CS252 Graduate Computer Architecture
Fall 2015
Lecture 11: Memory

Krste Asanovic
krste@eecs.berkeley.edu
http://inst.eecs.berkeley.edu/~cs252/fa15
Last Time in Lecture 10

VLIW Machines
- Compiler-controlled static scheduling
- Loop unrolling
- Software pipelining
- Trace scheduling
- Rotating register file
- Predication
- Limits of static scheduling
Early Read-Only Memory Technologies

Punched cards, From early 1700s through Jaquard Loom, Babbage, and then IBM

Punched paper tape, instruction stream in Harvard Mk 1

Diode Matrix, EDSAC-2 µcode store

IBM Card Capacitor ROS

IBM Balanced Capacitor ROS
Early Read/Write Main Memory Technologies

Babbage, 1800s: Digits stored on mechanical wheels

Williams Tube, Manchester Mark 1, 1947

Mercury Delay Line, Univac 1, 1951

Also, regenerative capacitor memory on Atanasoff-Berry computer, and rotating magnetic drum memory on IBM 650
MIT Whirlwind Core Memory
Core Memory

- Core memory was first large scale reliable main memory
  - invented by Forrester in late 40s/early 50s at MIT for Whirlwind project
- Bits stored as magnetization polarity on small ferrite cores threaded onto two-dimensional grid of wires
- Coincident current pulses on X and Y wires would write cell and also sense original state (destructive reads)

- Robust, non-volatile storage
- Used on space shuttle computers
- Cores threaded onto wires by hand (25 billion a year at peak production)
- Core access time ~ 1µs

DEC PDP-8/E Board, 4K words x 12 bits, (1968)
Semiconductor Memory

- Semiconductor memory began to be competitive in early 1970s
  - Intel formed to exploit market for semiconductor memory
  - Early semiconductor memory was Static RAM (SRAM). SRAM cell internals similar to a latch (cross-coupled inverters).

- First commercial Dynamic RAM (DRAM) was Intel 1103
  - 1Kbit of storage on single chip
  - charge on a capacitor used to hold value
  - Value has to be regularly read and written back, hence dynamic

*Semiconductor memory quickly replaced core in ’70s*
One-Transistor Dynamic RAM
[Dennard, IBM]

1-T DRAM Cell

Storage capacitor (FET gate, trench, stack)

VREF

word

access transistor

bit

TiN top electrode (VREF)

Ta2O5 dielectric

poly word line

W bottom electrode

access transistor

Wordline
Modern DRAM Cell Structure

[Samsung, sub-70nm DRAM, 2004]
- Bits stored in 2-dimensional arrays on chip
- Modern chips have around 4-8 logical banks on each chip
  - each logical bank physically implemented as many smaller arrays
DRAM Physical Layout

Figure 1. Physical floorplan of a DRAM. A DRAM actually contains a very large number of small DRAMs called sub-arrays.

[Vogelsang, MICRO-2010]
DRAM Packaging
(Laptops/Desktops/Servers)

- DIMM (Dual Inline Memory Module) contains multiple chips with clock/control/address signals connected in parallel (sometimes need buffers to drive signals to all chips)
- Data pins work together to return wide word (e.g., 64-bit data bus using 16x4-bit parts)
DRAM Packaging, Mobile Devices

[ Apple A4 package on circuit board]

[ Apple A4 package cross-section, iFixit 2010 ]
DRAM Operation

- Three steps in read/write access to a given bank
  - Row access (RAS)
    - decode row address, enable addressed row (often multiple Kb in row)
    - bitlines share charge with storage cell
    - small change in voltage detected by sense amplifiers which latch whole row of bits
    - sense amplifiers drive bitlines full rail to recharge storage cells
  - Column access (CAS)
    - decode column address to select small number of sense amplifier latches (4, 8, 16, or 32 bits depending on DRAM package)
    - on read, send latched bits out to chip pins
    - on write, change sense amplifier latches which then charge storage cells to required value
    - can perform multiple column accesses on same row without another row access (burst mode)
  - Precharge
    - charges bit lines to known value, required before next row access

