Outline

- 11 Advanced Cache Optimizations
- Memory Technology and DRAM optimizations
- Virtual Machines

Review: 6 Basic Cache Optimizations

- Reducing hit time
  1. Giving Reads Priority over Writes
     - E.g., Read complete before earlier writes in write buffer
  2. Avoiding Address Translation during Cache Indexing

- Reducing Miss Penalty
  3. Multilevel Caches

- Reducing Miss Rate
  4. Larger Block size (Compulsory misses)
  5. Larger Cache size (Capacity misses)
  6. Higher Associativity (Conflict misses)
11 Advanced Cache Optimizations

- **Reducing hit time**
  1. Small and simple caches
  2. Way prediction
  3. Trace caches

- **Increasing cache bandwidth**
  4. Pipelined caches
  5. Multibanked caches
  6. Nonblocking caches

- **Reducing Miss Penalty**
  7. Critical word first
  8. Merging write buffers

- **Reducing Miss Rate**
  9. Compiler optimizations

- **Reducing miss penalty or miss rate via parallelism**
  10. Hardware prefetching
  11. Compiler prefetching

1. Fast Hit times via Small and Simple Caches

- **Indexing tag memory and then comparing takes time**
- **Small cache - faster to index**
  - E.g., L1 caches same size for 3 generations (each about 3 years apart) of AMD microprocessors: K6, Athlon, and Opteron (64 KB)
  - Also L2 cache small enough to fit on chip with the processor avoids time penalty of going off chip
- **Simple - direct mapping**
  - Can overlap tag check with data transmission since no choice
- **Access time estimate for 90 nm using CACTI model 4.0**
  - Median ratios of access time relative to the direct-mapped caches are 1.32, 1.39, and 1.43 for 2-way, 4-way, and 8-way caches

3. Fast Hit times via Trace Cache
(Pentium 4 only; and last time?)

- **Find more instruction level parallelism? How avoid translation from x86 to microops? - $ them**
- **Trace cache in Pentium 4 does two things**
  1. Dynamic traces of the executed instructions
    - Very different from memory layout with static sequences of instructions in cache are determined by layout in memory
    - Built-in branch predictor
  2. Cache the micro-ops vs. x86 instructions
    - Decode/translate from x86 to micro-ops on trace cache miss

2. Fast Hit times via Way Prediction

- **How to combine fast hit time of Direct Mapped and have the lower conflict misses of 2-way SA cache? Best of both worlds!**
- **Way prediction**: keep extra bits in cache to predict the “way,” or block within the set, of next cache access.
  - Multiplexer is set early to select desired block, only 1 tag comparison performed that clock cycle in parallel with reading the cache data
  - Miss $ \rightarrow 1^{st}$ check other blocks for matches in next clock cycle
  - Hit Time
  - Way-Miss Hit Time
  - Miss Penalty

- **Accuracy $\approx$ 85%**
- **Drawback**: CPU pipeline is hard if hit time is variable
  - Used for instruction caches vs. data caches
3. Fast Hit times via Trace Cache
(Pentium 4 only; and last time?)

+ better utilization of large blocks
  - Utilization may be poor in large blocks
    » don’t exit in middle of block, don’t enter at label in middle of block

- complicated address mapping since addresses no longer aligned to power-of-2 multiples of word size

- instructions may appear multiple times in multiple dynamic traces due to different branch outcomes

- Complicated to design

4: Increasing Cache Bandwidth by Pipelining

- Pipeline cache access to maintain bandwidth, but higher latency
- Instruction cache access pipeline stages:
  - Pentium - 1 stage
  - Pentium Pro through Pentium III - 2 stages
  - Pentium 4 - 4 stages

- Disadvantages
  - greater penalty on mispredicted branches because of longer pipeline
  - more clock cycles between the issue of the load and the use of the data

5. Increasing Cache Bandwidth: Non-Blocking Caches

- Non-blocking cache or lockup-free cache - Don’t stall, just keep going
  - reduces the miss penalty continuing to service CPU requests allowing data cache to continue to supply cache hits during a miss
  - requires out-of-order execution
  - Requires multi-bank memories = more bandwidth

- 2 types
  - “hit under miss”
  - “hit under multiple miss” or “miss under miss”
    » further lower the effective miss penalty by overlapping multiple misses

- Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses

- Pentium Pro allows 4 outstanding memory misses

6: Increasing Cache Bandwidth via Multiple Banks

- Rather than treat the cache as a single monolithic block, divide into independent banks that can support simultaneous accesses
  - E.g., T1 (“Niagara”) L2 has 4 banks

- Most effective if accesses are spread across banks

- sequential interleaving
  - Simple mapping that works well.
  - Spread block addresses sequentially across banks
  - E.g., if there are 4 banks, bank 0 has all blocks whose address modulo 4 is 0; bank 1 has all blocks whose address modulo 4 is 1; ...
    » Sort of like how set associativity works except now with banks
  - Good for instructions and arrays

- More complex methods use hash functions
7. Reduce Miss Penalty: 
Early Restart and Critical Word First

- Don't wait for full block before restarting CPU
  - Early restart — As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
  - Critical Word First — Ask for blocks out of order
    » Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block
    » Long blocks more popular today ⇒ Critical Word 1st Widely used

- No clear notion of benefit because of spatial locality
  - May have to wait for next block anyway

8. Merging Write Buffer to 
Reduce Miss Penalty

- Write buffer
  - allows processor to continue while waiting to write to memory

- Merging
  - Check buffer to see if a write can be merged into an existing write
    » I.e. two writes to different words or bytes of the same cache block
  - If so, new data are combined with that entry
  - Eliminates writing the same memory location multiple times

- Widely used
  - The Sun T1 (Niagara) processor, among many others, uses write merging

---

9. Reducing Misses by Compiler 
Optimizations

- Compiler optimizations - hardware designers love it!

- McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache, 4 byte blocks in software

- Instructions
  - Reorder procedures in memory so as to reduce conflict misses
  - Profiling to look at conflicts(using tools they developed)

- Data - 4 standard algorithms
  - Merging Arrays: improve spatial locality by single array of compound elements vs. 2 arrays
  - Loop Interchange: change nesting of loops to access data in order stored in memory
  - Loop Fusion: Combine 2 independent loops that have same looping and some variables overlap
  - Blocking: Improve temporal locality by accessing “blocks” of data repeatedly vs. going down whole columns or rows

---

Merging Arrays Example

```c
/* Before: 2 sequential arrays */
int val[SIZE];
int key[SIZE];
/* After: 1 array of structures */
struct merge {
  int val;
  int key;
};
struct merge merged_array[SIZE];
```

Reducing conflicts between val & key and improve spatial locality
Loop Interchange Example

/* Before */
for (k = 0; k < 100; k = k+1)
  for (j = 0; j < 100; j = j+1)
    for (i = 0; i < 5000; i = i+1)
      \[x[i][j] = 2 \times x[i][j]\];
/* After */
for (k = 0; k < 100; k = k+1)
  for (i = 0; i < 5000; i = i+1)
    for (j = 0; j < 100; j = j+1)
      \[x[i][j] = 2 \times x[i][j]\];

• Sequential accesses instead of striding through memory every 100 words; improved spatial locality
• Depends a lot on how programming language stores things in memory
  – Column major ordering vs. row major ordering

Loop Fusion Example

/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
    \[a[i][j] = 1/b[i][j] \times c[i][j]\];
  for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
      \[d[i][j] = a[i][j] + c[i][j]\];
/* After */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
  {
    \[a[i][j] = 1/b[i][j] \times c[i][j]\];
    \[d[i][j] = a[i][j] + c[i][j]\];
  }

2 misses per access to \(a\) & \(c\) vs. one miss per access; improve spatial locality

Blocking Example

/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
  {r = 0;
    for (k = 0; k < N; k = k+1)
      \[r = r + y[i][k] \times z[k][j]\];
    \[x[i][j] = r\];
  }

• Two Inner Loops:
  – Read all NxN elements of \(z[]\)
  – Read N elements of 1 row of \(y[]\) repeatedly
  – Write N elements of 1 row of \(x[]\)
• Capacity Misses a function of N & Cache Size:
  – \(2N^3 + N^2\) \(\Rightarrow\) (assuming no conflict; otherwise …)
• Idea: compute on BxB submatrix that fits

