EEL 5764 Graduate Computer Architecture

Chapter 5 – Advanced Memory Hierarchy

Ann Gordon-Ross
Electrical and Computer Engineering
University of Florida

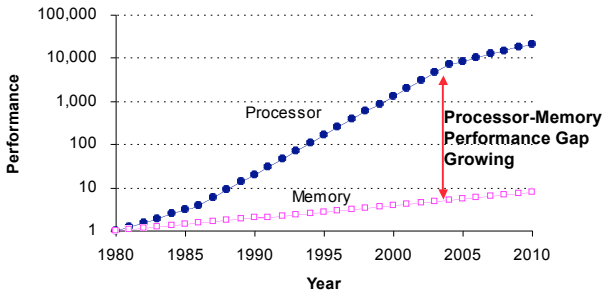
http://www.ann.ece.ufl.edu/

These slides are provided by:
David Patterson
Electrical Engineering and Computer Sciences, University of California, Berkeley
Modifications/additions have been made from the originals

Outline

- 11 Advanced Cache Optimizations
- Memory Technology and DRAM optimizations
- Virtual Machines
- Xen VM: Design and Performance
- AMD Opteron Memory Hierarchy

Why More on Memory Hierarchy?



Review: 6 Basic Cache Optimizations

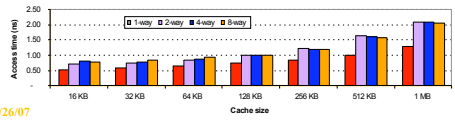
- **Reducing hit time**
  1. Giving Reads Priority over Writes
     - E.g., Read complete before earlier writes in write buffer
  2. Avoiding Address Translation during Cache Indexing
- **Reducing Miss Penalty**
  3. Multilevel Caches
- **Reducing Miss Rate**
  4. Larger Block size (Compulsory misses)
  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**
  1. Pipelined caches
  2. Multibanked caches
  3. Nonblocking caches
- **Reducing Miss Penalty**
  1. Critical word first
  2. Merging write buffers
- **Reducing Miss Rate**
  1. Compiler optimizations
- **Reducing miss penalty or miss rate via parallelism**
  1. Hardware prefetching
  2. Compiler prefetching

1. Fast Hit times via Small and Simple Caches

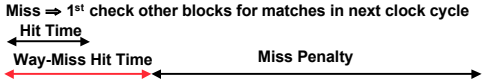
- Indexing tag memory and then comparing takes time
- ⇒ Small cache - faster to index
  - E.g., L1 caches same size for 3 generations of AMD microprocessors: K6, Athlon, and Opteron (64 KB)
  - Also L2 cache small enough to fit on chip with the processor avoids time penalty of going off chip
- Simple ⇒ 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



Biggest change from DM to SA

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.
  - Multiplexor is set early to select desired block, only 1 tag comparison performed that clock cycle in parallel with reading the cache data
  - Miss ⇒ 1st check other blocks for matches in next clock cycle



- Accuracy ≈ 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?)

- 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 - 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

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 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

```
/* Before: 2 sequential arrays */
int val[SIZE];
int key[SIZE];
```

If access both in the same loop, val and key may conflict in the cache

```
/* After: 1 array of stuctures */
struct merge {
  int val;
  int key;
};
struct merge merged_array[SIZE];
```

Will not conflict if grouped in structure together.

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 * 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];
```

For fast changing row and slow changing column

- 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

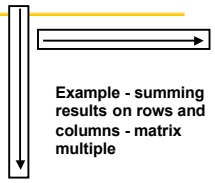
Loop Fusion Example

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

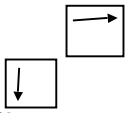
2 misses per access to `a` & `c` vs. one miss per access; improve spatial locality

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;
    };
```



Example - summing results on rows and columns - matrix multiple



- 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:
  - $2N^3 + N^2$ => (assuming no conflict; otherwise …)
- Idea: compute on BxB submatrix that fits

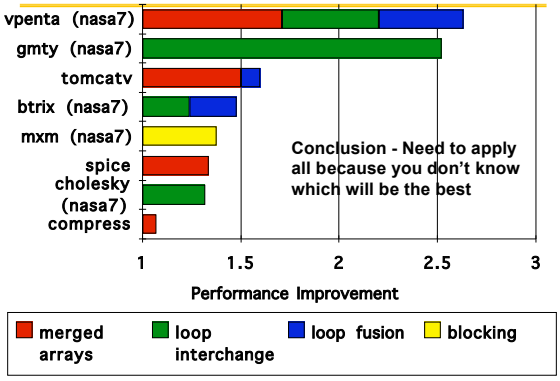
Blocking Example

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

Soln - N/B loops of short size of B

- 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)



