Memory Hierarchy

Improving Cache Performance
Cache Performance (per instruction)

\[ t_{CPU} = (N_{CPU} + N_{MEM}) \times T \]

- \( t_{CPU} \): CPU time [sec]
- \( N_{CPU} \): CPU execution clock cycles [cycles]
- \( N_{MEM} \): Memory stall clock cycles [cycles]
- \( T \): Clock cycle time [sec]

\[ N_{MEM} = N_{R} \times MR_{R} \times P_{R} + N_{W} \times MR_{W} \times P_{W} \]

- \( N_{MEM} \): Memory stall clock cycles [cycles]
- \( N_{R/W} \): Number of Reads/Writes [#]
- \( MR_{R/W} \): Read/Write miss rate [fraction]
- \( P_{R/W} \): Read/Write miss penalty [cycles]

Combining read and writes

\[ N_{MEM} = N_{M} \times MR \times P \]

- \( N_{M} \): Memory accesses [#]
- \( MR \): Miss rate [fraction]
- \( P \): Miss penalty [cycles]
Cache Performance
(Entire program assuming one instruction at a time)

\[ t_{CPU} = IC \times (CPI_{\text{execution}} + MPI \times MR \times P) \times T \]

- **IC** = Instruction Count [\#]
- **t_{CPU}** = CPU time [sec]
- **MPI** = Memory access per Instruction [\#/instr]

\[ MPI = MPI \times MR \]

- **MPI** = Misses per instruction [\#/instruction]

\[ t_{CPU} = IC \times (CPI_{\text{execution}} + MPI \times P) \times T \]

- **CPI_{\text{execution}}** includes cache hit time (100% L1 hit rate)
Improving Cache Performance

\[ t_M = AMAT = t_H + MR \times P \]

AMAT = Average Memory-Access Time
\( t_H = \) Hit time

Improve performance* by:
1. Reducing the miss rate (MR)
2. Reducing the miss penalty (P)
3. Reducing the cache hit time \( t_H \)

*Part of the latency can be overlap with other parallel activities

Remark: be careful with computation of “savings” whenever latency can be masked (processor can continue work on other things)
Memory Hierarchy
Improving Cache Performance

- Reducing miss rate
- Reducing miss penalty
- Reducing hit time in the caches
Classifying Misses: 3 Cs

– **Compulsory**—The first access to a block is not in the cache, so the block must be brought into the cache. These are also called *cold start misses* or *first reference misses*. *(Misses in Infinite Cache)*

– **Capacity**—If the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur due to blocks being discarded and later retrieved. *(Misses in optimally-organized Finite Cache)*

– **Conflict**—If the block-placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory and capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. These are also called *collision misses* or *interference misses*. *(Misses in N-way Associative, Finite Cache)*
3Cs: Absolute Miss Rate
3Cs: Relative Miss Rate

The graph shows the relative miss rate for different cache sizes and cache capacities.

- **Capacity**: This layer represents the overall capacity of the cache, which decreases as the cache size increases.
- **Compulsory**: This layer represents the compulsory misses, which remain constant as the cache size increases.
- **Conflict**: This layer represents the conflict misses, which decrease as the cache size increases.

The graph is divided into three main regions:

- **1-way**: The least efficient, suffering from high miss rates at all cache sizes.
- **2-way**: Slightly more efficient than 1-way, with lower miss rates as cache size increases.
- **4-way**: More efficient than 2-way, with further reductions in miss rates as cache size increases.
- **8-way**: The most efficient, with the lowest miss rates across the board.

The horizontal axis represents the cache size in kilobytes (KB), while the vertical axis represents the relative miss rate from 0% to 100%.
Cache miss rate reduction techniques

- Increase block (line) size
- Increase level of set associativity
- Victim cache
- Way prediction
- Hardware prefetching
- Software prefetching
- Compiler techniques
- ...
Block Size Impact on Miss Rate

![Graph showing the impact of block size on miss rate. The x-axis represents block size (16, 32, 64, 128, 256 bytes) and the y-axis represents miss rate (0%, 5%, 10%, 15%, 20%, 25%). Each line corresponds to a different block size: 1K, 4K, 16K, 64K, and 256K. The graph illustrates how miss rate increases with block size for each of the block sizes shown.](image-url)
Associativity impact

Purpose: Reduce conflict misses
Concept: Provide multiple way set associative cache

• **2:1 Cache Rule**
  Miss Rate with a Direct Mapped cache of size $N$ [B] = —
  Miss Rate with 2-way Associative cache of size $N/2$ [B]

• **Impact on clock timing**
  hit time of 2-way vs. 1-way (Hill [1988]): —
  external cache +10% «
  internal + 2% for «

