Advanced Memory Hierarchy

Prof. Mikko H. Lipasti
University of Wisconsin-Madison

Lecture notes based on notes by John P. Shen and Mark Hill
Updated by Mikko Lipasti

Readings

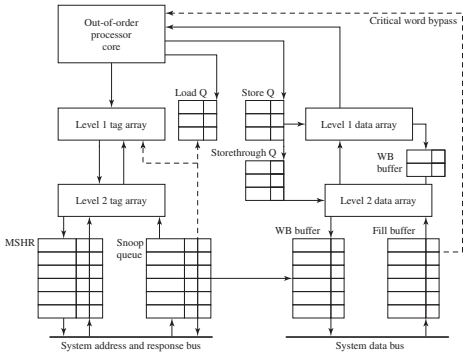
- Read on your own:
  - Review: Shen & Lipasti Chapter 3
  - W.-H. Wang, J.-L. Baer, and H. M. Levy. Organization of a two-level virtual-real cache hierarchy, *Proc. 16th ISCA,* pp. 140-148, June 1989 **(B6)** Online PDF
  - D. Kroft. Lockup-Free Instruction Fetch/Prefetch Cache Organization, *Proc. International Symposium on Computer Architecture* , May 1981 **(B6)**. Online PDF
  - N.P. Jouppi. Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers, *Proc. International Symposium on Computer Architecture* , June 1990 **(B6)**. Online PDF
- Discuss in class:
  - Review due 3/24/2010: Benjamin C. Lee, Ping Zhou, Jun Yang, Youtao Zhang, Bo Zhao, Engin Ipek, Onur Mutlu, Doug Burger, "Phase-Change Technology and the Future of Main Memory," IEEE Micro, vol. 30, no. 1, pp. 143-143, Jan./Feb. 2010
  - Read Sec. 1, skim Sec. 2, read Sec. 3: Bruce Jacob, "The Memory System: You Can't Avoid It, You Can't Ignore It, You Can't Fake It," Synthesis Lectures on Computer Architecture 2009 4:1, 1-77.

2

Advanced Memory Hierarchy

- Coherent Memory Interface
- Evaluation methods
- Better miss rate: skewed associative caches, victim caches
- Reducing miss costs through software restructuring
- Higher bandwidth: Lock-up free caches, superscalar caches
- Beyond simple blocks
- Two level caches
- Prefetching, software prefetching
- Main Memory, DRAM
- Virtual Memory, TLBs
- Interaction of caches, virtual memory

Coherent Memory Interface



Coherent Memory Interface

- Load Queue
  - Tracks inflight loads for aliasing, coherence
- Store Queue
  - Defers stores until commit, tracks aliasing
- Storethrough Queue or Write Buffer or Store Buffer
  - Defers stores, coalesces writes, must handle RAW
- MSHR
  - Tracks outstanding misses, enables *lockup-free caches* [Kroft ISCA 91]
- Snoop Queue
  - Buffers, tracks incoming requests from coherent I/O, other processors
- Fill Buffer
  - Works with MSHR to hold incoming partial lines
- Writeback Buffer
  - Defers writeback of evicted line (demand miss handled first)

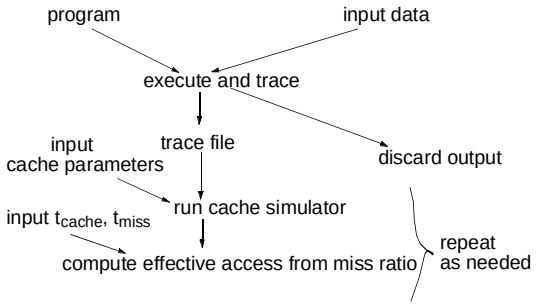
Evaluation Methods - Counters

- Counts hits and misses in hardware
  - see [Clark, TOCS 1983]
  - Intel VTune tool
- Accurate
- Realistic workloads - system, user, everything
- Requires machine to exist
- Hard to vary cache parameters
- Experiments not deterministic

Evaluation Methods - Analytical

- Mathematical expressions
  - Insight - can vary parameters
  - Fast
  - Absolute accuracy suspect for models with few parameters
  - Hard to determine many parameter values
  - Not widely used today

Evaluation: Trace-Driven Simulation



Evaluation: Trace-Driven Simulation

- Experiments repeatable
- Can be accurate
- Much recent progress
- Reasonable traces are very large ~gigabytes
- Simulation can be time consuming
- Hard to say if traces representative
- Don't model speculative execution

