Advanced Memory Hierarchy
COSC 5351 Advanced Computer Architecture
Slides modified from Hennessy CS252 course slides

Outline
- 11 Advanced Cache Optimizations
- Memory Technology and DRAM optimizations
- Virtual Machines
- Xen VM: Design and Performance
- AMD Opteron Memory Hierarchy
- Opteron Memory Performance vs. Pentium 4
- Fallacies and Pitfalls
- Conclusion

Review Quiz
- How does a memory hierarchy improve performance?
- What costs are associated with a memory access?

Why More on Memory Hierarchy?

Virtual address to L2 Cache (fig 5.3)

Virtual address to L2 Cache (fig 5.3)

VM is $2^{44}$ or $16\text{Eb}$
Virtual address to L2 Cache (fig 5.3)

Physical Mem is $2^{41}$ or 2Tb

Page size is $2^{13}$ or 8Kb

2\(^{13}\) (8Kb) direct mapped L1 lines with 64b blocks

2\(^{8}\) TLB entries direct mapped in this case (often fully assoc)

If in TLB you check the L1 cache tag in the appropriate line to see if in L1

If not in L1, build PA with 28bit TLB data + page offset. Use this to access L2 cache
10/26/2011

COSC5351 Advanced Computer Architecture

11 Advanced Cache Optimizations

- Reducing hit time
  1. Small and simple caches
  2. Way prediction
  3. Trace caches
- Reducing Miss Penalty
  1. Critical word first
  2. Merging write buffers
- Reducing Miss Rate
  1. Compiler optimizations
  2. Compiler prefetching
- Reducing miss penalty or miss rate via parallelism
  1. Hardware prefetching
  2. Compiler prefetching

Review: 6 Cache Optimizations

- Reducing hit time
  1. Giving Reads Priority over Writes
     - E.g., Read completes before earlier writes in write buffer
  2. Avoiding Address Translation during Cache indexing (use page offset)
- Reducing Miss Penalty
  1. Multilevel Caches (avoid larger vs faster)
- Reducing Miss Rate
  1. Larger Block size (Compulsory misses)
  2. Larger Cache size (Capacity misses)
  3. Higher Associativity (Conflict misses)

Do these always improve performance?

1. Fast Hit times via Small and Simple Caches

- Index tag memory and then compare takes time
  - \( \Rightarrow \) Small cache can help hit time since smaller memory takes less time to index
  - E.g., L1 caches same size for 3 generations of AMD microprocessors: K6, Athlon, and Opteron
  - Also L2 cache small enough to fit on chip with the processor avoids time penalty of going off chip
- Simple \( \Rightarrow \) 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

10/26/2011

COSC5351 Advanced Computer Architecture

Does simpler help?

- Assume 2-way hit time is 1.1x faster than 4-way
- Miss rate will be .049 and .044 (from C.8)
- Hit is 1 clock cycle, miss penalty is 10 clocks (to go to L2 and it hits)
- Avg Mem Acces = Hit time + Miss Rate X Miss pen
  - 2-way
    - Avg Mem Acces = \( 1 + .049 \times 10 = 1.49 \)
  - 4-way
    - Avg Mem Acces = \( 1.1 + .044 \times 9 = 1.50 \)

Elapsed time should be about same, \( \approx 9.9 \pm 10 \)

This means the clock would be slower through to everything runs slower.
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?
- 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 $\Rightarrow$ 1st check other blocks for matches in next clock cycle

Way-Miss Hit Time | Miss Penalty
---|---

- Accuracy = 85% (seen 97.9%)
- Drawback: CPU pipeline is harder if variable hit times
- Used for instruction caches (speculative) vs. data 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?
- Trace cache in Pentium 4
  1. Dynamic traces of the executed instructions vs. static sequences of instructions as determined by layout in memory
  2. Cache the micro-ops vs. x86 instructions
    - Decode/translate from x86 to micro-ops on trace cache miss
    - 1. $\Rightarrow$ better utilize long blocks (don’t exit in middle of block, don’t enter at label in middle of block)
    - 1. $\Rightarrow$ complicated address mapping since addresses no longer aligned to power-of-2 multiples of word size
    - 1. $\Rightarrow$ instructions may appear in multiple dynamic traces due to different branch outcomes decreasing cache space usage efficiency