• In large caches, increase associativity may **increase**
  Average Memory Access Time ($AMAT=t_M$)
Example: Avg. Memory Access Time vs. Miss Rate

- Example: assume CCT = 1.10 for 2-way, 1.12 for 4-way, 1.14 for 8-way vs. CCT direct mapped

<table>
<thead>
<tr>
<th>Cache Size (KB)</th>
<th>1-way</th>
<th>2-way</th>
<th>4-way</th>
<th>8-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.33</td>
<td>2.15</td>
<td>2.07</td>
<td>2.01</td>
</tr>
<tr>
<td>2</td>
<td>1.98</td>
<td>1.86</td>
<td>1.76</td>
<td>1.68</td>
</tr>
<tr>
<td>4</td>
<td>1.72</td>
<td>1.67</td>
<td>1.61</td>
<td>1.53</td>
</tr>
<tr>
<td>8</td>
<td>1.46</td>
<td>1.48</td>
<td>1.47</td>
<td>1.43</td>
</tr>
<tr>
<td>16</td>
<td>1.29</td>
<td>1.32</td>
<td>1.32</td>
<td>1.32</td>
</tr>
<tr>
<td>32</td>
<td>1.20</td>
<td>1.24</td>
<td>1.25</td>
<td>1.27</td>
</tr>
<tr>
<td>64</td>
<td>1.14</td>
<td>1.20</td>
<td>1.21</td>
<td>1.23</td>
</tr>
<tr>
<td>128</td>
<td>1.10</td>
<td>1.17</td>
<td>1.18</td>
<td>1.20</td>
</tr>
</tbody>
</table>

(Red: A.M.A.T. not improved by higher associativity)
Victim Cache

- How to combine fast hit time of Direct Mapped yet still avoid conflict misses?
- Add buffer to place data discarded from cache
- Example: 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache (Jouppi [1990])

FIGURE 5.15 Placement of victim cache in the memory hierarchy.
Reducing Miss Rate (and hit time) via “Way” Prediction

- General idea: combine fast hit time of Direct Mapped and have the lower conflict misses of n-way SA
- The “way” is “guessed” based on prior accesses
- Multiplexer is set to read a block from the likely option (according to the guess)
- If the tag matches, we effectively have the performance of direct mapped.
- If not, there is a penalty.

Benefit:
- Can be faster –
- Less energy consumption –

- Example: Alpha 21264 instruction cache; MIPS R4300
Hardware Prefetching
(Instruction & Data)

Purpose: reduce miss rate/penalty
Concept: fetch ahead (before actually needed) of I and D

• E.g., Instruction Prefetching
  – Alpha 21064 fetches 2 blocks on a miss
  – Extra block placed in stream buffer
  – On miss, check stream buffer

• Works with Data blocks too:
  – 1 data stream buffer got 25% misses from 4KB cache; 4 streams got 43% Jouppi [1990]
  – Scientific programs: 8 streams got 50% to 70% of misses from two 64KB, 4-way set associative caches Palacharla & Kessler [1994]

• Disadvantage: uses memory bandwidth and power (like any prediction). May cause cache pollution
SW Data Prefetching

Purpose: reduce data miss rate/penalty
Concept: (SW) fetch ahead (before actually needed) Data

• Data Prefetch
  – Load data into register (careful intrusive!)
  – Load data into cache (via special instruction)
    MIPS IV, PowerPC, SPARC, IA32)
  – Special prefetching instructions cannot cause faults; a form of
    speculative execution

• Issuing Prefetch Instructions takes time
  – Is cost of prefetch issues < savings in reduced misses?
Compiler Optimizations

Purpose: improve performance
Concept: Compiler optimization techniques

- **Instructions**
  - Reorder procedures so as to reduce misses
  - Profiling to look at conflicts
  - McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache with 4 byte blocks

- **Data**
  - *Merging Arrays*: improve spatial locality by single array of compound elements instead of two arrays
  - *Loop Interchange*: change nesting of loops to access data in order stored in memory
  - *Loop Fusion*: Combine two independent loops that have same looping and some variables overlap
  - *Blocking*: Improve temporal locality by accessing “blocks” of data repeatedly vs. going down whole columns or rows
Merging Arrays: Example

/* Before */
int val[SIZE];
int key[SIZE];

/* After */
struct merge {
    int val;
    int key;
};
struct merge merged_array[SIZE];

Reducing conflicts between val & key
Loop Interchange: Example

/* Before */
for (k = 0; k < 100; k = k+1)
    for (j = 0; j < 100; j = j+1)
        for (i = 0; i < 5000; i = i+1)
            x[i][j] = 2 * x[i][j];

