Lecture 14: Memory Systems
-- Virtual Memory

CSE 564 Computer Architecture Fall 2016

Department of Computer Science and Engineering
Yonghong Yan
yan@oakland.edu
www.secs.oakland.edu/~yan
Topics for Memory Systems

- Memory Technology and Metrics
  - SRAM, DRAM, Flash/SSD, 3-D Stack Memory, Phase-change memory
  - Latency and Bandwidth, Error Correction
  - Memory wall

- Cache
  - Cache basics
  - Cache performance and optimization
  - Advanced optimization
  - Multiple-level cache, shared and private cache, prefetching

- Virtual Memory
  - Protection, Virtualization, and Relocation
  - Page/segment, protection
  - Address Translation and TLB

- Parallelism (to be discussed in TLP)
  - Memory Consistency model
    - Instructions for fence, etc
  - Cache coherence
  - NUMA and first touch
  - Transactional memory

- Implementation
  - Software/Hardware interface
  - Cache/memory controller
  - Bus systems and interconnect
Virtual Memory: Contents

• Motivation and Background
  – Protection, Virtualization and Relocation
• Paging and Address Translation
  – Linear page table
  – Multi-level page table
  – Page fault
  – Page replacement and write
• Making Address Translation Fast: TLB

• Memory Protection

• Memory Hierarchy: Putting VM and Cache together
Virtual Memory: Motivations and Benefits

• **Protection**: to allow efficient and safe sharing of memory among multiple programs
  – Conventional Multi-programming, time-sharing OS
  – Today for the memory needed by multiple virtual machines for cloud computing

• **Virtualization**: to remove the programming burdens of a small, limited amount of main memory.
  – 4G memory space of 32-bit OS/machine even with $\ll 4$GB physical memory, e.g. 512MB
    • Each sees 0x00000000 – 0xFFFFFFFF memory

• **Relocation**: simplifies loading the program for execution.
  – allows the same program to run in any location in physical memory.

• It is called **Virtual Memory**, thus **NOT REAL** memory
• In a bare machine, the only kind of address is a physical address
Dynamic Address Translation

• Motivation
  – In early machines, I/O was slow and each I/O transfer involved the CPU (programmed I/O)
  – Higher throughput possible if CPU and I/O of 2 or more programs were overlapped, how?
    => multiprogramming with DMA I/O devices, interrupts

• Location-independent programs
  – Programming and storage management ease
    => need for a base register

• Protection
  – Independent programs should not affect each other inadvertently
    => need for a bound register

• Multiprogramming drives requirement for resident supervisor software to manage context switches between multiple programs
Base and bounds registers are visible/accessible only when processor is running in the *supervisor mode*.
Separate Areas for Program and Data
(Scheme used on all Cray vector supercomputers prior to X1, 2002)

Load X

Program Address Space

Data Bound Register
Mem. Address Register
Data Base Register

Program Bound Register
Program Counter
Program Base Register

Logical Address
Physical Address

≥

Bounds Violation?

≥

Bounds Violation?

+}

What is an advantage of this separation?
Can fold addition of base register into (register+immediate) address calculation using a carry-save adder (sums three numbers with only a few gate delays more than adding two numbers)
What if the next request is for 32K?
At some stage programs have to be moved around to compact the storage.
Paged Memory Systems

- Processor-generated address can be split into:

<table>
<thead>
<tr>
<th>Page Number</th>
<th>Offset</th>
</tr>
</thead>
</table>

- A Page Table contains the physical address at the start of each page

```
0 1 2 3
0 1 2 3
```

Address Space of User-1

```
0 1 2 3
```

Page Table of User-1

```
1 0 3 2
```

Physical Memory

Page tables make it possible to store the pages of a program non-contiguously.
Private Address Space per Process (User)

- Each user (process) has a page table
- Page table contains an entry for each user page
Virtual Memory

• Use main memory as a “cache” for secondary (disk) storage
  – Managed jointly by CPU hardware and the operating system (OS)

• Programs share main memory
  – Each gets a private virtual address space holding its frequently used code and data
  – Protected from other programs

• CPU and OS translate virtual addresses to physical addresses
  – VM “block” is called a page
  – VM translation “miss” is called a page fault
VM: On-Demand Paging and Swap: Protection, Virtualization and Relocation

- **Fixed-size pages** (e.g., 4K)

\[ \text{ld x10, 0x3540(x5)} \]

- **Protection**: with multiple virtual address spaces, errors are confined to one address space
  - Between programs (processes)

- **Virtualization** via on-demand paging: move only frequently used pages to VM
  - Principle of locality

- **Relocation**: pages on disk can be loaded to any free physical pages
VM: Address Translation & Protection

- Fixed-size pages (e.g., 4K)

- Every instruction and data access needs address translation and protection checks
  - Within a program: writes to EXE or Read-only segment are violations

- A good VM design needs to be fast (~ one cycle) and space efficient
Each user (process) has a page table.

