CS6290
Memory
Views of Memory

• Real machines have limited amounts of memory
  – 640KB? A few GB?
  – (This laptop = 2GB)

• Programmer doesn’t want to be bothered
  – Do you think, “oh, this computer only has 128MB so I’ll write my code this way…”
  – What happens if you run on a different machine?
Programmer’s View

• Example 32-bit memory
  – When programming, you don’t care about how much *real* memory there is
  – Even if you use a lot, memory can always be paged to disk

AKA Virtual Addresses
Programmer’s View

- Really “Program’s View”
- Each program/process gets its own 4GB space
CPU’s View

• At some point, the CPU is going to have to load-from/store-to memory... all it knows is the real, A.K.A. **physical** memory

• ... which unfortunately is often < 4GB

• ... and is never **4GB per process**
### Pages

- Memory is divided into pages, which are nothing more than fixed sized and aligned regions of memory.
  - Typical size: 4KB/page (but not always)

<table>
<thead>
<tr>
<th>Address Range</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-4095</td>
<td>Page 0</td>
</tr>
<tr>
<td>4096-8191</td>
<td>Page 1</td>
</tr>
<tr>
<td>8192-12287</td>
<td>Page 2</td>
</tr>
<tr>
<td>12288-16383</td>
<td>Page 3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Page Table

- Map from virtual addresses to physical locations

Virtual Addresses

"Physical Location" may include hard-disk

Physical Addresses
Page Tables

Physical Memory

0K
4K
8K
12K
16K
20K
24K
28K
Need for Translation

Virtual Address: 0xFC51908B

Virtual Page Number: 0xFC519
Page Offset: 0x00152

Page Table: 0x0015208B

Physical Address: 0x0015208B

Main Memory
Simple Page Table

- Flat organization
  - One entry per page
  - Entry contains physical page number (PPN) or indicates page is on disk or invalid
  - Also meta-data (e.g., permissions, dirtiness, etc.)
Multi-Level Page Tables

Virtual Page Number

Level 1  Level 2  Page Offset

Physical Page Number
Choosing a Page Size

• Page size inversely proportional to page table overhead

• Large page size permits more efficient transfer to/from disk
  – vs. many small transfers
  – Like downloading from Internet

• Small page leads to less fragmentation
  – Big page likely to have more bytes unused
CPU Memory Access

- Program deals with virtual addresses
  - “Load R1 = 0[R2]”
- On memory instruction
  1. Compute virtual address (0[R2])
  2. Compute virtual page number
  3. Compute physical address of VPN’s page table entry
  4. Load* mapping
  5. Compute physical address
  6. Do the actual Load* from memory

Could be more depending on page table organization
Impact on Performance?

• Every time you load/store, the CPU must perform two (or more) accesses!

• Even worse, every *fetch* requires translation of the PC!

• Observation:
  – Once a virtual page is mapped into a physical page, it’ll likely stay put for quite some time
Idea: Caching!

- Not caching of data, but caching of translations
Translation Cache: TLB

• TLB = Translation Look-aside Buffer

If TLB hit, no need to do page table lookup from memory

Note: data cache accessed by physical addresses now
PAPT Cache

• Previous slide showed Physically-Addressed Physically-Tagged cache
  – Sometimes called PIPT (I=Indexed)

• Con: TLB lookup and cache access serialized
  – Caches already take > 1 cycle

• Pro: cache contents valid so long as page table not modified
Virtually Addressed Cache

- **Pro**: latency – no need to check TLB
- **Con**: Cache must be flushed on process change

How to enforce permissions?
Virtually Indexed Physically Tagged

- **Pro:** latency – TLB parallelized
- **Pro:** don’t need to flush $ on process swap
- **Con:** Limit on cache indexing (can only use bits *not* from the VPN/PPN)

Big page size can help here
TLB Design

• Often fully-associative
  – For latency, this means few entries
  – However, each entry is for a whole page
  – Ex. 32-entry TLB, 4KB page... how big of working set while avoiding TLB misses?

• If many misses:
  – Increase TLB size (latency problems)
  – Increase page size (fragmentation problems)
Process Changes

- With physically-tagged caches, don’t need to flush cache on context switch
  - But TLB is no longer valid!
  - Add process ID to translation

<table>
<thead>
<tr>
<th>PID</th>
<th>VPN</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>28</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>44</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>52</td>
</tr>
<tr>
<td>0</td>
<td>12</td>
<td>36</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>32</td>
</tr>
<tr>
<td>1</td>
<td>12</td>
<td>36</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>28</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>44</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>52</td>
</tr>
</tbody>
</table>

Only flush TLB when Recycling PIDs
SRAM vs. DRAM

- DRAM = Dynamic RAM

- SRAM: 6T per bit
  - built with normal high-speed CMOS technology

- DRAM: 1T per bit
  - built with special DRAM process optimized for density
Hardware Structures

