Memory Systems: Address Translation

CPS 220
Professor Alvin R. Lebeck
Fall 2001

Outline

• Main Memory
• Address Translation
  – basics
• Managing memory

Main Memory Background

• Performance of Main Memory:
  – Latency: Cache Miss Penalty
    - Access Time: time between request and word arrives
    - Cycle Time: time between requests
  – Bandwidth: I/O & Large Block Miss Penalty (L2)
• Main Memory is DRAM: Dynamic Random Access Memory
  – Dynamic since needs to be refreshed periodically (8 ms)
  – Addresses divided into 2 halves (Memory as a 2D matrix):
    - RAS or Row Access Strobe
    - CAS or Column Access Strobe
• Cache uses SRAM: Static Random Access Memory
  – No refresh (6 transistors/bit vs. 1 transistor/bit)
  – Address not divided
• Size: DRAM/SRAM - 4-8
  – Cost/Cycle time: SRAM/DRAM - 8-16

Memory Hierarchy 101

Very fast 1ns clock
Multiple Instructions per cycle

SRAM, Fast, Small
Expensive

DRAM, Slow, Big,Cheap
(called physical or main)

Magnetic, Really Slow,
Really Big, Really Cheap

=> Cost Effective Memory System (Price/Performance)

Main Memory Performance

• Simple:
  – CPU, Cache, Bus, Memory
    same width (1 word)

• Wide:
  – CPU,Mux 1 word:
    Mux,Cache, Bus, Memory N words (Alpha: 64 bits & 256 bits)

• Interleaved:
  – CPU, Cache, Bus 1 word:
    Memory N Modules
    (4 Modules); example is word interleaved

System Organization

© Alvin R. Lebeck 2001
Main Memory Performance

- Timing model
  - 1 to send address,
  - 6 access time, 1 to send data
- Cache Block is 4 words
  - Simple = \((1+6+1) \times 4 = 32\)
  - Wide = \(1 + 6 + 1 = 8\)
  - Interleaved = \(1 + 6 + 4 \times 1 = 11\)

<table>
<thead>
<tr>
<th>Address</th>
<th>Bank 0</th>
<th>Address</th>
<th>Bank 1</th>
<th>Address</th>
<th>Bank 2</th>
<th>Address</th>
<th>Bank 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Avoiding Bank Conflicts

- Lots of banks
  - \(x[256[j \times 512] + j]\)
  - for \(j = 0, 1, \ldots, 511\)
  - \(x[i][j] = x[i][j + 1]\)
- Even with 128 banks, since 512 is multiple of 128, conflict
  - structural hazard
- SW: loop interchange, nonlinear layout, declaring array not power of 2
- HW: Prime number of banks
  - bank number = address mod number of banks
  - address within bank = address / number of banks
  - modulo & divide per memory access?
  - address within bank = address mod number words in bank (3, 7, 31)
  - bank number? easy if 2N words per bank

Chinese Remainder Theorem

As long as two sets of integers \(a_i\) and \(b_i\) follow these rules
- \(b_i \mod a_i = 0\)
- \(b_i \div a_i \mod a_j = 0\)
and that \(a_i\) and \(a_j\) are co-prime if \(i \neq j\), then the integer \(x\) has only one solution (unambiguous mapping):
- bank number = \(b_i\), number of banks = \(a_i\) (\(= 3\) in example)
- address within bank = \(b_i\), number of words in bank = \(a_i\) (\(= 8\) in example)
- N word address 0 to N-1, prime no. banks, # of words is power of 2

Independently Memory Banks

- Memory banks for independent accesses vs. faster sequential accesses
  - Multiprocessor
  - I/O
  - Non-blocking (lockup-free) Cache
- Superbank: all memory active on one block transfer
- Bank: portion within a superbank that is word interleaved

<table>
<thead>
<tr>
<th>Superbank Number</th>
<th>Bank Number</th>
<th>Bank Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fast Bank Number

- Prime Mapping Example

<table>
<thead>
<tr>
<th>Bank Number:</th>
<th>Seq. Interleaved</th>
<th>Modulo Interleaved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>0 1 2</td>
<td>0 1 2</td>
</tr>
<tr>
<td>within Bank:</td>
<td>0 1 2</td>
<td>0 1 2</td>
</tr>
<tr>
<td>0</td>
<td>0 1 2</td>
<td>0 16 8</td>
</tr>
<tr>
<td>1</td>
<td>3 4 5</td>
<td>1 17</td>
</tr>
<tr>
<td>2</td>
<td>6 7 8</td>
<td>18 10 2</td>
</tr>
<tr>
<td>3</td>
<td>9 10 11</td>
<td>3 19 11</td>
</tr>
<tr>
<td>4</td>
<td>12 13 14</td>
<td>12 4 20</td>
</tr>
<tr>
<td>5</td>
<td>15 16 17</td>
<td>21 13 5</td>
</tr>
<tr>
<td>6</td>
<td>18 19 20</td>
<td>6 22 14</td>
</tr>
<tr>
<td>7</td>
<td>21 22 23</td>
<td>15 7 23</td>
</tr>
</tbody>
</table>
Fast Memory Systems: DRAM specific

