Chapter 2 (and Appendix B)

Memory Hierarchy Design Cont.
Cache Optimization Metrics

- Reduce hit time
- Increase cache bandwidth
- Reduce miss penalty
- Reduce miss rate
Ten Advanced Optimizations

- Small and simple 1\textsuperscript{st}-level caches (hit time)
- Way Prediction (hit time)
- Pipelining Cache (bandwidth)
- Nonblocking Caches (bandwidth)
- Multibanked Caches (bandwidth)
- Critical Word First, Early Restart (miss penalty)
- Merging Write Buffer (miss penalty)
- Compiler Optimizations* (miss rate)
- Hardware Prefetching (miss penalty, miss rate)
- Compiler Prefetching (miss penalty, miss rate)

* Indicates topic of interest
Small and simple 1st-level caches

- Critical timing path:
  - addressing tag memory, then
  - comparing tags, then
  - selecting correct set
- Direct-mapped caches can overlap tag compare and transmission of data
- Lower associativity reduces power because fewer cache lines are accessed
L1 Size and Associativity

Access time vs. size and associativity
L1 Size and Associativity

Energy per read vs. size and associativity
Way Prediction

- To improve hit time, predict the way
  - Similar to branch prediction (chapter 3)
  - Correct predictions have reduced hit times, but misprediction gives longer hit time
  - Prediction accuracy
    - > 90% for two-way
    - > 80% for four-way
    - I-cache has better accuracy than D-cache
  - First used on MIPS R10000 in mid-90s
  - Used on ARM Cortex-A8

- Extend to access only the predicted block
  - “Way selection”
  - Lower power, but increases misprediction penalty
Pipelining Cache

- Pipeline cache access to improve bandwidth
  - Examples:
    - Pentium: 1 cycle
    - Pentium Pro – Pentium III: 2 cycles
    - Pentium 4 – Core i7: 4 cycles
  - Similar to processor pipelining (chapter 3)
  - Access requires several cycles but cycles can be faster
  - When pipeline is full, one request completes per cycle, improving bandwidth
  - Increases branch misprediction penalty
Nonblocking Caches

- Allow hits before previous misses complete
  - “Hit under miss”
  - “Hit under multiple miss”
- L2 must support this
- In general, processors can hide L1 miss penalty but not L2 miss penalty
Multibanked Caches

- Organize cache as independent banks to support simultaneous access
  - ARM Cortex-A8 supports 1-4 banks for L2
  - Intel i7 supports 4 banks for L1 and 8 banks for L2
- Interleave banks according to block address

![Diagram of block addressing](image)

**Figure 2.6** 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.
Critical Word First, Early Restart

- Critical word first
  - Request missed word from memory first
  - Send it to the processor as soon as it arrives

- Early restart
  - Request words in normal order
  - Send missed word to the processor as soon as it arrives

- Effectiveness of these strategies depends on block size and likelihood of another access to the portion of the block that has not yet been fetched
Merging Write Buffer

- When storing to a block that is already pending in the write buffer, update write buffer
- Reduces stalls due to full write buffer

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

No buffer merging

<table>
<thead>
<tr>
<th>Write address</th>
<th>V</th>
<th>V</th>
<th>V</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Buffer merging
Compiler Optimizations

- Merging arrays
  - Make data more sequential
- Loop Fusion
  - Combine loops
- Loop Interchange
  - Swap nested loops to access memory in sequential order
- Blocking
  - Instead of accessing entire rows or columns, subdivide matrices into blocks
  - Requires more memory accesses but improves locality of accesses
Compiler Optimizations

- Merging arrays

/* Before: 2 sequential arrays */
int val[SIZE];
int key[SIZE];

/* After: 1 array of structures */
struct merge {
    int val;
    int key;
};
struct merge merged_array[SIZE];
Compiler Optimizations

- **Loop fusion**

/* Before */

```c
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];
        }
```

