Appendix B Again

Instructor: Josep Torrellas
CS433
Reducing Prefetching Overhead: Loop Unrolling + Software Pipelining

for(i=0,i<n;i++){
    prefetch(A[i])
    ...=a[i]
}

for(i=0,i<n;i++){
    prefetch(A[i])
    ...=a[i]
    ...=a[i+1]
    ...=a[i+2]
    ...=a[i+3]
}

for(i=0,i<n;i++){
    if(i%4==0)
        prefetch(A[i]);
    ...=a[i]
}

for(i=0,i<n;i+=4){
    prefetch(A[i])
    ...=a[i]
    ...=a[i+1]
    ...=a[i+2]
    ...=a[i+3]
}

for(i=0,i<n;i+=4){
    prefetch(A[i])
    ...=a[i]
    ...=a[i+1]
    ...=a[i+2]
    ...=a[i+3]
}

SW pipe

Unroll

2
Virtual Memory

- At any point: many processes running
  → each has a huge virtual address space
  → they share a single physical memory

Vmem: protection is ensured
  : program does not have to be completely loaded to run, can be relocated dynamically.
Terminology

- Page or Segment: block
- Page fault or address fault: miss in VA → PA translation
- CPU issues Vaddresses; translated to Paddresses
- Process of translation is called memory mapping or address translation

See Figure B.20
Difference between Caches and Vmemory

- Replacement → cache misses: in HW

  → VM: by the OS since long miss penalty
  and important make good choice

- Cache size is independent of address size, not VM
Types of VM systems

- Paged: uses pages; fixed size
-Segmented: use segments, variable sizes
-Most machines use paging or hybrid (paged segments) because of replacement ease

Figure B.21
Page Table

• Where can a block be placed in memory?
  For lower miss rate, fully assoc

• How to find a block in memory?
  Check page table, a data structure that is indexed by the VPage # and contains PPage #

<table>
<thead>
<tr>
<th>PP #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>PP #2</td>
</tr>
<tr>
<td>PP #3</td>
</tr>
<tr>
<td>PP #4</td>
</tr>
</tbody>
</table>
Page Table (II)

• Which block to replace on a miss:
  to minimize misses: use LRU (using a reference bit)
• What happens on a write?
  Obviously not write through.
  Include a dirty bit, when replace page, if dirty, write back to a disk
• fast address translation:
  → naïve approach: each mem access takes two accesses
    1. Access page table
    2. Access data
  → Solution: remember the last few translations on chip (TLB)
Translation Lookaside Buffer (TLB)

- Why does TLB work? Address translations have locality

<table>
<thead>
<tr>
<th>VPN</th>
<th>PPN</th>
</tr>
</thead>
</table>

Protection bits
- valid bit
- use bit
- dirty bit

- TLB smaller /faster than cache
- How to select page size?
  Larger → smaller pg table
    → can use virtual index cache
    → efficient transfer to I/O
    → lower TLB misses (can be variable size page)

Smaller: conserving storage
No translation during indexing of the cache (page B-36)

• Want to reduces hit time
• usually, caches are physically addressed
• could we use virtual caches? (use virtual addresses)
  – need to flush at context switches: bad
  – or add PIDS (recycle problems)
Virtually Indexed, Physically Tagged

• Alternative: virtually indexed, physically tagged cache

Overlap reading tag + data with VA to PA translation
* limitation; direct mapped cache <= 1 page
  or, if associative: each bank <= 1 page
  (other approaches possible)
For Fully VA Caches

→ What if OS and user pgm use 2 diff VA to refer to same PA ? (synonyms or aliases)
  – then : if both in cache , one may be modified and not the other

→ how to solve synonym problem ? **Page coloring**
  aliases need to have identical low X bits of their VA

e.g SUN: 18 bits

![Diagram showing PA, VA, and cache with arrows indicating synonyms or aliases and a note about cache size and time]
Other Issues

• For multiple issue processors: cache must supply lots of bandwidth. IBM RS/6000
  issue 6 inst/cycle
  data cache: 2*16B /cycle

• Pitfall: ignore impact of OS on performance of Mem hierarchy [Figure B.29: very famous researcher]