CS7810 Prefetching

Seth Pugsley
Predicting the Future

- Where have we seen prediction before?
  - Does it always work?
- Prefetching is prediction
  - Predict which cache line will be used next, and place it in the cache before it is used
  - A processor cache is required
Data Accesses

• What is it about data accesses in programs that makes them predictable?
  – Spatially predictable
  – Temporally predictable

• Virtual page boundaries

• Regular vs Irregular
Some patterns are very easy to accurately predict.
Irregular Data Access

- Pointer-chasing patterns are impossible to predict without large, comprehensive histories.
Some applications have a combination of both types. For example, a linked list where each item is several cache lines long.
Cache Line Size

• Overfetching and Prefetching
  – Main memory access granularity vs program access granularity

• Large cache lines have more spatial locality
  – Why not just use huge cache lines?
  – Where is the limit?
Next Line Prefetching

• For every cache line A that is fetched, also prefetch A+1
  – There is no intelligence or decision making, it always performs the same action
• How is this different from having twice the cache line size?
  – Alignment
  – Eviction granularity
Next Line Prefetching
Prefetch Aggressiveness

• Cache Capacity
  – Prefetched lines take up cache capacity

• Memory Bandwidth
  – Prefetched lines use memory bandwidth

• Accuracy vs Cache Miss Coverage
  – Fundamentally at odds with one another

• Timeliness
  – Does the prefetched line arrive on time?

• How does a Next Line Prefetcher fare in these metrics in the best case? Worst case?
Stream Prefetcher

• Prefetch multiple +1 lines ahead
• Requires confirmation before any action is taken
  – Stream started on access A
  – Stream direction determined on access A+1
  – Stream confirmed on access A+2
  – Begin prefetching A+3
• Intelligence and bookkeeping required to identify and confirm these streams
• Prefetch degree is how many cache lines are prefetched at a time
Stream Prefetchers are good at prefetching very long regular streams.
### Stream Prefetcher

<table>
<thead>
<tr>
<th>First Access</th>
<th>Second Access</th>
<th>Direction</th>
<th>Next Expected</th>
<th>Next Prefetch</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Accesses: 0, 1, 2, 10, 11, 3, 12, 4, 5
Prefetched:
Stride Prefetcher

• Like a stream prefetcher, but with variable access stride (not always +1)
  – More bookkeeping to determine stride
• Also requires confirmation before prefetching
  – Allocate stream on access A
  – Determine direction and stride on access A+X
  – Confirm stream on access A+2*X
  – Begin prefetching A+3*X
## Stride Prefetcher

<table>
<thead>
<tr>
<th>First Access</th>
<th>Second Access</th>
<th>Stride</th>
<th>Next Expected</th>
<th>Next Prefetch</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Accesses: 0, 4, 8, 53, 56, 101, 12, 16, 20, 59, 62, 65
Prefetched:
Common Regular Data Prefetchers

• If you understand these, then you’re off to a good start
  – Next Line
  – Stream
  – Stride

• Now we will introduce more advanced topics
Correlation Prefetchers

• Correlate events in a history
  – Assume history repeats itself

• History Table
  – Indexed by some key, e.g., PC or load address
  – Table entry tells prefetching algorithm what to do

• Can correlate a variety of things
  – Total access order (on a pair by pair basis)
  – Distances between accesses
Correlation Prefetchers

<table>
<thead>
<tr>
<th>Index</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

Accesses: 0, 7, 3, 6, 2, 1, 5, 4, 7, 3, 0, 7, 2, 1, 5, 0, 4, 7, 2, 1, 5, 0, 7, 3, 6
Global History Buffer

• Nesbit and Smith, 2005
• Instead of just one history table, uses an index table and global history buffer
  – Index table is accessed by directly indexing into it
  – GHB is a FIFO with pointers between entries
• This can be used as a framework to implement other prefetchers
• Still very popular basis for new prefetchers
## Global History Buffer

<table>
<thead>
<tr>
<th>Index</th>
<th>Ptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

### History

Accesses: 0, 2, 3, 1, 2, 0, 2, 3, 0, 2, 3, 1, 2, 0, 2, 3, 0, 2
Access Map Pattern Matching

- JILP Data Prefetching Championship 2009
- Exhaustive search on a history, looking for regular patterns
  - History stored as bit vector per physical page
  - Shift history to center on current access
  - Check for patterns for all +/- X strides
  - Prefetch matches with smallest prefetch distance
Summary

• Some data accesses are predictable
• Regular vs Irregular data access patterns
• Three must-know prefetchers
  – Next Line, Stream, Stride
• Fourth must-know prefetcher
  – Global History Buffer