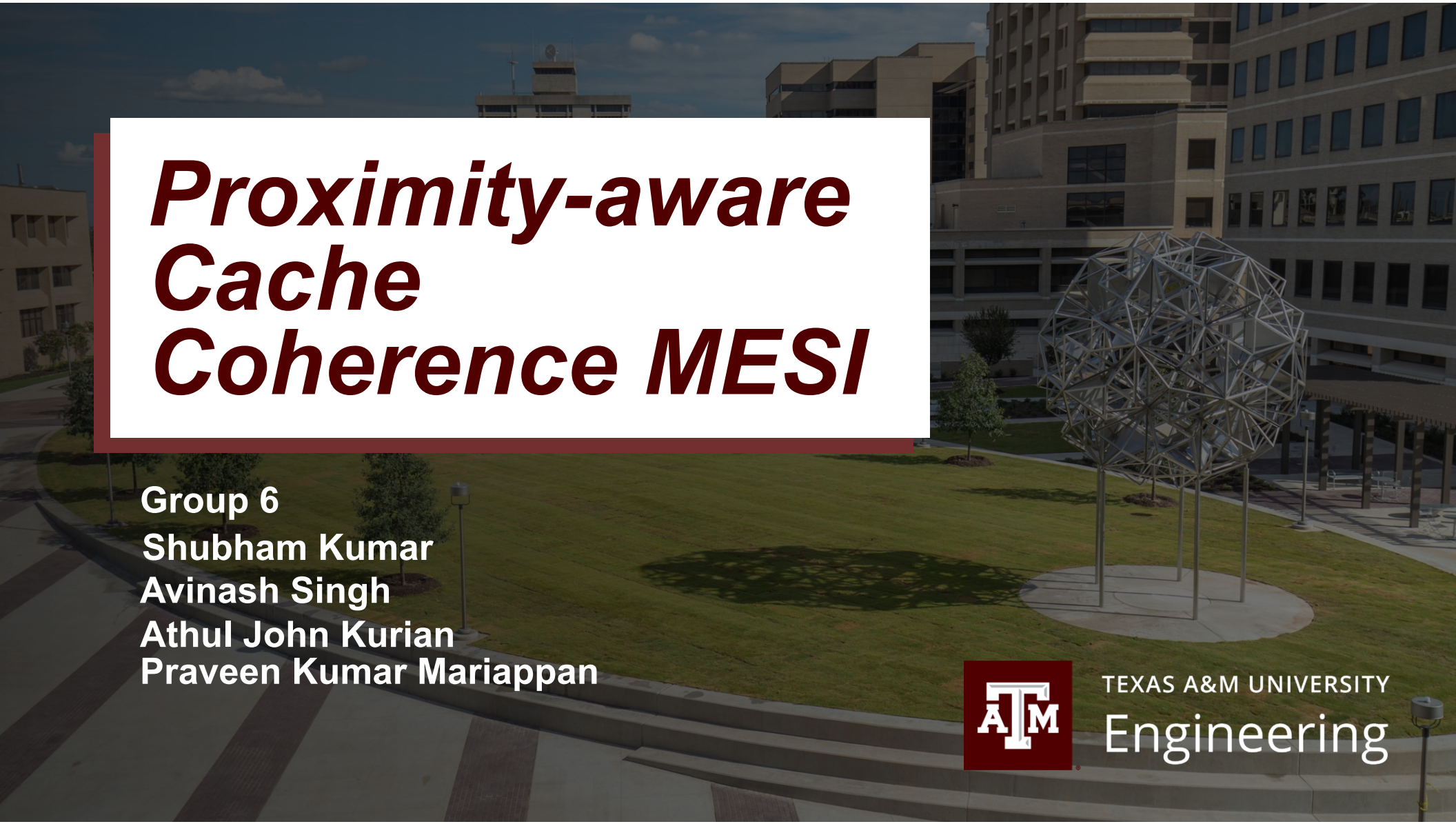


Proximity-aware Cache Coherence MESI

Group 6

Shubham Kumar

Avinash Singh

Athul John Kurian

Praveen Kumar Mariappan

TEXAS A&M UNIVERSITY

Engineering

Introduction

- Modern processors have multiple cores, each with private caches — keeping data consistent across them is a critical challenge.
- Cache coherence protocols like MESI ensure that all cores have a consistent view of memory, preventing bugs and stale data access.
- Simulating MESI helps us understand real-world coherence behavior, including memory traffic, invalidations, and latency.
- This project builds a custom directory-based MESI simulator, providing a foundation to explore proximity-aware enhancements and MOESI extensions.

Introduction

- Implemented a **directory-based architecture** that manages coherence using centralized metadata.
- Collected performance metrics like cache hits, memory accesses, invalidations, and latency.
- Designed the simulator to serve as a baseline for exploring proximity-aware coherence.
- 📍 *Why proximity-aware?* Directory protocols know all sharers — enabling **nearest-neighbor forwarding** to reduce memory traffic and latency

Background

- Many works have compared performance of Cache Coherence protocols like MSI, MESI,MOESI.
- Traditional MESI and MOESI protocols enforce coherence via directory or bus snooping.
- MOESI improves reuse with Owned state, reducing memory traffic.
- Prior research introduces locality-aware forwarding in mesh-based CMPs.
