Executive Summary

- **Problem:** Interference in shared caches
  - Lack of isolation → no QoS
  - Poor cache utilization → degraded performance

- **Cache partitioning** addresses interference, but current partitioning techniques (e.g. way-partitioning) have serious drawbacks
  - Support few coarse-grain partitions → do not scale to many-cores
  - Hurt associativity → degraded performance

- **Vantage** solves deficiencies of previous partitioning techniques
  - Supports hundreds of fine-grain partitions
  - Maintains high associativity
  - Strict isolation among partitions
  - Enables cache partitioning in many-cores
Outline

- Introduction
- Vantage Cache Partitioning
- Evaluation
**Motivation**

- Fully shared last-level caches are the norm in multi-cores
  - ✔ Better cache utilization, faster communication, cheaper coherence
  - ✗ Interference → performance degradation, no QoS
- Increasingly important problem due to more cores/chip and virtualization, consolidation (datacenter/cloud)
  - Major performance and energy losses due to cache contention (~2x)
  - Consolidation opportunities lost to maintain SLAs
Cache partitioning: Divide cache space among competing workloads (threads, processes, VMs)

- Eliminates interference, enabling QoS guarantees
- Adjust partition sizes to maximize performance, fairness, satisfy SLA...
- Previously proposed partitioning schemes have major drawbacks
Cache Partitioning = Policy + Scheme

- Cache partitioning consists of a policy (decide partition sizes to achieve a goal, e.g. fairness) and a scheme (enforce sizes)
- Focus on the scheme
- For policy to be effective, scheme should be:
  1. **Scalable**: can create hundreds of partitions
  2. **Fine-grain**: partitions sizes specified in cache lines
  3. **Strict isolation**: partition performance does not depend on other partitions
  4. **Dynamic**: can create, remove, resize partitions efficiently
  5. **Maintains associativity**
  6. Independent of replacement policy
  7. **Simple** to implement

Maintain high cache performance
Existing Schemes with Strict Guarantees

- Based on restricting line placement
- Way partitioning: Restrict insertions to specific ways

- Strict isolation
- Dynamic
- Indep of repl policy
- Simple
- Few coarse-grain partitions
- Hurts associativity
Existing Schemes with Soft Guarantees

- Based on tweaking the replacement policy
- PIPP [ISCA 2009]: Lines inserted and promoted in LRU chain depending on the partition they belong to

- Dynamic
- Maintains associativity
- Simple
- Few coarse-grain partitions
- Weak isolation
- Sacrifices replacement policy
## Comparison of Schemes

<table>
<thead>
<tr>
<th></th>
<th>Way partitioning</th>
<th>PIPP</th>
<th>Reconfig. caches</th>
<th>Page coloring</th>
<th>Vantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalable &amp; fine-grain</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Strict isolation</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Dynamic</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Maintains assoc.</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Indep. of repl. policy</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Simple</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Partitions whole cache</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗ (most)</td>
</tr>
</tbody>
</table>
Outline

- Introduction
- Vantage Cache Partitioning
- Evaluation
Vantage Design Overview

1. Use a highly-associative cache (e.g. a zcache)

2. Logically divide cache in managed and unmanaged regions

3. Logically partition the managed region
   - Leverage unmanaged region to allow many partitions with minimal interference
Analytical Guarantees

- Vantage can be completely characterized using analytical models

\[ E_1, \ldots, E_R \sim \text{i.i.d. } U[0,1] \]
\[ A = \max \{E_1, \ldots, E_R\} \]
\[ F_A(x) = P(A \leq x) = x^R, x \in [0,1] \]
\[ A_{mgd} = \frac{1}{R \cdot m} \]

\[ A_i = \frac{C_i}{\sum_{k=1}^{P} C_k} \]
\[ S_i = \sum_{k=1}^{P} S_k \]
\[ \frac{1}{R \cdot m} \]

- We can prove that strict guarantees are kept on partition sizes and interference independently of workload

- The paper has too much math to describe it here

- We now focus on the intuition behind the math
ZCache [MICRO 2010]