Evaluation: Execution-Driven Simulation

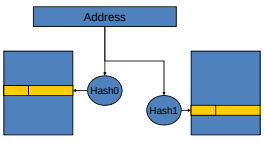
- Do full processor simulation each time
  - Actual performance; with ILP miss rate means nothing
    - Non-blocking caches
    - Prefetches (timeliness)
    - Pollution effects due to speculation
  - No need to store trace
  - Much more complicated simulation model
- Time-consuming - but good programming can help
- Very common today

Advanced Memory Hierarchy

- Coherent Memory Interface
- Evaluation methods
- **Better miss rate: skewed associative caches, victim caches**
- Reducing miss costs through software restructuring
- Higher bandwidth: Lock-up free caches, superscalar caches
- Beyond simple blocks
- Two level caches
- Prefetching, software prefetching
- Main Memory, DRAM
- Virtual Memory, TLBs
- Interaction of caches, virtual memory

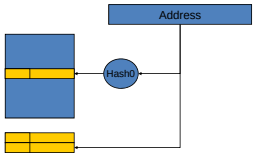
Seznec's Skewed Associative Cache

- Alleviates conflict misses in a conventional set assoc cache
- If two addresses conflict in 1 bank, they conflict in the others too
  - e.g., 3 addresses with same index bits will thrash in 2-way cache
- Solution: use different hash functions for each bank
- Works reasonably well: more robust conflict miss behavior
- But: how do you implement replacement policy?



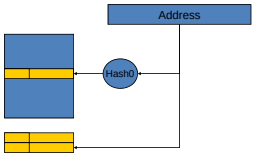
Jouppi's Victim Cache

- Targeted at conflict misses
- Victim cache: a small fully associative cache
  - holds victims replaced in direct-mapped or low-assoc
  - LRU replacement
  - a miss in cache + a hit in victim cache
    - => move line to main cache
- Poor man's associativity
  - Not all sets suffer conflicts; provide limited capacity for conflicts



Jouppi's Victim Cache

- Removes conflict misses, mostly useful for DM or 2-way
  - Even one entry helps some benchmarks
  - I-cache helped more than D-cache
- Versus cache size
  - Generally, victim cache helps more for smaller caches
- Versus line size
  - helps more with larger line size (why?)
- Used in Pentium Pro (P6) I-cache



Advanced Memory Hierarchy

- Coherent Memory Interface
- Evaluation methods
- Better miss rate: skewed associative caches, victim caches
- **Reducing miss costs through software restructuring**
- Higher bandwidth: Lock-up free caches, superscalar caches
- Beyond simple blocks
- Two level caches
- Prefetching, software prefetching
- Main Memory, DRAM
- Virtual Memory, TLBs
- Interaction of caches, virtual memory

Software Restructuring

- **If column-major (Fortran)**
  - **x[i+1, j] follows x [i,j] in memory**
  - **x[i,j+1] long after x[i,j] in memory**
- **Poor code**

```
for i = 1, rows
  for j = 1, columns
    sum = sum + x[i,j]
```

- **Conversely, if row-major (C/C++)**
- **Poor code**

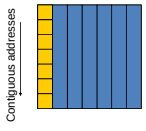
```
for j = 1, columns
  for i = 1, rows
    sum = sum + x[i,j]
```

Software Restructuring

- **Better column-major code**

```
for j = 1, columns
  for i = 1, rows
    sum = sum + x[i,j]
```

- **Optimizations - need to check if it is valid to do them**
  - **Loop interchange (used above)**
  - **Merging arrays**
  - **Loop fusion**
  - **Blocking**



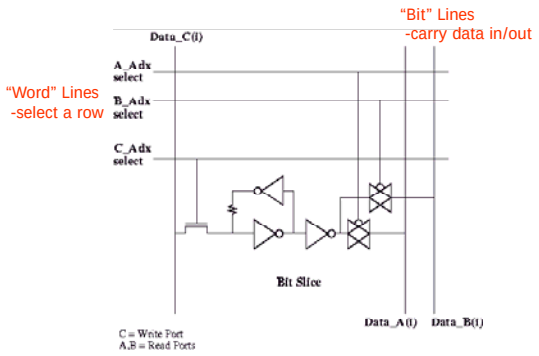
Advanced Memory Hierarchy

- Coherent Memory Interface
- Evaluation methods
- Better miss rate: skewed associative caches, victim caches
- Reducing miss costs through software restructuring
- **Higher bandwidth: Lock-up free caches, superscalar caches**
- Beyond simple blocks
- Two level caches
- Prefetching, software prefetching
- Main Memory, DRAM
- Virtual Memory, TLBs
- Interaction of caches, virtual memory

