Memory Hierarchy
Reducing Hit Time
Main Memory
and
Examples

Soner Onder
Michigan Technological University
Randy Katz & David A. Patterson
University of California, Berkeley
Review: Reducing Misses

3 Cs: Compulsory, Capacity, Conflict

1. Reduce Misses via Larger Block Size
2. Reduce Misses via Higher Associativity
3. Reducing Misses via Victim Cache
4. Reducing Misses via Pseudo-Associativity
5. Reducing Misses by HW Prefetching Instr, Data
6. Reducing Misses by SW Prefetching Data
7. Reducing Misses by Compiler Optimizations

Remember danger of concentrating on just one parameter when evaluating performance
Reducing Miss Penalty Summary

Five techniques

- Read priority over write on miss
- Subblock placement
- Early Restart and Critical Word First on miss
- Non-blocking Caches (Hit under Miss, Miss under Miss)
- Second Level Cache

Can be applied recursively to Multilevel Caches

- Danger is that time to DRAM will grow with multiple levels in between
- First attempts at L2 caches can make things worse, since increased worst case is worse

Out-of-order CPU can hide L1 data cache miss (-3-5 clocks), but stall on L2 miss (-40-100 clocks)?

\[ \text{CPUtime} = IC \times \left( \text{CPI}_{\text{Execution}} + \frac{\text{Memory accesses}}{\text{Instruction}} \right) \times \text{Miss rate} \times \text{Miss penalty} \times \text{Clock cycle time} \]
Review: Improving Cache Performance

1. Reduce the miss rate,

2. Reduce the miss penalty, or

3. Reduce the time to hit in the cache.
1. Fast Hit times via Small and Simple Caches

Why Alpha 21164 has 8KB Instruction and 8KB data cache + 96KB second level cache?

- Small data cache and clock rate

**Direct Mapped, on chip**
2. Fast hits by Avoiding Address Translation

Send virtual address to cache? Called **Virtually Addressed Cache** or just **Virtual Cache** vs. **Physical Cache**

- Every time process is switched logically must flush the cache; otherwise get false hits
  - Cost is time to flush + “compulsory” misses from empty cache
- Dealing with *aliases* (sometimes called *synonyms*); Two different virtual addresses map to same physical address
- I/O must interact with cache, so need virtual address

**Solution to aliases**

- HW guarantees covers index field & direct mapped, they must be unique; called *page coloring*

**Solution to cache flush**

- Add *process identifier tag* that identifies process as well as address within process: can't get a hit if wrong process
Virtually Addressed Caches

Conventional Organization

Virtually Addressed Cache
Translate only on miss
Synonym Problem

Overlap $ access with VA translation: requires $ index to remain invariant across translation
2. Fast Cache Hits by Avoiding Translation: Process ID impact

Black is uniprocess

Light Gray is multiprocess when flush cache

Dark Gray is multiprocess when use Process ID tag

Y axis: Miss Rates up to 20%

X axis: Cache size from 2 KB to 1024 KB
2. Fast Cache Hits by Avoiding Translation: Index with Physical Portion of Address

If index is physical part of address, can start tag access in parallel with translation so that can compare to physical tag.

Limits cache to page size: what if want bigger caches and uses same trick?

- Higher associativity moves barrier to right
- Page coloring
3. Fast Hit Times Via Pipelined Writes

Pipeline Tag Check and Update Cache as separate stages; current write tag check & previous write cache update

Only STORES in the pipeline; empty during a miss

Store r2, (r1)
Add
Sub
Store r4, (r3)
r2&

Check r1
--
--
M[r1] <-
check r3

In shade is “Delayed Write Buffer”; must be checked on reads; either complete write or read from buffer
4. Fast Writes on Misses Via Small Subblocks

If most writes are 1 word, subblock size is 1 word, & write through then always write subblock & tag immediately

- **Tag match and valid bit already set**: Writing the block was proper, & nothing lost by setting valid bit on again.

- **Tag match and valid bit not set**: The tag match means that this is the proper block; writing the data into the subblock makes it appropriate to turn the valid bit on.