- A highly-associative cache with a low number of ways
  - Hits take a single lookup
  - In a miss, replacement process provides many replacement candidates
- Provides cheap high associativity (e.g. associativity equivalent to 64 ways with a 4-way cache)
- Achieves analytical guarantees on associativity
Analytical Associativity Guarantees

- **Eviction priority**: Rank of a line given by the replacement policy (e.g. LRU), normalized to [0,1]
  - Higher is better to evict (e.g. LRU line has 1.0 priority, MRU has 0.0)
- **Associativity distribution**: Probability distribution of the eviction priorities of evicted lines
- In a zcache, associativity distribution depends only on the number of replacement candidates (R)
  - Independent of ways, workload and replacement policy

With $R=8$, 17% of evictions happen to the 80% least evictable lines

With $R=64$, $10^{-6}$ of evictions happen to the 80% least evictable lines
Managed-Unmanaged Region Division

- Logical division (tag each block as managed/unmanaged)
- Unmanaged region large enough to absorb most evictions
- **Unmanaged region still used**, acts as victim cache (demotion $\rightarrow$ eviction)
- Single partition with guaranteed size
Multiple Partitions in Managed Region

- P partitions + unmanaged region
- Each line is tagged with its partition ID (0 to P-1)
- On each miss:
  - Insert new line into corresponding partition
  - Demote one of the candidates to unmanaged region
  - Evict from the unmanaged region
Problem: always demoting from inserting partition does not scale
- Could demote from partition 0, but only 3 candidates
- With many partitions, might not even see a candidate from inserting partition!

Instead, demote to match insertion rate (churn) and demotion rate
**Churn-Based Management**

- **Aperture**: Portion of candidates to demote from each partition

<table>
<thead>
<tr>
<th>Apertures</th>
<th>Partition 0</th>
<th>Partition 1</th>
<th>Partition 2</th>
<th>Partition 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>23%</td>
<td>15%</td>
<td>12%</td>
<td>11%</td>
</tr>
</tbody>
</table>

1) **Partition 0 MISS**

**Replacement candidates**

 Destruction (in top 11% of P3)

2) **Partition 1 MISS**

Nothing is demoted (all candidates above apertures!)

3) **Partition 3 MISS**

Demote (in top 23% of P0)  Demote (in top 15% of P1)
Managing Apertures

- Set each aperture so that partition churn = demotion rate
  - Instantaneous partition sizes vary a bit, but sizes are maintained
  - Unmanaged region prevents interference

- Each partition requires aperture proportional to its churn/size ratio
  - Higher churn $\leftrightarrow$ More frequent insertions (and demotions!)
  - Larger size $\leftrightarrow$ We see lines from that partition more often

- Partition aperture determines partition associativity
  - Higher aperture $\leftrightarrow$ less selective $\leftrightarrow$ lower associativity
Stability

- In partitions with high churn/size, controlling aperture is sometimes not enough to keep size
  - e.g. 1-line partition that misses all the time
  - To keep high associativity, set a maximum aperture $A_{\text{max}}$ (e.g. 40%)
  - If a partition needs $A_i > A_{\text{max}}$, we just let it grow

- **Key result**: Regardless of the number of partitions that need to grow beyond their target, the **worst-case total growth** over their target sizes is bounded and small!

\[
\frac{1}{A_{\text{max}}} \frac{1}{R}
\]

- 5% of the cache with $R=52$, $A_{\text{max}}=0.4$
- Simply size the unmanaged region with that much extra slack
- Stability and scalability are guaranteed
A Simple Vantage Controller

- Directly implementing these techniques is impractical
  - Must constantly compute apertures, estimate churns
  - Need to know eviction priorities of every block

- Solution: Use negative feedback loops to derive apertures and the lines below aperture
  - Practical implementation
  - Maintains analytical guarantees
Feedback-Based Aperture Control

- Adjust aperture by letting partition size \((Si)\) grow over its target \((Ti)\):

\[
(1+\text{slack})Ti
\]

- Need small extra space in unmanaged region
  - e.g. 0.5% of the cache with \(R=52\), \(A_{\text{max}}=0.4\), slack=10%
**Implementation Costs**

