CHROME: Concurrency-Aware Holistic Cache Management Framework with Online Reinforcement Learning

Xiaoyang Lu, Hamed Najafi, Jason Liu, Xian-He Sun
Cache Management

Cache Management: Essential for bridging the performance gap between fast CPU and slower main memory

Cache Replacement
• Determines which cache blocks to evict when new data needs to be loaded

Cache Bypassing
• Decides whether incoming data should be stored in the cache

Prefetching
• Predictively loads data into the cache before it is actually requested by the CPU
Limitations of Current Cache Management Schemes

We observe there are **two common limitations** faced by traditional cache management techniques:

1. **Lack of Holistic View**
   - Current schemes often examine cache replacement, bypassing, and prefetching in isolation, overlooking the potential benefits that could arise from a joint optimization strategy.

2. **Lack of Adaptability**
   - Current schemes often rely on fixed heuristics that don't account for the changing access patterns of modern applications and system configurations.
Lack of Holistic View

Inspecting Unresued Blocks in LLC with Gilder management scheme [MICRO'19]. Next-line prefetcher is used at L1 and stride prefetcher is used at L2.

83.7% of evicted blocks in shared LLC are not reused before eviction; 70.0% of the blocks that are not reused before eviction are attributed to prefetching.
Inspecting Unresued Blocks in LLC with Gilder management scheme [MICRO’19]. Next-line prefetcher is used at L1 and stride prefetcher is used at L2.

83.7% of evicted blocks in shared LLC are **not reused before eviction**; 70.0% of the blocks that are not reused before eviction are attributed to **prefetching**.

28.0% of evicted blocks are **not reused before eviction**, but are **requested again** in the future.
Lack of Holistic View

Inspecting Unresued Blocks in LLC with Gilder management scheme [MICRO’19]. Next-line prefetcher is used at L1 and stride prefetcher is used at L2.

83.7% of evicted blocks in shared LLC are not reused before eviction; 70.0% of the blocks that are not reused before eviction are attributed to prefetching.

28.0% of evicted blocks are not reused before eviction, but are requested again in the future.

Possible enhancement: integrates cache bypassing and replacement policies with pattern-based prefetching.
A holistic cache management scheme is needed:

- **Cache bypassing** needs to be utilized to identify the blocks accessed only once.
- **Cache replacement** needs to be aware of prefetching, to avoid the eviction of vital data.

83.7% of evicted blocks in shared LLC are not reused before eviction; 70.0% of them come from prefetching. 28.0% of evicted blocks are not reused before eviction, but requested again in the future.
Lack of Adaptivity

Comparing speedup over LRU on a 4-core system between: (a) using next-line prefetcher at L1 and stride prefetcher at L2, and (b) using stride prefetcher at L1 and streamer prefetcher at L2.

Three state-of-the-art cache management schemes:
Hawkeye [ISCA’16]
Glider [MICRO’19]
Mockingjay [HPCA’22]

Inconsistent performance across different workloads
Lack of Adaptivity

Comparing speedup over LRU on a 4-core system between: (a) using next-line prefetcher at L1 and stride prefetcher at L2, and (b) using stride prefetcher at L1 and streamer prefetcher at L2.

Three state-of-the-art cache management schemes:
- Hawkeye [ISCA’16]
- Glider [MICRO’19]
- Mockingjay [HPCA’22]

Inconsistent performance across different workloads
Performance varies among diverse system configurations
Lack of Adaptness

Comparing speedup over LRU on a 4-core system between: (a) using next-line prefetcher at L1 and stride prefetcher at L2, and (b) using stride prefetcher at L1 and streamer prefetcher at L2.

Three state-of-the-art cache management schemes:
- Hawkeye [ISCA'16]
- Glider [MICRO’19]
- Mockingjay [HPCA'22]

Inconsistent performance across different workloads
Performance varies among diverse system configurations

Possible enhancement: adaptive framework to handle diverse workloads and system configurations
Lack of Adaptivity

Comparing speedup over LRU on a 4-core system between: (a) using next-line prefetcher at L1 and stride prefetcher at L2, and (b) using stride prefetcher at L1 and streamer prefetcher at L2.

An adaptive cache management scheme is needed:
• Automatically predict and adapt to various access patterns
• Aware the system and self-correct decisions dynamically

Possible enhancement: adaptive framework to handle diverse workloads and system configurations
Our Solution

A holistic cache management framework that dynamically adapts to various workloads and system configurations
Key Contributions: CHROME

**Holistic Integration**: Integrate cache bypassing and replacement with pattern-based prefetching

**Dynamic Online Learning**: Utilizes online reinforcement learning to adapt cache management to varying workloads and system configurations

**Multiple Program Features**: Employs multiple program features to achieve a thorough understanding of memory access patterns

**Concurrency-Aware Rewards**: Implements a reward system that is aware of concurrent accesses, factoring in system-level feedback for decision-evaluation

**Efficient Design**: Achieves a minimal hardware overhead
Reinforcement Learning (RL)

- **Autonomously** learn through **feedback** from actions and experiences in an **interactive** environment
- Algorithmic approach to learn to take an **action** in a given **situation** to **maximize** a numerical **reward**

- Agent stores **Q-values** for **every** state-action pair
  - **Expected return** for taking an action in a state
  - Given a state, selects action that provides **highest** Q-value
Why RL?