- Page table contains an entry for each user page.
  - Each entry stores the physical page address and other info.
Page Tables

• Stores placement information
  – Array of page table entries, indexed by virtual page number
  – Page table register in CPU points to page table in physical memory

• If page is present in memory
  – PTE stores the physical page number
  – Plus other status bits (referenced, dirty, ...)

• If page is not present
  – PTE can refer to location in swap space on disk
Where Should Page Tables Reside?

• Space required by the page tables (PT) is proportional to the address space, number of users, ...
  \[ \Rightarrow \text{Too large to keep in registers} \]

• Idea: Keep PTs in the main memory
  – needs one reference to retrieve the page base address and another to access the data word
  • \[ \Rightarrow \text{doubles the number of memory references!} \]
Page Tables in Physical Memory

User 1 Virtual Address Space

User 2 Virtual Address Space

increased memory access latency?
Linear Page Table Example (4K pages)

- 32-bit address
- 4K-size page
- 4-byte PTE
Linear Page Table

• With 32-bit addresses, 4-KB pages & 4-byte PTEs:
  – 220 PTEs, i.e, 4 MB page table per user
  – 4 GB of swap needed to back up full virtual address space

• Larger pages?
  – Internal fragmentation (Not all memory in page is used)
  – Larger page fault penalty (more time to read from disk)

• What about 64-bit virtual address space???
  – How many page table entries (PTEs)?
Hierarchical Page Table

Virtual Address from CPU

31  22  21  12  11  0
   p1  p2  offset

10-bit 10-bit
L1 index  L2 index

Root of the Current Page Table

(Processor Register)

Level 1 Page Table

Level 2 Page Tables

Physical Memory

Data Pages

What is the downside?

page in primary memory
page in secondary memory
PTE of a nonexistent page
Two-Level Page Tables in Physical Memory

Virtual Address Spaces

User 1

VA1

User 2

VA1

Physical Memory

Level 1 PT
User 1

Level 1 PT
User 2

Level 2 PT
User 1

User2/VA1

User1/VA1

Level 2 PT
User 2
4 Page Table Levels of 64-bit AMD Opteron

Figure B.27 The mapping of an Opteron virtual address. The Opteron virtual memory implementation with four page table levels supports an effective physical address size of 40 bits. Each page table has 512 entries, so each level
Mapping Pages to Storage

- Demand paging
  - Valid bit to indicate whether a page is in physical mem or not
Page Fault Handler

• Use faulting virtual address to find PTE
• Locate page on disk
• Choose page to replace
  – If dirty, write to disk first
• Read page into memory and update page table
• Make process runnable again
  – Restart from faulting instruction
Page Fault Penalty

• On page fault, the page must be fetched from disk
  – Takes millions of clock cycles
  – Handled by OS code

• Try to minimize page fault rate
  – Fully associative placement
  – Smart replacement algorithms
Replacement and Writes

- To reduce page fault rate, prefer least-recently used (LRU) replacement
  - Reference bit (aka use bit) in PTE set to 1 on access to page
  - Periodically cleared to 0 by OS
  - A page with reference bit = 0 has not been used recently

- Disk writes take millions of cycles
  - Block at once, not individual locations
  - Write through is impractical
  - Use write-back
  - Dirty bit in PTE set when page is written
Translation Lookaside Buffers (TLB)

Address translation is very expensive!
   In a two-level page table, each reference becomes several memory accesses

Solution: Cache translations in TLB, Principle of Locality for PTE

- TLB hit $\implies$ Single-Cycle Translation
- TLB miss $\implies$ Page-Table Walk to refill

W, R: Read and write permission bits
D: Dirty
V: Valid

(Virtual address)

<table>
<thead>
<tr>
<th>V</th>
<th>R</th>
<th>W</th>
<th>D</th>
<th>tag</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(PPN = physical page number)

Physical address

VPN

offset
Fast Translation Using a TLB

The TLB acts as a cache of the page table for the entries that map to physical pages only. The TLB contains a subset of the virtual-to-physical page mappings that are in the page table.
From VA to Byte via TLB and L-1 Cache
From VA to Data via TLB, L-1, and L2
How Would You Design the TLB?

• Associativity?

• Replacement policy?
TLB Designs

- Typically 32-128 entries, usually fully associative
  - Each entry maps a large page, hence less spatial locality across pages → more likely that two entries conflict
  - Sometimes larger TLBs (256-512 entries) are 4-8 way set-associative
  - Larger systems sometimes have multi-level (L1 and L2) TLBs
  - 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate
- Random or FIFO replacement policy
- No process information in TLB?
- **TLB Reach**: Size of largest virtual address space that can be simultaneously mapped by TLB
  - How much of the physical address space is the TLB mapping to?