Loop Fusion Example

/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
    \[a[i][j] = 1/b[i][j] \times c[i][j]\];
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
    \[d[i][j] = a[i][j] + c[i][j]\];
/* After */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
  {
    \[a[i][j] = 1/b[i][j] \times c[i][j]\];
    \[d[i][j] = a[i][j] + c[i][j]\];
  }

2 misses per access to \(a\) & \(c\) vs. one miss per access; improve spatial locality

Blocking Example

/* After */
for (jj = 0; jj < N; jj = jj+B)
  for (kk = 0; kk < N; kk = kk+B)
    for (i = 0; i < N; i = i+B)
      \[r = 0;
        for (kk = 0; kk < min(kk+B-1,N); k = k+1)
          \{\]
          \[r = r + y[i][k] \times z[k][jj];\]
          \[x[i][jj] = x[i][jj] + r;\]
      \};

• B called Blocking Factor - Should be based on cache size for best results
• Capacity Misses from \(2N^3 + N^2\) to \(2N^3/B + N^2\)
• Conflict Misses Too?
Summary of Compiler Optimizations to Reduce Cache Misses (by hand)

<table>
<thead>
<tr>
<th>Compiler Optimization</th>
<th>Performance Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>vpenta (nasa7)</td>
<td>1</td>
</tr>
<tr>
<td>gmy (nasa7)</td>
<td>1.5</td>
</tr>
<tr>
<td>tmc (nasa7)</td>
<td>2</td>
</tr>
<tr>
<td>btrix (nasa7)</td>
<td>2.5</td>
</tr>
<tr>
<td>mxm (nasa7)</td>
<td>3</td>
</tr>
<tr>
<td>spice (nasa7)</td>
<td></td>
</tr>
<tr>
<td>cholesky (nasa7)</td>
<td></td>
</tr>
<tr>
<td>compress</td>
<td></td>
</tr>
</tbody>
</table>

Conclusion - Need to apply all because you don't know which will be the best

10. Reducing Misses by Hardware Prefetching of Instructions & Data

- Prefetching relies on having extra memory bandwidth that can be used without penalty
- Instruction Prefetching
  - Typically, CPU fetches 2 blocks on a miss: the requested block and the next consecutive block.
  - Requested block in instruction cache. Prefetched block in instruction stream buffer

11. Reducing Misses by Software Prefetching Data

- Data Prefetch
  - Pentium 4 can prefetch data into L2 cache from up to 8 streams from 8 different 4 KB pages
    - Can be more aggressive since placing in L2 cache and it is big enough to accommodate
  - Prefetching invoked if 2 successive L2 cache misses to a page, if distance between those cache blocks is < 256 bytes
    - Array striding - Calculate stride and fetch data at next stride distance

Not always good, negative results not shown

- Data Prefetch
  - 2 types
    - Load data into register (HP PA-RISC loads)
    - Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
  - Difference between load and prefetch
    - Special prefetching instructions cannot cause faults; a form of speculative execution

- Tradeoff
  - Issuing Prefetch Instructions takes time
    - Is cost of prefetch issues < savings in reduced misses?
      - Higher superscalar reduces difficulty of issue bandwidth
  - Assumes we have extra memory cycles
Compiler Optimization vs. Memory Hierarchy Search (not in chapter)

- Compiler tries to figure out memory hierarchy optimizations
  - Hard to do, compilers MUST be accurate and thus conservative but potential savings are large

- New approach: “Auto-tuners”
  - First run variations of program on computer to find best combinations of optimizations (blocking, padding, …) and algorithms
  - Then produce C code to be compiled for that computer and execute
  - Gather data and compare to find best configuration
  - Typically targeted for a certain class of computers

- Example
  - “Auto-tuner” targeted to numerical method
    - E.g., PHIPAC (BLAS), Atlas (BLAS), Sparsity (Sparse linear algebra), Spiral (DSP), FFT-W

---

Sparse Matrix – Search for Blocking

for finite element problem [Im, Yelick, Vuduc, 2005]

900 MHz Itanium 2, Intel C v8; ref=275 MFlop/s

Best: 4x2

Reference

<table>
<thead>
<tr>
<th>row block size (r)</th>
<th>column block size (c)</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>8</td>
<td>1.00</td>
<td>1.35</td>
<td>1.12</td>
<td>1.39</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3.34</td>
<td>4.07</td>
<td>2.31</td>
<td>1.16</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1.91</td>
<td>2.52</td>
<td>2.54</td>
<td>2.23</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4.01</td>
<td>2.45</td>
<td>1.20</td>
<td>1.55</td>
</tr>
</tbody>
</table>

- All possible column block sizes selected for 8 computers; How could compiler know which is best?