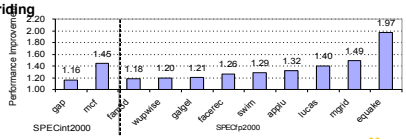
10. Reducing Misses by Hardware Prefetching of Instructions & Data

- 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

10. Reducing Misses by Hardware Prefetching of Instructions & Data

- Data Prefetching
  - 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
    - » Calculate stride and fetch data at next stride distance
      - Array striding

Not always good, negative results not shown



11. Reducing Misses by Software Prefetching Data

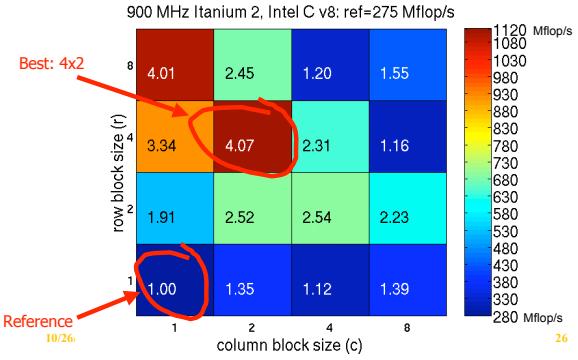
- Data Prefetch
  - 2 types
    - » Load data into register (HP PA-RISC loads)
    - » Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
  - Difference between load and prefetch
    - » Special prefetching instructions cannot cause faults; a form of speculative execution
- Tradeoff
  - Issuing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



Best Sparse Blocking for 8 Computers

row block size (r)				
8		Intel Pentium M		Sun Ultra 2, Sun Ultra 3, AMD Opteron
4	IBM Power 4, Intel/HP Itanium	Intel/HP Itanium 2	IBM Power 3	
2				
1				
	1	2	4	8

column block size (c)

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

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

Outline

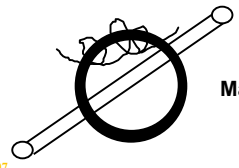
- 11 Advanced Cache Optimizations
- Memory Technology and DRAM optimizations
- Virtual Machines
- Xen VM: Design and Performance
- AMD Opteron Memory Hierarchy

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

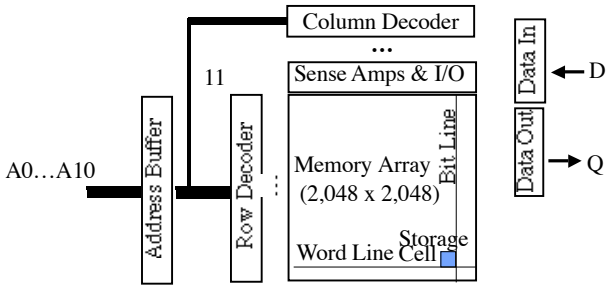
Main Memory Deep Background

- 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, hard to synchronize
   - Add a clock signal to DRAM interface
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

DRAM name based on Peak Chip Transfers / Sec
DIMM name based on Peak DIMM MBytes / Sec

Stan-dard	Clock Rate (MHz)	M transfers / second	DRAM Name	Mbytes/s/ DIMM	DIMM Name
DDR	133	266	DDR266	2128	PC2100
DDR	150	300	DDR300	2400	PC2400
DDR	200	400	DDR400	3200	PC3200
DDR2	266	533	DDR2-533	4264	PC4300
DDR2	333	667	DDR2-667	5336	PC5300
DDR2	400	800	DDR2-800	6400	PC6400
DDR3	533	1066	DDR3-1066	8528	PC8500
DDR3	666	1333	DDR3-1333	10664	PC10700
DDR3	800	1600	DDR3-1600	12800	PC12800

x 2 x 8

Need for Error Correction!

- Motivation:
  - At first errors were very common
    - » Failures/time *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

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

Outline

- 11 Advanced Cache Optimizations
- Memory Technology and DRAM optimizations
- Virtual Machines
- Xen VM: Design and Performance
- AMD Opteron Memory Hierarchy

Introduction to Virtual Machines

- VMs developed in late 1960s
  - Large demand for timesharing of computers
    - » Instead of sharing entire machine, run a virtual machine and each user gets the illusion of having the machine all to themselves
  - 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 (virus and attacks),
  - failures in security and reliability of standard operating systems,
  - sharing of a single computer among many unrelated users,
  - and the dramatic increases in CPU speed has made virtual machines 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
- More narrow view (we will talk about)
  - "(Operating) System Virtual Machines" provide a complete system level environment at binary ISA
  - Create many virtual machines with the same ISA as the machine they will run on - IBM's original interpretation
    - » Can have different ISA but we won't talk about those
  - E.g., IBM VM/370, VMware ESX Server, and Xen

What is a Virtual Machine (VM)?

