An Adaptive Filtering Mechanism for Energy Efficient Data Prefetching

Xianglei Dang, Xiaoyin Wang, Dong Tong, Zichao Xie, Lingda Li, Keyi Wang

Microprocessor Research & Development Center, Peking University, Beijing, China
Outline

- Background and Motivation
- Related Work
- The Proposed APF
- Design and Implementation
- Experimental Evaluation
- Conclusions
Outline

- Background and Motivation
- Related Work
- The Proposed APF
- Design and Implementation
- Experimental Evaluation
- Conclusions
Energy Efficient Data Prefetching

- With the strong consumer demand, embedded processors increasingly unitize techniques in general-purpose processor to improve performance
  - ARM Cortex-A15: superscalar out-of-order pipeline
- Prefetching: fetch data before the processor needs
  - Widely used in processors to tolerate memory latency
- Energy Efficiency is a major design constraint in embedded processor designs[ISCA’10]
  - As for data prefetching, the key to improve the energy efficiency is reducing the energy consumption while achieving good performance[CAL’11][TVLSI’11]
Useless Prefetches

- Useless prefetches: the prefetched data will be never used by the processor
  - Generated when the prediction of the future memory access addresses is wrong
  - Waste bandwidth and energy and also hurt performance due to the incurred cache pollution

- Aggressive prefetching[HPCA’07]
  - Use higher prefetch degree (the number of cache blocks prefetched each time): prefetch more data each time
  - Gains high performance but degrades the prefetch accuracy, thus issuing more useless prefetches
Impact of Aggressive Prefetching

- Prefetch degree (S/DC prefetcher): 4 vs 32
- It is critical for improving energy efficiency to reduce useless prefetches without hurting the performance

Performance: 48.57% vs 78.30%

Prefetch Accuracy: 82.73% vs 61.16%
Outline

- Background and Motivation
- Related Work
- The Proposed APF
- Design and Implementation
- Experimental Evaluation
- Conclusions
Hardware Data Prefetching Schemes

- Capture repetitive memory access patterns during running of programs to predict prefetch addresses
- Stride Prefetcher[TC’95]: captures sequences of addresses that differ by a constant value
- Markov Prefetcher[ISCA’97]: captures repetitive subsequences in the miss address stream
- Delta Correlation(DC) Prefetcher[HPCA&PACT’04]
  - Differences of consecutive addresses -> the delta stream
  - Captures repetitive subsequences in the delta stream
  - Have advantages in performance[MICRO’04]
- S/DC prefetcher[DATE’12]: handle stride & DC
Limit Useless Prefetches

- **DDPF (Dynamic Data Prefetch Filtering) [TC’07]**
  - Collects the history information about whether an issued prefetch has been used by the processor to determine issuing or filtering new generated prefetches.
  - It does not record whether a filtered prefetch is useful, thus filtering a great deal of useful prefetches, which hurts the performance of data prefetching.

- **PACMan [MICRO’11]:** combine prefetching with the cache insertion policy
  - Insert prefetched data into easily evicted blocks when data prefetching incurs performance loss.
  - It cannot reduce useless prefetches, which still waste the bandwidth and energy.
Outline

- Background and Motivation
- Related Work
- The Proposed APF
- Design and Implementation
- Experimental Evaluation
- Conclusions
Use counter to collect the utilization results of the issued and filtered prefetches by the processor to filter new generated useless prefetches.

For an issued prefetch: record the prefetch-victim address pair (only consider cache miss).

 Afterwards, three situations may happen:

- Situation1 – the prefetch address is first accessed by the processor: the prefetch is useful and should not be filtered (counter – 1)

- Situation2 – the victim address is first accessed by the processor: a useful block has been replaced by this prefetch, the prefetch should be filtered to avoid the loss (counter + 1)
APF – Essential Mechanism (1/2)

- Situation 3 – when a cache miss occurs, the prefetch block is replaced before the processor accesses it: the prefetch should be filtered to avoid the waste (counter + 1)

- For a filtered prefetch: record the prefetch address
  - Situation 1 – the prefetch address is accessed by the processor: the filtered prefetch is very likely useful and the filtering before is probably wrong (counter – 1)
  - As the feedback mechanism to avoid wrongly filtering more useful prefetches

- When a new prefetch address is generated
  - The value of the counter: determine issuing (counter <= 0) or filtering (counter > 0) this prefetch
Outline

- Background and Motivation
- Related Work
- The Proposed APF
  - Design and Implementation
- Experimental Evaluation
- Conclusions
APF – Structure

- Replaced/Filtered Table (RFT): record information of the issued and filtered prefetches
  - PV & PTag: prefetch address
  - VV & VTag: victim address
- Filter Counter (FC): identify the filtering status
  - Each entry is a 3-bit saturation counter, corresponding to a memory region
  - Update when searching RFT
Search and Update RFT

- **Search**: when the processor or the prefetcher access the cache
  - **Source1** – demand access address of the processor
    - Compare: the prefetch address and victim address in RFT
    - Found: *Situation1 & Situation2* of issued prefetches and *Situation1* of filtered prefetches
  - **Source2** – victim address of the new cache miss
    - Compare: the prefetch address in RFT
    - Found: *Situation3* of issued prefetches
  - Update the FC according the search results

- **Update**: when a prefetch is issued (record PAddr and VAddr) or filtered (only record PAddr)
Filtering Useless Prefetches

- Read counter value from FC: when a new prefetch address is generated (indexed by part of address)
  - counter value > 0: filter the prefetch
  - counter value <= 0: issue the prefetch

When a prefetch \( X \) is generated:
if \( FC[X.Addr] > 0 \)
   Filter the prefetch;
else if \( FC[X.Addr] <= 0 \)
   Issue the prefetch;
end if

