Line Sharing Cache: Exploring Cache Capacity with Frequent Line Value Locality

Keitarou Oka, Hiroshi Sasaki and Koji Inoue

Kyushu University, Japan
Outline

‣ Background
‣ Motivation
‣ Line Sharing Cache
‣ Evaluation
‣ Conclusions
Problem

- Memory wall problem
  - Off-chip memory bandwidth is limited by I/O pin counts (no longer scale)
  - Memory speed is much slower than processor speed
- Multicore processors aggregate the memory wall problem
  - Demands higher off-chip memory bandwidth because of frequent memory accesses
Problem

- Memory wall problem
  - Off-chip memory bandwidth is limited by I/O pin counts (no longer scale)
  - Memory speed is much slower than processor speed

- Multicore processors aggregate the memory wall problem
  - Important to reduce Last Level Cache (LLC) misses
Goal of Our Research

- Today’s approach
  - Integrating a large shared last-level cache (LLC)
    - e.g., Intel Core i7 with 4MB to 15MB L3 cache

- Problem
  - The large area requires high cost

Reducing LLC misses without increasing the size
Outline

- Background
- Motivation
  - *Frequent Line Value Locality*
- Line Sharing Cache (LSC)
- Evaluation
- Conclusions
Frequent Line Value Locality

- Frequent value locality [Yang and Gupta 2002]
  - A small number of values occupy a large fraction of memory access values

Our findings:

- Frequent line value locality (FLVL)
  - In some applications, locality exists even when the size of the value is expanded to a cache line
Analysis of FLVL

Data uniqueness ratio: proportion of unique values written to the cache

Even for **8 words** some benchmarks show high frequent line value locality
Outline

- Background
- Motivation
- Line Sharing Cache (LSC)
  - Concept
  - Operation
  - Structure
  - Pros and cons
- Evaluation
- Conclusions
**Concept**

- Associate tag entries with a line value
Concept

- Associate tag entries with a line value
Concept

- Associate tag entries with a line value
Concept

- Associate tag entries with a line value
- Remove multiple identical line values
Concept

- Associate tag entries with a line value
- Remove multiple identical line values

Tag Array       Data Array

<table>
<thead>
<tr>
<th>a</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
</tr>
</tbody>
</table>
Concept

- Associate tag entries with a line value
- Remove multiple identical line values
- Increase tag entries by reducing data entries

Tag Array  Data Array

```
| a |
| b |
| c |
```

```
| A |
```
Concept

- Associate tag entries with a line value
- Remove multiple identical line values
- Increase tag entries by reducing data entries

Tag Array   Data Array

a   A
b   
c
Concept

- Associate tag entries with a line value
- Remove multiple identical line values
- Increase tag entries by reducing data entries

Tag Array  Data Array

a
b
c
A
Concept

- Associate tag entries with a line value
- Remove multiple identical line values
- Increase tag entries by reducing data entries
Concept

- Associate tag entries with a line value
- Remove multiple identical line values
- Increase tag entries by reducing data entries

LSC can increase the effective cache size without increasing the physical size
Outline

- Background
- Motivation
- Line Sharing Cache (LSC)
  - Concept
  - Operation
  - Structure
  - Pros and cons
- Evaluation
- Conclusions
Operation

- Read hit operation
- Write hit operation
- Read miss operation
- Write miss operation

explained in this presentation
Read Hit Operation

- Read data associated with the tag entry

Diagonal: Address → Tag Array → Data Array

Tag Array: Tag a

Data Array: Data A

Equation: a = A

Result: Cache hit
Write Hit Operation

- Search the data array for the written value → value hit / miss
- Value hit → the tag entry map onto the written value
- Value miss → the tag entry map onto the written value after update of LSC
Write Hit Operation

- Search the data array for the written value → value hit / miss
- Value hit → the tag entry map onto the written value
- Value miss → the tag entry map onto the written value after update of LSC

![Diagram of Tag Array and Data Array with search and cache hit annotations]
Write Hit Operation

- Search the data array for the written value → value hit / miss
- Value hit → the tag entry map onto the written value
- Value miss → the tag entry map onto the written value after update of LSC
**Write Hit Operation**

- Search the data array for the written value → value hit / miss
- Value hit → the tag entry map onto the written value
- Value miss → the tag entry map onto the written value after update of LSC

![Diagram](image)

- **Tag Array**
  - Address: tag | index
  - Elements: a, b

- **Data Array**
  - Written value: A, B
  - Value hit: A

- **Cache hit**
  - Search for the written value
Update of LSC (Value Miss)

- Select a victim entry from the data array
- Invalidate tag entries mapped to the victim entry
- Store the written value in the victim entry
Update of LSC (Value Miss)

- Select a victim entry from the data array
- Invalidate tag entries mapped to the victim entry
- Store the written value in the victim entry
Update of LSC (Value Miss)

- Select a victim entry from the data array
- Invalidate tag entries mapped to the victim entry
- Store the written value in the victim entry
Update of LSC (Value Miss)