Copyright © 2012, Elsevier Inc. All rights reserved.
Compiler Optimizations

- Loop interchange
  - Rearrange to maximize sequential access
  - Assume row-major matrices

/* 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];
Compiler Optimizations

- Blocking (before)
  - Remember matrix multiplication?
  - Calculating each entry in $x$ requires requiring reading a row of $y$ and a column of $z$
  - Entries across the entire matrices in memory!

```c
/* 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;
  }
```
Compiler Optimizations

- Blocking (before)
Compiler Optimizations

- Blocking (after)
  - Divide matrices into blocks (of size $B$)
  - Read only elements within the block, then move on to next block
    - Blocks in $x$, $y$, and $z$ are not always coincident

```c
/* 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;
      }
```

Copyright © 2012, Elsevier Inc. All rights reserved.
Compiler Optimizations

- Blocking (after)
Hardware Prefetching

- Fetch two blocks on miss (include next sequential block)
Compiler Prefetching

- Insert prefetch instructions before data is needed
  - Register prefetch
    - Loads data into register
  - Cache prefetch
    - Loads data into cache

- Combine with loop unrolling and software pipelining (chapter 3)
### Summary

<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>Trivial; widely used</td>
<td></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>
<td></td>
</tr>
<tr>
<td>Pipelined cache access</td>
<td></td>
<td>+</td>
<td></td>
<td></td>
<td>1</td>
<td>Widely used</td>
<td></td>
</tr>
<tr>
<td>Nonblocking caches</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>3</td>
<td>Widely used</td>
<td></td>
</tr>
<tr>
<td>Banked caches</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td></td>
<td>1</td>
<td>Used in L2 of both i7 and Cortex-A8</td>
<td></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>
<td></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>
<td></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, but many compilers handle common linear algebra calculations</td>
<td></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>Most provide prefetch instructions; modern high-end processors also automatically prefetch in hardware.</td>
<td></td>
</tr>
<tr>
<td>Compiler controlled prefetching</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td>3</td>
<td>Needs nonblocking cache; possible instruction overhead; in many CPUs</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.11 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 done sufficiently early; if not, it can reduce miss penalty. + means that the technique improves the factor; − means it hurts that factor, and blank means it has no impact. The complexity measure is subjective, with 0 being the easiest and 3 being a challenge.
Core i7 Cache Hierarchy

- L1
  - 32KB I/32KB D, 4-way I/8-way D, 4-cycle access time, pipelined, Pseudo LRU

- L2
  - 256KB, 8-way, 10-cycle access time, Pseudo LRU

- L3 (shared)
  - 2MB per core, 16-way, 35-cycle access time, Pseudo LRU
Memory Technology

- Performance metrics
  - Latency is concern of cache
  - Bandwidth is concern of multiprocessors and I/O
  - Access time
    - Time between read request and when desired word arrives
  - Cycle time
    - Minimum time between unrelated requests to memory

- DRAM used for main memory, SRAM used for cache
Memory Technology

- **SRAM**
  - Requires low power to retain bit
  - Requires 6 transistors/bit

- **DRAM**
  - Must be re-written after being read
  - Must also be periodically refreshed
    - Every ~ 8 ms
    - Each row can be refreshed simultaneously
  - One transistor/bit
  - Address lines are multiplexed:
    - Upper half of address: row access strobe (RAS)
    - Lower half of address: column access strobe (CAS)
Memory Technology

- Amdahl:
  - Memory capacity should grow linearly with processor speed
  - Unfortunately, memory capacity and speed has not kept pace with processors

- Some optimizations:
  - Multiple accesses to same row
  - Synchronous DRAM
    - Added clock to DRAM interface
    - Burst mode with critical word first
  - Wider interfaces
  - Double data rate (DDR)
  - Multiple banks on each DRAM device
## Memory Optimizations

