Combining Local and Global History for High Performance Data Prefetching

Martin Dimitrov and Huiyang Zhou
Our Contributions

• New localities in the local and global address stream
• A high performance prefetcher design
• Mechanisms for eliminating redundant prefetches
• Advocating for L1-cache data prefetchers
Presentation Outline

• Contributions
• Novel data localities in the address stream
• Proposed data prefetcher
• Filtering of redundant prefetches
• Design Space Exploration
• Experimental Results
• Conclusions
Novel Data Localities: Global Stride

- **Global Stride** exists when there is a constant stride between addresses of two *different* instructions.

  global address stream →

  Load A: \( X \quad Y \quad Z \)
  Load B: \( X+d \quad Y+d \quad Z+d \)

- When does it occur
  - Load/store instructions access adjacent elements of a data structure
  - Address-Value Delta [MICRO-38] is also a form of global stride
Novel Data Localities: Most Common Stride

• **Most Common Stride** exists when a constant pattern is disrupted from time to time.

  local address delta stream →

  Store A: \( D \ X \ D \ Y \ D \ Z \ D \ldots \)

• **When does it occur**

  ```c
  for (j = lll = 0; j < ll; ++j){
    x = psv->value(j);
    if (isNotZero(x, eps)){
      k = psv->index(j);
      kk = u.row.start[k] + (u.row.len[k]++);
      u.col.idx[m++] = k;
      u.row.idx[kk] = i;
      u.row.val[kk] = x;
      ++lll;
    }
    ...
  }
  ```

  Code example from *Soplex*

Local address delta in bytes

University of Central Florida
Novel Data Localities: Scalar Stride

• **Scalar Stride** exists when the address is multiplied or divided by a constant

  local address stream →

  Load A: 32D 16D 8D 4D 2D D ...  

• When does it occur

```
long cmp;
while ( ... ){
  ...
  cmp *= 2;
  if( cmp + 1 <= net->max_residual_new_m )
    if( new[cmp-1].flow < new[cmp].flow )
      cmp++;
}
```

Code example from *mcf*

Local address delta in bytes

University of Central Florida
Global History Buffer (GHB) Prefetcher

- Few static instructions may occupy the whole GHB
- Requires sequential traversal of the linked list
Prefetch Function
Detecting Global Stride

global address stream →

Load A: \( X \quad Y \quad Z \)
Load B: \( X+d \quad Y+d \quad Z+d \)

Match?

Global delta

GHB (N entries)
Prefetch Function
Detecting Delta Correlation

local delta stream →

Load A: a b c d a b c d a b c d . . .
    a b

Match !

a b c d ← generate prefetches
Prefetch Function
Detecting Single Delta Match

local delta stream →

Load A: a x c d a z c d a y c d . . .

Match!

a x c d ← generate prefetches
Prefetch Function

• If no delta correlation is detected, generate 2 prefetches
  – Prefetch last matched stride to approximate most common stride.
  – Next line prefetch

• The output of the prefetch function is a buffer (up to max prefetch degree) filled with potential prefetch addresses.
Filtering of Redundant Prefetches

- Local redundant prefetches

  Load A address stream

  *time 1:* miss: a  
  prefetch: b, c, d, e

  *time 2:* hit (pref bit ON): b  
  prefetch: c, d, e, f

  *time 3:* hit (pref bit ON): c  
  prefetch: d, e, f, g

- Global redundant prefetches

  Load B prefetches: a + 8, x, y, etc.

  Load C prefetches: b + 16, w, z, etc.

  Other loads/stores use data in the same cache line as Load A.
Filtering of Redundant Prefetches

• Filtering local redundant prefetches
  - Add a confidence bit to each LDB to indicate that we have already prefetched the full prefetch degree
  - If conf bit is set, make only 1 prefetch

Load A address stream

*time 1:* miss: a  
prefetch: b, c, d, e  
conf: ON

*time 2:* hit (pref bit ON): b  
prefetch: f  
conf: ON

• Filtering global redundant prefetches
  - Use a MSHR
  - Use a Bloom filter. On a Bloom filter hit, drop the prefetch. Reset the Bloom filter periodically.
Design Space Exploration
Prefetch into the L1 or L2 Cache?

