Two-level Cache Architecture to Reduce Memory Accesses for IP Lookups

Sunil Ravinder, Mario Nascimento, Mike MacGregor

Department of Computing Science
University of Alberta
Outline

- IP Lookups: Longest-prefix matching
- Prefix caching
- Substride caching, 2-level architecture
- Modelling cache hit rates
- Optimal two-level designs
- Experimental results
- Conclusions
IP Lookups

- IP lookups provide forwarding decisions based on the incoming packet's destination address.
- Involves searching the longest prefix that matches the packet's destination address.
- Successful prefix search determines the output port (next hop).
IP Lookups

Incoming IP Destination Address

Routing Table

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.16.28.0/24</td>
<td>P1</td>
</tr>
<tr>
<td>138.196.0.0/16</td>
<td>P2</td>
</tr>
</tbody>
</table>

Next Hop Decision

Next Hop
A classic solution: Prefix caching

- Cache prefixes (and next hops) from previous lookups.
- A hit in the prefix cache gives the next hop.
- Benefits from temporal locality.
- Better than just caching full addresses because it also captures spatial locality.
Assisting prefix cache misses

- Missing the PC usually forces a table traversal.
- Let’s try to decrease the cost of a PC miss!
- Dynamic substride cache: A new cache organization that stores “substrides”.
- Lookups that hit in the DSC proceed directly from an internal node within the routing table - skipping several memory accesses.
Dynamic substride cache (DSC)

- We obtain substrides by shortening a (recently looked up) prefix by $k$ bits.
- e.g. for prefix 0001* and $k = 1$ we obtain the substride 000*.
- DSC stores $<$substride, table node addr$>$ pairs.
Dynamic substride cache (DSC)

- A substride captures a wider IP address space than its parent prefix.
- Larger values of $k$ increase the space spanned (↑DSC hit rate), but also increase the number of memory accesses after a DSC hit (↑hit cost)
Proposed architecture

IP dest. address

Prefix Cache

Dynamic Substridge Cache (DSC)

Next Hop

Low level memory
(contains IP routing table in a trie data structure)

hit

miss → full table traversal

hit

shortcut into routing table

hit

miss
Proposed architecture

- Prefix cache sees destination address stream first. Focused on exploiting temporal and spatial locality.
- DSC assists lookups that fall in “popular” IP address ranges. Assists lookups that miss the prefix cache.
- Lookups that miss the PC, but hit in the DSC will skip many memory references.
## A short example

<table>
<thead>
<tr>
<th>[A]</th>
<th>111100 101000 (PC) 000101</th>
<th>New reference: 000100111</th>
<th>111100 101000 (PC) 000101</th>
<th>Miss in PC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1111</td>
<td></td>
<td>1111</td>
<td>Hit in DSC</td>
</tr>
<tr>
<td></td>
<td>1010 (DSC)</td>
<td></td>
<td>1010 (DSC)</td>
<td>Miss in DSC</td>
</tr>
<tr>
<td></td>
<td>0001</td>
<td></td>
<td>0001</td>
<td>Miss in DSC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[B]</th>
<th>000100 111100 (PC) 101000</th>
<th>New reference: 01111001010</th>
<th>000100 111100 (PC) 101000</th>
<th>Miss in PC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0001</td>
<td></td>
<td>0001</td>
<td>Miss in DSC</td>
</tr>
<tr>
<td></td>
<td>1111 (DSC)</td>
<td></td>
<td>1111 (DSC)</td>
<td>Miss in DSC</td>
</tr>
<tr>
<td></td>
<td>1010</td>
<td></td>
<td>1010</td>
<td>Miss in DSC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[C]</th>
<th>011110 000100 (PC) 111100</th>
<th>New reference: 111100110011</th>
<th>011110 000100 (PC) 111100</th>
<th>Hit in PC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0111</td>
<td></td>
<td>0111</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0001 (DSC)</td>
<td></td>
<td>0001 (DSC)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1111</td>
<td></td>
<td>1111</td>
<td></td>
</tr>
</tbody>
</table>
Optimal design

- Partition a given number of cache lines between the PC and DSC.
- Our objective is to minimize the average number of memory accesses.
- Requires a model for cache hit rates.
- Start from the footprint function:
  \[ u(k) = Wk^a \]
- \( u(k) \) is the number of unique references in a stream at the time of the \( k \)th reference
- \( W \) reflects working set size, \( a \) reflects locality
Hit rate modelling