4: Increasing Cache Bandwidth by Pipelining

- Pipeline cache access
  - Allows higher clock
  - Gives higher bandwidth
  - But multiple clocks for a hit $\Rightarrow$ higher latency
- Cycles to access instruction cache Example:
  1. Pentium
  2. Pentium Pro through Pentium III
  4. Pentium 4
     $\Rightarrow$ greater penalty on mispredicted branches
     $\Rightarrow$ more cycles between load issue & data use
     $\Rightarrow$ easier to have higher associativity

5. Increasing Cache Bandwidth: Non-Blocking Caches

- Non-blocking cache or lockup-free cache allow data cache to continue to supply cache hits during a miss
  - Requires F/E bits on registers or out-of-order execution
  - Requires multi-bank memories
  - "Hit under miss" reduces the effective miss penalty by working during miss vs. ignoring CPU requests
  - "Hit under multiple miss" or "miss under miss" may 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
    - Requires multiple memory banks (otherwise cannot support)
    - Pentium Pro allows 4 outstanding memory misses

Value of Hit Under Miss for SPEC (old data)

- FP programs on average: AMAT = 0.68 $\rightarrow$ 0.52 $\rightarrow$ 0.34 $\rightarrow$ 0.26
- Int programs on average: AMAT = 0.24 $\rightarrow$ 0.20 $\rightarrow$ 0.19 $\rightarrow$ 0.19
- 8 KB Data Cache, Direct Mapped, 32B block, 16 cycle miss, SPEC 92

Value of Hit Under Miss for SPEC (94)
Updated Study (2011)

- Previous study old with smaller cache
- New study: 32KL1 4cal, 256KBL2 10CAL, L3 2M 36CAL

Li, Chen, Brockman, Jouppi (2011)

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 access
  - E.g., T1 ("Niagara") and Arm Cortex-A8 have 4 L2 banks
  - Intel Core i7 has four L1 banks, L2 has 8.
- Banking works best when accesses spread across banks ⇒ mapping of addresses to banks affects behavior of memory system
- Simple mapping that works well is sequential interleaving
  - Spread block addresses sequentially across banks
  - E.g., if there 4 banks, Bank 0 has all blocks whose address modulo 4 is 0; bank 1 has all blocks whose address modulo 4 is 1;...

7. Reduce Miss Penalty: Early Restart & 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
    - Spatial locality ⇒ tend to want next sequential word, so not clear size of benefit of just early restart
  - Critical Word First—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

8. Merging Write Buffer to Reduce Miss Penalty

- Write buffer allows processor to continue while waiting to write to memory
  - If buffer contains modified blocks, the addresses can be checked to see if address of new data matches the address of a valid write buffer entry
  - If so, new data are combined with that entry
  - Increases block size of write for write-through cache of writes to sequential words since multiword writes more efficient to memory
  - Used by many processors including Sun T1 (Niagara) and Intel Core i7

9. Reducing Misses by Compiler Optimizations

- 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
    - 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 Fission: Combine 2 independent loops that have some 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

/* 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; 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 * 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 * x[i][j];

Sequential accesses instead of striding through memory every 100 words; improved spatial locality

Loop Fusion Example

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

Two Inner Loops:
- Read all N x N 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 => (assuming no conflict; otherwise ...)
- Idea: compute on B x B submatrix that fits

Reducing Conflict Misses by Blocking

Conflict misses in caches not FA vs. Blocking size
- Lam et al [1991] a blocking factor of 24 had a fifth the misses vs. 48 despite both fit in cache

