Rethinking Belady’s Algorithm to Accommodate Prefetching

Akanksha Jain
Calvin Lin

June 4, 2018
Motivation

• Belady’s MIN algorithm (1966) is provably optimal for minimizing cache misses
  – Evicts the line that is reused furthest in the future

• *Is MIN optimal in the presence of a prefetcher?*
  – Surprisingly, the answer is no
Belady’s MIN With Prefetching

- MIN does not distinguish between demand loads and prefetches

<table>
<thead>
<tr>
<th>Prefetch X</th>
<th>Prefetch X</th>
<th>Prefetch X</th>
<th>Prefetch X</th>
<th>Prefetch X</th>
<th>Load X</th>
</tr>
</thead>
<tbody>
<tr>
<td>t = 0</td>
<td>t = 1</td>
<td>t = 2</td>
<td>t = 3</td>
<td>t = 5</td>
<td>t = 6</td>
</tr>
</tbody>
</table>
Belady’s MIN With Prefetching

- MIN does not distinguish between demand loads and prefetches

<table>
<thead>
<tr>
<th>Time</th>
<th>Hit</th>
<th>Load X</th>
<th>Extra space to cache other demand requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>t = 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t = 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t = 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t = 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t = 5</td>
<td>Prefetch X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t = 6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Motivation

• Belady’s MIN algorithm (1966) is provably optimal for minimizing cache misses
  – Evicts the line that is reused furthest in the future

• *Is MIN still optimal in the presence of a prefetcher?*
  – Surprisingly, the answer is no
  – It minimizes total misses, not demand misses
An Alternative Solution: Demand-MIN

- Demand-MIN minimizes demand misses

Not cached with Demand-MIN

<table>
<thead>
<tr>
<th>Prefetch X</th>
<th>Prefetch X</th>
<th>Prefetch X</th>
<th>Prefetch X</th>
<th>Prefetch X</th>
<th>Prefetch X</th>
</tr>
</thead>
<tbody>
<tr>
<td>t = 0</td>
<td>t = 1</td>
<td>t = 2</td>
<td>t = 3</td>
<td>t = 5</td>
<td>t = 6</td>
</tr>
</tbody>
</table>

Hit

Load X

Demand miss reduction over LRU (%)

Averaged over 100s of multiprogrammed SPEC workloads
An Alternative Solution: Demand-MIN

- Demand-MIN minimizes demand misses
- But it increases prefetcher traffic

4 extra prefetches

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>t = 0</td>
<td>Prefetch</td>
</tr>
<tr>
<td>t = 1</td>
<td>Prefetch</td>
</tr>
<tr>
<td>t = 2</td>
<td>Prefetch</td>
</tr>
<tr>
<td>t = 3</td>
<td>Prefetch</td>
</tr>
<tr>
<td>t = 5</td>
<td>Prefetch</td>
</tr>
<tr>
<td>t = 6</td>
<td>Load X</td>
</tr>
</tbody>
</table>

Traffic reduction over LRU (%)

MIN
Demand-MIN

Averaged over 100s of multiprogrammed SPEC workloads
Our Contributions

• We introduce Demand-MIN, a new algorithm that minimizes demand misses in the presence of prefetches

• We show that the space between MIN and Demand-MIN is complex
Our Contributions

• We introduce **Demand-MIN**, a new algorithm that minimizes demand misses in the presence of prefetches

• We show that the space between MIN and Demand-MIN is complex, and we introduce **Flex-MIN** to navigate this space

• We introduce **Harmony**, a new practical cache replacement policy that builds on our prior work
Outline

• Motivation
• Demand-MIN and Flex-MIN
• Harmony
• Evaluation
Demand-MIN

- Demand-MIN evicts lines that will be prefetched in the future
- Evict the line that will be *prefetched* furthest in the future. If no such line exists, default to MIN
Usage Intervals

• We view cache accesses as *usage intervals*
  – The period between two consecutive accesses to X
  – Endpoints of usage intervals can be demand or prefetch requests
Usage Intervals

• We view cache accesses as *usage intervals*
  – The period between two consecutive accesses to X
  – Endpoints of usage intervals can be demand or prefetch requests

```
P-P intervals

Prefetch X  Prefetch X  Prefetch X  Prefetch X  Prefetch X  Load X

P-D intervals

t = 0  t = 1  t = 2  t = 3  t = 5  t = 6

Time
```
Demand-MIN

- Demand-MIN evicts usage intervals that end with a prefetch
  - Demand-MIN evicts \( *-P \) intervals
Demand-MIN reduces demand MPKI by 28% over MIN.