- This work builds on proximity-aware coherence design.

Prior Work

- SGI Origin and DASH used directory-based coherence for scalable DSM systems.
- Directory caches were introduced to reduce directory access latency and memory overhead.
- CC-NUMA and COMA architectures minimized latency using spatial data placement.
- Token Coherence decoupled correctness from performance to reduce memory access.
- Cooperative caching leveraged neighboring caches to reduce off-chip accesses.
- CMP directory-based protocols assumed fixed directory placement, not proximity.
- In-network coherence moved control into routers for latency optimization.

Current Work

- Many different approaches are ongoing in Cache Coherence protocols to handle latency, memory accesses and other performance metrics.
- **Hybrid and Adaptive Coherence Protocols :** Dynamically switch between coherence actions based on workload and uses the closest sharer to reduce cache-to-cache latency.
- **Peer-to-Peer Cache Forwarding :** enables the home node to select the nearest valid sharer from the directory and forward data directly to the requester, reducing memory access latency and coherence overhead—an early form of proximity-aware coherence.

Current Work

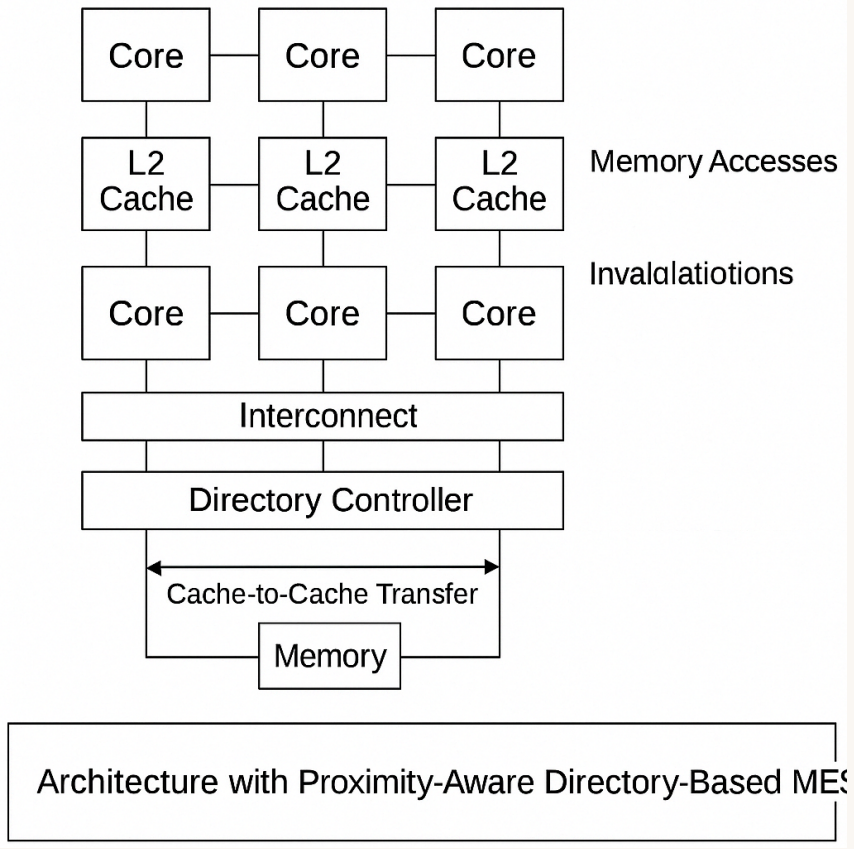
- **Timestamp-Based Coherence (Library Cache Coherence – LCC) :** Shift toward **temporal coordination** instead of spatial invalidation using auto-expiring timestamps instead of multicast invalidations. Achieves 1.85× lower latency vs MESI directory, even with a simple timestamp policy.