Summary of Compiler Optimizations to Reduce Cache Misses (by hand)
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 is placed in instruction cache when it returns, and prefetched block is placed into instruction stream buffer
- Data Prefetching
  - Pentium 4 can prefetch data into L2 cache from up to 8 streams from 8 different 4 KB pages
  - Prefetching invoked if 2 successive L2 cache misses to a page, if distance between those cache blocks is < 256 bytes

11. Reducing Misses by Software Prefetching Data

- Data Prefetch
  - Load data into register (HP PA-RISC loads)
  - Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
  - Special prefetching instructions cannot cause faults; a form of speculative execution
    - If a fault would occur, turn into no-op. Why?
- Issuing Prefetch Instructions takes time
  - Is cost of prefetch issues < savings in reduced misses?

Compiler Optimization vs. Memory Hierarchy Search

- Compiler tries to figure out memory hierarchy optimizations
- New approach: Auto-tuners 1st 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
- “Auto-tuner” targeted to numerical methods
  - E.g., PHiPAC (BLAS), Atlas (BLAS), Sparsity (Sparse linear algebra), Spiral (DSP), FFT

Sparse Matrix – Search for Blocking for finite element problem [Im, Yelick, Vuduc, 2005]

Best Sparse Blocking for 8 Computers

<table>
<thead>
<tr>
<th>row block size (r)</th>
<th>Intel Pentium M</th>
<th>Intel/HP Itanium 2</th>
<th>Sun Ultra 2, Sun Ultra 3, AMD Opteron</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Performance Improvement

<table>
<thead>
<tr>
<th>SPECint2000</th>
<th>SPECfp2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.16</td>
<td>1.45</td>
</tr>
<tr>
<td>1.18</td>
<td>1.20</td>
</tr>
<tr>
<td>1.21</td>
<td>1.26</td>
</tr>
<tr>
<td>1.29</td>
<td>1.32</td>
</tr>
<tr>
<td>1.40</td>
<td>1.49</td>
</tr>
<tr>
<td>1.97</td>
<td>1.00</td>
</tr>
<tr>
<td>1.20</td>
<td>1.40</td>
</tr>
<tr>
<td>1.60</td>
<td>1.80</td>
</tr>
<tr>
<td>2.00</td>
<td>2.20</td>
</tr>
</tbody>
</table>

Gap

mcf  fam3d  wupwise  galgel  facerec  swim  applu  lucas  mgrid  equake
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 (should be <5% time)
  - 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

- "Out–of–Core", "In–Core", "Core Dump"?
- "Core memory"?
- Non–volatile, magnetic
- Lost to 4 Kbit DRAM (today using 512Mbit DRAM)
  - Access time 750 ns, cycle time 1500–3000 ns

Quest for DRAM Performance

1. **Fast Page mode**
   - Add timing signals that allow repeated accesses to row buffer without another row access time
   - Such a buffer comes naturally, as each array will buffer 1024 to 4096 bits for each access
2. **Synchronous DRAM (SDRAM)**
   - Add a clock signal to DRAM interface, so that the repeated transfers would not bear overhead to synchronize with DRAM controller
   - Burst mode allows 8 or more xfers without new address
3. **Wider DRAMS (4-bit to 16-bit)**
4. **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 lowers power by dropping the voltage from 2.5 to 1.8 volts + offers higher clock rates: up to 400 MHz
   - DDR3 drops to 1.5 volts + higher clock rates: up to 800 MHz
   - DDR4 (2014) drops to 1.1-1.2volts + 1600MHz
5. **Banks** allow interleaving
   - Improved Bandwidth, not Latency

What About Graphics?

