Review (#1/3)

° Apply Principle of Locality Recursively
° Reduce Miss Penalty? add a (L2) cache
° Manage memory to disk? Treat as cache
  • Included protection as bonus, now critical
  • Use Page Table of mappings vs. tag/data in cache
° Virtual memory to Physical Memory Translation too slow?
  • Add a cache of Virtual to Physical Address Translations, called a TLB

Review (#2/3)

° Virtual Memory allows protected sharing of memory between processes with less swapping to disk, less fragmentation than always swap or base/bound via segmentation
° Spatial Locality means Working Set of Pages is all that must be in memory for process to run fairly well
° TLB to reduce performance cost of VM
° Need more compact representation to reduce memory size cost of simple 1-level page table (especially 32 – 64-bit addresses)

Why Caches?

µProc 60%/yr.
DRAM 7%/yr.

“Moore’s Law”

Processor-Memory Performance Gap:
(grows 50% / year)

° 1989 first Intel CPU with cache on chip;
° 1999 gap “Tax”; 37% area of Alpha 21164, 61% StrongArm SA110, 64% Pentium Pro
Memory Hierarchy Pyramid

Central Processor Unit (CPU)

“Upper”

Levels in memory hierarchy

Level 1
Level 2
Level 3
... 
Level n

“Lower”

Size of memory at each level

Principle of Locality (in time, in space) +
Hierarchy of Memories of different speed, cost; exploit to improve cost-performance

Why virtual memory? (#1/2)

° Protection
  • regions of the address space can be read only, execute only, ...

° Flexibility
  • portions of a program can be placed anywhere, without relocation (changing addresses)

° Expandability
  • can leave room in virtual address space for objects to grow

° Storage management
  • allocation/deallocation of variable sized blocks is costly and leads to (external) fragmentation; paging solves this

Why virtual memory? (#2/2)

° Generality
  • ability to run programs larger than size of physical memory

° Storage efficiency
  • retain only most important portions of the program in memory

° Concurrent I/O
  • execute other processes while loading/dumping page

Virtual Memory Review (#1/4)

° User program view of memory:
  • Contiguous
  • Start from some set address
  • Infinitely large
  • Is the only running program

° Reality:
  • Non-contiguous
  • Start wherever available memory is
  • Finite size
  • Many programs running at a time
Virtual Memory Review (#2/4)

°Virtual memory provides:
  • illusion of contiguous memory
  • all programs starting at same set address
  • illusion of infinite memory
  • protection

Virtual Memory Review (#3/4)

°Implementation:
  • Divide memory into “chunks” (pages)
  • Operating system controls pagetable that maps virtual addresses into physical addresses
  • Think of memory as a cache for disk
  • TLB is a cache for the pagetable

Why Translation Lookaside Buffer (TLB)?

°Paging is most popular implementation of virtual memory (vs. base/bounds in segmentation)
°Every paged virtual memory access must be checked against Entry of Page Table in memory to provide protection
°Cache of Page Table Entries makes address translation possible without memory access (in common case) to make translation fast

Virtual Memory Review (#4/4)

°Let’s say we’re fetching some data:
  • Check TLB (input: VPN, output: PPN)
    - hit: fetch translation
    - miss: check pagetable (in memory)
      ➢ pagetable hit: fetch translation
      ➢ pagetable miss: page fault, fetch page from disk to memory, return translation to TLB
  • Check cache (input: PPN, output: data)
    - hit: return value
    - miss: fetch value from memory
Three Advantages of Virtual Memory

1) Translation:
- Program can be given consistent view of memory, even though physical memory is scrambled
- Makes multiple processes reasonable
- Only the most important part of program ("Working Set") must be in physical memory
- Contiguous structures (like stacks) use only as much physical memory as necessary yet still grow later

2) Protection:
- Different processes protected from each other
- Different pages can be given special behavior
  - (Read Only, Invisible to user programs, etc).
- Privileged data protected from User programs
- Very important for protection from malicious programs ⇒ Far more "viruses" under Microsoft Windows

3) Sharing:
- Can map same physical page to multiple users ("Shared memory")

4 Questions for Memory Hierarchy

° Q1: Where can a block be placed in the upper level? (Block placement)
° Q2: How is a block found if it is in the upper level? (Block identification)
° Q3: Which block should be replaced on a miss? (Block replacement)
° Q4: What happens on a write? (Write strategy)
Q1: Where block placed in upper level?

- Block 12 placed in 8 block cache:
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = Block Number Mod Number of Sets

<table>
<thead>
<tr>
<th>Set Select</th>
<th>Data Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>11111111</td>
<td>11111111</td>
</tr>
<tr>
<td>01010101</td>
<td>01010101</td>
</tr>
<tr>
<td>10101010</td>
<td>10101010</td>
</tr>
</tbody>
</table>

Direct indexing (using index and block offset), and tag comparing

Increasing associativity shrinks index, expands tag

Q2: How is a block found in upper level?

- Direct indexing (using index and block offset), and tag comparing

Q3: Which block replaced on a miss?

- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

Miss Rates

<table>
<thead>
<tr>
<th>Associativity</th>
<th>2-way</th>
<th>4-way</th>
<th>8-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 KB</td>
<td>5.2%</td>
<td>5.7%</td>
<td>4.7%</td>
</tr>
<tr>
<td></td>
<td>4.7%</td>
<td>5.3%</td>
<td>4.4%</td>
</tr>
<tr>
<td></td>
<td>5.0%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64 KB</td>
<td>1.9%</td>
<td>2.0%</td>
<td>1.5%</td>
</tr>
<tr>
<td></td>
<td>1.5%</td>
<td>1.7%</td>
<td>1.4%</td>
</tr>
<tr>
<td></td>
<td>1.5%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>256 KB</td>
<td>1.15%</td>
<td>1.17%</td>
<td>1.13%</td>
</tr>
<tr>
<td></td>
<td>1.13%</td>
<td>1.12%</td>
<td>1.12%</td>
</tr>
</tbody>
</table>

Q4: What happens on a write?