- take derivative and evaluate at $u(k)=C$
- derivative of $u(k)$ is the instantaneous rate of unique references
- evaluate at point where unique refs. observed have just filled the cache – this gives the miss rate

$$M(C) = aW^{1/a}C^{(1-1/a)}$$

- this does not obey the boundary condition at $C=0$
Hit rate modelling

- We propose:
  \[ H(x) = 1 - \left( \frac{x}{A} + 1 \right)^{-\theta} \]

- \( \theta \) captures the locality in the trace
- \( A \) captures the initial hit rate behavior
- \( H(0) = 0, \ H(\infty) = 1 \)
- use the same general form of model for both the PC and the DSC
Optimization Tableau

\[ \min (F_0 m_0 + F_1 m_1 + F_2 m_2) \]

- \( F_i \) are the number of references resolved by the PC, DSC and routing table, respectively
- \( m_i \) is the number of memory references required when resolving a reference at level \( i \)
- \( m_0 = 0, \ m_1 = 5, \ m_2 = 32 \)

Subject to:

**total capacity** \[ c_0 + c_1 = C \]

**DSC hits** \[ F_1 = \left[ L - L \times H_0(c_0) \right] \times H_1(c_1) \leq L \]

**full table traversals** \[ F_2 = L - F_0 - F_1 \leq L \]
Hit rate models in the tableau

- PC and DSC hit rates are modeled as
  \[ H(x) = 1 - \left( \frac{x}{A} + 1 \right)^{-\theta} \]
- \( A_0(c_0), A_1(c_1), \theta_0(c_0) \) and \( \theta_1(c_1) \) are all nonlinear functions (rather than just constants)
- break them into linear segments and add selector variables to choose the correct segment
- interpolate linearly within a segment
Experiments

- seven traces: four high locality, three low locality
- all from public sources, paired with a routing table from the same router
- 6K to 292K prefixes in the tables
- verified optimization tableau by comparing to exhaustive search
Example optimization surface

ISP3 trace

Global hit rate

Prefix cache size

DSC size
### Average number of memory accesses per lookup

<table>
<thead>
<tr>
<th>Scheme</th>
<th>upcb.2</th>
<th>ISP3</th>
<th>FUNET</th>
<th>ISP2</th>
<th>ISP1</th>
<th>upcb.1</th>
<th>bell</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC+DSC</td>
<td>4.26</td>
<td>3.26</td>
<td>2.64</td>
<td>1.38</td>
<td>0.89</td>
<td>0.44</td>
<td>0.37</td>
</tr>
<tr>
<td>MPC</td>
<td>6.03</td>
<td>4.23</td>
<td>2.78</td>
<td>1.52</td>
<td>1.36</td>
<td>0.61</td>
<td>0.48</td>
</tr>
<tr>
<td>Shyu</td>
<td>6.42</td>
<td>4.85</td>
<td>4.25</td>
<td>1.71</td>
<td>1.49</td>
<td>1.13</td>
<td>0.97</td>
</tr>
<tr>
<td>Akhbari-zadeh</td>
<td>7.68</td>
<td>5.62</td>
<td>4.32</td>
<td>1.79</td>
<td>3.68</td>
<td>1.92</td>
<td>1.56</td>
</tr>
<tr>
<td>Kasnavi</td>
<td>8.34</td>
<td>7.63</td>
<td>5.69</td>
<td>2.84</td>
<td>5.18</td>
<td>4.39</td>
<td>3.67</td>
</tr>
<tr>
<td>Multizone</td>
<td>10.91</td>
<td>8.47</td>
<td>6.04</td>
<td>2.95</td>
<td>5.77</td>
<td>6.13</td>
<td>5.02</td>
</tr>
<tr>
<td>Address cache</td>
<td>16.67</td>
<td>9.73</td>
<td>6.76</td>
<td>3.26</td>
<td>6.65</td>
<td>9.16</td>
<td>8.94</td>
</tr>
</tbody>
</table>
Conclusions

- Adding the DSC following the PC reduces average number of memory accesses per lookup.
- Reductions up to 40% compared to other current proposals.
- Works well even with low-locality traffic.
- Incremental updates possible.
- Cache hit rate model obeys boundary conditions, usable for caches in general.
- Optimization tableau verified as accurate by comparing to exhaustive search.