Demand-MIN increases traffic by 60% over MIN.
Tradeoff Depends On Workload

Memory traffic (% increase)

- sphinx3_883B
- tonto_2834B
- calculix_2670B
- bwaves_1609B
- cactusADM_734B

Demand hit rate %
Tradeoff Depends On Workload

Demand-MIN
Belady’s MIN

- sphinx3_883B
- tonto_2834B
- calculix_2670B
- bwaves_1609B
- cactusADM_734B

Memory traffic (% increase)

Demand hit rate %
Tradeoff Depends On Workload

- **sphinx3_883B**
- **tonto_2834B**
- **calculix_2670B**
- **bwaves_1609B**
- **cactusADM_734B**

**Demand-MIN**

**Belady’s MIN**
Tradeoff Depends On Workload

Memory traffic (% increase)

Demand hit rate %

- sphinx3_883B
- tonto_2834B
- calculix_2670B
- bwaves_1609B
- cactusADM_734B

- Demand-MIN
- Belady’s MIN
How Can We Navigate This Design Space?

• **Key Idea:** Some *-P intervals are more profitable to evict than others

![Diagram showing protected lines and undesirable/desirable tradeoffs for Load X and Load Y]
Flex-MIN

• Evict the line that is prefetched furthest in the future and is not protected. If no such line exists, default to MIN.
  – Short *-P intervals are good candidates for protection
  – Protected lines: Starting points of *-P intervals whose length is below a certain threshold

• Picks a point between MIN and Demand-MIN
Flex-MIN

Miss reduction over LRU (%)

Traffic reduction over LRU (%)

MIN  Demand-MIN  Flex-MIN

-20  -10  0  10  20  30  40

-20  -10  0  10  20
Flex-MIN strikes a good balance between Demand-MPKI and traffic.
Our Contributions

• We introduce Demand-MIN, a new algorithm that minimizes demand misses in the presence of prefetches.

• We show that the space between MIN and Demand-MIN is complex, and we introduce Flex-MIN to navigate this space.

• We introduce Harmony, a new practical cache replacement policy that builds on our prior work.
Harmony: A Practical Solution

- Builds on our recently proposed Hawkeye Cache [ISCA 2016]
  - Learns from the *optimal solution* for the past to predict future caching decisions
Harmony: A practical solution

• Builds on our recently proposed Hawkeye Cache

• Instead of learning from MIN, Harmony learns from Flex-MIN
Experimental Methodology

- Last-Level Cache
- Baseline: LRU, PACMan, SHiP+PACMan
PACMan

- Addresses cache pollution
- Lower priority for prefetchable lines
  - No update on prefetch cache hits
- Does not reason about the tradeoff between hit rate and traffic
Experimental Methodology

• Last-Level Cache
• Baseline: LRU, PACMan, SHiP+PACMan
• Prefetcher: PC-based stride prefetcher (more in paper)
• ChampSim simulator (Cache Replacement Championship)
  – Out-of-order core with a detailed memory system
• Multi-programmed SPEC workloads
  – 1-core, 4-cores (100 mixes) and 8-core (50 mixes)
Evaluation

![Graph showing speedup over LRU (%) for different core counts for PACMan and PACMan+SHiP. The graph indicates a higher speedup for PACMan+SHiP compared to PACMan as the core count increases. At 1 core, PACMan has 1.8% speedup, while PACMan+SHiP has 4.4% speedup. At 8 cores, PACMan+SHiP maintains a 4.4% speedup, whereas PACMan's speedup decreases to 1.8%.](image)
Evaluation

The chart shows the speedup over LRU (%) for different core counts for three different systems:

- **PACMan**
- **PACMan+SHiP**
- **Harmony**

The speedup percentages are as follows:

- **PACMan**:
  - 1 core: 3.3%
  - 4 cores: 7.7%
- **PACMan+SHiP**:
  - 1 core: 4.4%
  - 4 cores: 7.7%
- **Harmony**:
  - 1 core: 1.8%
  - 4 cores: 4.4%
  - 8 cores: 9.4%
Harmony Explores A Larger Design Space
Harmony Explores A Larger Design Space

Reduction in traffic vs. Reduction in MPKI

PACMan
Harmony Explores A Larger Design Space
Harmony Explores A Larger Design Space

Reduction in traffic vs. Reduction in MPKI for PACMan and Harmony.
Cache Replacement Championship [ISCA 2017]

Performance improvement over LRU (%) on 1 core

- Leeway
- MPPPB
- ReD
- SHiP++
- Harmony
Conclusions

• New design space to explore when considering cache replacement with prefetching

• Cache replacement policies can modulate prefetcher traffic

• Harmony is a practical solution that navigates this design space
Thank You

• Questions?
Other Prefetchers

![Graph showing MPKI and TPKI metrics for different prefetchers across Stride, AMPM, and BO contexts.](image)

- MPKI (lower is better)
- TPKI (lower is better)
Other Prefetchers

![Chart showing speedup over LRU (%)]

- Stride
- AMPM
- BO

- SHiP+PACMan
- Harmony
Interval Distribution

![Interval Distribution Diagram](image)

- **DD Intervals**
  - Stride: 43.5%
  - AMPM: 44.6%
  - BO: 49.4%

- **PD Intervals**
  - Stride: 12.8%
  - AMPM: 13.1%
  - BO: 14.0%

- **PP Intervals**
  - Stride: 30.8%
  - AMPM: 13.3%
  - BO: 12.9%

- **DP Intervals**
  - Stride: 14.7%
  - AMPM: 21.9%
  - BO: 14.0%