A Fully Associative Software-Managed Cache Design

Erik G. Hallnor and Steven K. Reindhardt

Presented By:
Maryam Sadooghi-Alvandi
Motivation

• Two Trends:
  • Growing DRAM access latency
  • Multi-megabyte on-chip caches
Motivation

• Two Trends:
  • Growing DRAM access latency
  • Multi-megabyte on-chip caches

Re-examine how caches are organized and managed
Motivation (II)

- Cache-DRAM relationship similar to DRAM-disk storage relationship
Virtual Memory

- Two mechanisms used in Virtual Memory:
  - Full Associativity
  - Software Management
Virtual Memory

- Two mechanisms used in Virtual Memory:
  - Full Associativity
  - Software Management

Apply these to Caches
Goal

- A fully associative software-managed cache
The System

- Consists of 2 parts:
  - The Indirect Index Cache (IIC)
  - The Generational Replacement Algorithm
Potential Uses of A Software-Managed Cache

• More sophisticated replacement algorithms
• Reduced penalty for locking data
• Arbitrary partitioning of the data store
## Conventional Caches

<table>
<thead>
<tr>
<th>TAG</th>
<th>SET</th>
<th>OFFSET</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**MATCH?**

**MATCH?**
Conventional Caches

TAG | SET | OFFSET

MATCH? | MATCH?
Conventional Caches

Table structure:
- **TAG**
- **SET**
- **OFFSET**

Decision points:
- MATCH?
Full Associativity

DATA ARRAY
Full Associativity

Data block may be placed in any location
Full Associativity

Data block may be placed in any location
Full Associativity

Data block may be placed in any location
Full Associativity

Data block may be placed in any location
<table>
<thead>
<tr>
<th>TAG</th>
<th>OFFSET</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>HASH</th>
</tr>
</thead>
</table>

|--------|--------|--------|--------|--------|

**IIC**
<table>
<thead>
<tr>
<th>TAG</th>
<th>OFFSET</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### IIC

<table>
<thead>
<tr>
<th>TAG</th>
<th>OFFSET</th>
<th>TAG</th>
<th>STATUS</th>
<th>INDEX</th>
<th>REPL.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **HASH**
  - TE
  - TE
  - TE
  - TE

**MATCH?**
- MATCH?
- MATCH?
- MATCH?
- MATCH?
- MATCH?
IIC: Storage Overhead
IIC: Storage Overhead
IIC: Storage Overhead
IIC: Storage Overhead

TAG | OFFSET
--- | ---

TAG | STATUS | INDEX | REPL.
--- | --- | --- | ---

HASH

TE

MATCH?

TE

MATCH?

TE

MATCH?

TE

MATCH?
IIC: Storage Overhead

TAG | OFFSET
--- | ---

TAG | STATUS | INDEX | REPL.
--- | --- | --- | ---

TE
TE
TE
TE
TE

MATCH?
MATCH?
MATCH?
MATCH?
MATCH?
IIC: Storage Overhead

TAG | OFFSET
---|---

TAG | STATUS | INDEX | REPL.
---|---|---|---

MATCH?

MATCH?

MATCH?

MATCH?

MATCH?
IIC: Timing Overhead

- Accessing tag and data arrays sequentially
- Traversing the hash chain
- Overhead of software management
Generational Replacement: Motivation

L1

L2
Generational Replacement: Motivation

repeatedly referenced

L1

L2
Generational Replacement: Motivation

L1

repeatedly referenced

L2

L2 does not see all references
Generational Replacement: Motivation

- LRU looks at \textit{when} block was last accessed
- GR emphasizes \textbf{how many} times a block is accessed over when it was accessed
- Priority for staying given to blocks that are repeatedly accessed
  - even though they may not be the most recently used
- Cope with long intervals between accesses
On A Hit ...

- FRESH POOL
- POOL 0
  - least frequently used
- POOL 1
  - POOL 2
  - POOL 3
  - most frequently used
On A Miss ...

FRESH POOL

POOL 0

POOL 1

POOL 2

POOL 3

least frequently used

most frequently used
On A Miss ...

<table>
<thead>
<tr>
<th>FRESH POOL</th>
<th>POOL 0</th>
<th>POOL 1</th>
<th>POOL 2</th>
<th>POOL 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>least frequently used</td>
<td>least frequently used</td>
<td>least frequently used</td>
<td>most frequently used</td>
<td>most frequently used</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
On A Miss ...