---

<table>
<thead>
<tr>
<th>Technique</th>
<th>Hit</th>
<th>Band-width</th>
<th>Miss penalty</th>
<th>Miss rate</th>
<th>HW cost/complexity</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small and simple caches</td>
<td>+</td>
<td>-</td>
<td>0</td>
<td>Trivial; widely used</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Way-predicting caches</td>
<td>+</td>
<td>-</td>
<td>1</td>
<td>Used in Pentium 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trace caches</td>
<td>+</td>
<td>-</td>
<td>3</td>
<td>Used in Pentium 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pipelined cache access</td>
<td>+</td>
<td>-</td>
<td>1</td>
<td>Widely used</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nonblocking caches</td>
<td>+</td>
<td>3</td>
<td>1</td>
<td>Widely used</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Banked caches</td>
<td>+</td>
<td>1</td>
<td>Used in L2 of Opteron and Niagara</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Critical word first and early restart</td>
<td>+</td>
<td>2</td>
<td>Widely used</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Merging write buffer</td>
<td>1</td>
<td>1</td>
<td>Widely used with write through</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiler techniques to reduce cache misses</td>
<td>+</td>
<td>0</td>
<td>Software is a challenge; some computers have compiler option</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware prefetching of instructions and data</td>
<td>+</td>
<td>+</td>
<td>2 instr., 3 data</td>
<td>Many prefetch instructions; AMD Opteron prefetches data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiler-controlled prefetching</td>
<td>+</td>
<td>+</td>
<td>3</td>
<td>Needs nonblocking cache; in many CPUs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Outline

• 11 Advanced Cache Optimizations
• Memory Technology and DRAM optimizations
• Virtual Machines

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: is a factor of I/O & Large Block Miss Penalty (L2)
• Main Memory is DRAM: Dynamic Random Access Memory
  – Dynamic since needs to be refreshed periodically (8 ms, 1% time)
  – 1 transistor and 1 capacitor
  – 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)

Main Memory Deep Background

• Used to be called “Core memory”
• “Out-of-Core”, “In-Core,” “Core Dump”
• Non-volatile, magnetic - stored memory via polarity
• Lost to 4 Kbit DRAM (today using 512Mbit DRAM)
• Access time 750 ns, cycle time 1500-3000 ns

Magnetic ceramic ring

DRAM logical organization (4 Mbit)

• Address transferred in 2 pieces
Quest for DRAM Performance

1. Fast Page mode
   - If subsequent access are to the same row, just read from row buffer instead of fetching into row buffer again
   - Buffers are large - 1024 to 2048 bits

2. Synchronous DRAM (SDRAM)
   - DRAM didn’t used to be clocked (for flexibility), hard to synchronize
   - Add a clock signal to DRAM interface which is multiple of CPU clock

3. Double Data Rate (DDR SDRAM)
   - Transfer data on both the rising edge and falling edge of the DRAM clock signal → doubling the peak data rate
   - DDR2 - lower voltage (1.8) and higher clock rate: up to 400 MHz
   - DDR3 - drops to 1.5 volts + higher clock rates: up to 800 MHz
   - All 3 improved Bandwidth, not Latency

DRAM standards

• Commodity market for success
  – If only one manufacturer, companies will be reluctant to use DRAM just in case supply disappears
  – Solution, standardize and allow many companies to make
    » I.e. Intel licensed x86 architecture for same reason

Need for Error Correction!