/* After */
for (k = 0; k < 100; k = k+1)
    for (i = 0; i < 5000; i = i+1)
        for (j = 0; j < 100; j = j+1)
            x[i][j] = 2 * x[i][j];

Sequential accesses Instead of striding through memory every 100 words
Loop Fusion Example (H&Pe2: p. 407)

/* Before */
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        d[i][j] = a[i][j] + c[i][j];

/* After */
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        { a[i][j] = 1/b[i][j] * c[i][j];
          d[i][j] = a[i][j] + c[i][j]; }

2 misses per access to a & c vs. one miss per access
Blocking Example (H&P2e: p. 408)

/* Before */
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        {r = 0;
         for (k = 0; k < N; k = k+1){
             r = r + y[i][k]*z[k][j];};
         x[i][j] = r;
        }

• Two Inner Loops:
  – Read all NxN elements of z[]
  – Read N elements of 1 row of y[] repeatedly
  – Write N elements of 1 row of x[]

• Capacity Misses a function of N & Cache Size:
  – 3 NxN => no capacity misses; otherwise ...

• Idea: compute on BxB submatrix that fits
Blocking Example

/* After */
for (jj = 0; jj < N; jj = jj+B)
for (kk = 0; kk < N; kk = kk+B)
for (i = 0; i < N; i = i+1)
  for (j = jj; j < min(jj+B-1,N); j = j+1)
    {r = 0;
     for (k = kk; k < min(kk+B-1,N); k = k+1) {
       r = r + y[i][k]*z[k][j];
     }
    x[i][j] = x[i][j] + r;
  };

• Capacity Misses from $2N^3 + N^2$ to $2N^3/B + N^2$
• B called \textit{Blocking Factor}
• Conflict Misses Too?
Reducing Conflict Misses by Blocking

- Conflict misses in caches not FA vs. Blocking size
  - Lam et al [1991] a blocking factor of 24 had a fifth the misses vs. 48 despite both fitting in cache
Summary of Compiler Optimizations to Reduce Cache Misses

- vpenta (nasa7)
- gmtty (nasa7)
- tomcatv
- btrix (nasa7)
- mxm (nasa7)
- spice cholesky (nasa7)
- compress

Performance Improvement

- merged arrays
- loop interchange
- loop fusion
- blocking
**Summary**

\[ t_{CPU} = IC \times (CPI_{\text{execution}} + MPI \times MR \times P) \times T \]

\[ CPU_{time} = IC \times \left( CPI_{\text{Execution}} + \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss rate} \times \text{Miss penalty} \right) \times \text{Clock cycle time} \]

3 Cs: Compulsory, Capacity, Conflict
- Reducing miss rate
- Reducing miss penalty
- Reducing hit time in the caches
Read Priority over Write

Purpose: Reduce read stalls (miss penalty)
Concept: Writes do not stall the CPU, Reads DO

• Give Read priority on Writes

• Memory writes ➔ into Write Buffer

• Write Buffer may cause RAW conflicts with main memory reads on cache misses
  – Wait for write buffer to empty will increase read miss penalty
  – Check write buffer contents before read; no conflicts ➔ access memory

• Example: Write Back (read miss)
  – Normal procedure:
    » Write dirty block to memory
    » Read from memory
  – Improved procedure
    » Copy dirty block to Write buffer
    » Check if read address is in Write Buffer
    » If there, Read from write buffer
    » If not in Write Buffer, read from memory
    » Write from Write Buffer to memory
Subblock Placement to Reduce Miss Penalty

Purpose: Reduce block transfer penalty
Concept: On miss, load only the requested data

- Don’t have to load full block on a miss
- Have bits per subblock to indicate valid
- Advantage over small block size: less tag storage

![Diagram of subblock placement](image)
Early Restart and Critical Word First

Purpose: Reduce read latency
Concept: Prioritize the data – more important data first

- Don’t wait for full block to be loaded before restarting CPU
  - *Early restart*—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
  - *Critical Word First*—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block. Also called *wrapped fetch* and *requested word first*

- Generally useful only in large blocks,
- Actual benefit is small since first word in block is likely to be requested first anyhow
Non-blocking Caches

Purpose: Reduce penalty of read after miss
Concept: Enable read to bypass previous read miss