Superscalar Caches

- Increasing issue width => wider caches
- Parallel cache accesses are harder than parallel functional units
- Fundamental difference:
  - Caches have **state**, functional units don't
  - Operation thru one port affects future operations thru others
- Several approaches used
  - True multi-porting
  - Multiple cache copies
  - Virtual multi-porting
  - Multi-banking (interleaving)

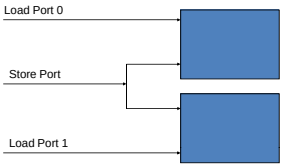
True Multiporting of SRAM



True Multiporting of SRAM

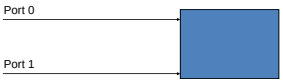
- Would be ideal
- Increases cache area
  - Array becomes wire-dominated
- Slower access
  - Wire delay across larger area
  - Cross-coupling capacitance between wires
- SRAM access difficult to pipeline

Multiple Cache Copies



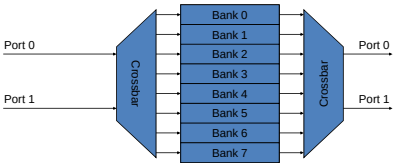
- Used in DEC Alpha 21164, IBM Power4
- Independent load paths
- Single shared store path
  - May be exclusive with loads, or internally dual-ported
- Bottleneck, not practically scalable beyond 2 paths
- Provides some fault-tolerance
  - Parity protection per copy
  - Parity error: restore from known-good copy
  - Avoids more complex ECC (no RMW for subword writes), still provides SEC

Virtual Multiporting



- Used in IBM Power2 and DEC 21264
  - Wave pipelining: pipeline wires WITHOUT latches
- Time-share a single port
- Not scalable beyond 2 ports
- Requires very careful array design to guarantee balanced paths
  - Second access cannot catch up with first access
- Short path constraint limits maximum clock period
- Complicates and reduces benefit of *speed binning*

Multi-banking or Interleaving



- Used in Intel Pentium (8 banks)
- Need routing network
- Must deal with bank conflicts
  - Bank conflicts not known till address generated
  - Difficult in non-data-capture machine with speculative scheduling
    - Replay – looks just like a cache miss
  - Sensitive to bank interleave: fine-grained vs. coarse-grained
- Spatial locality: many temporally local references to same block
  - Combine these with a "row buffer" approach?

Combined Schemes

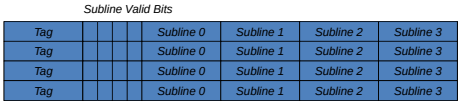
- Multiple banks with multiple ports
- Virtual multiporting of multiple banks
- Multiple ports and virtual multiporting
- Multiple banks with multiply virtually multiported ports
- Complexity!
- No good solution known at this time
  - Current generation superscalars get by with 1-3 ports

Advanced Memory Hierarchy

- Coherent Memory Interface
- Evaluation methods
- Better miss rate: skewed associative caches, victim caches
- Reducing miss costs through software restructuring
- Higher bandwidth: Lock-up free caches, superscalar caches
- **Beyond simple blocks**
- Two level caches
- Prefetching, software prefetching
- Main Memory, DRAM
- Virtual Memory, TLBs
- Interaction of caches, virtual memory

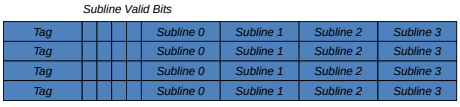
Sublines

- Break blocks into
  - Address block associated with tag
  - Transfer block to/from memory (subline, sub-block)
- Large address blocks
  - Decrease tag overhead
  - But allow fewer blocks to reside in cache (fixed mapping)



Sublines

- Larger transfer block
  - Exploit spatial locality
  - Amortize memory latency
  - But take longer to load
  - Replace more data already cached (more conflicts)
  - Cause unnecessary traffic
- Typically used in large L2/L3 caches to limit tag overhead
- Sublines tracked by MSHR during pending fill



Latency vs. Bandwidth

- Latency can be handled by
  - Hiding (or tolerating) it - out of order issue, nonblocking cache
  - Reducing it – better caches
- Parallelism helps to hide latency
  - MLP – multiple outstanding cache misses overlapped
- But increases bandwidth demand
- Latency ultimately limited by physics