• Motivation:
  – At first errors were very common
    » Failures in time (FIT) proportional to number of bits!
  – As DRAM cells shrink, more vulnerable
  – Even in 80’s when memory was scarce, extra bits were used to detect and correct errors

• Result - designers worked very hard and for 5-8 years went through period in which failure rate was low enough - dropped EC
  – DRAM banks too large now
  – Servers always corrected memory systems

<table>
<thead>
<tr>
<th>Standard</th>
<th>Clock Rate (MHz)</th>
<th>Mtransfers / second</th>
<th>DRAM Name</th>
<th>Mbytes/s/ DIMM Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDR</td>
<td>133</td>
<td>266</td>
<td>DDR266</td>
<td>2128</td>
</tr>
<tr>
<td>DDR</td>
<td>150</td>
<td>300</td>
<td>DDR300</td>
<td>2400</td>
</tr>
<tr>
<td>DDR</td>
<td>200</td>
<td>400</td>
<td>DDR400</td>
<td>3200</td>
</tr>
<tr>
<td>DDR2</td>
<td>266</td>
<td>533</td>
<td>DDR2-533</td>
<td>4264</td>
</tr>
<tr>
<td>DDR2</td>
<td>333</td>
<td>667</td>
<td>DDR2-667</td>
<td>5336</td>
</tr>
<tr>
<td>DDR2</td>
<td>400</td>
<td>800</td>
<td>DDR2-800</td>
<td>6400</td>
</tr>
<tr>
<td>DDR3</td>
<td>533</td>
<td>1066</td>
<td>DDR3-1066</td>
<td>8528</td>
</tr>
<tr>
<td>DDR3</td>
<td>666</td>
<td>1333</td>
<td>DDR3-1333</td>
<td>10664</td>
</tr>
<tr>
<td>DDR3</td>
<td>800</td>
<td>1600</td>
<td>DDR3-1600</td>
<td>12800</td>
</tr>
</tbody>
</table>

x 2  x 8  (transfer width of 8 bytes)
Error Correction!

- Error correction mechanism: add redundancy through parity bits
  - Common configuration: Random error correction
    » SEC-DED (single error correct, double error detect)
    » One example: 64 data bits + 8 parity bits (11% overhead)
  - Really want to handle failures of physical components as well
    » Organization is multiple DRAMs/DIMM, multiple DIMMs
    » Want to recover from completely failed DRAM and failed DIMM!
    » “Chip kill” handle major failures width of single DRAM chip
      • Need to detect these!

Protection via Virtual Memory

- Virtual Memory
  - Paged-based virtual memory allows translation from a processes address space to main memory
  - Translation look-aside buffer caches these translations
  - Primary mechanism that protects processes from each other
- Process
  - Includes running program and state needed to run
  - Process switch/context switch - switch from one process to the next
  - OS and Architecture work together to allow processes to share hardware and not interfere with other processes
    » Architecture limits what a user process can do yet allow OS to access more

Outline

- 11 Advanced Cache Optimizations
- Memory Technology and DRAM optimizations
- Virtual Machines

Protection via Virtual Memory

- To limit what user processes can access, architecture must do the following:
  - Provide at least two modes - user and OS (kernel/supervisor)
  - Provide a portion of the processor state that the user process can read but not write
  - Mechanism to allow a process to transition between modes
    » I.e. system call
  - Limit memory access
    » Virtual memory
      • Includes protection restrictions - read, write, execute
      • Only OS can update page table
Protection via Virtual Memory

• Paged access takes twice as long - one access to get physical address and one access to get data
  – Takes too long
  – Use TLB to take advantage of locality
• Can’t assume the computer faithfully obeys restrictions
  – Problems arise from inaccuracies in hardware and OS
    » Bugs measured in number per thousand lines
• Ensuring protection is enforced is becoming more important.
• Virtual machines provide a protection model with a much smaller code base

Protection via Virtual Machines

• VMs were developed in the 1960s
  – Large demand for timesharing of computers
  – Users get the illusion of having sole access to the machine
  – Largely ignored in single user computers
• Recently gaining popularity
  – Isolation and security
  – Failures in security and reliability of OS
  – Sharing of single computers among many unrelated users
  – Increase in processor speed to support

What is a Virtual Machine (VM)?

• Broadest definition
  – includes all emulation methods that provide a standard software interface, such as the Java VM
• More narrow view (we will talk about)
  – provide a complete system level environment at binary ISA
  – Same ISA as host machine – System/Operating Virtual Machine
  – Can have different ISA, but we won’t talk about them
• Create the illusion that a user has a machine to themselves including a copy of the OS

What is a Virtual Machine (VM)?