- Select a victim entry from the data array
- Invalidate tag entries mapped to the victim entry
- Store the written value in the victim entry
Update of LSC (Value Miss)

- Select a victim entry from the data array
- Invalidate tag entries mapped to the victim entry
- Store the written value in the victim entry
**Update of LSC (Value Miss)**

- Select a victim entry from the data array
- Invalidate tag entries mapped to the victim entry
- Store the written value in the victim entry
Outline

- Background
- Motivation
- Line Sharing Cache (LSC)
  - Concept
  - Operation
  - Structure
  - Pros and cons
- Evaluation
- Conclusions
Structure (1/3)

- **Tag Array**
  - Tag, *Forward pointer (FPTR), Tag entry list (TLIST)*

- **Data Array**
  - Line, *Reverse pointer (RPTR)*

![Diagram](image.png)
How does each tag identify its associated line?

- FPTR stores the pointer to the associated line.
How does a line identify its associated tag?

- TLIST manages doubly linked list of the tag entries associated with a line value.
- RPTR identifies the head and tail of a list.
How does a line identify its associated tag?

- TLIST manages doubly linked list of the tag entries associated with a line value
- RPTR identifies the head and tail of a list
How does a line identify its associated tag?

- TLIST manages doubly linked list of the tag entries associated with a line value
- RPTR identifies the head and tail of a list
Outline

- Background
- Motivation
- Line Sharing Cache (LSC)
  - Concept
  - Operation
  - Structure
  - Pros and cons
- Evaluation
- Conclusions
Pros and Cons

- **Cache misses**
  - :-)
    - The effective size of the LLC increases
  - :-(
    - On a value miss, tag entries associated with a victim data entry are invalidated
    - But value misses decreases when FLVL is high

- **Access Latency**
  - :-(
    - Additional operations make write latency longer
    - Search for the written value
    - Update operation
Outline

‣ Background
‣ Motivation
‣ Line Sharing Cache (LSC)
‣ Evaluation
‣ Conclusions
Simulation Setup

- M5 processor simulator
  - ISA: Alpha

- SPEC CPU 2000 benchmark suit
  - Selected benchmarks
    - SPEC INT: mcf, twolf, vpr, parser, vortex
    - SPEC FP: ammp, apsi, art, applu, sixtrack, mgrid, swim
  - Input size: Train
### System Configuration

<table>
<thead>
<tr>
<th>Core architecture</th>
<th>Single-core, one-IPC model</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 I cache</td>
<td>32 KB, 2-way, 64B lines, 1 cycle latency</td>
</tr>
<tr>
<td>L1 D cache</td>
<td>32 KB, 2-way, 64B lines, 2 cycle latency</td>
</tr>
<tr>
<td>Main memory latency</td>
<td>200 cycles</td>
</tr>
<tr>
<td>Conventional LLC</td>
<td>256 KB, 16-way, 64 B lines, 12 cycles latency</td>
</tr>
</tbody>
</table>
Evaluated Cache Configuration

- **Constraint:** # of SRAM bits of LSC is less than that of the conventional LLC

<table>
<thead>
<tr>
<th></th>
<th># of tag entries</th>
<th># of data entries</th>
<th># of SRAM bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv. LLC</td>
<td>4 K</td>
<td>4 K</td>
<td>288 KB</td>
</tr>
<tr>
<td>LSC-2</td>
<td>8 K</td>
<td>2 K</td>
<td>221 KB</td>
</tr>
<tr>
<td>LSC-4</td>
<td>16 K</td>
<td>1 K</td>
<td>240 KB</td>
</tr>
</tbody>
</table>

In LSC additional write latency is ignored
Performance

- LSC-4 outperforms LSC-2
- Some benchmarks show large performance improvement
The performance improvement of LSC comes from MPKI reduction.
Conclusions

‣ LLC managements which reduce the number of misses are required

‣ Our proposal: LSC
  • *Allocates a single entry for lines which stores an identical value*
  • *Reduces the number of data entries and allows more tag entries*

‣ LSC outperforms the conventional LLC by up to 35%
Back up Slides
Write Hit Operation on Value Hit (1/4)

- Tag comparison → Cache hit
- Searching written line value → Value hit
- Update of LSC
Write Hit Operation on Value Hit (1/4)

- Tag comparison → Cache hit
- Searching written line value → Value hit
- Update of LSC
**Write Hit Operation on Value Hit (2/4) Update of LSC**

- Remove the accessed tag from the TLIST
- Associate the accessed tag with the written value
- Add the accessed tag to the TLIST of the written value

```
<table>
<thead>
<tr>
<th>Tag Array</th>
<th>Data Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>Line</td>
</tr>
<tr>
<td>FPTR</td>
<td>RPTR</td>
</tr>
<tr>
<td>TLIST</td>
<td></td>
</tr>
</tbody>
</table>

| a | A |
| b |   |
| c | C |
| d |   |
| e | D |
| f |   |
```

**Hit Operation**
- Remove tag c from the TLIST of A
Write Hit Operation on Value Hit (2/4) Update of LSC

- Remove the accessed tag from the TLIST
- Associate the accessed tag with the written value
- Add the accessed tag to the TLIST of the written value