- **Graphics Data RAMs (GDRAMs)** or **Graphics Synchronous Data RAMs (GSDRAMs)**
- **GDDR5** (based on DDR3), but deals with higher bandwidth demand of GPUs
  - Wider interface: 32-bits instead of 4,8 or 16
  - Higher max clock rate
    - Achieved by attaching GDRAMs directly to GPU to remove signaling penalty
  - Get about 2x to 5x the bandwidth of DDR3
Flash Memory

- Electronically Erasable Programmable Read–Only Memory (EEPROM) – NAND most common
  - Read–only but erasable!
  - Erase first (in blocks), then can write to it
- Static, No power need to hold contents
  - Also needs less power in standby; none if inactive
- Used for backup and part of memory hierarchy in mobile devices
- Limited number of writes; data moves
- Cheaper than SDRAM (10x) but not disks (20x)
- Slower than SDRAM (4x reads 10–100x writes)

Dealing With Memory Errors

- Soft Errors – changes to a cells contents (not a change in circuitry) primarily due to cosmic rays striking cell. These are dynamic errors
- Hard Errors – can happen during fabrication or during operation
  - Make extra rows and use one if a main row is bad

Need for Error Correction!

- Motivation:
  - Failures/time proportional to number of bits!
  - As DRAM cells shrink, more vulnerable (why?)
  - DRAM banks too large now
  - Servers always corrected memory systems
- Basic idea: 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 failed DRAM and failed DIMM!
  - “Chip kill” can handle failures of a single DRAM chip

How Often do Errors Occur

- IBM analyzed error rates for a 10,000 processor server with 4GB per processor over 3 years
  - Parity only errors
    - 90,000 or one every 17 minutes
  - ECC only
    - 3500 or one every 7.5 hours
  - Chipkill
    - 6 or one every 2 months

Introduction to Virtual Machines

- VMs developed in late 1960s
  - Remained important in mainframe computing over the years
  - Largely ignored in single user computers of 1980s and 1990s
  - Recently regained popularity due to
    - increasing importance of isolation and security in modern systems,
    - failures in security and reliability of standard operating systems,
    - sharing of a single computer among many unrelated users,
    - and the dramatic increases in raw speed of processors, which makes the overhead of VMs more acceptable

What is a Virtual Machine (VM)?

- Broadest definition includes all emulation methods that provide a standard software interface, such as the Java VM
  - “(Operating) System Virtual Machines” provide a complete system level environment at binary ISA
    - Here assume ISAs always match the native hardware ISA
    - E.g., IBM VM/370, VMware ESX Server, and Xen
    - Present illusion that VM users have entire computer to themselves, including a copy of OS
    - Single computer runs multiple VMs, and can support multiple, different OSes
    - On conventional platform, single OS “owns” all HW resources
    - With a VM, multiple OSes all share HW 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
- VMM determines how to map virtual resources to physical resources
- Physical resource may be time-shared, partitioned, or emulated in software
- VMM is much smaller than a traditional OS;
  - isolation portion of a VMM is ~10,000 lines of code

Other Uses of VMs

- Focus here on protection
- 2 Other commercially important uses of VMs
  1. **Managing Software**
     - VMs provide an abstraction that can run the complete SW stack, even including old OSes like DOS
     - Typical deployment: some VMs running legacy OSes, many running current stable OS release, few testing next OS release
  2. **Managing Hardware**
     - VMs allow separate SW stacks to run independently yet share HW, thereby consolidating number of servers
     - Some run each application with compatible version of OS on separate computers, as separation helps dependability
     - Migrate running VM to a different computer
     - Either to balance load or to evacuate from failing HW

Requirements of a Virtual Machine Monitor

- VMM must be at higher privilege level than guest VM, which generally run in user mode
  - Execution of privileged instructions handled by VMM
- E.g., Timer interrupt: VMM suspends currently running guest VM, saves its state, handles interrupt, determine which guest VM to run next, and then load its state
- Guest VMs that rely on timer interrupt provided with virtual timer and an emulated timer interrupt by VMM
- Requirements of system virtual machines are same as paged–virtual memory:
  1. At least 2 processor modes, system and user
  2. Privileged subset of instructions available only in system mode, trap if executed in user mode
     - All system resources controllable only via these instructions

VMM Overhead?