• Single computer runs multiple VMs, and can support a multiple, different OSes
  – VMs share hardware resources
• Underlying HW platform is called the host, and its resources are shared among the guest VMs
Virtual Machine Monitors (VMMs)

- Virtual machine monitor (VMM) or hypervisor is software that supports VMs - workhorse of the system
- VMM determines how to map virtual resources to physical resources
- Physical resource may be:
  - time-shared
  - Partitioned
  - emulated in software
- VMM is much smaller than a traditional OS;
  - isolation portion of a VMM is ~ 10,000 lines of code
  - Why is this good?
    » Smaller so fewer bugs, thus fewer security holes
    » Can even verify formally

Cost of Processor Virtualization

- Depends on the workload - difficult to determine
  - User-level processor bound (doesn't invoke system calls e.g. SPEC)
    » Zero virtualization overhead
    » Runs directly on the architecture
  - I/O intensive workloads
    » High virtualization overhead
    » OS intensive
    » Many system calls and privileged instructions
    » Overhead – native vs. emulated instructions
      • Emulated instructions by the VMM - how many and how long
    » Overhead can be hidden if processor is also I/O bound - waiting on I/O

Requirements of a Virtual Machine Monitor

- Need a VM Monitor (VMM) to …
  - Presents a SW interface to guest software,
  - Isolates state of guests from each other, and
  - Protects itself from guest software (including guest OSes)
- Guest software should behave on a VM exactly as if running on the native HW, don’t want to have to modify OS
- Guest software should not be able to change allocation of real system resources directly
- Control everything
  • Accessed to privileged state
  • Address translation
  • I/O
  • Exceptions and interrupts

Other Uses of VMs

1. Managing Software
   - Backward compatibility for legacy codes - i.e. DOS
2. Managing Hardware
   - Easiest machine to manage only runs 1 application
   - Consolidating servers
   - Migrate running VM to a different computer
Requirements of a Virtual Machine Monitor

- Must run at higher privilege level than guest OS
  - Guest OS runs in user mode
  - Need VMM to run in higher privilege mode to take care of privileged instructions
- Basic requirements (similar to virtual memory)
  - At least 2 processor modes - system and user
  - Privileged set of instructions only available in privileged mode.
    » VMM traps these instructions and takes control
    » All system resources are controlled via these instructions

ISA Support for Virtual Machines

- Virtualizable - able to run VM directly on hardware and only invoke VMM when needed
  - Must consider during ISA design, not hard to do
  - Can reduce number of instructions that must be executed by VMM and how long it takes to emulate them
- Since desktop VMs are fairly new, most ISAs were created without virtualization in mind
- Example:
  - Guest OS can only interact with virtual resources via privileged instructions - can trap these instructions
  - If instructions aren't privileged, then VMM must take special precautions to locate these instructions

Impact of Virtual Machines on Virtual Memory

- Each guest OS manages its own page tables, but it can’t really since privileged
- Added level of memory:
  - Virtual memory, real memory and physical memory
  - Guest OS page tables map virtual memory to real memory via its own page tables
  - VMM page tables map real memory to physical memory
- Extra level of indirection is too much overhead
  - VMM manages Shadow Page Table
    » Maps directly from guest virtual memory to physical memory
    » Must trap any guest OS access to its page tables via write protecting the page tables
      • Natural if access to page table is privileged

Impact of Virtual Machines on I/O

- Most difficult part of virtualization
  - A lot of devices (hard drives, NICs, mouse, keyboards, etc)
  - All of them very different
  - Share among many VMs
  - Device drivers are prevalent and buggy
- Solution: Give each VM generic versions of each type of I/O device driver, and let VMM to handle real I/O
- Mapping hard, depends on device
  - Disks partitioned by VMM to create virtual disks for guest VMs
  - Network interface time shared
Paravirtualization

- Early in VM development, guest OS not modified.
  - Developers recognized inefficiencies
- Paravirtualization -
  - Small modifications to guest OS to make virtualization more efficient.

Protection and Instruction Set Architecture - What are the problems?

- Why is virtualization so hard to fix?
- Example Problem: 80x86 POPF (pop flags) instruction loads flag registers from top of stack in memory
  - One such flag is Interrupt Enable (IE)
  - In system mode, POPF changes IE
  - In user mode, POPF simply changes all flags except IE
  - Problem: guest OS runs in user mode inside a VM, so it expects to see changed a IE, but it won't
    » Guest OS should not be able to change
    » Could cause different results
    » Should trap this instruction instead of allowing to change