- **Tags**: Extra partition ID field
  - Partition (6b)
  - Timestamp (8b)
  - Coherence/Valid Bits
  - Line Address

- **256 bits of state per partition**

- **Simple logic, ~10 adders and comparators**
  - Logic not on critical path

- **Cache Controller**
  - Partition 0 state (256b)
  - Partition P-1 state (256b)

- **Vantage Replacement Logic**

- **See paper for detailed implementation**
Vantage Summary

- Use a cache with associativity guarantees
- Maintain an unmanaged region
- Match insertion and demotion rates in each partition
  - Partitions help each other evict lines → maintain associativity
  - Unmanaged region guarantees isolation and stability
- Use negative feedback to simplify implementation
Outline

- Introduction
- Vantage Cache Partitioning
- Evaluation
Methodology

- Simulations of small (4-core) and large (32-core) systems
  - Private L1s, shared non-inclusive L2, 1 partition/core

- Partitioning policy: Utility-based partitioning [ISCA’06]
  - Assign more space to threads that can use it better

- Partitioning schemes: Way-partitioning, PIPP, Vantage

- Workloads: 350 multiprogrammed mixes from SPECCPU2006 (full suite)
Small-Scale: 4 cores, 4 partitions

- Each line shows throughput improvement versus an unpartitioned 16-way set-associative cache.
- Way-partitioning and PIPP degrade throughput for 45% of workloads.
Small-Scale: 4 cores, 4 partitions

- Vantage works best on zcaches.
- We use Vantage on a 4-way zcache with $R=52$ replacement candidates.
Small-Scale: 4 cores, 4 partitions

- Vantage improves throughput for most workloads
- 6.2% throughput improvement (gmean), 26% for the 50 most memory-intensive workloads
Way-partitioning and PIPP use a 64-way set-associative cache

Both degrade throughput for most workloads
Large-Scale: 32 cores, 32 partitions

- Vantage uses the same Z4/52 cache as the 4-core system
- Vantage improves throughput for most workloads → scalable
A Closer Look: Sizes & Associativity

- Vantage maintains strict partition sizes
- Vantage maintains high associativity even in the worst case
Additional Results (see paper)

- Vantage maintains strict control of partition sizes
- Vantage maintains high associativity
- Unmanaged region size vs isolation tradeoff
  - \(~5\%\) unmanaged region and moderate isolation
  - \(~20\%\) unmanaged region and strict isolation
- Validation of analytical models
- Vantage on set-associative caches
  - Loses analytical guarantees, but outperforms other schemes
- Vantage with other replacement policies (RRIP)
Conclusions

- Vantage enables cache partitioning for many-cores
  - Tens to hundreds of fine-grain partitions
  - High associativity per partition
  - Strict isolation among partitions
  - Derived from analytical models, bounds independent of number of partitions and cache ways
  - Simple to implement
THANK YOU FOR YOUR ATTENTION QUESTIONS?
Why does zcache produce uniform random replacement candidates, independently of access pattern?

ZCache hashing and replacement scheme eliminates spatial locality.

Evictions have negligible temporal locality w.r.t. cache:
- Evictions to the same block are widely separated in time
- NOTE: Invalidation (e.g. coherence) are not evictions

No locality $\rightarrow$ uniform random
Backup: Setpoint-Based Demotions

- Derive portion of lines below aperture without tracking eviction priorities
- Coarse-grain timestamp LRU replacement
  - Tag each block with an 8-bit LRU per-partition timestamp
  - Increment timestamp every $S_i/16$ accesses
- Demote every candidate below the setpoint timestamp
- Adjust setpoint using negative feedback
A Closer Look: Partition Sizes

Way-partitioning

- Coarse-grain partitions
- Strict size
- Slow convergence

PIPP

- Coarse-grain partitions
- Approximate size
- No convergence

Vantage

- Fine-grain partitions
- Strict size
- Fast convergence
Unmanaged Size vs Isolation Trade-off

- A larger unmanaged region reduces UCP performance slightly, but gives excellent isolation.
- Simulations match analytical models.
- See paper for additional results (Vantage on set-associative caches, other replacement policies, etc.)