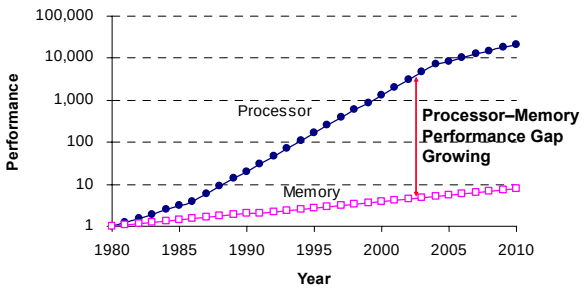
Lecture 10
Advanced Memory Hierarchy

Slides were used during lectures by
David Patterson, Berkeley, spring 2006

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
- Conclusion

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

4. Pipelined caches
5. Multibanked caches
6. Nonblocking caches

Reducing Miss Penalty

7. Critical word first
8. Merging write buffers

Reducing Miss Rate

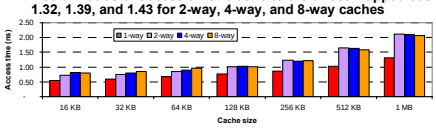
9. Compiler optimizations

Reducing miss penalty or miss rate via parallelism

10. Hardware prefetching
11. Compiler prefetching

1. Fast Hit times via Small and Simple Caches

- Index tag memory and then compare takes time
- ⇒ 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 ⇒ 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



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.
  - 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 takes 1 or 2 cycles
  - 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?
- Trace cache in Pentium 4
  1. Dynamic traces of the executed instructions vs. static sequences of instructions as 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
- \+ 1. ⇒ better utilize long blocks (don't exit in middle of block, don't enter at label in middle of block)
- \- 1. ⇒ complicated address mapping since addresses no longer aligned to power-of-2 multiples of word size
- \- 1. ⇒ instructions may appear multiple times in multiple dynamic traces due to different branch outcomes

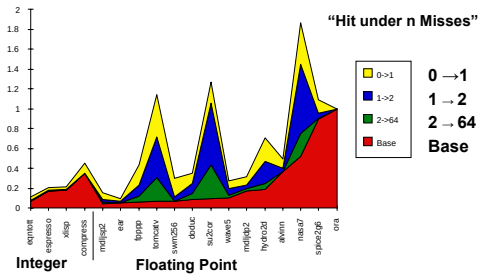
4. Increasing Cache Bandwidth by Pipelining

- Pipeline cache access to maintain bandwidth, but higher latency
- Instruction cache access pipeline stages:
  - 1: Pentium
  - 2: Pentium Pro through Pentium III
  - 4: Pentium 4
- \- ⇒ greater penalty on mispredicted branches
- \- ⇒ 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* 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 → 0.52 → 0.34 → 0.26
- Int programs on average: AMAT= 0.24 → 0.20 → 0.19 → 0.19
- 8 KB Data Cache, Direct Mapped, 32B block, 16 cycle miss, SPEC 92

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
- Banking works best when accesses naturally spread themselves 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 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
  - 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

block

8. Merging Write Buffer to Reduce Miss Penalty

- Write buffer to allow 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, bytes since multiword writes more efficient to memory
- The Sun T1 (Niagara) processor, among many others, uses write merging

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 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];

/* After: 1 array of stuctures */
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

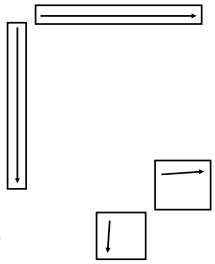
Loop Fusion Example

```
/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
      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;
     };
```



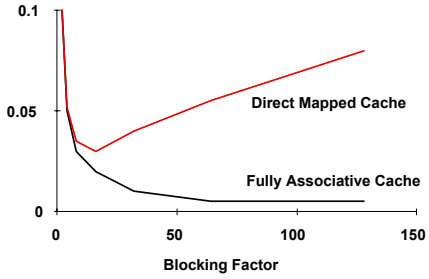
- Two Inner Loops:
  - Read all NxN elements of z[]
  - Read N elements of 1 row of y[] repeatedly
  - Write N elements of 1 row of x[]
- Capacity Misses a function of N & Cache Size:
  - $2N^3 + N^2$ => (assuming no conflict; otherwise …)
- Idea: compute on BxB submatrix that fits