FRESH POOL

POOL 0

POOL 1

POOL 2

POOL 3

least frequently used

most frequently used
On A Miss ...

<table>
<thead>
<tr>
<th>FRESH POOL</th>
<th>POOL 0</th>
<th>POOL 1</th>
<th>POOL 2</th>
<th>POOL 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Least frequently used

Most frequently used
### On A Miss ...

<table>
<thead>
<tr>
<th>FRESH POOL</th>
<th>POOL 0</th>
<th>POOL 1</th>
<th>POOL 2</th>
<th>POOL 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>least frequently used</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>most frequently used</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **POOL 0**: 0
- **POOL 1**: Empty
- **POOL 2**: Empty
- **POOL 3**: 0
On A Miss ...

FRESH POOL

![Diagram showing pool usage]

least frequently used

most frequently used
On A Miss ...

<table>
<thead>
<tr>
<th>FRESH POOL</th>
<th>POOL 0</th>
<th>POOL 1</th>
<th>POOL 2</th>
<th>POOL 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>least frequently used</td>
<td>most frequently used</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
GR vs. LRU

POOL 0
- B1: 0
- B2: 0
- B3: 0

POOL 1

POOL 2

HIT

1
GR vs. LRU

POOL 0

B1 | 1
B2 | 0
B3 | 0

POOL 1

POOL 2

HIT
GR vs. LRU

<table>
<thead>
<tr>
<th>POOL 0</th>
<th>POOL 1</th>
<th>POOL 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>B2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B3</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
### GR vs. LRU

<table>
<thead>
<tr>
<th>POOL 0</th>
<th>POOL 1</th>
<th>POOL 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B3</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**MISS**

1. **POOL 0**
   - B1: 1
2. **MISS**
GR vs. LRU

- Pool 0:
  - B2: 0
  - B3: 0

- Pool 1:
  - B1: 1

- Pool 2:

MISS

Note: The diagram shows the comparison between GR (Group Replacement) and LRU (Least Recently Used) algorithms for cache management.
GR vs. LRU

POOL 0
- B2: 0
- B3: 0

POOL 1
- B1: 1

POOL 2

MISS

MISS
GR vs. LRU

<table>
<thead>
<tr>
<th>POOL 0</th>
<th>POOL 1</th>
<th>POOL 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>B2 0</td>
<td>B1 0</td>
<td></td>
</tr>
<tr>
<td>B3 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
GR vs. LRU

POOL 0
- B2: 0
- B3: 0

POOL 1
- B1: 0

POOL 2

3 hits
GR vs. LRU
GR vs. LRU

POOL 0
B2 0
B3 0

POOL 1
B1 I

POOL 2
GR vs. LRU

POOL 0

<table>
<thead>
<tr>
<th>B2</th>
<th>0</th>
</tr>
</thead>
</table>

| B3 | 0 |

POOL 1

| B1 | 1 |

POOL 2

HIT
GR vs. LRU

POOL 0

B2 | 1
B3 | 0

POOL 1

B1 | 1

POOL 2

HIT

The diagram illustrates a comparison between GR (Guaranteed Replacement) and LRU (Least Recently Used) cache algorithms. The image shows the status of different cache pools and how cache blocks are managed.
GR vs. LRU

POOL 0
- B2: 1
- B3: 0

POOL 1
- B1: 1

POOL 2
- Empty
GR vs. LRU

POOL 0

B2 | 1
B3 | 0

POOL 1

B1 | 1

POOL 2
GR vs. LRU

**POOL 0**
- B2: 1
- B3: 0

**POOL 1**
- Empty

**POOL 2**
- B1: 1

MISS 5
GR vs. LRU

POOL 0
- B2
- B3

MISS

POOL 1

MISS

POOL 2
- B1
- 0
GR vs. LRU

POOL 0

POOL 1
B2 | 1

POOL 2
B1 | 0

MISS
GR vs. LRU

POOL 0
B3 0

POOL 1
B2 0

POOL 2
B1 0

5 MISS
GR vs. LRU

POOL 0
- B3: 0

POOL 1
- B2: 0

POOL 2
- B1: 0

MISS 5

MISS
GR vs. LRU

POOL 0
- B3: 0

POOL 1
- B2: 0

POOL 2
- B1: 0
GR vs. LRU

MISS

POOL 0
- B3: 0

POOL 1
- B2: 0

POOL 2
- B1: 0

MISS 6
GR vs. LRU

POOL 0: B3 0
POOL 1: B2 0
POOL 2: B1 0

EVICTION

MISS
GR vs. LRU

POOL 0

POOL 1

B2 0

POOL 2

B1 0

MISS 6
GR vs. LRU

MISS

POOL 0

4
B2  0

POOL 1

POOL 2

3
B1  0

MISS
GR vs. LRU

POOL 0
4
B2 0

POOL 1
3
B1 0

POOL 2

MISS
GR vs. LRU
GR vs. LRU

POOL 0

B2 | 0

POOL 1

B1 | 0

POOL 2

MISS
• Number of operations on a miss: proportional to the number of priority queues
GR: Storage Overhead