• **Non-blocking cache** or **lockup-free cache** allowing the data cache to continue to supply cache hits during a miss
• “**Hit under miss**” reduces the effective miss penalty by being helpful during a miss instead of ignoring the requests of the CPU
• “**Hit under multiple miss**” or “**miss under miss**” may further lower the effective miss penalty by overlapping multiple misses
  – Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses.
FP programs on average: AMAT= 0.68 -> 0.52 -> 0.34 -> 0.26
Int programs on average: AMAT= 0.24 -> 0.20 -> 0.19 -> 0.19
8 KB Data Cache, Direct Mapped, 32B block, 16 cycle miss
Value of Hit Under Miss for SPEC

FIGURE 5.22 Ratio of the average memory stall time for a blocking cache to hit-under-miss schemes as the number of outstanding misses is varied for 18 SPEC92 programs.
Multilevel Caches
(Example: 2nd Level Cache)

• L1+L2:
  \[ AMAT = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1} \]

  \[ \text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2} \]

  \[ AMAT = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times (\text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2}) \]

• Definitions:
  – Local miss rate—misses in this cache divided by the total number of memory accesses to this cache (Miss rate_{L2})
  – Global miss rate—misses in this cache divided by the total number of memory accesses generated by the CPU (Miss Rate_{L1} \times Miss Rate_{L2})
Multilevel Cache – L1 Design Considerations

• Tied directly to CPU clock cycle: must be small and fast

• This is changing and is not always as critical:
  – Instructions:
    » prefetch units pull instructions into buffers
    » large decoding units (decoding required especially for Intel X86 Instruction set, e.g., PC processors)
  – Data
    » Read: try to use registers + buffers
    » Write: latency is not so critical (use write buffers etc.)
    » Therefore: growing emphasis on throughput (pipelining + fast clock) rather than only on latency
Multilevel Cache – L2 Design Considerations

• Latency is less critical (sees only L1 misses)

• Can be optimized differently:
  – Larger
  – Higher degree of associativity
  – Unified Instruction + Data for better space utilization
  – Main purpose: prevent need to go to main memory

• Can be shared among cores in the case of multi-core machines (chip multi-processors: CMP)
Multilevel Cache – Joint Design Considerations

- Get speed from small, direct-mapped L1 cache

- Get low probability of having to go to main memory from large, hit-rate optimized L2 cache (e.g., higher associativity)

- Inclusive vs. Exclusive: exclusive iff L2 is not much larger than L1

- If inclusive: use L2 tag checking mechanism, which is underutilized, for cache coherence protocols in shared memory multi-processor machines.
Memory Hierarchy

- Reducing miss rate
- Reducing miss penalty
- Reducing hit time in caches
“Way” Prediction

Purpose: Improve hit time
Concept: Predict the “way” (in N way set associative cache) based on History

• Multiplexer is set to read the predicted block
• Hit time is close to a direct map cache when tags match
• No tag match → penalty
• Benefits: Faster, less energy consumption

Example: Alpha 21264 instruction cache; MIPS R4300, Pentium Duo 2
Small and Simple Caches

- Alpha 21164 has 8KB Instruction and 8KB data cache + 96KB second level cache
- Direct Mapped, on chip
Reducing hit time in caches
Small and simple L1

Purpose: Improve hit time
Concept: Small/simple L1

• Harvard architecture
  Split data and Instruction cache → smaller, parallel accesses)
  
• Relatively small cache

• Physically close to execution units

Pentium Core 2 Duo has 32KB+32KB I and D caches (8 ways) + 2-6MB L2 cache
Reducing hit time in caches

Avoid Address Translation

**Purpose:** Improve hit time

**Concept:** L1 accessed via non translated address bits

- Physically addressed cache requires virtual → physical translation
- In L1, avoid virtual → physical slow down
  - Index and offset bits points to within a page (no need for translations)
- Use parallel translation with cache access

The concept of virtual memory will be introduced in the next chapter where we will extensively discuss this problem.
Pipelined Writes

Purpose: Improve write hit time

Concept: Pipelining write activities

• **Writes should be pipelined:**
  – Tag check and cache update = separate pipeline stages

• **Delayed Write Buffer; must be checked on reads**
**Fast Writes on Misses Via Small Subblocks**

- If most writes are 1 word, subblock size is 1 word, Direct Mapped, & 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. As this is a write-through cache, however, 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
Trace Cache
(instruction*)

Purpose: Enable feeding wide machines
Concept: cache holds the dynamic flow of executed instructions

- Enable feed of wide machine [Peleg, Weiser 1995]

- Cache a sequence of recently accessed instructions, including predicted branch path
- Cache a sequence of recently accessed data, including non-contiguous addresses (!)
- Non efficient use of cache space, more complicated to manage

Example: Intel NetBurst architecture (used in Pentium 4)
## 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>“Way” prediction and pseudoassoc.</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Trace 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>Other Compiler techniques</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>