- Depends on the workload
  - **User-level processor-bound** programs (e.g., SPEC) have zero virtualization overhead
  - Runs at native speeds since OS rarely invoked
  - **I/O-intensive workloads** ⇒ **OS-intensive**
  - execute many system calls and privileged instructions
  - can result in high virtualization overhead
  - For System VMs, goal of architecture and VMM is to run almost all instructions directly on native hardware
- If I/O-intensive workload is also I/O-bound
  - low processor utilization since waiting for I/O
  - processor virtualization can be hidden
  - low virtualization overhead

Requirements of a Virtual Machine Monitor

- A VM Monitor
  - 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
  - Except for performance-related behavior or limitations of fixed resources shared by multiple VMs
- Guest software should not be able to change allocation of real system resources directly
  - Hence, VMM must control everything even though guest VM and OS currently running is temporarily using them
  - Access to privileged state, Address translation, I/O, Exceptions and Interrupts, ...

ISA Support for Virtual Machines

- If plan for VM during design of ISA, easy to reduce instructions executed by VMM, speed to emulate
  - ISA is virtualizable if can execute VM directly on real machine while letting VMM retain control of CPU: "direct execution"
  - Since VMs have been considered for 40 years, PC server apps only recently, many ISAs were created ignoring virtualization, including 80x86 and most RISC architectures
- VMM must ensure that guest system only interacts with virtual resources ⇒ conventional guest OS runs as user mode program on top of VMM
  - If guest OS accesses or modifies information related to HW resources via a privileged instruction—e.g., reading or writing the page table pointer—it will trap to VMM
  - If not, VMM must intercept instruction and support a virtual version of sensitive information as guest OS expects
Impact of VMs on Virtual Memory
- Virtualization of virtual memory if each guest OS in every VM manages its own set of page tables?
- VMM separates real and physical memory
  - Makes real memory a separate, intermediate level between virtual memory and physical memory
  - Some use the terms virtual memory, physical memory, and machine memory to name the 3 levels
  - Guest OS maps virtual memory to real memory via its page tables, and VMM page tables map real memory to physical memory
- VMM maintains a shadow page table that maps directly from the guest virtual address space to the physical address space of HW
  - Rather than pay extra level of indirection on every memory access
  - VMM must trap any attempt by guest OS to change its page table or to access the page table pointer

Impact of I/O on Virtual Memory
- I/O most difficult part of virtualization
  - Increasing number of I/O devices attached to the computer
  - Increasing diversity of I/O device types
  - Sharing of a real device among multiple VMs
  - Supporting many device drivers that are required, especially if different guest OSes are supported on same VM system
- Give each VM generic versions of each type of I/O device driver, and let VMM to handle real I/O
- Method for mapping virtual to physical I/O device depends on the type of device:
  - Disks partitioned by VMM to create virtual disks for guest VMs
  - Network interfaces shared between VMs in short time slices, and VMM tracks messages for virtual network addresses to ensure that guest VMs only receive their messages

ISA Support for VMs & Virtual Memory
- IBM 370 architecture added additional level of indirection that is managed by the VMM
  - Guest OS keeps its page tables as before, so the shadow pages are unnecessary
  - (AMD Pacifica proposes same improvement for 80x86)
- To virtualize software TLB, VMM manages the real TLB and has a copy of the contents of the TLB of each guest VM
  - Any instruction that accesses the TLB must trap
  - TLBs with Process ID tags support a mix of entries from different VMs and the VMM, thereby avoiding flushing of the TLB on a VM switch