- **Write through**—The information is written to both the block in the cache and to the block in the lower-level memory.
- **Write back**—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.

- is block clean or dirty?

Pros and Cons of each?

- **WT**: read misses cannot result in writes
- **WB**: no writes of repeated writes
3D - Graphics For Mobile Phones

- Developed in collaboration with Imagination Technologies, MBX 2D and 3D accelerator cores deliver PC and console-quality 3D graphics on embedded ARM-based devices.
- Supporting the feature-set and performance-level of commodity PC hardware, MBX cores use a unique screen-tiling technology to reduce the memory bandwidth and power consumption to levels suited to mobile devices, providing excellent price-performance for embedded SoC devices.
  - 660K gates (870K with optional VGP geometry processor)
  - 80MHz operation in 0.18µm process
  - Over 120MHz operation in 0.13µm process
  - Up to 500 mega pixel/sec effective fill rate
  - Screen tiling and deferred texturing - only visible pixels are rendered
  - Internal Z-buffer tile within the MBX core
  - "Up to 2.5 million triangle/sec rendering rate"
  - "Suitable to QVGA (320x240) up to VGA (640x480) resolution screens"
  - "<1mW/MHz in 0.13µm process and <2mW in 0.18 µm process"
  - "Optional VGP floating point geometry engine compatible with Microsoft VertexShader specification"
  - "2D and 3D graphics acceleration and video acceleration"

Address Translation & 3 Exercises

Virtual Address

- VPN-tag
- INDEX
- Offset

- TLB
- TLB-tag
- PPN

Physical Address

- PPN
- Offset

VPN = VPN-tag + Index

Address Translation Exercise 1 (#1/2)

Exercise:

- 40-bit VA, 16 KB pages, 36-bit PA

Number of bits in Virtual Page Number?

- a) 18; b) 20; c) 22; d) 24; e) 26; f) 28

Number of bits in Page Offset?

- a) 8; b) 10; c) 12; d) 14; e) 16; f) 18

Number of bits in Physical Page Number?

- a) 18; b) 20; c) 22; d) 24; e) 26; f) 28

Address Translation Exercise 1 (#2/2)

- 40- bit virtual address, 16 KB (2^{14} B)

Virtual Page Number (26 bits) Page Offset (14 bits)

- 36- bit virtual address, 16 KB (2^{14} B)

Physical Page Number (22 bits) Page Offset (14 bits)
Address Translation Exercise 2 (#1/2)
° Exercise:
  • 40-bit VA, 16 KB pages, 36-bit PA
  • 2-way set-assoc TLB: 256 "slots", 2 per slot

° Number of bits in TLB Index?
  a) 8; b) 10; c) 12; d) 14; e) 16; f) 18

° Number of bits in TLB Tag?
  a) 18; b) 20; c) 22; d) 24; e) 26; f) 28

° Approximate Number of bits in TLB Entry?
  a) 32; b) 36; c) 40; d) 42; e) 44; f) 46

Address Translation 2 (#2/2)
° 2-way set-associ data cache, 256 $2^8$ "slots",
  2 TLB entries per slot => 8 bit index

° Data Cache Entry: Valid bit, Dirty bit,
  Access Control (2-3 bits?),
  Virtual Page Number, Physical Page Number

° Number of bits in Cache Offset?
  a) 6; b) 8; c) 10; d) 12; e) 14; f) 16

° Number of bits in Cache Index?
  a) 6; b) 9; c) 10; d) 12; e) 14; f) 16

° Number of bits in Cache Tag?
  a) 18; b) 20; c) 21; d) 24; e) 26; f) 28

° Approximate No. of bits in Cache Entry?

Address Translation Exercise 3 (#1/2)
° Exercise:
  • 40-bit VA, 16 KB pages, 36-bit PA
  • 2-way set-associ TLB: 256 "slots", 2 per slot
  • 64 KB data cache, 64 Byte blocks, 2 way S.A.

° Number of bits in Cache Offset?
  a) 6; b) 8; c) 10; d) 12; e) 14; f) 16

° Number of bits in Cache Index?
  a) 6; b) 9; c) 10; d) 12; e) 14; f) 16

° Number of bits in Cache Tag?
  a) 18; b) 20; c) 21; d) 24; e) 26; f) 28

° Approximate No. of bits in Cache Entry?
The Principle of Locality:
- Program access a relatively small portion of the address space at any instant of time.
  - Temporal Locality: Locality in Time
  - Spatial Locality: Locality in Space

Caches, TLBs, Virtual Memory all understood by examining how they deal with 4 questions:
1) Where can block be placed?
2) How is block found?
3) What block is replaced on miss?
4) How are writes handled?

Virtual Memory allows protected sharing of memory between processes with less swapping to disk, less fragmentation than always swap or base/bound in segmentation.

3 Problems:
1) Not enough memory: Spatial Locality means small Working Set of pages OK
2) TLB to reduce performance cost of VM
3) Need more compact representation to reduce memory size cost of simple 1-level page table, especially for 64-bit address (See COMP3231)

Virtual memory was controversial at the time: can SW automatically manage 64KB across many programs?
- 1000X DRAM growth removed controversy

Today VM allows many processes to share single memory without having to swap all processes to disk; VM protection today is more important than memory hierarchy

Today CPU time is a function of (ops, cache misses) vs. just f(ops): What does this mean to Compilers, Data structures, Algorithms?