Memory Prefetching for the GreenDroid Microprocessor

David Curran
December 10, 2012
Outline

- Memory Prefetchers Overview
- GreenDroid Overview
- Problem Description
- Design
  - Placement
  - Prediction Logic
- Simulation
- Conclusions
Memory Prefetchers Overview
Memory Prefetchers Overview

- Fast Processor
  vs.
- High Latency Large Memory
Memory Prefetchers Overview

- Fast Processor
- Increasing Latency Memory Hierarchy
  - Low Latency Cache Levels (L1, L2, L3, ...)
  - High Latency Large Main Memory
  - Very High Latency Backing Store
Memory Prefetchers Overview

- Fast Processor
- Increasing Latency Memory Hierarchy

<table>
<thead>
<tr>
<th>Prefetcher</th>
<th>Low Latency Cache Levels (L1, L2, L3, …)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Medium Latency Prefetcher Buffer</td>
</tr>
<tr>
<td></td>
<td>• High Latency Large Main Memory</td>
</tr>
<tr>
<td></td>
<td>• Very High Latency Backing Store</td>
</tr>
</tbody>
</table>
GreenDroid Overview
GreenDroid Overview

• Low-Power Tiled Microprocessor
  – Array of tiles connected by a mesh network
  – Devices on sides of mesh network
  – Three networks
    • Static Network
    • General Dynamic Network (GDN)
    • Memory Dynamic Network (MDN)
  – Conservation-Cores (C-Cores) in Tiles
    • Specialized Fast / Power-Efficient Processing Units
    • Dark Silicon Allocations
GreenDroid Overview

- Tiled Microprocessor

GreenDroid Overview

- C-Cores

Problem Description
Problem Description

- Design a memory prefetcher for the GreenDroid microprocessor
- Simulate the design in the GreenDroid cycle-accurate-simulator
- Evaluate the prefetcher based on the simulation results
Design Goals

- Low Complexity / Footprint
- Ease of Integration / Removal
- Capability
Placement

- At the Tile Cache
Placement

- At the Edge of the Mesh Network
Placement

- At the DRAM Controller
Prefetcher Logic
Prediction Logic

- **Stride Prefetchers**
  - Detect memory access patterns of uniform stride length
  - Typically use saturating counters to detect and validate streams
  - Typically associate streams with PC locations
Prediction Logic

- Condensed-Stride Prefetcher
  - Detects memory access patterns of uniform stride length
  - Does not use saturating counters
  - Does not have access to PC location
Condensed-Stride Logic

- Keeps a record of the last 32 processor requested addresses
- Performs two nested traversals of the list in reverse order to find stride patterns
  - Each location in the list may be matched with one other used or unused location in the list before being marked as having been used
  - A location is marked as used if the distance between its address and another location's address is less than the maximum allowed stride length
  - The first pair of addresses to qualify are matched
  - The difference between the original unused address and the matched address is added to the original unused address, and the sum is prefetched if it does not already exist in the prefetch buffer
  - The prefetcher is dormant if all addresses are used
Condensed-Stride Logic
Simulation
Simulation Environment

- Cycle Accurate Simulator
  - .bc Language
- C-Compiler
- Standard Benchmark Code
Simulator Constraints

- DRAM CAS Latency: 27 cycles (2 cache lines)
- DRAM Write Penalty: 70 cycles
- Prefetcher Buffer Find Latency: 4 cycles
- Prefetcher Address Calculation Time: 8 cycles
  + (1 x numFullLowEyeSwings)
  + (bufferFindLatency x numBufferLookups)
Simulation

- Examine the prefetcher's effectiveness for
  - runs of several common computational tasks (PROJECT benchmarks)
  - sections of several standard benchmark runs (SPEC CINT2000 benchmarks)
Selected Benchmarks

- **PROJECT**
  - 1.traversal
  - 2.qsro
  - 3.qsoo
  - 4.bts
- **SPEC CINT2000**
  - 175.vpr
  - 181.mcf
  - 256.bzip2
  - 300.twolf
Cycle Count Since Start vs Hit Ratio (PROJECT 2.qsro)

- **Cumulative (Entire Program Run)**
- **Cumulative (Sorting Procedure Only)**
- **Local**
- **Finished Setting Array Values**
- **Cache Cleared; Began Sorting**

**Axes:**
- **Y-axis:** Hit Ratio
- **X-axis:** Cycle Count Since Start

**Legend:**
- Orange line: Cumulative (entire program run)
- Brown line: Cumulative (sorting procedure only)
- Blue diamond: Local
- Yellow triangle: Finished setting array values
- Green triangle: Cache cleared; began sorting

**Graph Details:**
- The graph shows the relationship between cycle count since start and hit ratio.
- Key events are marked on the graph with different symbols and lines.
Cache Misses Since Start vs Cycle Count Since Start (PROJECT 2.qsro)

- **Red line**: without prefetcher
- **Blue line**: with prefetcher
- **Yellow triangle**: Finished setting array values
- **Green triangle**: Cache cleared; began sorting

The graph shows a comparison between the number of cache misses since the start and cycle counts since the start, with two lines representing different conditions: one with and one without prefetching. The graph illustrates how the cycle count increases with the number of cache misses.
Cycle Count Since Start vs Hit Ratio (CINT2000 256.bzip2)
Cache Misses Since Start vs Cycle Count Since Start (CINT2000 256.bzimpl2)

- **Cycle Count Since Start**
  - Y-axis: 0.0E+0 to 8.0E+8

- **Cache Misses Since Start**
  - X-axis: 0.0E+0 to 1.4E+6

- **Legend**
  - Red line: without prefetcher
  - Blue line: with prefetcher
Cycle Count Since Start vs Hit Ratio (CINT2000 300.twolf)
Cache Misses Since Start vs Cycle Count Since Start (CINT2000 300.twolf)

- **Red Line**: without prefetcher
- **Blue Line**: with prefetcher
Conclusions

● Hit Ratios
  – Wide range over selected benchmarks from nearly 0% to nearly 100%

● Performance Effects
  – Best: Nearly 2:1 Speedup (hit ratio ~100%)
  – Worst: Minimal Speed Deficit (hit ratio ~0%)