Chapter 2
Memory Hierarchy Design
Some Advanced Optimizations of Caches

1. *Reducing the hit time*—Small and simple first-level caches and way-prediction. Both techniques also generally decrease power consumption.

2. *Increasing cache bandwidth*—Pipelined caches, multibanked caches, and non-blocking caches. These techniques have varying impacts on power consumption.

3. *Reducing the miss penalty*—Critical word first and merging write buffers. These optimizations have little impact on power.

4. *Reducing the miss rate*—Compiler optimizations. Obviously any improvement at compile time improves power consumption.

5. *Reducing the miss penalty or miss rate via parallelism*—Hardware prefetching and compiler prefetching. These optimizations generally increase power consumption, primarily because of prefetched data that are unused.
Small, Simple Caches
Reduces hit time, power

Figure 2.8 Relative access times generally increase as cache size and associativity are increased. These data come from the CACTI model 6.5 by Tarjan et al. (2005).

Figure 2.9 Energy consumption per read increases as cache size and associativity are increased. As in the previous figure, CACTI is used for the modeling with the same
Pipelined Access, Multibank Caches
Improves Bandwidth

Figure 2.10 Four-way interleaved cache banks using block addressing. Assuming 64 bytes per block, each of these addresses would be multiplied by 64 to get byte addressing.
Non-Blocking Caches
Increases Cache Bandwidth

“Hit under Miss” - cache can supply hits while waiting for a miss

Figure 2.11 The effectiveness of a nonblocking cache is evaluated by allowing 1, 2, or 64 hits under a cache miss with 9 SPECINT (on the left) and 9 SPECFP (on the right) benchmarks. The data memory system modeled after the Intel i7 consists of a 32 KiB L1
Critical Word First, Early Restart

Reduces Effective Miss penalty

- *Critical word first*—Request the missed word first from memory and send it to the processor as soon as it arrives; let the processor continue execution while filling the rest of the words in the block.

- *Early restart*—Fetch the words in normal order, but as soon as the requested word of the block arrives, send it to the processor and let the processor continue execution.

- Most effective for caches with large block sizes

- Effective due to spacial locality
Merging Write Buffers
Reduces Miss Penalty

Figure 2.12 In this illustration of write merging, the write buffer on top does not use write merging while the write buffer on the bottom does. The four writes are merged.
Compiler Optimizations
Reduces miss rate, cache activity in general

• Loop Interchange
  
  • Arrange code so subsequent loops access data consecutively

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

Reduces miss rate, cache activity in general

- Blocking

Figure 2.13 A snapshot of the three arrays $x$, $y$, and $z$ when $N=6$ and $i=1$. The age of accesses to the array
Code utilizing blocking

/* 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;
        };

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


Hardware Prefetching

Reduces Miss Penalty and/or Miss Rate

Figure 2.15: Speedup because of hardware prefetching on Intel Pentium 4 with hardware prefetching turned on for 2 of 12 SPECint2000 benchmarks and 9 of 14 SPECfp2000 benchmarks. Only the programs that benefit most from prefetching are shown; prefetching speeds up the missing 15 SPEC CPU benchmarks by less than 15% (Boggs et al., 2004).
Compiler-Controlled Prefetching

Architecture contains “Prefetch” instructions that can be utilized by the compiler to cause soon-to-be-accessed data values to be fetched in advance

- *Register prefetch* loads the value into a register.
- *Cache prefetch* loads data only into the cache and not the register.
<table>
<thead>
<tr>
<th>Technique</th>
<th>Hit time</th>
<th>Bandwidth</th>
<th>Miss penalty</th>
<th>Miss rate</th>
<th>Power consumption</th>
<th>Hardware 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></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></td>
<td>Used in Pentium 4</td>
</tr>
<tr>
<td>Pipelined &amp; banked caches</td>
<td>–</td>
<td></td>
<td>+</td>
<td></td>
<td>1</td>
<td></td>
<td>Widely used</td>
</tr>
<tr>
<td>Nonblocking caches</td>
<td>+</td>
<td></td>
<td>+</td>
<td></td>
<td>3</td>
<td></td>
<td>Widely used</td>
</tr>
<tr>
<td>Critical word first and early restart</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td>Widely used</td>
</tr>
<tr>
<td>Merging write buffer</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></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></td>
<td>Software is a challenge, but many compilers handle common linear algebra calculations</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></td>
<td>Most provide prefetch instructions; modern high-end processors also automatically prefetch in hardware</td>
</tr>
<tr>
<td>Compiler-controlled prefetching</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td>Needs nonblocking cache; possible instruction overhead; in many CPUs</td>
</tr>
<tr>
<td>HBM as additional level of cache</td>
<td>±</td>
<td>–</td>
<td></td>
<td>+</td>
<td>3</td>
<td></td>
<td>Depends on new packaging technology. Effects depend heavily on hit rate improvements</td>
</tr>
</tbody>
</table>

Figure 2.18 Summary of 10 advanced cache optimizations showing impact on cache performance, power consumption, and complexity. Although generally a technique helps only one factor, prefetching can reduce misses if
Virtual Machines

Motivation

- the increasing importance of isolation and security in modern systems;
- the failures in security and reliability of standard operating systems;
- the sharing of a single computer among many unrelated users, such as in a data center or cloud; and
- the dramatic increases in the raw speed of processors, which make the overhead of VMs more acceptable.
Virtual Machine Requirements

- Guest software should behave on a VM exactly as if it were running on the native hardware, except for performance-related behavior or limitations of fixed resources shared by multiple VMs.
- Guest software should not be able to directly change allocation of real system resources.

Requirements

- At least two processor modes, system and user.
- A privileged subset of instructions that is available only in system mode, resulting in a trap if executed in user mode. All system resources must be controllable only via these instructions.