Chapter 2 Solutions

Case Study 1: Optimizing Cache Performance via Advanced Techniques

2.1

2.1 a. Each element is 8B. Because a 64B cacheline has 8 elements, and each column access will result in fetching a new line for the nonideal matrix, we need a minimum of 8×8 (64 elements) for each matrix. Hence, the minimum cache size is $128 \times 8B = 1$ KB.

b. The blocked version only has to fetch each input and output element once. The unblocked version will have one cache miss for every 64B/8B = 8 row elements. Each column requires $64B \times 256$ of storage, or 16 KB. Thus, column elements will be replaced in the cache before they can be used again. Hence, the unblocked version will have 9 misses (1 row and 8 columns) for every 2 in the blocked version.

c.
```
for (i = 0; i < 256; i = i + B) {
      for (j = 0; j < 256; j = j + B) {
          for (m = 0; m < B; m+ +) {
              for (n = 0; n < B; n+ +) {
                  output[j + n][i + m] = input[i + m][j + n];
              }
          }
      }
  }
```

d. 2-way set associative. In a direct-mapped cache, the blocks could be allocated so that they map to overlapping regions in the cache.

e. You should be able to determine the level-1 cache size by varying the block size. The ratio of the blocked and unblocked program speeds for arrays that do not fit in the cache in comparison to blocks that do is a function of the cache block size, whether the machine has out-of-order issue, and the bandwidth provided by the level-2 cache. You may have discrepancies if your machine has a write-through level-1 cache and the write buffer becomes a limiter of performance.

Exercises

2.17

a. The access time of the direct-mapped cache is 0.86 ns, while the 2-way and 4-way are 1.12 and 1.37 ns, respectively. This makes the relative access times 1.12/0.86 = 1.30 or 30% more for the 2-way and 1.37/0.86 = 1.59 or 59% more for the 4-way.

b. The access time of the 16 KB cache is 1.27 ns, while the 32 and 64 KB are 1.35 and 1.37 ns, respectively. This makes the relative access times 1.35/1.27 = 1.06 or 6% larger for the 32 KB and 1.37/1.27 = 1.078 or 8% larger for the 64 KB.

c. Avg. access time = hit% × hit time + miss% × miss penalty, miss% = misses per

instruction/references per instruction = 2.2% (DM), 1.2% (2-way), 0.33% (4-way), 0.09% (8-way).

Direct mapped access time = 0.86 ns @ 0.5 ns cycle time = 2 cycles
2-way set associative = 1.12 ns @ 0.5 ns cycle time = 3 cycles
4-way set associative = 1.37 ns @ 0.83 ns cycle time = 2 cycles
8-way set associative = 2.03 ns @ 0.79 ns cycle time = 3 cycles
Miss penalty = (10/0.5) = 20 cycles for DM and 2-way; 10/0.83 = 13 cycles for 4-way; 10/0.79 = 13 cycles for 8-way.

Direct mapped—$(1 - 0.022) \times 2 + 0.022 \times (20) = 2.39\,6$ cycles $\Rightarrow 2.396 \times 0.5 = 1.2$ ns
2-way—$(1 - 0.012) \times 3 + 0.012 \times (20) = 3.2$ cycles $\Rightarrow 3.2 \times 0.5 = 1.6$ ns
4-way—$(1 - 0.0033) \times 2 + 0.0033 \times (13) =$
2.036 cycles $\Rightarrow 2.06 \times 0.83 = 1.69$ ns
8-way—$(1 - 0.0009) \times 3 + 0.0009 \times 13 = 3$ cycles $\Rightarrow 3 \times 0.79 = 2.37$ ns

Direct mapped cache is the best.

2.19

a. The access time is 1.12 ns, while the cycle time is 0.51 ns, which could be potentially pipelined as finely as 1.12/0.51 = 2.2 pipestages.

b. The pipelined design (not including latch area and power) has an area of 1.19 mm^2 and energy per access of 0.16 nJ. The banked cache has an area of 1.36 mm^2 and energy per access of 0.13 nJ. The banked design uses slightly more area because it has more sense amps and other circuitry to support the two banks, while the pipelined design burns slightly more power because the memory arrays that are active are larger than in the banked case.