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;
     };
```

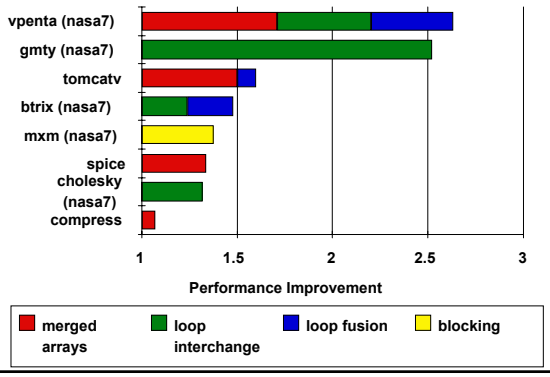
- B called *Blocking Factor*
- Capacity Misses from $2N^3 + N^2$ to $2N^3/B + N^2$
- Conflict Misses Too?

Reducing Conflict Misses by Blocking



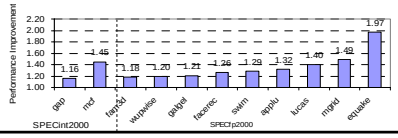
- Conflict misses in caches not FA vs. Blocking size
  - Lam et al [1991] a blocking factor of 24 had a fifth the misses vs. 48 despite both fit in cache

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

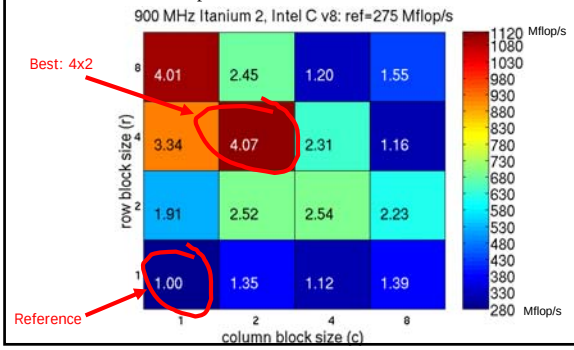
- Data Prefetch
  - Load data into register (HP PA-RISC loads)
  - Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
  - Special prefetching instructions cannot cause faults; a form of speculative execution
- Issuing Prefetch Instructions takes time
  - Is cost of prefetch issues < savings in reduced misses?
  - Higher superscalar reduces difficulty of issue bandwidth

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 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)	1	2	4	8
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				

column block size (c)

All possible column block sizes selected for 8 computers; How could compiler know?

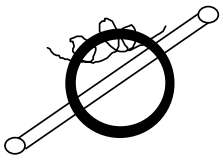
Technique	Hit Time	Band-width	Miss penalty	Miss rate	HW cost/ complexity	Comment
Small and simple caches	+			–	0	Trivial; widely used
Way-predicting caches	+				1	Used in Pentium 4
Trace caches	+				3	Used in Pentium 4
Pipelined cache access	–	+			1	Widely used
Nonblocking caches		+	+		3	Widely used
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

Main Memory Background

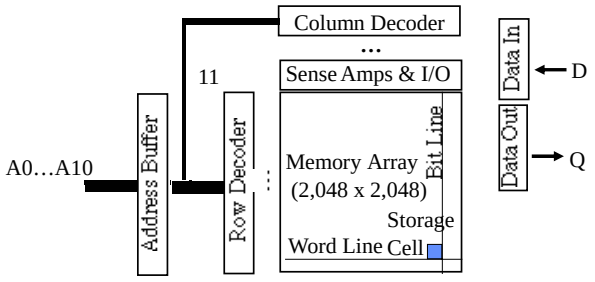
- Performance of Main Memory:
  - Latency: Cache Miss Penalty
    - » *Access Time*: time between request and word arrives
    - » *Cycle Time*: time between requests
  - Bandwidth: I/O & Large Block Miss Penalty (L2)
- Main Memory is *DRAM*: Dynamic Random Access Memory
  - Dynamic since needs to be refreshed periodically (8 ms, 1% 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
    Size: DRAM/SRAM - *4-8*,
    Cost/Cycle time: SRAM/DRAM - *8-16*

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



DRAM logical organization (4 Mbit)



Square root of bits per RAS/CAS

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 2048 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
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 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

Improved Bandwidth, not Latency

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

Fastest for sale 4/06 ($125/GB)

Fastest for sale 1/07 ($200/GB)

Need for Error Correction!

- **Motivation:**
  - Failures/time *proportional* to number of bits!
  - As DRAM cells shrink, more vulnerable
- **Went through period in which failure rate was low enough without error correction that people didn't do correction**
  - 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" handle 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**
- **Opteron Memory Performance vs. Pentium 4**
- **Conclusion**

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 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
- 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

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, …

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

- 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 ultimate control of CPU: "direct execution"
  - Since VMs have been considered for desktop/PC server apps only recently, most 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

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