- **Tag mismatch**: This is a miss and will modify the data portion of the block. Since write-through cache, no harm was done; memory still has an up-to-date copy of the old value. Only the tag to the address of the write and the valid bits of the other subblock need be changed because the valid bit for this subblock has already been set.

Doesn’t work with write back due to last case
## Cache Optimization Summary

<table>
<thead>
<tr>
<th>Technique</th>
<th>MR</th>
<th>MP</th>
<th>HT</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Larger Block Size</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Higher Associativity</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>Victim Caches</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Pseudo-Associative Caches</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>HW Prefetching of Instr/Data</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Compiler Controlled Prefetching</td>
<td>+</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Compiler Reduce Misses</td>
<td>+</td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Priority to Read Misses</td>
<td>+</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Subblock Placement</td>
<td>+</td>
<td>+</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Early Restart &amp; Critical Word 1st</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Non-Blocking Caches</td>
<td>+</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Second Level Caches</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Small &amp; Simple Caches</td>
<td>-</td>
<td>+</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Avoiding Address Translation</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Pipelining Writes</td>
<td>+</td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
What is the Impact of What You've Learned About Caches?

1960-1985: Speed = \( f(\text{no. operations}) \)

1990

- Pipelined Execution & Fast Clock Rate
- Out-of-Order execution
- Superscalar Instruction Issue

1998: Speed = \( f(\text{non-cached memory accesses}) \)

What does this mean for

- Compilers?, Operating Systems?, Algorithms?, Data Structures?
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 64Kbit DRAM)  
Access time 750 ns, cycle time 1500-3000 ns
DRAM logical organization (4 Mbit)

Square root of bits per RAS/CAS
DRAM physical organization (4 Mbit)

Block 0

... Block 3
4 Key DRAM Timing Parameters

$t_{RAC}$: minimum time from RAS line falling to the valid data output.

- Quoted as the speed of a DRAM when buy
- A typical 4Mb DRAM $t_{RAC} = 60$ ns
- Speed of DRAM since on purchase sheet?

$t_{RC}$: minimum time from the start of one row access to the start of the next.

- $t_{RC} = 110$ ns for a 4Mbit DRAM with a $t_{RAC}$ of 60 ns

$t_{CAC}$: minimum time from CAS line falling to valid data output.

- 15 ns for a 4Mbit DRAM with a $t_{RAC}$ of 60 ns

$t_{PC}$: minimum time from the start of one column access to the start of the next.

- 35 ns for a 4Mbit DRAM with a $t_{RAC}$ of 60 ns
A 60 ns ($t_{RAC}$) DRAM can

- perform a row access only every 110 ns ($t_{RC}$)
- perform column access ($t_{CAC}$) in 15 ns, but time between column accesses is at least 35 ns ($t_{PC}$).
  - In practice, external address delays and turning around buses make it 40 to 50 ns

These times do not include the time to drive the addresses off the microprocessor nor the memory controller overhead!
DRAM History

DRAMs: capacity +60%/yr, cost -30%/yr

- 2.5X cells/area, 1.5X die size in -3 years

'98 DRAM fab line costs $2B

- DRAM only: density, leakage v. speed

Rely on increasing no. of computers & memory per computer (60% market)

- SIMM or DIMM is replaceable unit
  => computers use any generation DRAM

Commodity, second source industry
=> high volume, low profit, conservative

- Little organization innovation in 20 years

Order of importance: 1) Cost/bit 2) Capacity

- First RAMBUS: 10X BW, +30% cost => little impact
**DRAM Future: 1 Gbit DRAM (ISSCC '96; production '02?)**

<table>
<thead>
<tr>
<th></th>
<th>Mitsubishi</th>
<th>Samsung</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocks</td>
<td>512 x 2 Mbit</td>
<td>1024 x 1 Mbit</td>
</tr>
<tr>
<td>Clock</td>
<td>200 MHz</td>
<td>250 MHz</td>
</tr>
<tr>
<td>Data Pins</td>
<td>64</td>
<td>16</td>
</tr>
<tr>
<td>Die Size</td>
<td>24 x 24 mm</td>
<td>31 x 21 mm</td>
</tr>
<tr>
<td></td>
<td>Sizes will be much smaller in production</td>
<td></td>
</tr>
<tr>
<td>Metal Layers</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Technology</td>
<td>0.15 micron</td>
<td>0.16 micron</td>
</tr>
</tbody>
</table>