<table>
<thead>
<tr>
<th>Production year</th>
<th>Chip size</th>
<th>DRAM Type</th>
<th>Slowest DRAM (ns)</th>
<th>Fastest DRAM (ns)</th>
<th>Column access strobe (CAS)/data transfer time (ns)</th>
<th>Cycle time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1980</td>
<td>64K bit</td>
<td>DRAM</td>
<td>180</td>
<td>150</td>
<td>75</td>
<td>250</td>
</tr>
<tr>
<td>1983</td>
<td>256K bit</td>
<td>DRAM</td>
<td>150</td>
<td>120</td>
<td>50</td>
<td>220</td>
</tr>
<tr>
<td>1986</td>
<td>1M bit</td>
<td>DRAM</td>
<td>120</td>
<td>100</td>
<td>25</td>
<td>190</td>
</tr>
<tr>
<td>1989</td>
<td>4M bit</td>
<td>DRAM</td>
<td>100</td>
<td>80</td>
<td>20</td>
<td>165</td>
</tr>
<tr>
<td>1992</td>
<td>16M bit</td>
<td>DRAM</td>
<td>80</td>
<td>60</td>
<td>15</td>
<td>120</td>
</tr>
<tr>
<td>1996</td>
<td>64M bit</td>
<td>SDRAM</td>
<td>70</td>
<td>50</td>
<td>12</td>
<td>110</td>
</tr>
<tr>
<td>1998</td>
<td>128M bit</td>
<td>SDRAM</td>
<td>70</td>
<td>50</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>2000</td>
<td>256M bit</td>
<td>DDR1</td>
<td>65</td>
<td>45</td>
<td>7</td>
<td>90</td>
</tr>
<tr>
<td>2002</td>
<td>512M bit</td>
<td>DDR1</td>
<td>60</td>
<td>40</td>
<td>5</td>
<td>80</td>
</tr>
<tr>
<td>2004</td>
<td>1G bit</td>
<td>DDR2</td>
<td>55</td>
<td>35</td>
<td>5</td>
<td>70</td>
</tr>
<tr>
<td>2006</td>
<td>2G bit</td>
<td>DDR2</td>
<td>50</td>
<td>30</td>
<td>2.5</td>
<td>60</td>
</tr>
<tr>
<td>2010</td>
<td>4G bit</td>
<td>DDR3</td>
<td>36</td>
<td>28</td>
<td>1</td>
<td>37</td>
</tr>
<tr>
<td>2012</td>
<td>8G bit</td>
<td>DDR3</td>
<td>30</td>
<td>24</td>
<td>0.5</td>
<td>31</td>
</tr>
</tbody>
</table>

**Figure 2.13** Times of fast and slow DRAMs vary with each generation. (Cycle time is defined on page 95.) Performance improvement of row access time is about 5% per year. The improvement by a factor of 2 in column access in 1986 accompanied the switch from NMOS DRAMs to CMOS DRAMs. The introduction of various burst transfer modes in the mid-1990s and SDRAMs in the late 1990s has significantly complicated the calculation of access time for blocks of data; we discuss this later in this section when we talk about SDRAM access time and power. The DDR4 designs are due for introduction in mid- to late 2012. We discuss these various forms of DRAMs in the next few pages.
Memory Optimizations