Latency vs. Bandwidth

- Bandwidth can be handled by "spending" more (hardware cost)
  - Wider buses, interfaces
  - Banking/interleaving, multiporting
- Ignoring cost, a well-designed system should never be bandwidth-limited
  - Can't ignore cost!
- Bandwidth improvement usually increases latency
  - No free lunch
- Hierarchies decrease bandwidth demand to lower levels
  - Serve as traffic filters: a hit in L1 is filtered from L2
- Parallelism puts more demand on bandwidth
- If average b/w demand is not met => infinite queues
  - Bursts are smoothed by queues
- If burst is much larger than average => long queue
  - Eventually increases delay to unacceptable levels

Advanced Memory Hierarchy

- Coherent Memory Interface
- Evaluation methods
- Better miss rate: skewed associative caches, victim caches
- Reducing miss costs through software restructuring
- Higher bandwidth: Lock-up free caches, superscalar caches
- Beyond simple blocks
- **Multilevel caches**
- Prefetching, software prefetching
- Main Memory, DRAM
- Virtual Memory, TLBs
- Interaction of caches, virtual memory

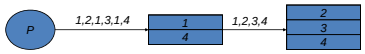
Multilevel Caches

- Ubiquitous in high-performance processors
  - Gap between L1 (core frequency) and main memory too high
  - Level 2 usually on chip, level 3 on or off-chip, level 4 off chip
- Inclusion in multilevel caches
  - Multi-level inclusion holds if L2 cache is superset of L1
  - Can handle virtual address synonyms
  - Filter coherence traffic: if L2 misses, L1 needn't see snoop
  - Makes L1 writes simpler
    - For both write-through and write-back

Multilevel Inclusion

- Example: local LRU not sufficient to guarantee inclusion
  - Assume L1 holds two and L2 holds three blocks
  - Both use local LRU
- Final state: L1 contains 1, L2 does not
  - Inclusion not maintained
- Different block sizes also complicate inclusion

Multilevel Inclusion



- Inclusion takes effort to maintain
  - Make L2 cache have bits or pointers giving L1 contents
  - Invalidate from L1 before replacing from L2
  - In example, removing 1 from L2 also removes it from L1
- Number of pointers per L2 block
  - L2 blocksize/L1 blocksize
- Reading list: [Wang, Baer, Levy ISCA 1989]

Multilevel Miss Rates

- Miss rates of lower level caches
  - Affected by upper level filtering effect
  - LRU becomes LRM, since "use" is "miss"
  - Can affect miss rates, though usually not important
- Miss rates reported as:
  - Miss per instruction
  - Global miss rate
  - Local miss rate
  - "Solo" miss rate
    - L2 cache sees all references (unfiltered by L1)

Advanced Memory Hierarchy

- Coherent Memory Interface
- Evaluation methods
- Better miss rate: skewed associative caches, victim caches
- Reducing miss costs through software restructuring
- Higher bandwidth: Lock-up free caches, superscalar caches
- Beyond simple blocks
- Two level caches
- **Prefetching, software prefetching**
- Main Memory, DRAM
- Virtual Memory, TLBs
- Interaction of caches, virtual memory

Prefetching

- Even "demand fetching" prefetches other words in block
  - Spatial locality
- Prefetching is useless
  - Unless a prefetch costs less than demand miss
- Ideally, prefetches should
  - Always get data before it is referenced
  - Never get data not used
  - Never prematurely replace data
  - Never interfere with other cache activity

Software Prefetching

- Use compiler to try to
  - Prefetch early
  - Prefetch accurately
- Prefetch into
  - Register (binding)
    - Use normal loads? Stall-on-use (Alpha 21164)
    - What about page faults? Exceptions?
  - Caches (non-binding) – preferred
    - Needs ISA support

Software Prefetching

- For example:

```
do j= 1, cols
  do ii = 1 to rows by BLOCK
    prefetch (&(x[i,j])+BLOCK)    # prefetch one block ahead
    do i = ii to ii + BLOCK-1
      sum = sum + x[i,j]
```

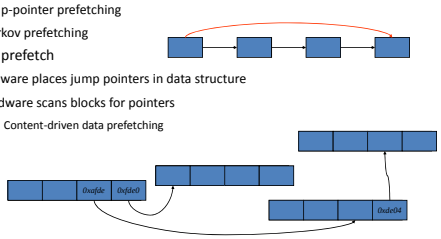
- How many blocks ahead should we prefetch?
  - Affects timeliness of prefetches
  - Must be scaled based on miss latency