Overview

- Implemented three coherence protocols in a custom C++ simulator following paper “Proximity-Aware Directory-based Coherence for Multi-core Processor Architectures” :
    1. Baseline MESI (Directory-based)
    2. MOESI (adds Owned state for reuse)
    3. Proximity-Aware MESI (forwards from nearest sharer)
- Used a 4x4 2D mesh of 16 cores with private L2 caches
- Custom Benchmarks: private, shared read, write-sharing, hybrid, distance-aware, etc.
- Comparison Metrics: Memory Accesses, Cache-to-Cache Transfers, Latency, Effective Hit Rate

System Architecture

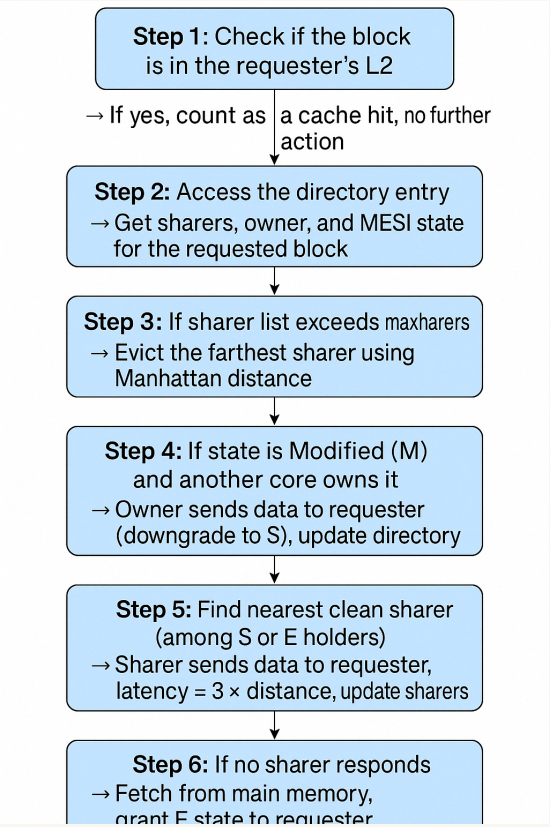
- Each core has a private L2 cache, forming a tiled multi-core architecture.
- A centralized Directory Controller maintains MESI states and sharer lists for cache blocks.
- On a cache miss, the directory:
  - Identifies sharers of the requested block.
  - Selects the closest sharer to minimize data transfer latency.
- The selected sharer performs a cache-to-cache transfer directly to the requester.
- If no sharer has the block, data is fetched from main memory.
- Invalidation messages are sent via the interconnect to maintain coherence on writes.
- All communication (data, invalidations, memory access) flows through the on-chip interconnect.



