hard to implement one). Here I mean fence is a hardware intrinsic, while barrier is software defined.
- Unlike the array-based strategy, which successfully isolated iterations via memory allocation, the scalar reuse strategy exposed the difficulty of flushing cache states between iterations. Future work will focus on refining the scalar implementation, as its memory reuse pattern provides a more rigorous stress test of the GPU's cache coherence protocols compared to the strided array approach.

References

[1] NVIDIA Corporation, "PTX ISA :: CUDA Toolkit Documentation," 2025.

[2] D. Lustig, S. Sahasrabuddhe, and O. Giroux, "A formal analysis of the nvidia ptx memory consistency model," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 257–270. [Online]. Available: https://doi.org/10.1145/3297858.3304043