- We advocate for prefetching into the L1 cache
  + L1-cache hits are better than L2-cache hits
  + More accurate address stream
  + Access to the program counter (PC)

  - Latency is more critical
Design Space Exploration
Three Prefetcher Design Points

• GHB-LDB-v1: Highest performance design, using MSHRs to remove redundant prefetches.
• GHB-LDB-v2: Scaled down design, using Bloom filter to remove redundant prefetches.
• LDB-only: Very complexity and latency efficient design.
Design Space Exploration
LDB-only Design

- Each entry in the table is an LDB. (a FIFO of last several deltas, last address and a confidence bit)
- Can detect all the stride patterns, except global stride
- Latency efficient: no linked list traversal, quick Bloom filter access
## Storage Cost

<table>
<thead>
<tr>
<th>Storage Cost</th>
<th>GHB-LDB-1</th>
<th>GHB-LDB-2</th>
<th>LDB-only</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index Table</td>
<td>256-entry 8-way 9728 bits</td>
<td>256-entry 8-way 9728 bits</td>
<td>64-entry 8-way</td>
</tr>
<tr>
<td>GHB</td>
<td>192 entry 192 * (32+8) = 7680 bits</td>
<td>128 entry 128 * (32+7) = 4992 bits</td>
<td>N/A</td>
</tr>
<tr>
<td>Prefetch Func</td>
<td>1120 bits</td>
<td>1120 bits</td>
<td>1120 bits</td>
</tr>
<tr>
<td>Prefetch MSHR</td>
<td>256-entry 8-way 256*(21+3)=6144 bits</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Bloom filter</td>
<td>N/A</td>
<td>2048 + 8-bit reset counter</td>
<td>4096 + 9-bit reset counter</td>
</tr>
<tr>
<td>LDBs</td>
<td>16 LDBs 16*(7*32+32+32+32+5) =5200 bits</td>
<td>16 LDBs 16*(7*32+32+32+32+4+1) =5200 bits</td>
<td>64 LDBs 64*(7*24+32+32+3+1) =15104 bits</td>
</tr>
<tr>
<td>Counters</td>
<td>100 bits</td>
<td>100 bits</td>
<td>N/A</td>
</tr>
<tr>
<td>Total</td>
<td>29972 bits (3.7kB)</td>
<td>23196 bits (2.9kB)</td>
<td>20329 bits (2kB)</td>
</tr>
</tbody>
</table>
Experimental Results

Speedup for best performing design point GHB-LDB-v1

<table>
<thead>
<tr>
<th>Speedup</th>
<th>bzip2</th>
<th>lbm</th>
<th>mcf</th>
<th>mlc</th>
<th>omnetpp</th>
<th>splex</th>
<th>xalan</th>
<th>Gmean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conf1</td>
<td>1.07</td>
<td>2.89</td>
<td>2.65</td>
<td>1.97</td>
<td>1.13</td>
<td>1.54</td>
<td>0.99</td>
<td><strong>1.61</strong></td>
</tr>
<tr>
<td>Conf2</td>
<td>1.08</td>
<td>2.98</td>
<td>1.90</td>
<td>2.83</td>
<td>1.10</td>
<td>1.46</td>
<td>0.97</td>
<td><strong>1.60</strong></td>
</tr>
<tr>
<td>Conf3</td>
<td>1.02</td>
<td>2.98</td>
<td>1.88</td>
<td>2.83</td>
<td>1.11</td>
<td>1.48</td>
<td>1.37</td>
<td><strong>1.67</strong></td>
</tr>
</tbody>
</table>

Avg. speedup for other two designs: 1.60X and 1.56X
Conclusions

- We introduce a high performance prefetcher design for prefetching into the L1 cache.
- Discover and utilize novel localities in the global and local address streams
- Emphasize the importance of filtering redundant prefetches and proposing mechanisms to accomplish the task
Questions?