KPart: A Hybrid Cache Sharing-Partitioning Technique for Commodity Multicores

Nosayba El-Sayed  Anurag Mukkara  Po-An Tsai  Harshad Kasture

Xiaosong Ma  Daniel Sanchez
Partitioning the last-level cache among co-running apps can reduce interference and improve system performance.

Recent processors offer hardware cache-partitioning support!

Two key challenges limit its usability:

1. Current hardware implements coarse-grained way-partitioning, which hurts system performance.
2. Lacks hardware monitoring units to collect cache-profiling data.

KPart tackles these limitations, unlocking significant performance on real hardware (avg gain: 24%, max: 79%), and is publicly available.
Limitations of hardware cache partitioning

1. Implements coarse-grained **way-partitioning** ➔ hurts system performance

- Real-system example (benchmarks: SPEC-CPU2006, PBBS)
Limitations of hardware cache partitioning

1. Implements coarse-grained **way-partitioning** ⇒ hurts system performance

- Real-system example (benchmarks: SPEC-CPU2006, PBBS)
- Baseline: **NoPart** (All apps share all ways)
Limitations of hardware cache partitioning

1. Implements coarse-grained **way-partitioning** \(\Rightarrow\) hurts system performance

- Real-system example (benchmarks: SPEC-CPU2006, PBBS)
1. Implements coarse-grained **way-partitioning** \(\rightarrow\) hurts system performance

- Real-system example (benchmarks: SPEC-CPU2006, PBBS)
- Conventional policy: Per-app, utility-based cache part (UCP)
Limitations of hardware cache partitioning

1. Implements coarse-grained **way-partitioning** \(\Rightarrow\) hurts system performance

- Real-system example (benchmarks: SPEC-CPU2006, PBBS)
- Conventional policy: Per-app, utility-based cache part (UCP)

### Application Cache-Profiles

<table>
<thead>
<tr>
<th>Application</th>
<th>Cache-Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>sphinx3</td>
<td></td>
</tr>
<tr>
<td>leslie3d</td>
<td></td>
</tr>
<tr>
<td>matching</td>
<td></td>
</tr>
<tr>
<td>hmmmer</td>
<td></td>
</tr>
<tr>
<td>libquantum</td>
<td></td>
</tr>
<tr>
<td>delaunay</td>
<td></td>
</tr>
<tr>
<td>leslie3d</td>
<td></td>
</tr>
<tr>
<td>GemsFDTD</td>
<td></td>
</tr>
</tbody>
</table>

**Smallest partition size**

- Last-Level Cache (12MB)
Limitations of hardware cache partitioning

1. Implements coarse-grained **way-partitioning** → hurts system performance
   - Real-system example (benchmarks: SPEC-CPU2006, PBBS)
   - Conventional policy: Per-app, utility-based cache part (UCP)

Application Cache-Profiles

- Smallest partition size

Conventional policies yield small partitions with few ways:
  - low associativity → more misses
  - This example: throughput degrades by **3.8%**
Prior work on cache partitioning

- **Hardware way-partitioning**: restrict insertions into subsets of ways
  - Available in commodity hardware
  - Small number of coarsely-grained partitions!

- **Page coloring**
  - No hardware support required
  - Not compatible with superpages; costly repartitioning due to recoloring; heavy OS modifications

- **High-performance, fine-grained hardware partitioners** (e.g. Vantage [ISCA'11], Futility Scaling [MICRO’14])
  - Support hundreds of partitions
  - Not available in existing hardware

- **Hybrid technique: Set and WAy Partitioning (SWAP)** [HPCA’17]
  - Combines page coloring and way-partitioning ➔ fine-grained partitions
  - Inherits page coloring limitations
Prior work on cache partitioning

- **Hardware way-partitioning**: restrict insertions into subsets of ways
  - Available in commodity hardware
  - Small number of coarsely-grained partitions

- **Page coloring**
  - No hardware support required
  - Not compatible with superpages; costly repartitioning due to recoloring; heavy OS modifications

- **High-performance, fine-grained hardware partitioners** (e.g. Vantage [ISCA'11], Futility Scaling [MICRO'14])
  - Support hundreds of partitions
  - Not available in existing hardware

- **Hybrid technique: Set and WAy Partitioning (SWAP)** [HPCA'17]
  - Combines page coloring and way-partitioning ➔ fine-grained partitions
  - Inherits page coloring limitations
KPart performs hybrid cache sharing-partitioning to make use of coarse-grained partitions.

Cache-Aware App Grouping

Avoids significant reduction in cache associativity ➔ throughput improves by 17%

Grouping must be done carefully!
KPart overview: Hybrid cache sharing-partitioning

How?