Example: Xen VM
- Xen: Open-source System VMM for 80x86 ISA
  - Original vision of VM is running unmodified OS
    - “paravirtualization” – small modifications to guest OS to simplify virtualization
  - 3 Examples of paravirtualization in Xen:
    1. To avoid flushing TLB when invoke VMM, Xen mapped into upper 64 MB of address space of each VM
    2. Guest OS allowed to allocate pages, just check that didn’t violate protection restrictions
    3. To protect the guest OS from user programs in VM, Xen takes advantage of 4 protection levels available in 80x86
      - Most OSes for 80x86 keep everything at privilege levels 0 or at 3.
      - Xen VM runs at the highest privilege level (0)
      - Guest OS runs at the next level (1)
      - Applications run at the lowest privilege level (3)

Xen changes for paravirtualization
- Port of Linux to Xen changed ≈ 3000 lines, or ≈ 1% of 80x86–specific code
  - Does not affect application–binary interfaces of guest OS
- OSes supported in Xen 2.0

<table>
<thead>
<tr>
<th>OS</th>
<th>Runs as host OS</th>
<th>Runs as guest OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux 2.4</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Linux 2.6</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>NetBSD 2.0</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>NetBSD 3.0</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Plan 9</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>FreeBSD 5</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Xen and I/O
- To simplify I/O, privileged VMs assigned to each hardware I/O device: “driver domains”
- Driver domains run physical device drivers, although interrupts still handled by VMM before being sent to appropriate driver domain
- Regular VMs (“guest domains”) run simple virtual device drivers that communicate with physical devices drivers in driver domains over a channel to access physical I/O hardware
- Data sent between guest and driver domains by page remapping
Xen Performance

- Performance relative to native Linux for Xen for 6 benchmarks from Xen developers


Xen Performance, Part II

- Subsequent study noticed Xen experiments based on 1 Ethernet network interfaces card (NIC), and single NIC was a performance bottleneck

Xen Performance, Part III

- > 2X instructions: page remapping and page transfer between driver and guest VMs and due to communication between the 2 VMs over a channel

- 4X L2 cache misses: Linux uses zero-copy network interface that depends on ability of NIC to do DMA from different locations in memory

- 12X – 24X Data TLB misses: 2 Linux optimizations

Xen Performance, Part IV

1. > 2X instructions: page remapping and page transfer between driver and guest VMs and due to communication between the 2 VMs over a channel
2. 4X L2 cache misses: Linux uses zero-copy network interface that depends on ability of NIC to do DMA from different locations in memory

- Since Xen does not support “gather DMA” in its virtual network interface, it can’t do true zero-copy in the guest VM
3. 12X – 24X Data TLB misses: 2 Linux optimizations

- Superpages for part of Linux kernel space, and 4MB pages lowers TLB misses versus using 1024 4 KB pages. Not in Xen

- PTEs marked global are not flushed on a context switch, and Linux uses them for its kernel space. Not in Xen

- Future Xen may address 2. and 3., but 1. inherent?

Protection and Instruction Set Architecture

- Example Problem: 80x86 POPF 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

- Historically, IBM mainframe HW and VMM took 3 steps:
  1. Reduce cost of processor virtualization
     - Intel/AMD proposed ISA changes to reduce this cost
  2. Reduce interrupt overhead cost due to virtualization
  3. Reduce interrupt cost by steering interrupts to proper VM directly without invoking VMM

- 2. and 3. not yet addressed by Intel/AMD; in the future?