- Update the RFT whether the new prefetch is issued or filtered
Outline

- Background and Motivation
- Related Work
- The Proposed APF
- Design and Implementation
  - *Experimental Evaluation*
- Conclusions
Simulation Environment

- Simulator: Simplescalar+Wattch+Simpoint
- Benchmarks: SEPC CPU2000 (Ref inputs)
- Baseline: A Typical 4-issue Superscalar Processor
- Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Feature</th>
<th>Parameters</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>S/DC</td>
<td>Pattern Prediction Table</td>
<td>32-entry</td>
<td>~0.32KB</td>
</tr>
<tr>
<td></td>
<td>Global History Buffer</td>
<td>64-entry</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Prefetch Degree</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>DDPF</td>
<td>Prefetch History Table</td>
<td>4096-entry</td>
<td>~1KB</td>
</tr>
<tr>
<td>PACMan</td>
<td>Set Dueling Monitor</td>
<td>3-entry</td>
<td>~0.01KB</td>
</tr>
<tr>
<td>APF</td>
<td>RFT</td>
<td>128-entry</td>
<td>~0.88KB</td>
</tr>
<tr>
<td></td>
<td>FC</td>
<td>16-entry</td>
<td></td>
</tr>
</tbody>
</table>
## Prefetch Accuracy

<table>
<thead>
<tr>
<th></th>
<th>S/DC</th>
<th>S/DC+DDPF</th>
<th>S/DC+PACMan</th>
<th>S/DC+APF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefetch Accuracy</td>
<td>61.81%</td>
<td>66.91%</td>
<td>59.56%</td>
<td>69.37%</td>
</tr>
</tbody>
</table>

DDPF and APF improve the prefetch accuracy by filtering useless prefetches.
**Reduction of Useless/Useful Prefetches**

<table>
<thead>
<tr>
<th>Method</th>
<th>Useless Pref</th>
<th>Useful Pref</th>
</tr>
</thead>
<tbody>
<tr>
<td>S/DC+DDPF</td>
<td>92.59%</td>
<td>63.69%</td>
</tr>
<tr>
<td>S/DC+PACMan</td>
<td>11.99%</td>
<td>3.55%</td>
</tr>
<tr>
<td>S/DC+APF</td>
<td>53.81%</td>
<td>5.28%</td>
</tr>
</tbody>
</table>

DDPF reduces a large number of useless prefetches as well as useful prefetches.

Our APF also reduces a large number of useless prefetches, and its adverse effect on useful prefetches is very limited.
## Performance Comparison

<table>
<thead>
<tr>
<th>S/DC</th>
<th>S/DC+DDPF</th>
<th>S/DC+PACMan</th>
<th>S/DC+APF</th>
</tr>
</thead>
<tbody>
<tr>
<td>48.10%</td>
<td>18.26%</td>
<td>48.85%</td>
<td>51.25%</td>
</tr>
</tbody>
</table>

DDPF degrades the performance of S/DC since it filters too many useful prefetches.

Our APF’s adverse effect on useful prefetches is very limited. It improves the performance of S/DC by an average of 2.12%.
## Reduction of L2 Cache Accesses

### Increased L2 Accesses by Prefetching

<table>
<thead>
<tr>
<th>Method</th>
<th>S/DC</th>
<th>S/DC+ DDPF</th>
<th>S/DC+ PACMan</th>
<th>S/DC+ APF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>79.80%</td>
<td>17.42%</td>
<td>73.60%</td>
<td>56.18%</td>
</tr>
</tbody>
</table>

DDPF and APF can significantly reduce the increased L2 cache accesses of S/DC by filtering useless prefetches.
### Reduction of Memory Accesses

#### Increased Memory Accesses by Prefetching

<table>
<thead>
<tr>
<th></th>
<th>S/DC</th>
<th>S/DC+DDPF</th>
<th>S/DC+PACMan</th>
<th>S/DC+APF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>174.56%</td>
<td>8.76%</td>
<td>105.52%</td>
<td>69.96%</td>
</tr>
</tbody>
</table>

DDPF and APF can significantly reduce the increased memory accesses of S/DC by filtering useless prefetches.
Reduction of the L2 Cache Energy

DDPF reduces the dynamic energy by filtering useless prefetches but increases leakage energy due to the performance loss.

Our APF further reduces the L2 cache energy of S/DC by an average of 6.19%.
Energy Efficiency Comparison

<table>
<thead>
<tr>
<th>Energy Efficiency Speedup</th>
<th>S/DC</th>
<th>S/DC+DDPF</th>
<th>S/DC+PACMan</th>
<th>S/DC+APF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>15.97%</td>
<td>7.35%</td>
<td>17.07%</td>
<td>17.95%</td>
</tr>
</tbody>
</table>

Our APF further improves the energy efficiency of S/DC by filtering useless prefetches.

\[\text{ISCA'10}: \text{Energy Efficiency} = \frac{\text{Energy}}{\text{per Operation}}\]
Outline

- Background and Motivation
- Related Work
- The Proposed APF
- Design and Implementation
- Experimental Evaluation
- Conclusions
Conclusions

- **Adaptive Prefetch Filtering (APF)**
  - Adaptively determines issuing or filtering new generated prefetches by counteracting history information about whether the issued prefetches are useful
  - Builds the feedback mechanism of filtering using the history information about whether the filtered prefetches would be used by the processor

- **Benefits**
  - Reduces useless prefetches with very little negative effect on useful prefetches, thus reducing the wasted bandwidth and energy without hurting the performance
  - Improve the average performance of prefetching by reducing the cache pollution
Thanks!
(Q & A)