Remove tag c from the TLIST of A
Write Hit Operation on Value Hit (3/4) Update of LSC

- Remove the accessed tag from the TLIST
- Associate the accessed tag with the written value
- Add the accessed tag to the TLIST of the written value

Associate tag c with the written value: C
Write Hit Operation on Value Hit (3/4)

Update of LSC

- Remove the accessed tag from the TLIST
- Associate the accessed tag with the written value
- Add the accessed tag to the TLIST of the written value

Associate tag c with the written value: C
Write Hit Operation on Value Hit (4/4)

- Remove the accessed tag from the TLIST
- Associate the accessed tag with the written value
- Add the accessed tag to the TLIST of the written value

TLIST of C is consist of tag b → Add tag c the list of line value C
Write Hit Operation on Value Hit (4/4)

- Remove the accessed tag from the TLIST
- Associate the accessed tag with the written value
- Add the accessed tag to the TLIST of the written value

### Tag Array

<table>
<thead>
<tr>
<th>Tag Array</th>
<th>FPTR</th>
<th>TLIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Data Array

<table>
<thead>
<tr>
<th>Data Array</th>
<th>Line</th>
<th>RPTR</th>
<th>Line</th>
<th>RPTR</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TLIST of C is consist of tag b ➔ Add tag c the list of line value C
Write Hit Operation on Value Miss (1/3)

- Tag comparison → Cache hit
- Search for written line value → Value miss
- LSC-Update:
Write Hit Operation on Value Miss (1/3)

- Tag comparison → Cache hit
- Search for written line value → Value miss
- LSC-Update:

![Diagram of Tag Array and Data Array]

- Tag comparison
- Search for written line value
- LSC-Update:

- Values: a, b, c, d, e, f
- Data Array: A, C, D
- Written value: B
- Value miss
Write Hit Operation on Value Miss (2/3)

Update of LSC

- Select a victim data entry
- Invalidate tags associated with the victim data entry
- Remove the accessed tag from the TLIST
- Associate the accessed tag with the written value
- Add the accessed tag to the TLIST of the written value

**Tag Array**

<table>
<thead>
<tr>
<th>Tag</th>
<th>FPTR</th>
<th>TLIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Data Array**

<table>
<thead>
<tr>
<th>Line</th>
<th>RPTR</th>
<th>Line</th>
<th>RPTR</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td>D</td>
</tr>
</tbody>
</table>

Victim data entry

hit
Write Hit Operation on Value Miss (3/3)

Update of LSC

- Select a victim data entry
- Invalidate tags associated with the victim data entry
- Remove the accessed tag from the TLIST
- Associate the accessed tag with the written value
- Add the accessed tag to the TLIST of the written value
Write Hit Operation on Value Miss (3/3)
Update of LSC

- Select a victim data entry
- Invalidate tags associated with the victim data entry
- Remove the accessed tag from the TLIST
- Associate the accessed tag with the written value
- Add the accessed tag to the TLIST of the written value

**Tag Array**

<table>
<thead>
<tr>
<th>Tag</th>
<th>FPTR</th>
<th>TLIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Written value**

B

**Data Array**

<table>
<thead>
<tr>
<th>Line</th>
<th>RPTR</th>
<th>Line</th>
<th>RPTR</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>D</td>
<td></td>
</tr>
</tbody>
</table>

Victim data entry
Write Hit Operation on Value Miss (3/3)

Update of LSC

- Select a victim data entry
- Invalidate tags associated with the victim data entry
- Remove the accessed tag from the TLIST
- Associate the accessed tag with the written value
- Add the accessed tag to the TLIST of the written value
Write Hit Operation on Value Miss (3/3)

Update of LSC

- Select a victim data entry
- Invalidate tags associated with the victim data entry
- Remove the accessed tag from the TLIST
- Associate the accessed tag with the written value
- Add the accessed tag to the TLIST of the written value
Structure

- How does LSC efficiently search for a line value?
  - Horizontally split the data array
  - Limit a placement of a line within a data set

Tag Array

<table>
<thead>
<tr>
<th>Tag</th>
<th>FPTR</th>
<th>TLIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data Array

<table>
<thead>
<tr>
<th>Line</th>
<th>R PTR</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>
How does LSC efficiently search for a line value?

- Horizontally split the data array
- Limit a placement of a line within a data set

**Tag Array**

<table>
<thead>
<tr>
<th>Tag</th>
<th>FPTR</th>
<th>TLIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Data Array**

<table>
<thead>
<tr>
<th>Line</th>
<th>RPTR</th>
<th>Line</th>
<th>RPTR</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How does LSC efficiently search for a line value?

- Horizontally split the data array
- Limit a placement of a line within a data set

Search only within a data set
Read / Write Latency

- **Read latency**
  - As long as that of the conventional cache
    - FPTR access can overlap with a tag comparison

- **Write latency**
  - Longer than that of the conventional cache
    - Additional operation is required

  ➡️ **Writeback buffer is useful**
    - Write operations can overlap with executing following instructions