Example: 64 TLB entries, 4KB pages, one page per entry

\[
\text{TLB Reach} = 64 \text{ entries} \times 4 \text{ KB} = 256 \text{ KB (if contiguous)}
\]
Page-Based Virtual-Memory Machine
(Hardware Page-Table Walk)

• Assumes page tables held in untranslated physical memory
Operon Data TLB

- Operon Data TLB

**Figure B.24** Operation of the Operon data TLB during address translation. The four steps of a TLB hit are shown as circled numbers. This TLB has 40 entries. Section B.5 describes the various protection and access fields of an Operon page table entry.
TLB Miss Handler

- TLB miss indicates
  - Page present, but PTE not in TLB
  - Page not preset
- Must recognize TLB miss before destination register overwritten
  - Raise exception
- Handler copies PTE from memory to TLB
  - Then restarts instruction
  - If page not present, page fault will occur
Handling a TLB Miss

• Software (MIPS, Alpha)
  – TLB miss causes an exception and the operating system walks the page tables and reloads TLB. A privileged “untranslated” addressing mode used for walk.

• Hardware (SPARC v8, x86, PowerPC, RISC-V)
  – A memory management unit (MMU) walks the page tables and reloads the TLB.
  – If a missing (data or PT) page is encountered during the TLB reloading, MMU gives up and signals a Page Fault exception for the original instruction.

• Tradeoffs?
  – The hardware needs to know the structure of the page table for hardware handling.
Address Translation: *putting it all together*

Virtual Address

- TLB Lookup
  - miss
  - hit
    - Protection Check
      - denied
      - permitted
        - Physical Address *(to cache)*
    - Page Table Walk
      - the page is
        - ∈ memory
        - ∉ memory
          - Page Fault
            - (OS loads page)
  - Update TLB

Where?

SEGFAULT
Handling VM-related exceptions

- Handling a TLB miss needs a hardware or software mechanism to refill TLB
- Handling a page fault (e.g., page is on disk) needs a restartable exception so software handler can resume after retrieving page
  - Precise exceptions are easy to restart
  - Can be imprecise but restartable, but this complicates OS software
- Handling protection violation may abort process
  - But often handled the same as a page fault
• Need to cope with additional latency of TLB:
  – slow down the clock?
  – pipeline the TLB and cache access?
  – virtual address caches
  – parallel TLB/cache access
Memory Protection

• Different tasks can share parts of their virtual address spaces
  – But need to protect against errant access
  – Requires OS assistance

• Hardware support for OS protection
  – Privileged supervisor mode (aka kernel mode)
  – Privileged instructions
  – Page/segment tables and other state information only accessible in supervisor mode
  – System call exception (e.g., syscall in MIPS)
# Page vs Segment

- **Pages**: fixed-size blocks
  - 4096 to 8192 bytes

- **Segments**: size varies.
  - The largest segment supported on any processor ranges from 216 bytes up to 232 bytes; the smallest segment is 1 byte.

<table>
<thead>
<tr>
<th></th>
<th>Page</th>
<th>Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Words per address</td>
<td>One</td>
<td>Two (segment and offset)</td>
</tr>
<tr>
<td>Programmer visible?</td>
<td>Invisible to application programmer</td>
<td>May be visible to application programmer</td>
</tr>
<tr>
<td>Replacing a block</td>
<td>Trivial (all blocks are the same size)</td>
<td>Difficult (must find contiguous, variable-size, unused portion of main memory)</td>
</tr>
<tr>
<td>Memory use inefficiency</td>
<td>Internal fragmentation (unused portion of page)</td>
<td>External fragmentation (unused pieces of main memory)</td>
</tr>
<tr>
<td>Efficient disk traffic</td>
<td>Yes (adjust page size to balance access time and transfer time)</td>
<td>Not always (small segments may transfer just a few bytes)</td>
</tr>
</tbody>
</table>
IA-32 Segment/Page Hybrid VM

- Segment descriptor table
  - *Present bit*—Equivalent to the PTE valid bit, used to indicate this is a valid translation
  - *Base field*—Equivalent to a page frame address, containing the physical address of the first byte of the segment
  - *Access bit*—Like the reference bit or use bit in some architectures that is helpful for replacement algorithms
  - *Attributes field*—Specifies the valid operations and protection levels for operations that use this segment
Protection Bits in AMD 64-bit Opteron PTE