Hardware Prefetching

- What to prefetch
  - One block spatially ahead
  - N blocks spatially ahead
  - Based on observed stride
- When to prefetch
  - On every reference
    - Hard to find if block to be prefetched already in the cache
  - On every miss
    - Better than doubling block size – why?
  - Tagged
    - Prefetch when prefetched item is referenced

Prefetching for Pointer-based Data Structures

- What to prefetch
  - Next level of tree: n+1, n+2, n+?
    - Entire tree? Or just one path
  - Next node in linked list: n+1, n+2, n+?
  - Jump-pointer prefetching
  - Markov prefetching
- How to prefetch
  - Software places jump pointers in data structure
  - Hardware scans blocks for pointers
    - Content-driven data prefetching



Stream or Prefetch Buffers

- Prefetching causes capacity and conflict misses (pollution)
  - Can displace useful blocks
- Aimed at compulsory and capacity misses
- Prefetch into buffers, NOT into cache
  - On miss start filling stream buffer with successive lines
  - Check both cache and stream buffer
    - Hit in stream buffer => move line into cache (promote)
    - Miss in both => clear and refill stream buffer
- Performance
  - Very effective for I-caches, less for D-caches
  - Multiple buffers to capture multiple streams (better for D-caches)
- Can use with any prefetching scheme to avoid pollution

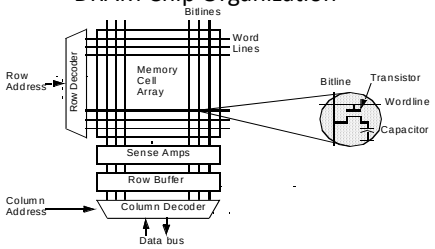
Advanced Memory Hierarchy

- Coherent Memory Interface
- Evaluation methods
- Better miss rate: skewed associative caches, victim caches
- Reducing miss costs through software restructuring
- Higher bandwidth: Lock-up free caches, superscalar caches
- Beyond simple blocks
- Two level caches
- Prefetching, software prefetching
- **Main Memory, DRAM**
- Virtual Memory, TLBs
- Interaction of caches, virtual memory

Main Memory

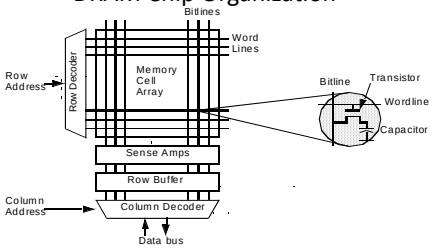
- DRAM chips
- Memory organization
  - Interleaving
  - Banking
- Memory controller design

DRAM Chip Organization



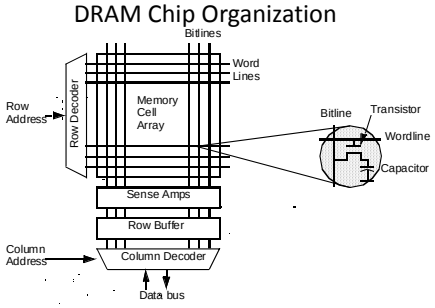
- Optimized for density, not speed
- Data stored as charge in capacitor
- Discharge on reads => destructive reads
- Charge leaks over time
  - refresh every few ms
- Cycle time roughly twice access time
- Need to precharge bitlines before access

DRAM Chip Organization



- Current generation DRAM.
  - 1Gbit, moving to 4 Gbit
  - 133 MHz synchronous interface
  - Data bus 2, 4, 8, 16 bits
- Address pins are time-multiplexed
  - Row address strobe (RAS)
  - Column address strobe (CAS)

DRAM Chip Organization



- New RAS results in:
  - Bitline precharge
  - Row decode, sense
  - Row buffer write (up to 8K)
- New CAS
  - Row buffer read
  - Much faster (3x)
- Streaming row accesses desirable

Simple Main Memory

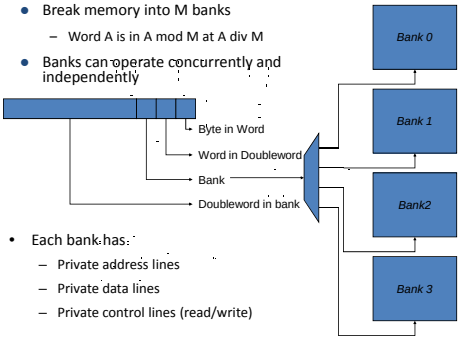
- Consider these parameters:
  - 1 cycle to send address
  - 6 cycles to access each word
  - 1 cycle to send word back
- Miss penalty for a 4-word block
  - (1 + 6 + 1) x 4 = 32
- How can we speed this up?