Architecture with Proximity-Aware Directory-Based MES

Proximity Aware Read Handling

- Each core is assigned (x, y) mesh coordinates for 2D distance calculation.
- Manhattan distance is used to measure proximity between requester and sharers.
- Manhattan Distance = $|x1-x2|+|y1-y2|$
- Directory tracks all current sharers for every block (via dirEntry.sharers).
- During a read miss:
  - Iterate over all sharers.
  - Skip the requester itself.
  - Select the sharer with the smallest distance who still has the block.
- Forward data from the nearest valid sharer directly to the requester.
- Latency is calculated as 3 × distance, modeling network hop cost.
- effectiveHits++ and cacheToCacheTransfers++ are incremented when a sharer responds.
- Fallback to memory occurs only if no sharer has the block.



Modelling Differences and Expectations

Aspect	Baseline MESI	Proximity-Aware MESI	MOESI	Expectations
Clean Sharer Forwarding	Not supported	Forward from nearest clean sharer	Forwarding from 'Owned' state	————
Sharer Selection	N/A	Nearest by Manhattan distance	N/A	————
Latency Modeling	Fixed	Distance-based (e.g., dist × 3)	Fixed	————
Cache-to-Cache Transfers	Only from dirty owner	From dirty owner + clean sharers	From dirty owner + O state	MESI < Proximity aware MESI ; MOESI should have more than MESI and comparable with proximity
Extra States	4 (M, E, S, I)	Same as MESI	5 (adds Owned 'O')	————
Write-back Behavior	M writes back to memory	Same	O retains clean copy and shares	————
Effective Hit Rate	Lower	Higher (reuse of clean sharers)	Higher (reuse of 'O' state)	MESI < MOESI && Proximity aware MESI
Overall Latency	Higher	Lower	Moderate	MESI>Proximity aware MESI MOESI should be better than MESI but not proximity aware MESI

Metrics Tracked

- **Cache-to-Cache Transfers** – Count of data transfers served by another core's cache.
- **Memory Accesses** – Number of requests that went to main memory.
- **Invalidation Messages** – Total invalidates sent to other sharers on a write.
- **Cache Hits** – Reads that hit in the local L2 cache.
- **Effective Hits** – Reads that hit in either local or remote cache.
- **Cache Hit Rate** – Percentage of reads that hit locally.
- **Effective Hit Rate** – Percentage of reads served by either local or remote caches.
- **Total Simulated Latency** – Sum of all latency incurred by memory transactions.

Methodology

Objective: Compare three coherence protocols — Base MESI, MOESI, and Proximity MESI — in a 16-core directory-based cache setup.

Simulation Framework:

- Cycle-accurate simulation with identical workloads across all protocols.
- Benchmarks cover various sharing and access patterns.

Metrics Tracked:

- **Latency** (average cycles per access)
- **Effective cache-to-cache hits**
- **Memory access**