Wish could do this for Microprocessors!
Main Memory Performance

**Simple:**
- CPU, Cache, Bus, Memory same width (32 or 64 bits)

**Wide:**
- CPU/Mux 1 word; Mux/Cache, Bus, Memory N words (Alpha: 64 bits & 256 bits; UltraSPARC 512)

**Interleaved:**
- CPU, Cache, Bus 1 word: Memory N Modules (4 Modules); example is word interleaved
Main Memory Performance

Timing model (word size is 32 bits)

- 1 to send address,
- 6 access time, 1 to send data
- Cache Block is 4 words

Simple M.P. = 4 x (1+6+1) = 32
Wide M.P. = 1 + 6 + 1 = 8
Interleaved M.P. = 1 + 6 + 4x1 = 11
Independent Memory Banks

Memory banks for independent accesses vs. faster sequential accesses

- Multiprocessor
- I/O
- CPU with Hit under n Misses, Non-blocking Cache

**Superbank**: all memory active on one block transfer (or **Bank**)

**Bank**: portion within a superbank that is word interleaved (or **Subbank**)

![Diagram of Superbank and Bank structure]
Independent Memory Banks

How many banks?

- number banks \cdot \text{number clocks to access word in bank}

- For sequential accesses, otherwise will return to original bank before it has next word ready

- (like in vector case)

Increasing DRAM $\Rightarrow$ fewer chips $\Rightarrow$ harder to have banks
**DRAMs per PC over Time**

<table>
<thead>
<tr>
<th>Minimum Memory Size</th>
<th>4 MB</th>
<th>8 MB</th>
<th>16 MB</th>
<th>32 MB</th>
<th>64 MB</th>
<th>128 MB</th>
<th>256 MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>'86</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>'89</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>'92</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>'96</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>'99</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>'02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DRAM generation**

- '86: 1 Mb
- '89: 4 Mb
- '92: 16 Mb
- '96: 64 Mb
- '99: 256 Mb
- '02: 1 Gb
Fast Memory Systems: DRAM specific

Multiple CAS accesses: several names (page mode)

- Extended Data Out (EDO): 30% faster in page mode

New DRAMs to address gap; what will they cost, will they survive?

- RAMBUS: startup company; reinvent DRAM interface
  - Each Chip a module vs. slice of memory
  - Short bus between CPU and chips
  - Does own refresh
  - Variable amount of data returned
  - 1 byte / 2 ns (500 MB/s per chip)

- Synchronous DRAM: 2 banks on chip, a clock signal to DRAM, transfer synchronous to system clock (66 - 150 MHz)
  - Intel claims RAMBUS Direct (16 b wide) is future PC memory

Niche memory or main memory?

- e.g., Video RAM for frame buffers, DRAM + fast serial output
DRAM Latency >> BW

More App Bandwidth =>
Cache misses
=> DRAM RAS/CAS

Application BW =>
Lower DRAM **Latency**

RAMBUS, Synch DRAM
increase BW but **higher** latency

EDO DRAM < 5% in PC
Potential DRAM Crossroads?

After 20 years of 4X every 3 years, running into wall? (64Mb - 1 Gb)

How can keep $1B fab lines full if buy fewer DRAMs per computer?

Cost/bit -30%/yr if stop 4X/3 yr?

What will happen to $40B/yr DRAM industry?
Main Memory Summary

Wider Memory

Interleaved Memory: for sequential or independent accesses

Avoiding bank conflicts: SW & HW

DRAM specific optimizations: page mode & Specialty DRAM

DRAM future less rosy?
Cache Cross Cutting Issues

Superscalar CPU & Number Cache Ports must match: number memory accesses/cycle?