Wider(Parallel) Main Memory

- Make memory wider
  - Read out all words in parallel
- Memory parameters
  - 1 cycle to send address
  - 6 to access a double word
  - 1 cycle to send it back
- Miss penalty for 4-word block: 1+6+1 = 16
- Costs
  - Wider bus
  - Larger minimum expansion unit (e.g. paired DIMMs)

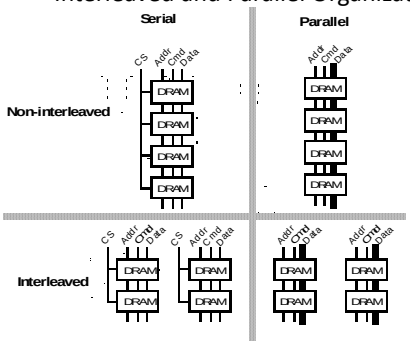
Interleaved Main Memory

- Break memory into M banks
  - Word A is in A mod M at A div M
- Banks can operate concurrently and independently



- Each bank has:
  - Private address lines
  - Private data lines
  - Private control lines (read/write)

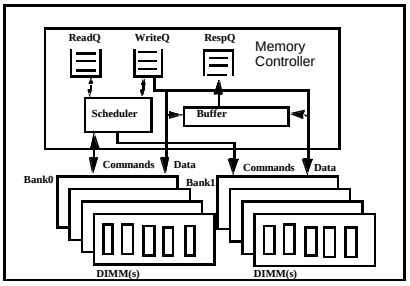
Interleaved and Parallel Organization



Interleaved Memory Summary

- Parallel memory adequate for sequential accesses
  - Load cache block: multiple sequential words
  - Good for writeback caches
- Banking useful otherwise
  - If many banks, choose a prime number
- Can also do both
  - Within each bank: parallel memory path
  - Across banks
    - Can support multiple concurrent cache accesses (nonblocking)

Memory Controller Organization



Memory Controller Organization

- **ReadQ**
  - Buffers multiple reads, enables scheduling optimizations
- **WriteQ**
  - Buffers writes, allows reads to bypass writes, enables scheduling opt.
- **RespQ**
  - Buffers responses until bus available
- **Scheduler**
  - FIFO? Or schedule to maximize for row hits (CAS accesses)
  - Scan queues for references to same page
  - Looks similar to issue queue with page number broadcasts for tag match
  - Fairness when sharing a memory controller [Nesbit, 2006]
- **Buffer**
  - Builds transfer packet from multiple memory words to send over processor bus

Advanced Memory Hierarchy

- Coherent Memory Interface
- Evaluation methods
- Better miss rate: skewed associative caches, victim caches
- Reducing miss costs through software restructuring
- Higher bandwidth: Lock-up free caches, superscalar caches
- Beyond simple blocks
- Two level caches
- Prefetching, software prefetching
- Main Memory, DRAM
- **Virtual Memory, TLBs**
- Interaction of caches, virtual memory

Main Memory and Virtual Memory

- Use of virtual memory
  - Main memory becomes another level in the memory hierarchy
  - Enables programs with address space or working set that exceed physically available memory
    - No need for programmer to manage overlays, etc.
    - Sparse use of large address space is OK
  - Allows multiple users or programs to timeshare limited amount of physical memory space and address space
- Bottom line: efficient use of expensive resource, and ease of programming

Virtual Memory

- Enables
  - Use more memory than system has
  - Think program is only one running
    - Don't have to manage address space usage across programs
    - E.g. think it always starts at address 0x0
  - Memory protection
    - Each program has private VA space: no-one else can clobber it
  - Better performance
    - Start running a large program before all of it has been loaded from disk

Virtual Memory – Placement

- Main memory managed in larger blocks
  - *Page size* typically 4K – 16K
- Fully flexible placement; fully associative
  - Operating system manages placement
  - Indirection through *page table*
  - Maintain mapping between:
    - Virtual address (seen by programmer)
    - Physical address (seen by main memory)