2.20

a. With critical word first, the miss service would require 120 cycles. Without crit- ical word first, it would require 120 cycles for the first 16B and 16 cycles for each of the next 3 16B blocks, or $120 + (3 \times 16) = 168$ cycles.

b. It depends on the contribution to Average Memory Access Time (AMAT) of the level-1 and level-2 cache misses and the percent reduction in miss service times provided by critical word first and early restart. If the percentage reduc- tion in miss service times provided by critical word first and early restart is roughly the same for both level-1 and level-2 miss service, then if level-1 misses contribute more to AMAT, critical word first would likely be more important for level-1 misses.

2.22

In all three cases, the time to look up the L1 cache will be the same. What differs is the time spent servicing L1 misses. In case (a), that time = 100 (L1 misses) × 16 cycles + 10 (L2 misses) × 200 cycles = 3600 cycles. In case (b), that time = $100 \times 4 + 50 \times 16 + 10 \times 200 = 3200$ cycles. In case (c), that time = $100 \times 2 + 80 \times 8 + 40 \times 16 + 10 \times 200 = 3480$ cycles. The best design is case (b) with a 3-level cache. Going to a 2-level cache can result in many long L2 accesses (1600 cycles looking up L2). Going to a 4-level cache can result in many futile look-ups in each level of the hierarchy.

2.24

Let's first assume an idealized perfect L1 cache. A 1000-instruction program would finish in 1000 cycles, that is, 1000 ns. The power consumption would be 1 W for the core and L1, plus 0.5 W of memory background power. The energy consumed would be 1.5 W × 1000 ns = 1.5 μJ.

Next, consider a PMD that has no L2 cache. A 1000-instruction program would finish in 1000 + 100 (MPKI) × 100 ns (latency per memory access) = 11,000 ns. The energy consumed would be 11,000 ns × 1.5 W (core, L1, background memory power) + 100 (memory accesses) × 35 nJ (energy per memory access) = 16,500 + 3500 nJ = 20 μJ.

For the PMD with a 256 KB L2, the 1000-instruction program would finish in 1000 + 100 (L1 MPKI) × 10 ns (L2 latency) + 20 (L2 MPKI) × 100 ns (memory latency) = 4000 ns. Energy = 1.7 W (core, L1, L2 background, memory back- ground power) × 4000 ns + 100 (L2 accesses) × 0.5 nJ (energy per L2 access) + 20 (memory accesses) × 35 nJ = 6800 + 50 + 700 nJ = 7.55 μJ.

For the PMD with a 1 MB L2, the 1000-instruction program would finish in $1000 + 100 \times 20$ ns $+ 10 \times 100$ ns = 4000 ns.

Energy = 2.3 W × 4000 ns + 100×0.7 nJ + 10×35 nJ = 9200 + 70 + 350 nJ = 9.62 μJ.

Therefore, of these designs, lowest energy for the PMD is achieved with a 256 KB L2 cache.

2.25

(a) A small block size ensures that we are never fetching more bytes than required by the processor. This reduces L2 and memory access power. This is slightly offset by the need for more cache tags and higher tag array power. However, if this leads to more application misses and longer program completion time, it may ultimately result in higher application energy. For example, in the previous exercise, notice how the

design with no L2 cache results in a long execution time and highest energy. (b) A small cache size would lower cache power, but it increases memory power because the number of memory accesses will be higher. As seen in the pre- vious exercise, the MPKIs, latencies, and energy per access ultimately decide if energy will increase or decrease. (c) Higher associativity will result in higher tag array power, but it should lower the miss rate and memory access power. It should also result in lower execution time, and eventually lower application energy.

2.42

An aggressive prefetcher brings in useful blocks as well as several blocks that are not immediately useful. If prefetched blocks are placed in the cache (or in a pre- fetch buffer for that matter), they may evict other blocks that are imminently useful, thus potentially doing more harm than good. A second significant downside is an increase in memory utilization, that may increase queuing delays for demand accesses. This is especially problematic in multicore systems where the bandwidth is nearly saturated and at a premium.