Adaptive online learning:
• Allows CHROME to continuously learn and adapt by receiving rewards from real-time interactions

Learning with multiple features
• Learning process is enriched by utilizing a wide range of program features

Environment-Derived Rewards
• Surpasses static, intuition-based methods by employing a dynamic reward system directly informed by environmental feedback

Acceptable overhead
• Does not require offline training and can be designed with smaller model size
• Q-values for state-action pairs can be stored in a lookup table
What is State?

- A vector of features for each access
- Feature: \{control-flow, data access\}
- Control-flow of demands examples:
  - PC (Program Counter), sequence of last 4 PCs, …
- Data-access examples:
  - memory address, page number, page offset, …
- \(S = (PC, \text{ page number})\)
- **Distinguish** between demand accesses and prefetch accesses
Formulating Cache Management as an RL Problem

What is Action?

• Eviction Priority Value (EPV)
  • Reflects the eviction priorities of the cache block
  • Three possible EPVs: low, moderate, high

• Cache miss (4 optional actions):
  • **Bypass** LLC
  • **Insert** the corresponding block in LLC with an EPV of low, moderate, or high

• Cache hit (3 optional actions):
  • **Update** the EPV of the corresponding block to low, moderate, or high
What is Reward?

- The rewards of CHROME:
  - Reflect the accuracy of each action
  - Distinguish between actions triggered by demand or prefetching
  - Take into account system-level feedbacks

- Eight distinct reward levels:
  - **Accuracy**: Encourages CHROME to make precise decisions, reducing cache misses
  - **Prefetching Awareness**: Motivates CHROME to prioritize blocks likely to be requested next by demand accesses over those that might be requested by prefetch accesses
  - **Concurrency-Aware System Feedback**: Identifies cores causing LLC obstruction at runtime, promoting actions that mitigate the obstruction
CHROME Overview

RL Decision

Bypass or assign EPV on a cache miss; update EPV on a cache hit

Last-Level Cache (LLC)

Sampled Set

Sampled Set

Sampled Set

Hit/Miss?

Observed Features

RL Decision

Assign reward to corresponding EQ entry

Evaluation Queue (EQ)

Update Q-Table

Insert corresponding memory address & state-action pair in EQ

Q-Table

A1 A2 A3 A4

S1

S2

Sn

RL Training

CHROME: Concurrency-Aware Holistic Cache Management Framework with Online Reinforcement Learning
**Q-Table:**

- Tracks the Q-values of all observed state-action pairs
- Given a state, CHROME picks a reasonable action based on the Q-Table

**Evaluation Queue:**

- Several first-in-first-out queues, each with a fixed capacity
- Records the actions of CHROME within a temporal window, which assists in rewarding
CHROME Workflow

**RL Decision**

- **D** Bypass or assign EPV on a cache miss; update EPV on a cache hit

**Last-Level Cache (LLC)**

- Sampled Set
- Sampled Set
- Sampled Set

**RL Training**

- **A** Assign reward to corresponding EQ entry

**Evaluation Queue (EQ)**

- Evaluation Queue (EQ)

**Q-Table**

- A1
- A2
- A3
- A4
- S1
- S2
- Sn

**Observed Features**

- Hit/Miss?

**Update Q-Table**

- **F** Update Q-Table

**Insert corresponding memory address & state-action pair in EQ**

- **E** Insert corresponding memory address & state-action pair in EQ
CHROME Workflow

**RL Decision**

- **D** Bypass or assign EPV on a cache miss; update EPV on a cache hit

**Last-Level Cache (LLC)**

- Sampled Set
- Sampled Set
- Sampled Set

**Hit/Miss?**

- Observed Features

**Q-Table**

<table>
<thead>
<tr>
<th></th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sn</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**State**