Virtual Memory – Placement

- Fully associative implies expensive lookup?
  - In caches, yes: check multiple tags in parallel
- In virtual memory, expensive lookup is avoided by using a level of indirection
  - Lookup table or hash table
  - Called a *page table*

Virtual Memory – Identification

Virtual Address	Physical Address	Dirty bit
0x20004000	0x2000	Y/N

- Similar to cache tag array
  - Page table entry contains VA, PA, dirty bit
- Virtual address:
  - Matches programmer view; based on register values
  - Can be the same for multiple programs sharing same system, without conflicts
- Physical address:
  - Invisible to programmer, managed by O/S
  - Created/deleted on demand basis, can change

Virtual Memory – Replacement

- Similar to caches:
  - FIFO
  - LRU; overhead too high
    - Approximated with reference bit checks
    - "Clock algorithm" intermittently clears all bits
  - Random
- O/S decides, manages
  - CS537

Virtual Memory – Write Policy

- Write back
  - Disks are too slow to write through
- Page table maintains dirty bit
  - Hardware must set dirty bit on first write
  - O/S checks dirty bit on eviction
  - Dirty pages written to backing store
    - Disk write, 10+ ms

Virtual Memory Implementation

- Caches have fixed policies, hardware FSM for control, pipeline stall
- VM has very different miss penalties
  - Remember disks are 10+ ms!
- Hence engineered differently

Page Faults

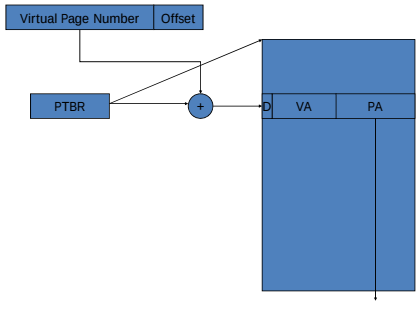
- A virtual memory miss is a page fault
  - Physical memory location does not exist
  - Exception is raised, save PC
  - Invoke OS page fault handler
    - Find a physical page (possibly evict)
    - Initiate fetch from disk
  - Switch to other task that is ready to run
  - Interrupt when disk access complete
  - Restart original instruction
- Why use O/S and not hardware FSM?

Address Translation

VA	PA	Dirty	Ref	Protection
0x20004000	0x2000	Y/N	Y/N	Read/Write/ Execute

- O/S and hardware communicate via PTE
- How do we find a PTE?
  - &PTE = PTBR + page number * sizeof(PTE)
  - PTBR is private for each program
    - Context switch replaces PTBR contents

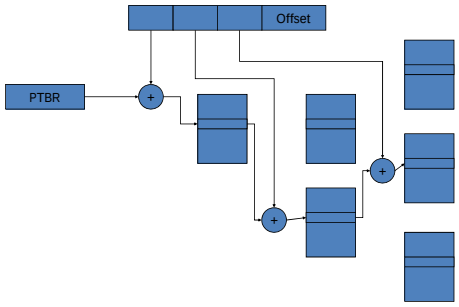
Address Translation



Page Table Size

- How big is page table?
  - 2^{32} / 4K * 4B = 4M per program (!)
  - Much worse for 64-bit machines
- To make it smaller
  - Use limit register(s)
    - If VA exceeds limit, invoke O/S to grow region
  - Use a multi-level page table
  - Make the page table pageable (use VM)

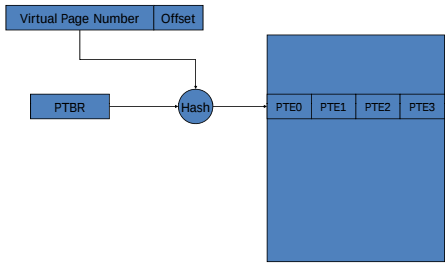
Multilevel Page Table



Hashed Page Table

- Use a hash table or inverted page table
  - PT contains an entry for each real address
    - Instead of entry for every virtual address
  - Entry is found by hashing VA
  - Oversize PT to reduce collisions: #PTE = 4 x (#phys. pages)