- Each step has a latency of around 15-20ns in modern DRAMs
- Various DRAM standards (DDR, RDRAM) have different ways of encoding the signals for transmission to the DRAM, but all share same core architecture
Memory Parameters

- Latency
  - Time from initiation to completion of one memory read (e.g., in nanoseconds, or in CPU or DRAM clock cycles)

- Occupancy
  - Time that a memory bank is busy with one request
  - Usually the important parameter for a memory write

- Bandwidth
  - Rate at which requests can be processed (accesses/sec, or GB/s)

- All can vary significantly for reads vs. writes, or address, or address history (e.g., open/close page on DRAM bank)
Processor-DRAM Gap (latency)

Four-issue 3GHz superscalar accessing 100ns DRAM could execute 1,200 instructions during time for one memory access!
Physical Size Affects Latency

- Signals have further to travel
- Fan out to more locations
Two predictable properties of memory references:

- **Temporal Locality**: If a location is referenced it is likely to be referenced again in the near future.

- **Spatial Locality**: If a location is referenced it is likely that locations near it will be referenced in the near future.
Memory Reference Patterns

Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192
Memory Hierarchy

- Small, fast memory near processor to buffer accesses to big, slow memory
  - Make combination look like a big, fast memory

- Keep recently accessed data in small fast memory closer to processor to exploit temporal locality
  - Cache replacement policy favors recently accessed data

- Fetch words around requested word to exploit spatial locality
  - Use multiword cache lines, and prefetching
Management of Memory Hierarchy

- **Small/fast storage, e.g., registers**
  - Address usually specified in instruction
  - Generally implemented directly as a register file
    - *but hardware might do things behind software’s back, e.g., stack management, register renaming*

- **Larger/slower storage, e.g., main memory**
  - Address usually computed from values in register
  - Generally implemented as a hardware-managed cache hierarchy (hardware decides what is kept in fast memory)
    - *but software may provide “hints”, e.g., don’t cache or prefetch*
Important Cache Parameters (Review)

- Capacity (in bytes)
- Associativity (from direct-mapped to fully associative)
- Line size (bytes sharing a tag)
- Write-back versus write-through
- Write-allocate versus write no-allocate
- Replacement policy (least recently used, random)
Improving Cache Performance

Average memory access time (AMAT) = Hit time + Miss rate x Miss penalty

To improve performance:
  • reduce the hit time
  • reduce the miss rate
  • reduce the miss penalty

What is best cache design for 5-stage pipeline?

Biggest cache that doesn’t increase hit time past 1 cycle (approx 8-32KB in modern technology)

[ design issues more complex with deeper pipelines and/or out-of-order superscalar processors]
Causes of Cache Misses: The 3 C’s

- **Compulsory**: first reference to a line (a.k.a. cold start misses)
  - misses that would occur even with infinite cache

- **Capacity**: cache is too small to hold all data needed by the program
  - misses that would occur even under perfect replacement policy

- **Conflict**: misses that occur because of collisions due to line-placement strategy
  - misses that would not occur with ideal full associativity
Effect of Cache Parameters on Performance

- Larger cache size
  + reduces capacity and conflict misses
  - hit time will increase

- Higher associativity
  + reduces conflict misses
  - may increase hit time

- Larger line size
  + reduces compulsory misses
  - increases conflict misses and miss penalty
**Multilevel Caches**

**Problem:** A memory cannot be large and fast

**Solution:** Increasing sizes of cache at each level

Local miss rate = misses in cache / accesses to cache

Global miss rate = misses in cache / CPU memory accesses

Misses per instruction = misses in cache / number of instructions
Presence of L2 influences L1 design

- Use smaller L1 if there is also L2
  - Trade increased L1 miss rate for reduced L1 hit time
  - Backup L2 reduces L1 miss penalty
  - Reduces average access energy