- **Presence**—Says that page is present in memory.
- **Read/write**—Says whether page is read-only or read-write.
- **User/supervisor**—Says whether a user can access the page or if it is limited to the upper three privilege levels.
- **Dirty**—Says if page has been modified.
- **Accessed**—Says if page has been read or written since the bit was last cleared.
- **Page size**—Says whether the last level is for 4 KB pages or 4 MB pages; if it’s the latter, then the Opteron only uses three instead of four levels of pages.
- **No execute**—Not found in the 80386 protection scheme, this bit was added to prevent code from executing in some pages.
- **Page level cache disable**—Says whether the page can be cached or not.
- **Page level write-through**—Says whether the page allows write-back or write-through for data caches.
The Memory Hierarchy: VM and Caching Together

• Common principles apply at all levels of the memory hierarchy
  – Based on notions of caching

• At each level in the hierarchy, 4 questions:
  1. Block placement
  2. Finding a block
  3. Replacement on a miss
  4. Write policy
Block Placement

• Determined by associativity
  – Direct mapped (1-way associative)
    • One choice for placement
  – n-way set associative
    • n choices within a set
  – Fully associative
    • Any location

• Higher associativity reduces miss rate
  – Increases complexity, cost, and access time
Finding a Block

- Hardware caches
  - Reduce comparisons to reduce cost
- Virtual memory
  - Full table lookup makes full associativity feasible
  - Benefit in reduced miss rate

<table>
<thead>
<tr>
<th>Associativity</th>
<th>Location method</th>
<th>Tag comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct mapped</td>
<td>Index</td>
<td>1</td>
</tr>
<tr>
<td>n-way set associative</td>
<td>Set index, then search entries within the set</td>
<td>n</td>
</tr>
<tr>
<td>Fully associative</td>
<td>Search all entries</td>
<td>#entries</td>
</tr>
<tr>
<td></td>
<td>Full lookup table</td>
<td>0</td>
</tr>
</tbody>
</table>
Replacement

• Choice of entry to replace on a miss
  – Least recently used (LRU)
    • Complex and costly hardware for high associativity
  – Random
    • Close to LRU, easier to implement

• Virtual memory
  – LRU approximation with hardware support
Write Policy

• Write-through
  – Update both upper and lower levels
  – Simplifies replacement, but may require write buffer

• Write-back
  – Update upper level only
  – Update lower level when block is replaced
  – Need to keep more state

• Virtual memory
  – Only write-back is feasible, given disk write latency
Interface Signals

- **CPU** to **Cache**:
  - Read/Write
  - Valid
  - Address (32 bits)
  - Write Data (32 bits)
  - Read Data (32 bits)
  - Ready

- **Cache** to **Memory**:
  - Read/Write
  - Valid
  - Address (32 bits)
  - Write Data (128 bits)
  - Read Data (128 bits)
  - Ready

- **Multiple cycles per access**
## Virtual Memory vs Caching

<table>
<thead>
<tr>
<th>Parameter</th>
<th>First-level cache</th>
<th>Virtual memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block (page) size</td>
<td>16–128 bytes</td>
<td>4096–65,536 bytes</td>
</tr>
<tr>
<td>Hit time</td>
<td>1–3 clock cycles</td>
<td>100–200 clock cycles</td>
</tr>
<tr>
<td>Miss penalty</td>
<td>8–200 clock cycles</td>
<td>1,000,000–10,000,000 clock cycles</td>
</tr>
<tr>
<td>(access time)</td>
<td>(6–160 clock cycles)</td>
<td>(800,000–8,000,000 clock cycles)</td>
</tr>
<tr>
<td>(transfer time)</td>
<td>(2–40 clock cycles)</td>
<td>(200,000–2,000,000 clock cycles)</td>
</tr>
<tr>
<td>Miss rate</td>
<td>0.1–10%</td>
<td>0.000001–0.001%</td>
</tr>
<tr>
<td>Address mapping</td>
<td>25–45-bit physical address to 14–20-bit cache address</td>
<td>32–64-bit virtual address to 25–45-bit physical address</td>
</tr>
</tbody>
</table>

**Figure B.20** Typical ranges of parameters for caches and virtual memory. Virtual memory parameters represent increases of 10 to 1,000,000 times over cache parameters. Normally, first-level caches contain at most 1 MB of data, whereas physical memory contains 256 MB to 1 TB.
Recap of Our Progress

- Preface
- Acknowledgments
- 1 Fundamentals of Quantitative Design and Analysis
- 2 Memory Hierarchy Design
- 3 Instruction-Level Parallelism and Its Exploitation
- 4 Data-Level Parallelism in Vector, SIMD, and GPU Architectures
- 5 Thread-Level Parallelism
- 6 Warehouse-Scale Computers to Exploit Request-Level and Data-Level Parallelism
- Appendix A. Instruction Set Principles
- Appendix B. Review of Memory Hierarchy
- Appendix C. Pipelining: Basic and Intermediate Concepts

References