80x86 VM Challenges

- 18 instructions cause problems for virtualization:
  1. Read control registers in user model that reveal that the guest operating system in running in a virtual machine (such as POPF), and
  2. Check protection as required by the segmented architecture but assume that the operating system is running at the highest privilege level

- Virtual memory: 80x86 TLBs do not support process ID tags ⇒ more expensive for VMM and guest OSes to share the TLB

  - each address space change typically requires a TLB flush
Intel/AMD address 80x86 VM Challenges

- Goal is direct execution of VMs on 80x86
- Intel’s VT-x
  - A new execution mode for running VMs
  - An architected definition of the VM state
  - Instructions to swap VMs rapidly
  - Large set of parameters to select the circumstances
  - where a VMM must be invoked
  - VT-x adds 11 new instructions to 80x86

Xen 3.0 plan proposes to use VT-x to run Windows on Xen

AMD’s Pacifica makes similar proposals

- Plus indirection level in page table like IBM VM 370

Ironic adding a new mode

- If OS start using mode in kernel, new mode would cause performance problems for VMM since ≈ 100 times too slow

Opteron Memory Hierarchy Performance

- For SPEC2000
  - I cache misses per instruction is 0.01% to 0.09%
  - D cache misses per instruction are 1.34% to 1.43%
  - L2 cache misses per instruction are 0.23% to 0.36%

Commercial benchmark (“TPC-C-like”)

- I cache misses per instruction is 1.83% (100X!)
- D cache misses per instruction are 1.39% (= same)
- L2 cache misses per instruction are 0.62% (2X to 3X)

How compare to ideal CPI of 0.33?

Pentium 4 vs. Opteron Memory Hierarchy

<table>
<thead>
<tr>
<th>CPU</th>
<th>Pentium 4 (3.2 GHz*)</th>
<th>Opteron (2.8 GHz*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction Cache</td>
<td>Trace Cache (8K micro-ops)</td>
<td>2-way associative, 64 KB, 64B block</td>
</tr>
<tr>
<td>Data Cache</td>
<td>8-way associative, 16 KB, 64B block, inclusive in L2</td>
<td>2-way associative, 64 KB, 64B block, exclusive to L2</td>
</tr>
<tr>
<td>L2 cache</td>
<td>8-way associative, 2 MB, 128B block</td>
<td>16-way associative, 1 MB, 64B block</td>
</tr>
<tr>
<td>Prefetch</td>
<td>8 streams to L2</td>
<td>1 stream to L2</td>
</tr>
<tr>
<td>Memory</td>
<td>200 MHz x 64 bits</td>
<td>200 MHz x 128 bits</td>
</tr>
</tbody>
</table>

*Clock rate for this comparison in 2005; faster versions existed
• D cache miss: P4 is 2.3X to 3.4X vs. Opteron
• L2 cache miss: P4 is 0.5X to 1.5X vs. Opteron
• Note: Same ISA, but not same instruction count

And in Conclusion [1/3] ...

» Memory wall inspires optimizations since so much performance lost there
  » Reducing hit time: Small and simple caches, Way prediction, Trace caches
  » Increasing cache bandwidth: Pipelined caches, Multibanked caches, Nonblocking caches
  » Reducing Miss Penalty: Critical word first, Merging write buffers
  » Reducing Miss Rate: Compiler optimizations
  » Reducing miss penalty or miss rate via parallelism: Hardware prefetching, Compiler prefetching
  » "Auto-tuners" search replacing static compilation to explore optimization space?
  » DRAM – Continuing Bandwidth innovations: Fast page mode, Synchronous, Double Data Rate

And in Conclusion [2/3] ...

» VM Monitor presents a SW interface to guest software, isolates state of guests, and protects itself from guest software (including guest OSes)
» Virtual Machine Revival
  » Overcome security flaws of large OSes
  » Manage Software, Manage Hardware
  » Processor performance no longer highest priority
» Virtualization challenges for processor, virtual memory, and I/O
  » Paravirtualization to cope with those difficulties
  » Xen as example VMM using paravirtualization
  » 2005 performance on non-I/O bound, I/O intensive apps: 80% of native Linux without driver VM, 34% with driver VM

And in Conclusion [3/3] ...

» “… VMMs give OS developers another opportunity to develop functionality no longer practical in today’s complex and ossified operating systems, where innovation moves at geologic pace”
  [Rosenblum and Garfinkel, 2005]
» Opteron memory hierarchy still critical to performance