- Multiple RAS accesses: several names (page mode)
  - 64 Mbit DRAM: cycle time = 100 ns, page mode = 20 ns
- New DRAMs to address gap; what will they cost, will they survive?
  - Synchronous DRAM: Provide a clock signal to DRAM, transfer synchronous to system clock
  - RAMBUS: reinvent DRAM interface (Intel was using it)
  - Each Chip a module vs. slice of memory
  - Short bus between CPU and chips
  - Does own refresh
  - Variable amount of data returned
  - 1 byte / 2 ns (500 MB/s per chip)
- Cached DRAM (CDRAM): Keep entire row in SRAM

Main Memory Summary

- Big DRAM + Small SRAM = Cost Effective
  - Cray C-90 uses all SRAM (how many sold?)
- Wider Memory
- Interleaved Memory: for sequential or independent accesses
- Avoiding bank conflicts: SW & HW
- DRAM specific optimizations: page mode & Specialty DRAM, CDRAM
  - Niche memory or main memory?
  - e.g., Video RAM for frame buffers, DRAM + fast serial output
- Embedded DRAM: Do you know what it is?

Big Memory

- Large amount of DRAM (e.g., 512 MB)
- Addresses are 32-64 bits (4GB for 32-bit)??
- How does this work?

Computer Architecture

- Interface Between Hardware and Software
  - Applications
  - Operating System
  - Compiler
  - Software
  - Hardware
  - This is IT
  - CPU
  - Memory
  - I/O
  - Multiprocessor Networks

Virtual Memory: Motivation

- Process = Address Space + thread(s) of control
- Address space = PA
  - programmer controls movement from disk
  - protection?
  - relocation?
- Linear Address space
  - larger than physical address space
  - 32, 64 bits v.s. 28-bit physical (256MB)
- Automatic management

Virtual Memory: Motivation

- Process = Address Space + thread(s) of control
- Address space = PA
  - programmer controls movement from disk
  - protection?
  - relocation?
- Linear Address space
  - larger than physical address space
  - 32, 64 bits v.s. 28-bit physical (256MB)
- Automatic management
Virtual Memory

- Process = virtual address space + thread(s) of control
- Translation
  - VA → PA
  - What physical address does virtual address A map to
  - Is VA in physical memory?
- Protection (access control)
  - Do you have permission to access it?

Virtual Memory: Questions

- How is data found if it is in physical memory?
- Where can data be placed in physical memory?
  - Fully Associative, Set Associative, Direct Mapped
- What data should be replaced on a miss?
  (Take CPS210 ...)

Segmented Virtual Memory

- Virtual address \(2^{32}, 2^{64}\) to Physical Address mapping \(2^{30}\)
- Variable size, base + offset, contiguous in both VA and PA

Intel Pentium Segmentation

- Logical Address
- Physical Address Space
- Segment Descriptors
- Global Descriptor Table (GDT)
- Segment Base Address

Intel Pentium Segmentation (Continued)

- Segment Descriptors
  - Local and Global
  - base, limit, access rights
  - Can define many
- Segment Registers
  - contain segment descriptors (faster than load from mem)
  - Only 6
- Must load segment register with a valid entry before segment can be accessed
  - generally managed by compiler, linker, not programmer

Paged Virtual Memory

- Virtual address \(2^{32}, 2^{64}\) to Physical Address mapping \(2^{28}\)
- virtual page to physical page frame
- Fixed Size units for access control & translation
Page Table

- Kernel data structure (per process)
- Page Table Entry (PTE)
  - VA -> PA translations (if none page fault)
  - access rights (Read, Write, Execute, User/Kernel, cached/uncached)
  - reference, dirty bits
- Many designs
  - Linear, Forward mapped, Inverted, Hashed, Clustered
- Design Issues
  - support for aliasing (multiple VA to single PA)
  - large virtual address space
  - time to obtain translation

Alpha VM Mapping (Forward Mapped)

- “64-bit” address divided into 3 segments
  - seg0 (bit 63=0) user code/heap
  - seg1 (bit 63 = 1, 62 = 1) user stack
  - kseg (bit 63 = 1, 62 = 0) kernel segment for OS
- Three level page table, each one page
  - Alpha 21064 only 43 unique bits of VA
  - future min page size up to 64KB => 55 bits of VA
- PTE bits: valid, kernel & user read & write enable (No reference, use, or dirty bit)
  - What do you do for replacement?

Intel Pentium Segmentation + Paging

The Memory Management Unit (MMU)

- Input
  - virtual address
- Output
  - physical address
  - access violation (exception, interrupts the processor)
- Access Violations
  - not present
  - user v.s. kernel
  - write
  - read
  - execute

Inverted Page Table (HP, IBM)

- One PTE per page frame
- only one VA per physical frame
- Must search for virtual address
- More difficult to support aliasing
- Force all sharing to use the same VA

Translation Lookaside Buffers (TLB)

- Need to perform address translation on every memory reference
  - 30% of instructions are memory references
  - 4-way superscaler processor
  - at least one memory reference per cycle
- Make Common Case Fast, others correct
- Throw HW at the problem
- Cache PTEs
Fast Translation: Translation Buffer

- Cache of translated addresses
- Alpha 21164 TLB: 48 entry fully associative

TLB Design

- Must be fast, not increase critical path
- Must achieve high hit ratio
- Generally small highly associative
- Mapping change
  - page removed from physical memory
  - processor must invalidate the TLB entry
- PTE is per process entity
  - Multiple processes with same virtual addresses
  - Context Switches?
- Flush TLB
- Add ASID (PID)
  - part of processor state, must be set on context switch