- Virtual memory creates illusion of private memory and virtual machines create illusion that VM users have entire computer to themselves, including a copy of OS
- Single computer runs multiple VMs, and can support a 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 - 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

VMM Overhead? Is it fast enough?

- Depends on the workload - difficult to determine
- 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
- Can hid overhead 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

Other Uses of VMs

- Focus here on protection
- 2 Other commercially important uses of VMs

1. Managing Software
   - Backward compatibility
   - Can run legacy OSes, current Oses, and beta releases of future Oses all at once with no change of system damage
2. Managing Hardware
   - Easiest machine to manage only runs 1 application
     - » VM allows this model without having to replicate hardware
     - » Thus fewer servers
   - Migrate running VM to a different computer
     - » Can move application without stopping it
       - Checkpoint and restart
     - » Either to balance load or to evacuate from failing HW

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
- 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, …

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

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
  - Since desktop VM is recent, ISAs where not built with support
- VMM must ensure that guest system only interacts with virtual resources
  - If guest OS attempts to access or modify information related to HW resources via a privileged instruction--for example, reading or writing the page table pointer--it will trap to the VMM
- VMM must intercept instruction and support a virtual version of the sensitive information as the guest OS expects (examples soon)

Impact of VMs on Virtual Memory

- Guest needs to manage virtual memory but can't do that
- VMM separates real and physical memory
  - Makes real memory a separate, intermediate level between virtual memory and physical memory - added level of indirection
  - 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

Impact of VMs on Virtual Memory

- Two levels of indirection is too slow so to speed things up
  - 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

ISA Support for VMs & Virtual Memory

- IBM has been working on VMs forever and have been perfecting their system since then
  - Works very well now
- In the beginning, 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
  - 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
  - Process ID tags avoid flushing - lower overhead in context switch

Impact of I/O on Virtual Memory

- Most difficult part of virtualization
  - A lot of devices
  - All of them very different
  - Share among many VMs
  - Device drivers are 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
  - Must deliver network packets to the correct VM
    - » Shared in sort time slices

Outline

- 11 Advanced Cache Optimizations
- Memory Technology and DRAM optimizations
- Virtual Machines
- Xen VM: Design and Performance
- AMD Opteron Memory Hierarchy

Example: Xen VM

- **Xen: Open-source System VMM for 80x86 ISA**
  - Project started at University of Cambridge, GNU license model
  - Growing in popularity
- **One way of running a VM is to run with unmodified version of OS**
  - Significant wasted effort just to keep guest OS happy
- **Xen creators said that wasn't necessary**
  - Why not make changes to OS to make virtualization easier and/or more efficient
  - "paravirtualization" - small modifications to guest OS to simplify virtualization

Example: Xen VM

3 Examples of paravirtualization in Xen:

1. Use existing address space for TLB
   - 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
   - Check to make sure protection restrictions are not violated
3. More protection levels
   - Xen takes advantage of 4 protection levels available in 80x86
     - Most OSes for 80x86 keep everything at privilege levels 0 or at 3.
     - Xen VMM runs at the highest privilege level (0)
     - Guest OS runs at the next level (1)
     - Applications run at the lowest privilege level (3)
   - More protection
     - Guest OS should have more access privileges than application running on guest OS

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 - Windows in future release**

OS	Runs as host OS	Runs as guest OS
Linux 2.4	Yes	Yes
Linux 2.6	Yes	Yes
NetBSD 2.0	No	Yes
NetBSD 3.0	Yes	Yes
Plan 9	No	Yes
FreeBSD 5	No	Yes

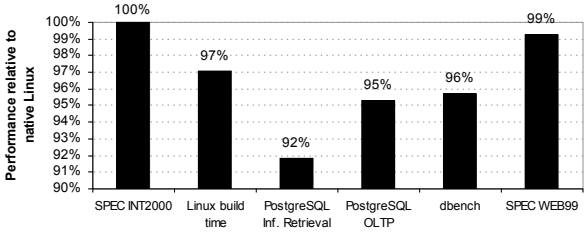
http://wiki.xensource.com/xenwiki/OSCompatibility

Xen and I/O - Biggest Challenge

- **Driver domains**
  - Driver associated with each hardware I/O device
  - Xen Jargon: "domains" = Virtual Machines
- **Driver domains run physical device drivers**
  - Interrupts by VM for device handled by VMM before being sent to the driver domain
- **Split I/O into 2 pieces**
  - Simple virtual device drivers in VM
  - Communicates with driver domain over a channel to access physical I/O hardware
- **Communication over a dedicated "channel"**
  - Data sent between guest and driver domains via memory by page remapping
  - Low cost communication because data isn't copied anywhere, just page remapping transfers control

