Advanced Cache Architectures and Virtual Memory

- Hiding cache miss latencies
  - Non-blocking caches – do not stall, or stop accessing cache, on a miss.
- DM hit time + set-associative hit rate
  - Victim caches
- High instruction fetch bandwidth
  - Trace caches

Victim Cache

Trace Cache

Conventional Cache
Other Cache Accelerators

- Software transformations (e.g., tiling) which change memory access order to increase locality.
- **Software Cache Prefetching** – bring data into the cache before the code executes the load.
  - Prefetch(A)
  - Helper thread prefetching
- **Hardware Cache Prefetching**
  - Next-line prefetcher – on a cache miss, or first access to a prefetched line, prefetch next line.
  - Stream buffers – detects address stride, and keeps a fifo full of next n (e.g., 4) accesses.

Virtual Memory

- It’s just another level in the cache/memory hierarchy
- **Virtual memory** is the name of the technique that allows us to view main memory as a cache of a larger memory space (on disk).

\[
\begin{array}{c}
\text{cpu} \\
\text{cacheing} \\
\text{cache} \\
\text{cacheing} \\
\text{memory} \\
\text{virtual memory} \\
\text{disk}
\end{array}
\]

Virtual Memory

- is just caching, but uses different terminology (and different storage/lookup techniques)

<table>
<thead>
<tr>
<th>cache</th>
<th>VM</th>
</tr>
</thead>
<tbody>
<tr>
<td>block</td>
<td>page</td>
</tr>
<tr>
<td>cache miss</td>
<td>page fault</td>
</tr>
<tr>
<td>address</td>
<td>virtual address</td>
</tr>
<tr>
<td>index</td>
<td>physical address (sort of)</td>
</tr>
</tbody>
</table>
Virtual Memory

- What happens if another program in the processor uses the same addresses that yours does?
- What happens if your program uses addresses that don’t exist in the machine?
- What happens to “holes” in the address space your program uses?

So, virtual memory provides
  - performance (through the caching effect)
  - protection
  - ease of programming/compilation
  - efficient use of memory

What makes VM different than memory caches

- **MUCH** higher miss penalty (millions of cycles)!
- Therefore
  - large pages [equivalent of cache line] (4 KB to MBs)
  - associative mapping of pages (typically fully associative)
  - software handling of misses (but not hits!!)
  - *write-through* not an option, only write-back
Address translation via the page table

- all page mappings are in the page table, so hit/miss is determined solely by the valid bit (i.e., no tag)
- so why is this fully associative???
- Biggest problem – this is slow. Why?

Making Address Translation Fast

- A cache for address translations: translation lookaside buffer (TLB)

TLBs and caches

Virtual Memory & Caches

- Cache lookup is now a serial process
  1. V->P translation through TLB
  2. Get index
  3. Read tag from cache
  4. Compare
- How can we make this faster?
  1.
  2.
**Virtual Caches**

- Which addresses are used to lookup data in cache/store in tag?
  - Virtual Addresses?
  - Physical Addresses?
- Pros/Cons?
  - Virtual
  - Physical

**Fast Index Translation**

- Can do
  1. V→P translation through TLB
  2. Get index in parallel, if the “virtual” index and the “physical” index are the same.

<table>
<thead>
<tr>
<th>virtual page number</th>
<th>page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>index</td>
</tr>
</tbody>
</table>

**Virtual Memory Key Points**

- How does virtual memory provide:
  - protection?
  - sharing?
  - performance?
  - illusion of large main memory?
- Virtual Memory requires twice as many memory accesses, so we cache page table entries in the TLB.
- Three things can go wrong on a memory access: cache miss, TLB miss, page fault.