Cache memory within memory hierarchy

- CPU: Registers, 500 bytes, 250 ps
- Cache: 64 KB, 1 ns
- Memory: 1 GB, 100 ns
- I/O bus: 1 TB, 10 ms

© 2007 Elsevier, Inc. All rights reserved.

Computer Architecture - 2014
Average access time to memory

- **Level 1**
  - Hit time + Miss rate x Miss Penalty

- **Level 2**
  - Hit time\(_{L1}\) + Miss rate\(_{L1}\) x (Hit time\(_{L2}\) + Miss rate\(_{L2}\) x Miss penalty\(_{L2}\) )

- ...

*Computer Architecture - 2014*
Basic optimizations

- Larger block size.
- Larger cache size.
- Higher associativity.
- Multilevel caches.
- Prioritize read misses.
- Avoid address translation during indexing.
Metrics to be reduced:
- Hit time.
- Miss rate.
- Miss penalty.
- Cache bandwidth.

All advanced optimizations try to improve some of this metrics.
Advanced optimizations

- Small and simple caches.
- Way prediction.
- Pipelined access to caches.
- Non-blocking caches.
- Multi-bank caches.
- Critical word first and early restart.
- Write buffer merge.
- Compiler optimizations.
- Data and instruction hardware prefetching.
- Compiler controlled prefetching.
1: Small and simple caches

- **Lookup:**
  - Line selection using **index**.
  - Read **tag** from line.
  - Compare **tag** and **address** (tag field).

- **Lookup time increases with cache size.**
  - Lookup hardware simpler with smaller cache.
  - Cache fits into processor chip.

- **A small cache** improves lookup time.
Access time and cache size

![Graph showing access time for different cache sizes and associativities.](image-url)
Cache simplification.

- Using mapping mechanisms as simple as possible.
- Direct mapping:
  - Allows to parallelize tag comparison and data transmission.

- Most modern processors more focused in using small caches than in simplifying them.
- **L1 cache (1 per core)**
  - 32 KB instructions
  - 32 KB data
  - Latency: 3 cycles
  - Associative 4(i), 8(d) ways.

- **L2 cache (1 per core)**
  - 256 KB
  - Latency: 9 cycles
  - Associative 8 ways.

- **L3 cache (shared)**
  - 8 MB
  - Latency: 39 cycles
  - Associative 16 ways.
Problem:
- Direct mapping → fast but many misses.
- Set associative → less misses but slower.

Way prediction:
- Store additional bits to predict way to be selected in next access.
- Block prefetching and compare to single tag.
  - On miss compare to other tags.
□ **Goal:** Increase cache bandwidth.

□ **Solution:** Pipeline cache access in several clock cycle.

□ **Effects:**
- Clock cycle can be shortened.
- An access may be initiated in every clock cycle.
- Cache bandwidth increases.
- Latency increases.

□ Positive effect in **superscalar processors**.
Problem: Cache miss leads to stall until block is obtained.

Solution: Out of order execution.
- How do we access to cache while miss is solved?

Hit during miss.
- Allow access with hit while waiting.
- Reduces miss penalty.

Hit during several misses / Miss during miss.
- Allow overlapped misses.
- Needs multi-channel memory.
- Highly complex.
Stall rate versus maximum number of misses

Percentage of the average memory stall time

- Hit under 1 miss
- Hit under 2 misses
- Hit under 64 misses

Benchmarks:
- swm256
- tomcatv
- fpopp
- su2cor
- hydro2d
- mdjisp2
- nasa7
- ducouc
- wave5
- ear
- mdijdp2
- alvinn
- spice2g6
- ora
- xisp
- espresso
- compress
- eqntott

© 2007 Elsevier, Inc. All rights reserved.
5: Multi-bank caches

- **Goal**: Allow simultaneous accesses to different cache locations.

- **Solution**: Divide memory into independent banks.

- **Effect**: Increases bandwidth.

- **Example**: Sun Niagara
  - L2: 4 banks
To **improve bandwidth** it is necessary that accesses are **distributed across banks**.

- **Simple approach**: Sequential interleaving.
  - Round-robin of blocks across banks.
□ **Observation:** Processor usually needs a single word to proceed.

□ **Solution:** Do not wait to get the whole block from memory.

□ **Alternatives:**
  - **Critical word first:** Block received ordered by word needed by processor first.
  - **Early restart:** Block received without reordering.
    - As soon as word of interest is received, CPU proceeds.

□ **Effects:** Depending on block size → **Larger is better.**
7: Write buffer merge

- Write buffer allows to decrease miss penalty.
  - When data written to buffer processor considers write done.
  - Simultaneous writes on memory more efficient than single write.

- **Uses:**
  - **Write-through:** In every write.
  - **Write-back:** When block is replaced
When buffer contains modified blocks, check addresses and overwrite if possible.

**Effects:**
- Reduces number of memory writes.
- Reduces stalls due to full buffer.
### 7: Write buffer merge

<table>
<thead>
<tr>
<th>Write address</th>
<th>V</th>
<th>V</th>
<th>V</th>
<th>V</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>Mem[100]</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>108</td>
<td>1</td>
<td>Mem[108]</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>116</td>
<td>1</td>
<td>Mem[116]</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>124</td>
<td>1</td>
<td>Mem[124]</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Write address</th>
<th>V</th>
<th>V</th>
<th>V</th>
<th>V</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
**Goal**: Generate code leading to less instruction and data misses.