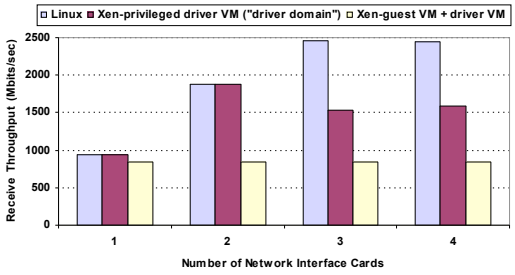
Xen Performance - Overhead of Virtualization

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

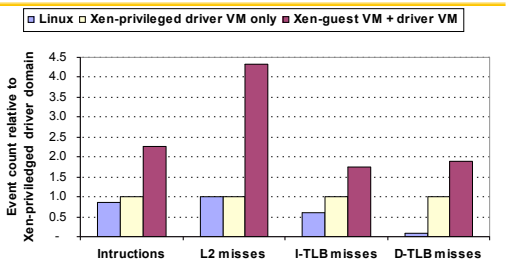


Xen Performance, Part II

- HP recreated results, discovered that apps where I/O bound on NIC and hid overhead of virtualization
  - Why? (next slide)



Xen Performance, Part III



1. > 2X instructions for guest VM + driver VM
2. > 4X L2 cache misses
3. 12X – 24X Data TLB misses

Xen Performance, Why?

1. > 2X instructions: Channel communication
   - 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 lowers TLB misses versus using 1024 4 KB pages. Not in Xen, used smaller pages
   - X86 marks page table entries so they aren't flushed during context swith, Not in Xen

- Basically, Xen did not implement things the same as x86 causing a large overhead
- Future Xen may address 2. and 3., but 1. inherent?

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

Protection and Instruction Set Architecture - What are the problems?

- **Overcome:**
  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 grouped into 2 types cause problems for virtualization:**
  1. Reading control registers in user mode
  2. Checking protection but assuming 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
  - Flushing overhead
    - » each address space change typically requires a TLB flush

Intel/AMD address 80x86 VM Challenges

- Goal is direct execution of VMs on 80x86
  - AMD and Intel are trying to solve the same problem but aren't working together
- 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
  - 11 new instructions
- 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 new mode may be 100 times too slow

Outline

- 11 Advanced Cache Optimizations
- Memory Technology and DRAM optimizations
- Virtual Machines
- Xen VM: Design and Performance
- **AMD Opteron Memory Hierarchy**

AMD Opteron Memory Hierarchy

- 12-stage integer pipeline yields a maximum clock rate of 2.8 GHz and fastest memory PC3200 DDR SDRAM
- 48-bit virtual and 40-bit physical addresses
- I and D cache: 64 KB, 2-way set associative, 64-B block, LRU
- L2 cache: 1 MB, 16-way, 64-B block, pseudo LRU, not inclusive
- Data and L2 caches use write back, write allocate
- L1 caches are virtually indexed and physically tagged
- L1 I TLB and L1 D TLB: fully associative, 40 entries
  - 32 entries for 4 KB pages and 8 for 2 MB or 4 MB pages
  - Separate for bandwidth reasons
- L2 I TLB and L2 D TLB: 4-way, 512 entities of 4 KB pages
- Memory controller allows up to 10 cache misses (hit under multiple misses)
  - 8 from D cache and 2 from I cache

Opteron Memory Hierarchy Performance

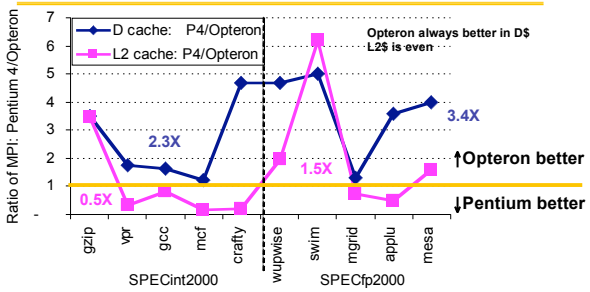
- **For SPEC2000**
  - I cache misses per instruction is 0.01% to 0.09%
  - D cache misses per instruction are 1.34% to 1.43%
  - L2 cache misses per instruction are 0.23% to 0.36%
- **Commercial benchmark ("TPC-C-like")**
  - I cache misses per instruction is 1.83% (100X!)
  - D cache misses per instruction are 1.39% (≈ same)
  - L2 cache misses per instruction are 0.62% (2X to 3X)

Pentium 4 vs. Opteron Memory Hierarchy

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

*Clock rate for this comparison in 2005; faster versions existed

Misses Per Instruction: Pentium 4 vs. Opteron



- **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** 69