Speculative Execution and non-faulting option on memory/TLB

Parallel Execution vs. Cache locality
  - Want far separation to find independent operations vs. want reuse of data accesses to avoid misses

I/O and consistency
  - Caches => multiple copies of data
    - Consistency
Alpha 21064

Separate Instr & Data TLB & Caches

TLBs fully associative

TLB updates in SW ("Priv Arch Libr")

Caches 8KB direct mapped, write thru

Critical 8 bytes first

Prefetch instr. stream buffer

2 MB L2 cache, direct mapped, WB (off-chip)

256 bit path to main memory, 4 x 64-bit modules

Victim Buffer: to give read priority over write
Alpha 21264 Memory Hierarchy

- 48 Bit virtual address & 44 bit physical address or
- 44 bit virtual address & 41 bit physical address.

Physical address space is halved, lower half memory addresses and upper half I/O addresses.
Alpha 21264 Memory Hierarchy
Alpha 21264 Memory Hierarchy
Alpha 21264 Memory Hierarchy
Alpha 21264 Memory Hierarchy
ASN : Address space number.

Instruction cache interface

Store queue out

Data cache interface
Alpha 21264 Memory Hierarchy

- Virtually indexed and virtually tagged
  - 8 bit ASN
  - I TLB access only on a miss.

- Uses way prediction
  - A way predict is prepended to 9 bit index -> 10 bit index
  - The cache looks like a 64 KB cache with 1024 blocks.

- Instruction cache tag = 48-9-6 = 33
- 11 bits to predict the next group of 16 bytes, updated:
  - Address of next sequential group on a cache miss
  - Non-sequential address by a dynamic branch predictor.
  - Called “Line prediction”.
The index field of the PC is compared with the predicted block address;
The tag field is compared to the address from the tag portion of the cache;
8 bit asn to the asn field.
Valid bit is checked.
Any of the above is wrong: cache miss.
An instruction cache miss causes:
Check the instruction TLB
Instruction prefetcher.
Alpha 21264 Memory Hierarchy

- Data cache is virtually indexed and physically tagged:
  - 9-bit index + 3 bits to select the appropriate 8 bytes are sent to the data cache;
  - Page frame of the address is sent to the TLB.
- Data TLB fully associative 128 PTEs.
Alpha 21264 Memory Hierarchy
Alpha Memory Performance: Miss Rates of SPEC92 (21064)

I$ miss = 6%
D$ miss = 32%
L2 miss = 10%

I$ miss = 2%
D$ miss = 13%
L2 miss = 0.6%

I$ miss = 1%
D$ miss = 21%
L2 miss = 0.3%
Alpha CPI Components

Instruction stall: branch mispredict (green);

Data cache (blue); Instruction cache (yellow); L2$ (pink)
Other: compute + reg conflicts, structural conflicts
Pitfall: Predicting Cache Performance from Different Prog. (ISA, compiler, ...)

4KB Data cache miss rate 8%, 12%, or 28%?

1KB Instr cache miss rate 0%, 3%, or 10%?

Alpha vs. MIPS for 8KB Data $: 17% vs. 10%

Why 2X Alpha v. MIPS?

---

**Graph Data**
- **D$, Tom**: Red line
- **D$, gcc**: Green line
- **D$, esp**: Blue line
- **I$, gcc**: Yellow line
- **I$, esp**: Magenta line
- **I$, Tom**: Cyan line

**X-axis (Cache Size (KB))**
- 1
- 2
- 4
- 8
- 16
- 32
- 64
- 128

**Y-axis (Miss Rate)**
- 0%
- 5%
- 10%
- 15%
- 20%
- 25%
- 30%
- 35%
Main Memory Summary

Wider Memory

Interleaved Memory: for sequential or independent accesses

Avoiding bank conflicts: SW & HW

DRAM specific optimizations: page mode & Specialty DRAM

DRAM future less rosy?
Practical Memory Hierarchy

Issue is NOT inventing new mechanisms

Issue is taste in selecting between many alternatives in putting together a memory hierarchy that fit well together

- e.g., L1 Data cache write through, L2 Write back
- e.g., L1 small for fast hit time/clock cycle,
- e.g., L2 big enough to avoid going to DRAM?