WHIRLPOOL!

IMPROVING DYNAMIC CACHE MANAGEMENT WITH STATIC DATA CLASSIFICATION

Anurag Mukkara, Nathan Beckmann, Daniel Sanchez

MIT CSAIL
Processors are limited by data movement

- Data movement often consumes >50% of time & energy
  - E.g., FP multiply-add: 20 pJ ↔ DRAM access: 20,000 pJ
- To scale performance, must keep data near where its used
- But how do programs use memory?

Diagram:
- Core
- Cache banks
- DRAM
- Good: nearby cache banks
- Bad: faraway cache banks
- Terrible: DRAM access
Static policies have limitations

- **Program Code**
  - Static analysis or profiling
  - **Fixed policy**
  - **Binary**

E.g., scratchpads, bypass hints

- Exploits program semantics
  - Can’t adapt to application phases, input-dependent behavior, or shared systems
Dynamic policies have limitations, too

E.g., data migration & replication

Responsive to actual application behavior

Difficult to recover program semantics from loads/stores

⇒ Expensive mechanisms (e.g., extra data movement & directories)
Combining static and dynamic is best

- Static analysis or profiling
- Observe loads/stores
- Exploits program semantics at low overhead
- Responsive to actual application behavior
Agenda

- Case study
- Manual classification
- Parallel applications
- WhirlTool
System configuration

Non-uniform cache access (NUCA):
Cache banks have different access latencies
We apply Whirlpool to Jigsaw [Beckmann PACT’13], a state-of-the-art NUCA cache

- Allocates virtual caches, collections of parts of cache banks
- Significantly outperforms prior D-NUCA schemes

Baseline dynamic NUCA scheme

Reduce cache misses
Reduce on-chip network traversals
Simple mechanisms
Dynamic policies can reduce data movement.

Static NUCA

App: Delaunay triangulation

Jigsaw

[Beckmann, PACT’13]

Dynamic policy performs somewhat better:

- 4% better performance
- 12% lower energy
Static analysis can help!

- Triangles
- Vertices
- Points

Accesses

Footprint (MB)

Access Intensity

6x difference

Points

Vertices

Triangles
Jigsaw with Static Classification

Few data structures accessed more frequently than others

Vs Jigsaw:
19% better performance
42% lower energy
Agenda

- Case study
- Manual classification
- Parallel applications
- WhirlTool
Organize application data into memory pools

**Points, Triangles**

```c
int poolPoints = pool_create();
Point* points = pool_malloc(sizeof(Point)*n, poolPoints);

int poolTris = pool_create();
Tri* smallTris = pool_malloc(sizeof(Tri)*m, poolTris);

Tri* largeTris = pool_malloc(sizeof(Tri)*M, poolTris);
```

Insight: Group semantically similar data into a pool
Minor changes to programs

PBBS

SPECCPU 2006

<table>
<thead>
<tr>
<th>Application</th>
<th>SPECCPU 2006</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delaunay triangulation</td>
<td>401.bzip2</td>
</tr>
<tr>
<td>Maximal matching</td>
<td>470.lbm</td>
</tr>
<tr>
<td>Delaunay refinement</td>
<td>429.mcf</td>
</tr>
<tr>
<td>Maximal independent set</td>
<td>436.cactusADM</td>
</tr>
<tr>
<td>Minimal spanning forest</td>
<td></td>
</tr>
</tbody>
</table>
Whirlpool on NUCA placement

- Use pools to improve Jigsaw’s decisions
  - Each pool is allocated to a virtual cache
  - Jigsaw transparently places pools in NUCA banks

- Whirlpool requires no changes to core Jigsaw
  - Increase size of structures (few KBs)
  - Minor improvements, e.g. bypassing (see paper)

- Pools useful elsewhere, eg to dynamic prefetching
Significant improvements on some apps

Up to 38% better performance

Up to 53% lower energy
Agenda

- Case study
- Manual classification
- Parallel applications
- WhirlTool
Conventional runtimes can harm locality

Optimize load balance, not locality
Whirlpool co-locates tasks and data

- Break input into *pools*
- Application indicates task affinity
- Schedule + steal tasks from nearby their data
- Dynamically adapt data placement
- Requires minimal changes to task-parallel runtimes
Whirlpool improves locality
Whirlpool adapts schedule dynamically

- Data placement implicitly schedules tasks
Significant improvements at 16 cores

Applications

Divide and conquer algorithms: Mergesort, FFT
Graph analytics: PageRank, Triangle Counting, Connected Components
Graphics: Delaunay Triangulation

Caveat: Splitting data into pools can be expensive!

Up to 67% better performance
Up to 2.6x lower energy
Agenda

- Case study
- Manual classification
- Parallel applications
- WhirlTool
WhirlTool – Automated classification

- Modifying program code is not always practical
- A profile-guided tool can automatically classify data into pools

WhirlTool
Profiler

Per-callpoint miss curves

WhirlTool
Analyzer

Callpoint-to-pool map

WhirlTool runtime

pool_malloc()

Application

malloc()
WhirlTool profiles miss curves

Groups allocations by callpoint
Profiles accesses to each pool
Periodically records per-callpoint miss curves

Application

Alloc Accs Alloc Accs Alloc

A B C

Time

Misses

Cache size

T i m e

Alloc Accs Alloc Accs Alloc
WhirlTool analyzes curves to find pools

- Hardware can only support a limited number of pools
  - Jigsaw uses 3 virtual caches / thread
    -> 0.6% area overhead over LLC
  - Whirlpool adds 4 pools (each mapped to a virtual cache)
    -> 1.2% total area overhead over LLC

- Must cluster callpoints into semantically similar groups

Per-callpoint miss curves  \hspace{2cm} \textbf{Agglomerative clustering} \hspace{2cm} \textbf{Callpoint-to-pool mapping}
Example of agglomerative clustering
How many misses are saved by separating pools?

WhirlTool’s distance metric
WhirlTool matches manual hints
Multiprogram mixes

- 4-core system with random SPECCPU2006 apps
  - Including those that do not benefit

- Whirlpool improves performance by (gmean over 20 mixes)
  - 35% over S-NUCA
  - 30% over idealized shared-private D-NUCA [Hererro, ISCA’10]
  - 26% over R-NUCA [Hardavellas, ISCA’09]
  - 18% over page placement by Awasthi et al. [Awasthi HPCA’09]
  - 5% over Jigsaw [Beckmann, PACT’13]
Conclusion

- Semantic information from applications improves performance of dynamic policies
- Coordinated data and task placement gives large improvements in parallel applications
- Automated classification reduces programmer burden
THANKS FOR YOUR ATTENTION!

QUESTIONS ARE WELCOME!