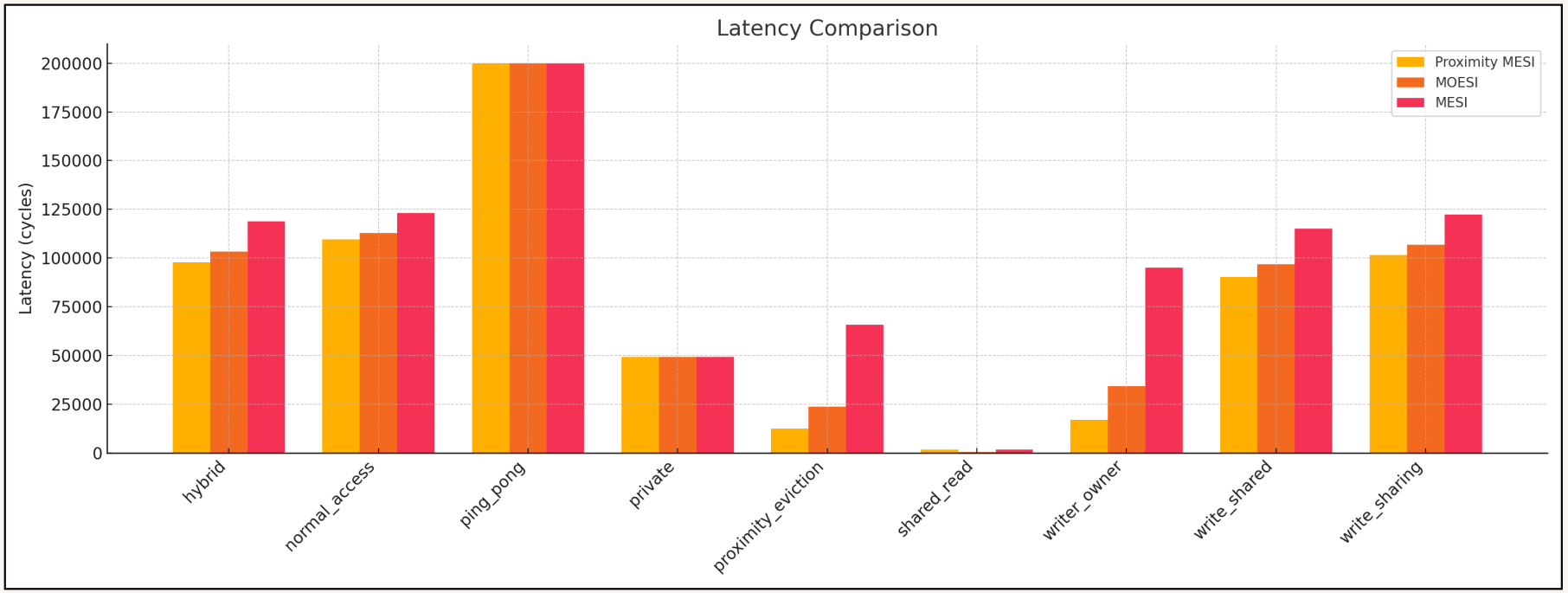
Protocols Compared:

- Base MESI (Simple, baseline)
- MOESI (Owner state for reducing memory traffic)
- Proximity MESI (Optimized for nearest sharer selection)