SRAM

wordline

b

DRAM

wordline

b
Implementing the Capacitor

"Trench Cell"

Cell Plate Si
Cap Insulator
Storage Node Poly
Field Oxide
...Refilling Poly

Si Substrate

DRAM figures from this slide were taken from Prof. Nikolic’s EECS141/2003 Lecture notes from UC-Berkeley
DRAM Chip Organization

Row Decoder

Memory Cell Array

Sense Amps

Row Buffer

Column Decoder

Data Bus

Row Address

Column Address
DRAM Chip Organization (2)

• Differences with SRAM
  • reads are destructive: contents are erased after reading
    – row buffer
  • read lots of bits all at once, and then parcel them out based on different column addresses
    – similar to reading a full cache line, but only accessing one word at a time
  • “Fast-Page Mode” FPM DRAM organizes the DRAM row to contain bits for a complete page
    – row address held constant, and then fast read from different locations from the same page
DRAM Read Operation

Accesses need not be sequential
Destructive Read

After read of 0 or 1, cell contains something close to 1/2
Refresh

• So after a read, the contents of the DRAM cell are gone
• The values are stored in the row buffer
• Write them back into the cells for the next read in the future
Refresh (2)

- Fairly gradually, the DRAM cell will lose its contents even if it’s not accessed
  - This is why it’s called “dynamic”
  - Contrast to SRAM which is “static” in that once written, it maintains its value forever (so long as power remains on)

- All DRAM rows need to be regularly read and re-written

If it keeps its value even if power is removed, then it’s “non-volatile” (e.g., flash, HDD, DVDs)
Accesses are asynchronous: triggered by RAS and CAS signals, which can in theory occur at arbitrary times (subject to DRAM timing constraints).
Double-Data Rate (DDR) DRAM transfers data on both rising and falling edge of the clock.

Command frequency does not change.

Burst Length

Timing figures taken from “A Performance Comparison of Contemporary DRAM Architectures” by Cuppu, Jacob, Davis and Mudge.
Rambus (RDRAM)

- Synchronous interface
- Row buffer cache
  - last 4 rows accessed cached
  - higher probability of low-latency hit
  - DRDRAM increases this to 8 entries
- Uses other tricks since adopted by SDRAM
  - multiple data words per clock, high frequencies
- Chips can self-refresh
- Expensive for PC’s, used by X-Box, PS2
Example Memory Latency Computation

- FSB freq = 200 MHz, SDRAM
- RAS delay = 2, CAS delay = 2

A0, A1, B0, C0, D3, A2, D0, C1, A3, C3, C2, D1, B1, D2

- What’s this in CPU cycles? (assume 2GHz)

- Impact on AMAT?
More Latency

Significant wire delay just getting from the CPU to the memory controller

Width/Speed varies depending on memory type

More wire delay getting to the memory chips

CPU and caches

128-bit 100MHz bus

Memory Controller

x16 DRAM

x16 DRAM

x16 DRAM

x16 DRAM

x16 DRAM

x16 DRAM

(plus the return trip…)
Memory Controller

Like Write-Combining Buffer, Scheduler may coalesce multiple accesses together, or re-order to reduce number of row accesses.

- Commands
- Data
- To/From CPU

Memory Controller

Read Queue

Write Queue

Response Queue

Scheduler

Buffer

Bank 0

Bank 1
Memory Reference Scheduling

• Just like registers, need to enforce RAW, WAW, WAR dependencies

• No “memory renaming” in memory controller, so enforce all three dependencies

• Like everything else, still need to maintain appearance of sequential access
  - Consider multiple read/write requests to the same address
Example Memory Latency Computation (3)

- FSB freq = 200 MHz, SDRAM
- RAS delay = 2, CAS delay = 2
- Scheduling in memory controller

A0, A1, B0, C0, D3, A2, D0, C1, A3, C3, C2, D1, B1, D2

- Think about hardware complexity...
So what do we do about it?

• Caching
  – reduces average memory instruction latency by avoiding DRAM altogether

• Limitations
  – Capacity
    • programs keep increasing in size
  – Compulsory misses
Faster DRAM Speed

• Clock FSB faster
  – DRAM chips may not be able to keep up
    • Latency dominated by wire delay
  – Bandwidth may be improved (DDR vs. regular) but latency doesn’t change much
    • Instead of 2 cycles for row access, may take 3 cycles at a faster bus speed

• Doesn’t address latency of the memory access
On-Chip Memory Controller

Memory controller can run at CPU speed instead of FSB clock speed

Also: more sophisticated memory scheduling algorithms

Disadvantage: memory type is now tied to the CPU implementation