- FIFO queues: doubly linked lists
- Two 12-bit pointers per block
- Head and tail pointers per queue
- Two full timestamps per queue
- 8-bit timestamps per block
Evaluation

- Windows NT system address traces on Intel Architecture platform
- S/390 trace running OLTP workload
Simulations

- 64KB, 2-way, split L1 cache with 32B blocks
- 1MB L2 cache
Miss vs. Associativity

- LRU 128
- LRU 256
- LRU 512
- OPT 128
- OPT 256
- OPT 512

Graph showing the comparison between LRU and OPT with varying associativities (128, 256, 512) and the TPCC_LONG benchmark.
Miss vs. Associativity

![Graph showing performance comparison between different cache sizes (LRU 128, LRU 256, LRU 512, OPT 128, OPT 256, OPT 512) across TPCC_LONG benchmarks. The graph indicates better performance with higher cache sizes.](image-url)
Miss vs. Associativity

Misses with various associativities and cache sizes for LRU and OPT algorithms.

- **LRU 128**: Green circles
- **LRU 256**: Blue circles
- **LRU 512**: Orange circles
- **OPT 128**: Red circles
- **OPT 256**: Purple circles
- **OPT 512**: Gray circles

The graph shows a comparison of misses for different cache sizes (4, 8, 16, 32, 64, 128, 256) and compares LRU and OPT algorithms. The label 'TPCC_LONG' is used for the x-axis labels.
Miss vs. Associativity

- LRU 128
- LRU 256
- LRU 512
- OPT 128
- OPT 256
- OPT 512

BETTER
Miss vs. Associativity

- LRU 128
- LRU 256
- LRU 512
- OPT 128
- OPT 256
- OPT 512

SpecWeb
Miss vs. Associativity

- LRU 128
- LRU 256
- LRU 512
- OPT 128
- OPT 256
- OPT 512

BETTER

SPECWEB
Miss vs. Associativity

SPECWEB

LRU 128  LRU 256  LRU 512
OPT 128  OPT 256  OPT 512

BETTER
Results (I)

![Graph showing millions of misses for different cache configurations]

- LRU 4-WAY
- LRU 8-WAY
- LRU 16-WAY
- CLOCK FA
- GEN FA
- LRU FA
- OPT FA

The graph indicates that adding a color to the LRU configuration results in better performance compared to the other configurations.
Miss vs. Associativity

Bar chart showing the number of misses for different cache configurations:

- LRU 4-WAY
- LRU 8-WAY
- LRU 16-WAY
- CLOCK FA
- GEN FA
- LRU FA
- OPT FA

The chart indicates that LRU 4-WAY has the highest number of misses, followed by LRU 8-WAY and LRU 16-WAY, with CLOCK FA, GEN FA, LRU FA, and OPT FA having significantly fewer misses. The arrow pointing to lower numbers indicates a better performance.
Results (II)

Bar chart showing the number of misses for different cache configurations.

- LRU 4-WAY
- LRU 8-WAY
- LRU 16-WAY
- CLOCK FA
- GEN FA
- LRU FA
- OPT FA

Legend:
- Better
- SPECWEB
Miss vs. Associativity

Thousands of Misses

- LRU 4-WAY
- + PG COLOR
- + VICTIM
- LRU 8-WAY
- LRU 16-WAY
- CLOCK FA
- GEN FA
- LRU FA
- OPT FA

SPECWEB

BETTER
Results (III)

- LRU 4-WAY
- LRU 8-WAY
- LRU 16-WAY
- CLOCK FA
- GEN FA
- LRU FA
- OPT FA

Thousands of Misses

TPCC_LONG

BETTER
Thank you.
Generational Replacement: Motivation

- L1 hits filtered from L2’s observed reference stream
- A block not referenced between 2 misses more likely to be in the program’s working set