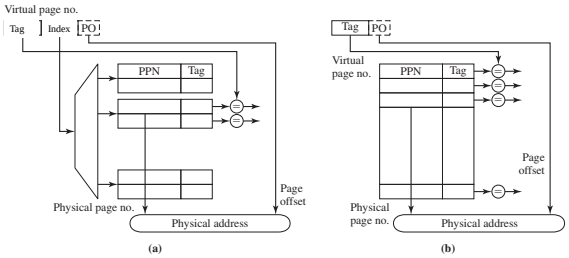
Hashed Page Table



High-Performance VM

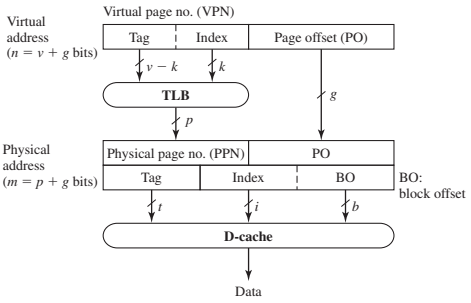
- VA translation
  - Additional memory reference to PTE
  - Each instruction fetch/load/store now 2 memory references
    - Or more, with multilevel table or hash collisions
  - Even if PTE are cached, still slow
- Hence, use special-purpose cache for PTEs
  - Called TLB (translation lookaside buffer)
  - Caches PTE entries
  - Exploits temporal and spatial locality (just a cache)

Translation Lookaside Buffer



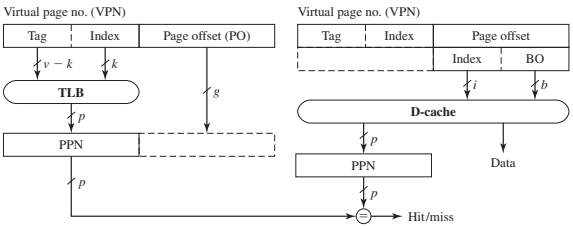
- Set associative (a) or fully associative (b)
- Both widely employed

Interaction of TLB and Cache



- Serial lookup: first TLB then D-cache
- Excessive cycle time

Virtually Indexed Physically Tagged L1



- Parallel lookup of TLB and cache
- Faster cycle time
- Index bits must be untranslated
  - Restricts size of n-associative cache to n x (virtual page size)
  - E.g. 4-way SA cache with 4KB pages max. size is 16KB

Virtual Memory Protection

- Each process/program has private virtual address space
  - Automatically protected from rogue programs
- Sharing is possible, necessary, desirable
  - Avoid copying, staleness issues, etc.
- Sharing in a controlled manner
  - Grant specific permissions
    - Read
    - Write
    - Execute
    - Any combination

Protection

- Process model
  - Privileged kernel
  - Independent user processes
- Privileges vs. policy
  - Architecture provided primitives
  - OS implements policy
  - Problems arise when h/w implements policy
- Separate policy from mechanism!

Protection Primitives

- User vs kernel
  - at least one privileged mode
  - usually implemented as mode bits
- How do we switch to kernel mode?
  - Protected "gates" or system calls
  - Change mode and continue at pre-determined address
- Hardware to compare mode bits to access rights
  - Only access certain resources in kernel mode
  - E.g. modify page mappings

Protection Primitives

- Base and bounds
  - Privileged registers
    base <= address <= bounds
- Segmentation
  - Multiple base and bound registers
  - Protection bits for each segment
- Page-level protection (most widely used)
  - Protection bits in page entry table
  - Cache them in TLB for speed

VM Sharing

- Share memory locations by:
  - Map shared physical location into both address spaces:
    - E.g. PA 0xC00DA becomes:
      - VA 0x2D000DA for process 0
      - VA 0x4D000DA for process 1
  - Either process can read/write shared location
- However, causes **synonym** problem

VA Synonyms

- Virtually-addressed caches are desirable
  - No need to translate VA to PA before cache lookup
  - Faster hit time, translate only on misses
- However, VA synonyms cause problems
  - Can end up with two copies of same physical line
  - Causes coherence problems [Wang, Baer reading]
- Solutions:
  - Flush caches/TLBs on context switch
  - Extend cache tags to include PID
    - Effectively a shared VA space (PID becomes part of address)
  - Enforce global shared VA space (PowerPC)
    - Requires another level of addressing (EA->VA->PA)

Summary: Advanced Memory Hierarchy

- Coherent Memory Interface
- Evaluation methods
- Better miss rate: skewed associative caches, victim caches
- Reducing miss costs through software restructuring
- Higher bandwidth: Lock-up free caches, superscalar caches
- Beyond simple blocks
- Two level caches
- Prefetching, software prefetching
- Main Memory, DRAM
- Virtual Memory, TLBs
- Interaction of caches, virtual memory