**Action**

**Assign reward to corresponding EQ entry**

**Update Q-Table**

**Insert corresponding memory address & state-action pair in EQ**

**Evaluation Queue (EQ)**
**CHROME Workflow**

**RL Decision**
- **Bypass or assign EPV on a cache miss; update EPV on a cache hit**

**Last-Level Cache (LLC)**
- Sampled Set
- Sampled Set
- Sampled Set

**Hit/Miss?**
- **Hit**/Miss?
  - **Hit**
  - **Miss**

**Evaluation Queue (EQ)**
- **Assign reward to corresponding EQ entry**

**Q-Table**
- **State**
  - S1
  - S2
  - Sn
- **Action**
  - A1
  - A2
  - A3
  - A4

**Update Q-Table**
- **Insert corresponding memory address & state-action pair in EQ**
CHROME Workflow

<table>
<thead>
<tr>
<th>RL Decision</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>LLC Request</td>
<td>Hit/ Miss?</td>
<td>Observed Features</td>
<td>Q-Table</td>
<td>Evaluation Queue (EQ)</td>
<td>Update Q-Table</td>
</tr>
<tr>
<td>Last-Level Cache (LLC)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sampled Set</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sampled Set</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sampled Set</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**RL Decision**
- **D**: Bypass or assign EPV on a cache miss; update EPV on a cache hit

**RL Training**
- **A**: Assign reward to corresponding EQ entry

**Q-Table**
- States: S1, S2, ..., Sn
- Actions: A1, A2, A3, A4

**Evaluation Queue (EQ)**
- Sampled Set

**Update Q-Table**
- Insert corresponding memory address & state-action pair in EQ
More in the Paper

- Details on concurrency-aware system-level feedback
- Insights into the reward systems
- Pipelined organization of Q-Table
- EQ organization and Q-value update
- Turing of the hyper-parameter
- Overhead analysis
Simulation Methodology

- **Champsim** trace-driven simulator
- **57** memory-intensive workload traces
  - SPEC CPU2006 and CPU2017
  - GAP

- **Homogeneous** and **heterogeneous** multi-core mixes

- **Prefetchers:**
  - L1D: Next-line prefetcher
  - L2: Stride prefetcher

- **Five** state-of-the-art LLC management schemes:
  - LRU
  - Hawkeye [ISCA’16]
  - Glider [MICRO’19]
  - Mockingjay [HPCA’22]
  - CARE [HPCA’23]
Performance with Varying Core Count

SPEC workloads homogeneous

<table>
<thead>
<tr>
<th>Core Count</th>
<th>Hawkeye</th>
<th>Glider</th>
<th>Mockingjay</th>
<th>CARE</th>
<th>CHROME</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-core</td>
<td>9.2%</td>
<td>9.2%</td>
<td>9.2%</td>
<td>9.2%</td>
<td>12.9%</td>
</tr>
<tr>
<td>8-core</td>
<td>10.6%</td>
<td>10.6%</td>
<td>10.6%</td>
<td>10.6%</td>
<td>12.9%</td>
</tr>
<tr>
<td>16-core</td>
<td>12.9%</td>
<td>12.9%</td>
<td>12.9%</td>
<td>12.9%</td>
<td>14.4%</td>
</tr>
</tbody>
</table>

SPEC workloads heterogeneous

<table>
<thead>
<tr>
<th>Core Count</th>
<th>Hawkeye</th>
<th>Glider</th>
<th>Mockingjay</th>
<th>CARE</th>
<th>CHROME</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-core</td>
<td>9.6%</td>
<td>9.6%</td>
<td>9.6%</td>
<td>9.6%</td>
<td>12.9%</td>
</tr>
<tr>
<td>8-core</td>
<td>12.9%</td>
<td>12.9%</td>
<td>12.9%</td>
<td>12.9%</td>
<td>14.4%</td>
</tr>
<tr>
<td>16-core</td>
<td>14.4%</td>
<td>14.4%</td>
<td>14.4%</td>
<td>14.4%</td>
<td>14.4%</td>
</tr>
</tbody>
</table>
Performance with Varying Core Count

CHROME can accurately provide cache management for different workloads

CHROME outperforms all other schemes across all system configurations

Performance advantage of CHROME over others increases with more cores
Performance on Unseen Traces

GAP workloads

<table>
<thead>
<tr>
<th></th>
<th>Hawkeye</th>
<th>Glider</th>
<th>Mockingjay</th>
<th>CARE</th>
<th>CHROME</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-core</td>
<td>9.5%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8-core</td>
<td></td>
<td></td>
<td>12.1%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16-core</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16.0%</td>
</tr>
</tbody>
</table>

CHROME: Concurrency-Aware Holistic Cache Management Framework with Online Reinforcement Learning
Performance on Unseen Traces

The holistic view provides a performance guarantee

Online RL provides good adaptability and scalability
Summary

CHROME is a holistic cache management framework.

CHROME continuously learns the policy by utilizing online RL.

CHROME considers multiple program features and concurrency-aware system-level feedback information.

CHROME outperforms state-of-the-art cache management schemes.
CHROME:
Concurrency-Aware Holistic Cache Management Framework with Online Reinforcement Learning

Xiaoyang Lu, Hamed Najafi, Jason Liu, Xian-He Sun