**Instructions:**
- Procedure reordering.
- Align code blocks to cache line start.
- Linearization.

**Data:**
- Array merge.
- Loop exchange.
- Loop merge.
- Block access.
8.1: Procedure reordering

- **Goal:**
  - Reduce conflict misses due to two procedures overlapping in time and mapped to the same cache line.

- **Technique:**
  - Reorder procedures in memory.
8.2: Basic block alignment

- **Definition:** A basic block is a set of instructions sequentially executed (no branches contained).

- **Goal:** Reduce cache misses possibility for sequential code.

- **Technique:** Align first instructions in basic block to first word in cache line.
8.3: Linearization

- **Goal:** Reduce cache misses due to conditional branching.

- **Technique:** If compiler knows it is likely to take a branch, it can invert the condition and interchange both alternatives basic blocks.
  - Most likely basic block likely not to cause a miss.
8.4: Array merging

vector<int> key{max};
vector<int> value{max};
...
for (int i=0;i<max;++i) {
    cout << key[i] << " , "
    << value[i] << endl;
}

struct entry {
    int key;
    int value;
};
vector<entry> v{max};
...
for (auto & e : v) {
    cout << e.key << " , "
    << e.value << endl;
}
8.5 Loop interchange

```
for (j=0; j<100; j++) {
    for (i=0;i<5000; i++) {
        v[i][j] = 2 * v[i][j];
    }
}
```

Striped accesses

```
for (i=0; i<5000; j++) {
    for (j=0;i<100; i++) {
        v[i][j] = 2 * v[i][j];
    }
}
```

Sequential accesses

Better spatial locality
for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
        a[i][j] = b[i][j] * c[i][j];
    }
}

for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
        d[i][j] = a[i][j] + c[i][j];
    }
}

Improves temporal locality

Beware: It could reduce spatial locality in some cases.
8.7: Blocked access

\[
\begin{align*}
\text{for (i=0; i<N; i++) { for (bj = 0; bj<N; bj+=B) { for (bk=0; bk<N; bk+=B) { for (i=0; i<N; i++) { for (j=bj; j<min(bj+B, N); j++) { for (k=bk; k<min(bk+B, N); k++) { } } } } } } } \\
\quad r=0; \\
\quad r+= y[i][k] * z[k][j]; \\
\quad x[i][j] = r; \\
\end{align*}
\]

\[B = \text{Blocking factor}\]
N=6, i=1
9: Data and instruction hardware prefetching

Instruction prefetching

- Instruction high spatial locality.
- Read **two blocks** on miss.
  - Block causing miss.
  - Next block.
- Location:
  - Block causing miss → Instruction cache.
  - Next block → Instruction buffer.

Data prefetching

- **Example**: Pentium 4.
- Allows 4KB prefetching to L2 cache.
- Prefetching invoked if:
  - 2 misses on L2 due to same page.
  - Distance between misses less than 256 bytes.
Prefetching speedup

Computer Architecture - 2014
Compiler generates prefetching instructions.

- **Register prefetch** $\rightarrow$ Load value into register.
- **Cache prefetch** $\rightarrow$ Load block into cache.

**Types:**

- **Faulting** $\rightarrow$ Can generate exception in VM.
- **Non-faulting** $\rightarrow$ Cannot generate exception in VM.

## Optimizations Summary

<table>
<thead>
<tr>
<th>Technique</th>
<th>Hit Time</th>
<th>Band-width</th>
<th>Miss penalty</th>
<th>Miss rate</th>
<th>HW cost/complexity</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small and simple caches</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>Trivial; widely used</td>
</tr>
<tr>
<td>Way-predicting caches</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Used in Pentium 4</td>
</tr>
<tr>
<td>Pipelined cache access</td>
<td>-</td>
<td>+</td>
<td></td>
<td></td>
<td>1</td>
<td>Widely used</td>
</tr>
<tr>
<td>Nonblocking caches</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td>3</td>
<td>Widely used</td>
</tr>
<tr>
<td>Banked caches</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Used in L2 of i7 and Cortex A8</td>
</tr>
<tr>
<td>Critical word first and early restart</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>Widely used</td>
</tr>
<tr>
<td>Merging write buffer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Widely used with write through</td>
</tr>
<tr>
<td>Compiler techniques to reduce cache misses</td>
<td></td>
<td></td>
<td>+</td>
<td></td>
<td>0</td>
<td>Software is a challenge; some computers have compiler option</td>
</tr>
<tr>
<td>Hardware prefetching of instructions and data</td>
<td></td>
<td>+</td>
<td></td>
<td></td>
<td>2 instr., 3 data</td>
<td>Many prefetch instructions; AMD Opteron prefetches data</td>
</tr>
<tr>
<td>Compiler-controlled prefetching</td>
<td></td>
<td>+</td>
<td></td>
<td></td>
<td>3</td>
<td>Needs nonblocking cache; in many CPUs</td>
</tr>
</tbody>
</table>

**Computer Architecture - 2014**
Reference

  Hennessy y Patterson.
  **Sections:** 2.1, 2.2

- **Exercises:** 2.1, 2.2, 2.3, 2.8, 2.9, 2.10, 2.11, 2.12