<table>
<thead>
<tr>
<th>Standard</th>
<th>Clock rate (MHz)</th>
<th>M transfers per second</th>
<th>DRAM name</th>
<th>MB/sec/DIMM</th>
<th>DIMM name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDR</td>
<td>133</td>
<td>266</td>
<td>DDR266</td>
<td>2128</td>
<td>PC2100</td>
</tr>
<tr>
<td>DDR</td>
<td>150</td>
<td>300</td>
<td>DDR300</td>
<td>2400</td>
<td>PC2400</td>
</tr>
<tr>
<td>DDR</td>
<td>200</td>
<td>400</td>
<td>DDR400</td>
<td>3200</td>
<td>PC3700</td>
</tr>
<tr>
<td>DDR2</td>
<td>266</td>
<td>533</td>
<td>DDR2-533</td>
<td>4264</td>
<td>PC4300</td>
</tr>
<tr>
<td>DDR2</td>
<td>333</td>
<td>667</td>
<td>DDR2-667</td>
<td>5336</td>
<td>PC5300</td>
</tr>
<tr>
<td>DDR2</td>
<td>400</td>
<td>800</td>
<td>DDR2-800</td>
<td>6400</td>
<td>PC6400</td>
</tr>
<tr>
<td>DDR3</td>
<td>533</td>
<td>1066</td>
<td>DDR3-1066</td>
<td>8528</td>
<td>PC8500</td>
</tr>
<tr>
<td>DDR3</td>
<td>666</td>
<td>1333</td>
<td>DDR3-1333</td>
<td>10,664</td>
<td>PC10700</td>
</tr>
<tr>
<td>DDR3</td>
<td>800</td>
<td>1600</td>
<td>DDR3-1600</td>
<td>12,800</td>
<td>PC12800</td>
</tr>
<tr>
<td>DDR4</td>
<td>1066–1600</td>
<td>2133–3200</td>
<td>DDR4-3200</td>
<td>17,056–25,600</td>
<td>PC25600</td>
</tr>
</tbody>
</table>

Figure 2.14 Clock rates, bandwidth, and names of DDR DRAMS and DIMMs in 2010. Note the numerical relationship between the columns. The third column is twice the second, and the fourth uses the number from the third column in the name of the DRAM chip. The fifth column is eight times the third column, and a rounded version of this number is used in the name of the DIMM. Although not shown in this figure, DDRs also specify latency in clock cycles as four numbers, which are specified by the DDR standard. For example, DDR3-2000 CL 9 has latencies of 9-9-9-28. What does this mean? With a 1 ns clock (clock cycle is one-half the transfer rate), this indicate 9 ns for row to columns address (RAS time), 9 ns for column access to data (CAS time), and a minimum read time of 28 ns. Closing the row takes 9 ns for precharge but happens only when the reads from that row are finished. In burst mode, transfers occur on every clock on both edges, when the first RAS and CAS times have elapsed. Furthermore, the precharge in not needed until the entire row is read. DDR4 will be produced in 2012 and is expected to reach clock rates of 1600 MHz in 2014, when DDR5 is expected to take over. The exercises explore these details further.
Memory Optimizations

- **DDR:**
  - DDR2
    - Lower power (2.5 V -> 1.8 V)
    - Higher clock rates (266 MHz, 333 MHz, 400 MHz)
  - DDR3
    - 1.5 V
    - 800 MHz
  - DDR4
    - 1-1.2 V
    - 1600 MHz

- GDDR5 is graphics memory based on DDR3
Memory Optimizations

- Graphics memory:
  - Achieve 2-5 X bandwidth per DRAM vs. DDR3
    - Wider interfaces (32 vs. 16 bit)
    - Higher clock rate
      - Possible because they are attached via soldering instead of socketted DIMM modules

- Reducing power in SDRAMs:
  - Lower voltage
  - Low power mode (ignores clock, continues to refresh)
Memory Power Consumption

- Low power mode
- Typical usage
- Fully active

Power in mW:
- Read, write, terminate power
- Activate power
- Background power
Flash Memory

- Type of EEPROM
- Must be erased (in blocks) before being overwritten
- Non volatile
- Limited number of write cycles
- Cheaper than SDRAM, more expensive than disk
- Slower than SRAM, faster than disk
Memory Dependability

- Memory is susceptible to cosmic rays
- *Soft errors*: dynamic errors
  - Detected and fixed by error correcting codes (ECC)
- *Hard errors*: permanent errors
  - Use spare rows to replace defective rows

- Chipkill: a RAID-like error recovery technique
Pitfalls

- Predicting cache performance of one program for another
- Not simulating enough instructions to get an accurate performance measure of the memory hierarchy
- Not delivering high memory bandwidth in a cache-based system