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

Example: Xen VM

- Xen: Open-source System VMM for 80x86 ISA
  - Project started at University of Cambridge, GNU license model
- Original vision of VM is running unmodified OS
  - Significant wasted effort just to keep guest OS happy
- "paravirtualization" – small modifications to guest OS to simplify virtualization

Three 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 VMM 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

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

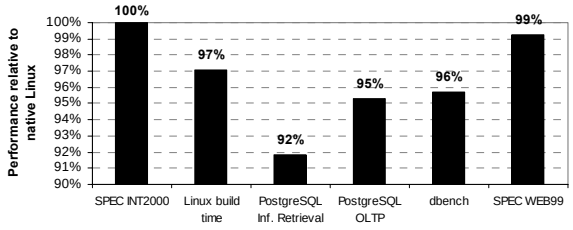
- More OSes in Xen 3.0

Xen and I/O

- To simplify I/O, privileged VMs assigned to each hardware I/O device: "driver domains"
  - Xen Jargon: "domains" = Virtual Machines
- 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

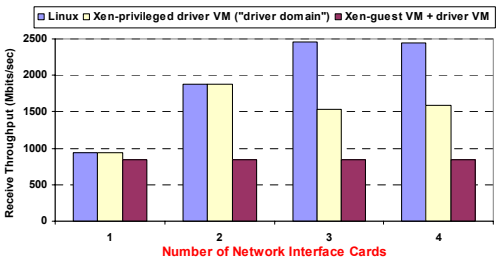
- Performance relative to native Linux for Xen for 6 benchmarks from Xen developers



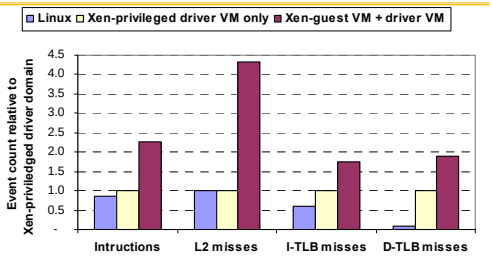
- User-level processor-bound programs? I/O-intensive workloads? I/O-Bound I/O-Intensive?
- Detailed performance analysis: see book

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



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

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

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
- Conclusion

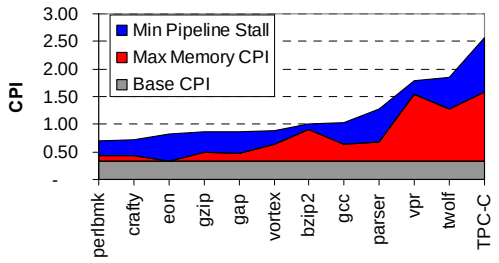
AMD Opteron Memory Hierarchy

- 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
- 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
- L2 I TLB and L1 D TLB: 4-way, 512 entities of 4 KB pages
- Memory controller allows up to 10 cache misses
  - 8 from D cache and 2 from I cache

Opteron Memory Hierarchy Performance

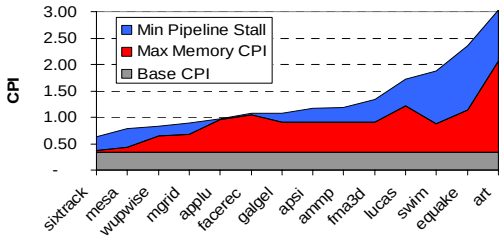
- For SPEC2000
  - I cache misses per instruction is 0.01% to 0.09%
  - D cache misses per instruction are 1.34% to 1.43%
  - L2 cache misses per instruction are 0.23% to 0.36%
- Commercial benchmark ("TPC-C-like")
  - I cache misses per instruction is 1.83% (100X!)
  - D cache misses per instruction are 1.39% (≈ same)
  - L2 cache misses per instruction are 0.62% (2X to 3X)
- How compare to ideal CPI of 0.33?

CPI breakdown for Integer Programs



- CPI above base attributable to memory ≈ 50%
- L2 cache misses ≈ 25% overall (50% memory CPI)
  - Assumes misses are *not* overlapped with the execution pipeline or with each other, so the pipeline stall portion is a lower bound

CPI breakdown for FP Programs



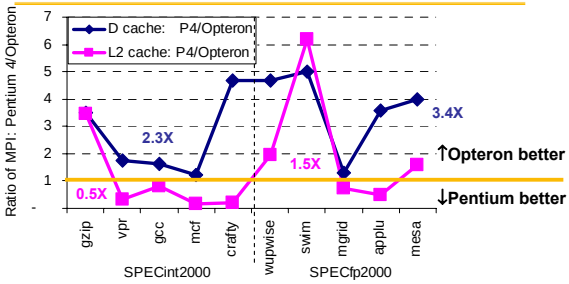
- CPI above base attributable to memory ≈ 60%
- L2 cache misses ≈ 40% overall (70% memory CPI)
  - Assumes misses are *not* overlapped with the execution pipeline or with each other, so the pipeline stall portion is a lower bound

Pentium 4 vs. Opteron Memory Hierarchy

CPU	Pentium 4 (3.2 GHz*)	Opteron (2.8 GHz*)
Instruction Cache	Trace Cache (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

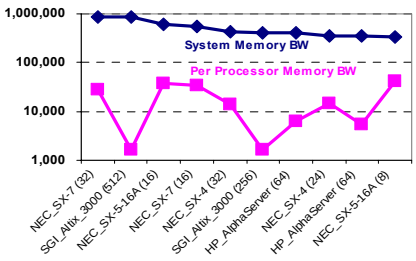
Misses Per Instruction: Pentium 4 vs. Opteron



- 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

Fallacies and Pitfalls

- Not delivering high memory bandwidth in a cache-based system
  - 10 Fastest computers at Stream benchmark [McCalpin 2005]
  - Only 4/10 computers rely on data caches, and their memory BW per processor is 7X to 25X slower than NEC SX7



And in Conclusion [1/2] …

- 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 [2/2] …

- 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
- Opteron memory hierarchy still critical to performance

Reading

- This lecture:
  - chapter 5: *Memory Hierarchy Design*
- Next lecture:
  - chapter 6: *Storage Systems*