- Use simpler write-through L1 with on-chip L2
  - Write-back L2 cache absorbs write traffic, doesn’t go off-chip
  - At most one L1 miss request per L1 access (no dirty victim write back) simplifies pipeline control
  - Simplifies coherence issues
  - Simplifies error recovery in L1 (can use just parity bits in L1 and reload from L2 when parity error detected on L1 read)
Inclusion Policy

- Inclusive multilevel cache:
  - Inner cache can only hold lines also present in outer cache
  - External coherence snoop access need only check outer cache

- Exclusive multilevel caches:
  - Inner cache may hold lines not in outer cache
  - Swap lines between inner/outer caches on miss
  - Used in AMD Athlon with 64KB primary and 256KB secondary cache

- Why choose one type or the other?
Power 7 On-Chip Caches [IBM 2009]

- 32KB L1 I$/core
- 32KB L1 D$/core
- 3-cycle latency

- 256KB Unified L2$/core
- 8-cycle latency

- 32MB Unified Shared L3$
- Embedded DRAM (eDRAM)
- 25-cycle latency to local slice
Prefetching

- Speculate on future instruction and data accesses and fetch them into cache(s)
  - Instruction accesses easier to predict than data accesses
- Varieties of prefetching
  - Hardware prefetching
  - Software prefetching
  - Mixed schemes
- What types of misses does prefetching affect?
Issues in Prefetching

- Usefulness – should produce hits
- Timeliness – not late and not too early
- Cache and bandwidth pollution
Hardware Instruction Prefetching

- Instruction prefetch in Alpha AXP 21064
  - Fetch two lines on a miss; the requested line (i) and the next consecutive line (i+1)
  - Requested line placed in cache, and next line in instruction stream buffer
  - If miss in cache but hit in stream buffer, move stream buffer line into cache and prefetch next line (i+2)
Hardware Data Prefetching

- **Prefetch-on-miss:**
  - Prefetch \( b + 1 \) upon miss on \( b \)

- **One-Block Lookahead (OBL) scheme**
  - Initiate prefetch for block \( b + 1 \) when block \( b \) is accessed
  - Why is this different from doubling block size?
  - Can extend to N-block lookahead

- **Strided prefetch**
  - If observe sequence of accesses to line \( b, b+N, b+2N \), then prefetch \( b+3N \) etc.

- **Example:** IBM Power 5 [2003] supports eight independent streams of strided prefetch per processor, prefetching 12 lines ahead of current access
Software Prefetching

```c
for(i=0; i < N; i++) {
    prefetch( &a[i + 1] );
    prefetch( &b[i + 1] );
    SUM = SUM + a[i] * b[i];
}
```
Software Prefetching Issues

- Timing is the biggest issue, not predictability
  - If you prefetch very close to when the data is required, you might be too late
  - Prefetch too early, cause pollution
  - Estimate how long it will take for the data to come into L1, so we can set P appropriately
  - Why is this hard to do?

```c
for(i=0; i < N; i++) {
    prefetch( &a[i + P] );
    prefetch( &b[i + P] );
    SUM = SUM + a[i] * b[i];
}
```

*Must consider cost of prefetch instructions*
Compiler Optimizations

- Restructuring code affects the data access sequence
  - Group data accesses together to improve spatial locality
  - Re-order data accesses to improve temporal locality

- Prevent data from entering the cache
  - Useful for variables that will only be accessed once before being replaced
  - Needs mechanism for software to tell hardware not to cache data ("no-allocate" instruction hints or page table bits)

- Kill data that will never be used again
  - Streaming data exploits spatial locality but not temporal locality
  - Replace into dead cache locations
Acknowledgements

This course is partly inspired by previous MIT 6.823 and Berkeley CS252 computer architecture courses created by my collaborators and colleagues:

- Arvind (MIT)
- Joel Emer (Intel/MIT)
- James Hoe (CMU)
- John Kubiatowicz (UCB)
- David Patterson (UCB)