Group applications into clusters

Cache-Sharing Clusters

Cluster#1

Cluster#2

Cluster#3

Assign cache partitions to clusters

Application Profiles

Collected online or offline

Cache Misses

Miss Curves

cache capacity

Per-Cluster Cache Partition Plan
Clustering apps based on cache-compatibility:

**Distance metric**

- How many additional cache misses are expected when two apps *share* cache capacity vs. when it’s *partitioned*?

- Use cache miss curves to estimate:
  - [Mukkara et al., ASPLOS’16]
  - [divide cap using UCP]

**Area** ➔ expected *performance degradation* when apps *share* cache capacity (due to additional misses)
Hierarchical clustering:

- Start with the applications as individual clusters
- At each step, merge the closest pair of clusters until only one cluster is left.

How do we find the best $K$ without running the mix?
Automatic selection of $K$ in KPart

Application Profiles

Performance Estimator

- Estimate throughput under all possible $K$s
- Account for bandwidth contention
- Estimate speedup curves
- Return $K_{auto}$ that produces best result

How?

Cluster#1

Cluster#2

Cluster#K_{auto}

Per-Cluster Cache Partition Plan
Cache-partitioning in commodity multicores

Partitioning the last-level cache among co-running apps can reduce interference and improve system performance.

Recent processors offer hardware cache-partitioning support!

Two key challenges limit its usability:

1. Implements coarse-grained way-partitioning, which hurts system performance.
2. Lacks hardware monitoring units to collect cache-profiling data.
How do we profile applications online at low overhead and high accuracy?

Prior work mostly simulated hardware monitors that don’t exist in real systems, or used expensive software-based mem address sampling.

DynaWay exploits hardware partitioning support to adjust partition sizes periodically → measure performance (misses, IPC, bandwidth).

We applied optimizations to reduce measurement points and interval length (see paper) → less than 1% profiling overhead (8-app workloads).
**KPart+DynaWay** profiles applications online, partitions the cache dynamically.

**Per-Cluster Partition Plan**

- **Cluster#1**
- **Cluster#Kauto**

Generate online profiles + update periodically
KPart+DynaWay profiles applications online, partitions the cache dynamically.
KPart Evaluation
Evaluation methodology

- **Platform**: 8-core Intel Broadwell D-1540 processor (12MB LLC)
- **Benchmarks**: SPEC-CPU2006, PBBS
- **Mixes**: 30 different mixes of 8 apps (randomly selected), each app running at least 10B instr.
- **Experiments**:
  - KPart on real system with offline profiling
  - KPart on real system with online profiling (using DynaWay)
  - KPart in simulation compared against high-performance techniques
  - KPart with mix of batch and latency-critical applications
KPart unlocks significant performance on real hardware

- Evaluation results on a real system with offline profiling

- KPart improves system performance by 24% on average!

- NoClust hurts ~30% of mixes

- Important to use $K_{auto}$ instead of fixed $K$
KPart unlocks significant performance on real hardware

- Evaluation results on a real system with offline profiling
- Case studies of individual mixes:
KPart evaluation with DynaWay’s online profiles

KPart+DynaWay can even outperform static KPart with offline profiling
(adapts to application phase changes!)
KPart bridges the gap between current and future hardware partitioners

- In simulation: we compared KPart to a high-performance fine-grained hardware partitioner, Vantage [ISCA'11]

**KPart** achieves **most of the gains** obtained by fine-grained partitioning!
KPart helps LC apps when combined with QoS-oriented techniques

- KPart focuses on batch apps, but data centers colocate latency-critical (LC) and batch

- Prior work uses cache partitioning to provide QoS guarantees for LC apps
  - but does not improve batch apps throughput

Combining KPart with QoS-oriented technique can improve both batch throughput and LC latency:
- KPart improves batch throughput which leads to reduced memory traffic
- LC apps benefit from more bandwidth and cache

**Evaluation:** On same 8-core system running both LC and batch apps, up to 28% improvement in batch throughput and up to 7% improvement in LC tail latency
KPart summary

- KPart unlocks the potential of hardware way-partitioning using a hybrid sharing-partitioning approach.
- KPart improves throughput significantly (avg: 24%) & bridges the gap between current and future partitioning techniques.
- DynaWay exploits existing way-partitioning support to perform lightweight & accurate cache-profiling.
- KPart+DynaWay can be combined with QoS-oriented policies to colocate latency-critical apps and batch apps effectively.

KPart is open-sourced and publicly available at http://kpart.csail.mit.edu
Thank you! Questions?

✓ KPart unlocks the potential of hardware way-partitioning using a hybrid sharing-partitioning approach

✓ KPart improves throughput significantly (avg: 24%) & bridges the gap between current and future partitioning techniques

✓ DynaWay exploits existing way-partitioning support to perform lightweight & accurate cache-profiling

✓ KPart+DynaWay can be combined with QoS-oriented policies to colocate latency-critical apps and batch apps effectively

KPart is open-sourced and publicly available at

http://kpart.csail.mit.edu