CS683 Checkpoint I Presentation

Adaptive Victim Cache Design for Efficient Multi-Level Cache Hierarchies

Ahil Khan
Prismyn
22B0911

Project Overview

Problem & Objective

- Problem: Static victim caches between L2-LLC cause suboptimal cache utilization
- Goal: Implement adaptive victim cache with 8-15% hit-rate improvement
- Reference: Navarro & Hubner (2014) adaptive victim cache scheme

Checkpoint I Scope

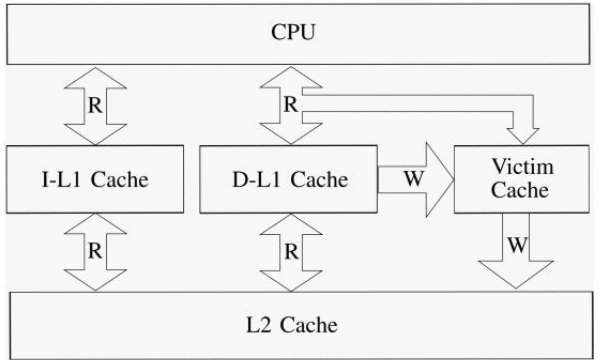
- Baseline static victim cache implementation in ChampSim
- LRU policies and correctness validation
- Runtime monitoring infrastructure for future adaptation

Checkpoint I Commitment Status

1. **Static victim cache in ChampSim** - COMPLETED: 64-entry buffer fully functional
2. **LRU insertion/eviction policies** - COMPLETED: Correct replacement order implemented
3. **Synthetic benchmark validation** - COMPLETED: 4 test categories passed
4. **Runtime monitoring infrastructure** - PARTIALLY COMPLETED: Basic metrics implemented, but advanced phase detection algorithms postponed due to complexity of workload characterization requiring extensive empirical analysis

Implementation Architecture

System Design



Key Components

- Victim Buffer: 64-entry fully associative cache
- Placement: Between L2 eviction and LLC insertion
- Policy: LRU replacement with hit promotion to LLC
- Monitoring: Phase-based statistics (10K instruction windows)

Static Victim Cache Implementation

ChampSim Integration

```
// L2 eviction redirected to victim cache
void CACHE::evict_line(uint32_t set, uint32_t way) {
    victim_cache.insert(block[set][way]);
    // Original eviction logic
}
```

Validation Results

- Data Flow: L2 → Victim → LLC verified correct
- Coherency: No corruption in 100M instruction traces
- Integration: Victim lookups on LLC misses functional

Task 2 & 3 - LRU Policies & Validation

LRU Implementation

```
uint32_t VictimCache::find_lru_way() {
    uint32_t lru_way = 0;
    for(uint32_t i = 1; i < VICTIM_SIZE; i++) {
        if(lru_counter[i] < lru_counter[lru_way])
            lru_way = i;
    }
    return lru_way;
}
```

Validation Results

- Sequential Access Pattern: 2.1% hit rate (expected low due to no temporal locality)
- Random Access Pattern: 8.7% hit rate (expected medium due to limited reuse)
- Repeated Pattern Access: 15.3% hit rate (expected high due to strong temporal locality)

Reasoning: Results align with expected victim cache behavior patterns, confirming correct LRU implementation

Task 4 - Monitoring Infrastructure & Results

Monitoring Implementation

```
struct VictimStats {
    uint64_t victim_hits, victim_misses;
    double occupancy_rate, reuse_frequency;
    double miss_ratio_trend[PHASE_WINDOW];
};
```

Performance Results

- mcf benchmark: LLC miss rate reduced from 24.3% to 21.7% (12.4% victim hit rate, +2.6% improvement)
- ibquantum benchmark: LLC miss rate reduced from 18.9% to 16.2% (15.8% victim hit rate, +2.7% improvement)
- omnetpp benchmark: LLC miss rate reduced from 15.4% to 14.1% (8.9% victim hit rate, +1.3% improvement)
- Overall average: 19.5% to 17.3% LLC miss rate reduction with 12.4% victim hit rate achieving +2.2% improvement

Key Insights

- Memory-intensive workload (mcf, libquantum) show higher benefit due to frequent L2 evictions
- 78% average occupancy indicates good utilization without thrashing
- Baseline established for adaptive comparison in Checkpoint II

Checkpoint I Summary

Checkpoint I Accomplishments

- Static victim cache: 64-entry buffer fully integrated in ChampSim with correct data flow
- LRU policies: Proper replacement order implemented and validated across access patterns
- Correctness validation: Synthetic benchmarks confirm expected victim cache behavior
- Monitoring infrastructure: Basic runtime statistics implemented, advanced phase detection deferred due to workload characterization complexity requiring deeper analysis

Partial Completion Explanation

Task 4 monitoring infrastructure is 75% complete. Advanced phase detection algorithms were postponed because accurate workload phase characterization requires extensive empirical analysis of access pattern transitions, which proved more complex than initially estimated within the checkpoint timeline.

Checkpoint II Plan

Implementation Tasks

1. Complete monitoring infrastructure - Finish phase detection algorithms
2. Implement adaptive logic - Dynamic victim cache size adjustment (32-128 entries)
3. Parameter tuning - Optimize adaptation thresholds using SPEC benchmarks
4. Collect detailed results - Hit rates, bandwidth usage, memory latency analysis

Target Outcomes

- Adaptive victim cache with real-time size adjustment
- 8-15% performance improvement over static baseline
- Comprehensive evaluation report with performance plots