Prefetching

Advanced Topics in Computer Architecture

Timothy Jones
Caching

• We’re all familiar with caching
• Caches store data close to the core
• Caches take advantage of locality
  • Spatial locality
  • Temporal locality

<table>
<thead>
<tr>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag match and valid?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hit / miss</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Select byte(s)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tag</td>
<td>Valid</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cache performance

• Cache hit and miss rates give an indication of cache performance
  • But they fail to capture the impact of the cache on the overall system
• We therefore prefer to incorporate timing into the cache performance
  • For example, including the time take to access the cache
  • And the time taken to service a miss
• This can give us a value for the average memory access time (AMAT)
Characterising cache performance

• From the CPU’s point of view, we want to reduce the average memory access time (AMAT)
  • This is the average time it takes to load data
  • Including a cache in the system should lead to reducing AMAT, otherwise it is doing more harm than good!

\[
AMAT = \text{Cache hit time} + \text{Cache miss rate} \times \text{Cache miss penalty}
\]
Improving cache performance

AMAT = Cache hit time + Cache miss rate * Cache miss penalty

• Let’s consider the equation further to see how to reduce AMAT
• We can’t improve the cache hit time without reducing cache size
• The cache miss penalty depends on where else the data is
  • I.e. whether it is in other caches or main memory
  • The AMAT of that cache dictates this!
• We have the most control over the cache miss rate
  • We can classify cache misses into four categories
    • Compulsory, capacity, conflict & coherence
Reducing cache misses

• We can reduce the number of misses in some of these classes directly
• For example, conflict misses
  • These can be reduced by increasing the size of each set
• Or capacity misses
  • These could be reduced by increasing the size of the cache
• However, we’re going to focus here on schemes to improve all misses
  • All schemes employ some notion of *prefetching*
Prefetching

• This is a technique to bring data into the cache before it is needed
• The idea is to make a prediction about what data the program will use in the near future
  • Then load that data into the cache so that it arrives before required
• Prefetching can be performed in hardware or software
  • Processors often provide special instructions to do this in software
• We’re going to look at a variety of hardware techniques
A simple prefetcher

- Next-line is a simple prefetcher
  - Does what it says on the tin!
- Stride prefetchers are also relatively simple
- The prefetcher identifies simple patterns in the accesses made
  - E.g. 0x1000, 0x1100, 0x1200
- It learns this stride and prefetches based on it
Stride prefetcher

- Stride prefetchers are great for accessing dense matrices / arrays
  - They usually provide little help for pointers
- Although simple, there are complexities
  - For example, how to distinguish between multiple interleaved streams
  - What prefetch degree or depth to use (# of blocks to prefetch)
  - Whether there should be new stream buffers to avoid cache pollution
More complex prefetching

• Stride prefetchers are effective for a lot of workloads
  • Think array traversals
• But they can’t pick up more complex patterns
• In particular, two types of access pattern are problematic
  • Those based on pointer chasing
  • Those that are dependent on the value of the data
• More complex prefetchers are required for these
Correlation prefetching

• Irregular access patterns are not picked up well by stride prefetchers
• Correlation prefetching means correlating pairs / groups of addresses
  • When you see the first address, you can prefetch the next address
Correlation prefetching

- Irregular access patterns are not picked up well by stride prefetchers
- Correlation prefetching means correlating pairs / groups of addresses
  - When you see the first address, you can prefetch the next address
Correlation prefetching

• These work by exploiting temporal correlation
  • Essentially a hardware lookup table mapping one address to another

• They have their downsides too though
  • Must have seen the accesses before and require enormous state

• The Markov prefetcher is the most simple design
Correlation prefetching

• Markov prefetcher prefetches all successor addresses
  • LRU to manage the table entries
  • Limited depth and coverage

• Depth can be addressed by associating a stream with each load PC
  • This has its own downsides too, of course

• Depth can also be addressed by dead-block prediction
  • Based on code correlation or time keeping

• Coverage can be addressed by increasing storage
  • Perhaps by moving the correlation table into main memory
Correlation prefetching

- The Markov predictor structure limits only to fixed-length streams
- Enter the global history buffer!
Correlation prefetching

• The Markov predictor structure limits only to fixed-length streams
• Enter the global history buffer!
Helper-thread-based prefetching

- Instead of dedicated hardware, we could use spare execution resources
  - A different SMT context
  - Continuing execution when the main thread stalls (runahead)
- Using a helper thread we can explore the future control-flow graph
Software prefetching

• Sometimes the programmer is better placed to help
  • Should know what’s accessed next
  • Often this isn’t the case!
• The architecture provides prefetching hint primitives
• When seen in code, the processor decides whether to prefetch or not

```c
for (i=0; i<NUM; i++) {
    A[B[i]]++;
}
```

```c
for (i=0; i<NUM; i++) {
    A[B[i]]++;
    SWPF(B[i + offset*2]);
    SWPF(A[B[i + offset]]);
    A[B[i]]++;
}
```
Software prefetching

for (i=0; i<NUM; i++) {
    A[B[i]]++;
}

for (i=0; i<NUM; i++) {
    SWPF(B[i + offset*2]);
    SWPF(A[B[i + offset]]);
    A[B[i]]++;  
}
Prefetching questions

• Whilst reading the papers for next week, here are some questions you might like to think about to judge each approach

• How do the prefetchers make their predictions?
  • Does this have a bearing on the access patterns that can be prefetched?

• What are the hardware requirements of the schemes?
  • I.e. what structures are needed to implement it and how costly are they?

• Where does the data get prefetched to?
  • Most of the time you’d like it brought into your own L1 cache

• What is the impact on other parts of the system (core, caches, etc)?