Benchmark Expectations

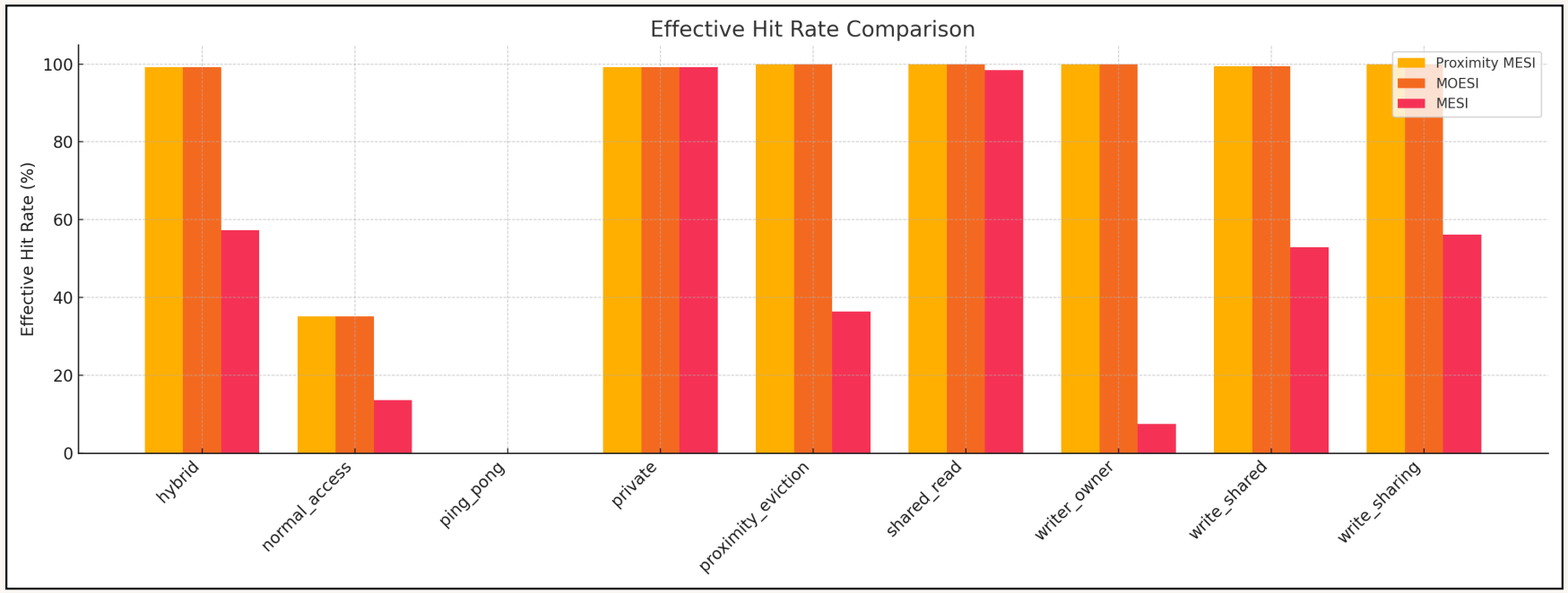
- **hybrid**: Mixed sharing and writes—Proximity MESI expected to reduce some latency via nearest transfers.
- **normal_access**: Irregular, random access—cache reuse is low; all protocols struggle with hit rates.
- **ping_pong**: 2 cores frequent writes to same address- causes continuous invalidations and poor caching performance.
- **private**: Each core uses its own data—high local hit rates across all protocols.
- **proximity_eviction**: Proximity MESI should evict farthest sharers, lowering latency versus MESI.
- **shared_read**: Many cores read shared data—effective hits expected to be high in MOESI and Proximity MESI.
- **writer_owner**: MOESI benefits from Owner state; Proximity MESI reduces access time by using nearest sharer.
- **write_shared**: Frequent writes to shared data—Proximity MESI reduces transfer delay but invalidation traffic dominates.
- **write_sharing**: Alternating writes and reads—heavy invalidation, proximity helps with write propagation.

Latency Comparison



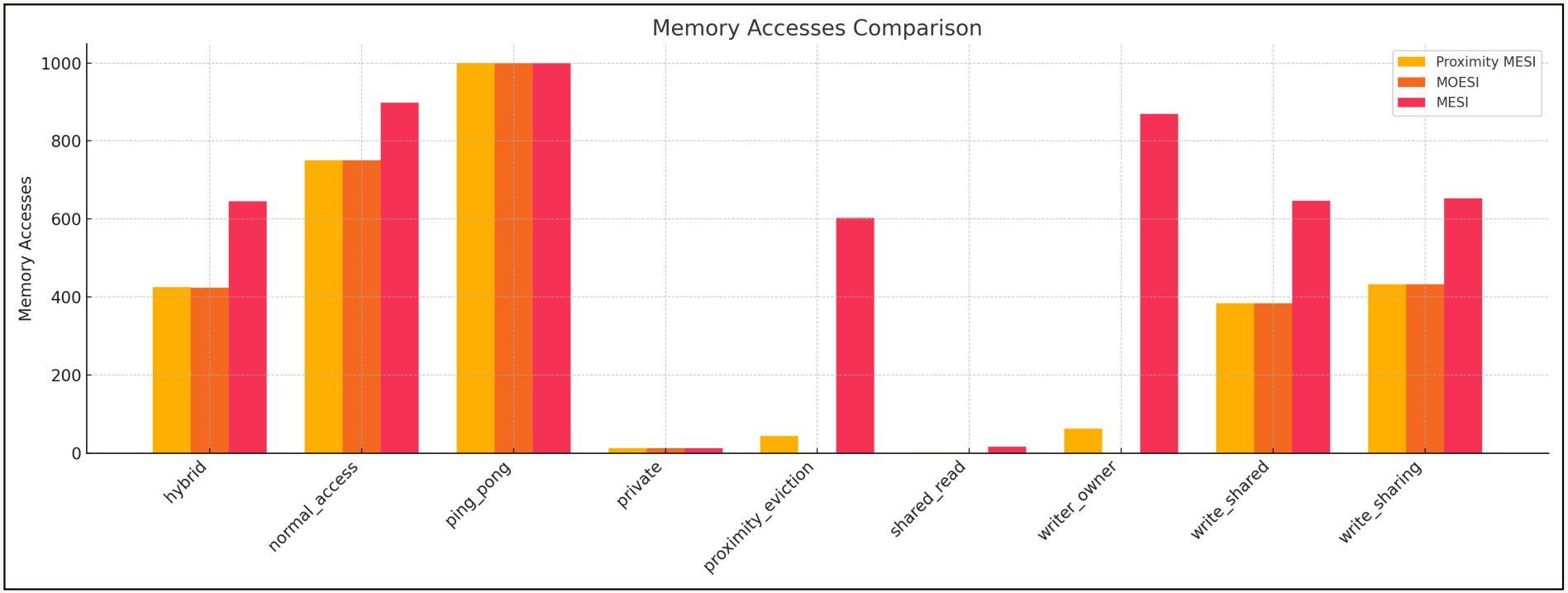
We can observe that MESI is consistently having highest latency throughout all benchmarks due to continuous off-chip memory access whereas Proximity aware and MOESI shows lower latency due to Data Forwarding from selected sharers based on distance or Owned state in MOESI

EHR Comparison



Proximity MESI and MOESI achieve high effective hit rates by forwarding data from nearby or Owner sharers. MESI falls behind due to frequent memory accesses and lack of remote data reuse.

Memory Access Comparison



Proximity MESI and MOESI minimize memory accesses through cache-to-cache transfers.
MESI shows the highest memory traffic due to lack of sharing and reliance on off-chip memory.

Result Analysis

- **Proximity MESI** consistently achieves **lower latency** across most benchmarks by selecting the **nearest sharer** for cache-to-cache transfers and intelligently **evicting distant sharers**. This results in faster access times, especially under stress and mixed workloads (hybrid, proximity_eviction, write_owner).
- **MOESI** performs similarly in terms of **effective hit rate**, often matching Proximity MESI, particularly in read-dominant workloads. However, its reliance on the **Owner state**, regardless of proximity, leads to **higher latency** in many cases.
- **Base MESI** shows clear performance degradation in most complex scenarios. It incurs **the highest memory access count** and suffers from **low effective hit rates** in workloads with frequent invalidations or shared data (write_shared, write_sharing, writer_owner).
- Benchmarks like writer_owner, proximity_eviction, and hybrid demonstrate the **performance gap** between traditional coherence strategies and proximity-aware optimization, reinforcing the benefit of **topology-aware coherence**in multicore systems.

Conclusion

We evaluated **Proximity MESI**, **MOESI**, and **Base MESI** across a wide range of synthetic benchmarks representing real-world multicore behavior: shared reads, private data, write contention, and stress cases.

Proximity MESI consistently outperformed MESI in both **latency** and **effective hit rate**, especially in mixed workloads (hybrid), writer contention (writer_owner), and eviction-heavy scenarios (proximity_eviction).

While **MOESI** matched Proximity MESI in hit rate, it lacked proximity awareness, resulting in **higher latency** when data resided far from requesting cores.

MESI's reliance on memory for coherence enforcement caused **significant performance penalties** in high-contention or shared scenarios.

The **visual plots** clearly illustrated:

- **Lower latency** and **fewer memory accesses** for Proximity MESI.
- **Near-100% effective hit rate** in most benchmarks.

Key Insight: Proximity-aware protocols adapt to hardware topology, enabling scalable, low-latency communication in multicore systems.

TEXAS A&M UNIVERSITY

Engineering

THANK YOU