The Art of Multiprocessor Programming
Solutions to Exercises
Appendix 2

July 14, 2009

Exercise 219. Thread A must wait for a thread on another processor to change a flag bit in memory. The scheduler can either allow A to spin, repeatedly retesting the flag, or it can deschedule A, allowing some other thread to run. Suppose it takes a total of 10 milliseconds for the operationg system to switch a processor from one thread to another. If the operating system deschedules thread A and immediately reschedules it, then it wastes 20 milliseconds. If, instead, A starts spinning at time t_0, and the flag changes at t_1, then the operating system will have wasted $t_1 - t_0$ time doing unproductive work.

A *prescient* scheduler is one that can predict the future. If it foresees that the flag will change in less than 20 milliseconds, it makes sense to have A spin, wasting less than 20 milliseconds, because descheduling and rescheduling A wastes 20 milliseconds. If, on the other hand, it takes more than 20 milliseconds for the flag to change, it makes sense to replace A with another thread, wasting no more than 20 milliseconds.

Your assignment is to implement a scheduler that never wastes more than *twice* the time a prescient scheduler would have wasted under the same circumstances.

Solution. Your scheduler should spin for 20 milliseconds and then deschedule A. There are two cases:

- If the flag changes in less than 20 milliseconds, you guessed correctly, and wasted the minimum amount of time.
- If the flag changes in more than 20 milliseconds, then you should have descheduled A right away, so you wasted at most 20 milliseconds more than the prescient scheduler, which itself was forced to waste more than 20 milliseconds.

Exercise 220. Imagine you are a lawyer, paid to make the best case you can for a particular point of view. How would you argue the following claim: if context switches took negligible time, then processors would not need caches, at least for applications that encompass large numbers of threads.

Extra credit: critique your argument.

Solution. If context switches took negligible time, then every time a thread made a memory reference, the processor could set that thread aside and run another. When the memory reference finished, the processor could reschedule the original thread.

The weakness in this argument is the assumption that an application always has another thread to run. If the application encompasses a limited number of threads, then this argument is less convincing.

Exercise 221. Consider a direct-mapped cache with 16 cache lines, indexed 0 to 15, where each cache line encompasses 32 words.

- Explain how to map an address a to a cache line in terms of bit shifting and masking operations. Assume for this question that addresses refer to words, not bytes: address 7 refers to the 7th word in memory.
- Compute the best and worst possible hit ratios for a program that loops 4 times through an array of 64 words.
- Compute the best and worst possible hit ratios for a program that loops 4 times through an array of 512 words.

Solution. Assume for this question that addresses refer to words, not bytes: address 7 refers to the 7^{th} word in memory. Consider a direct-mapped cache with 16 cache lines, indexed 0 to 15, where each cache line encompasses 32 words.

- Since each cache line contains 32 words, we first discard the lowest 5 bits from the address

  ```
  b = a >> 5;
  ```

  Now, the lowest 4 bits determine a cache line:

  ```
  c = b & 0xF;
  ```

  Here, the hexadecimal constant `0x0F` is the value represented in binary as four 1s. Putting everything together:

  ```
  c = ((a >> 5) & 0x0F)
  ```

- The best ratio occurs when the array occupies only two cache lines. The first time though, the processor takes a cache miss the first time it accesses a new block, which happens twice, at array elements 0 and 32. The program takes 2 cache misses while accessing $4 \times 64 = 512$ memory locations, yielding a hit ratio of:

  $$1.0 - \frac{2}{512} = 0.996.$$

  The worst ratio occurs when the array is not aligned with the cache lines. The first time though, the processor takes a cache miss the first time it accesses a new block, which happens three times. The program takes 3 cache misses while accessing 512 memory location, yielding a hit ratio of

  $$1.0 - \frac{3}{512} = 0.994$$

- *Compute the best and worst possible hit ratios for a program that loops 4 times through an array of 512 words.* The best occurs when the array fits exactly in the cache. The program takes 16 misses out of 2048 references:

  $$1.0 - \frac{16}{2048} = 0.992.$$

The worst occurs when the array starts in the middle of the block, so it takes up 17 blocks, which does not fit in the cache. On the first iteration, the program takes 17 misses the first time it reads each line. When it reads the last line, it evicts the first line from the cache. On each subsequent iteration, it takes two cache misses, one each on the first and last line, for a total of $17 + (3 \cdot 2) = 23$ misses. The ratio is:

$$1.0 - \frac{23}{2048} = 0.989.$$

Exercise 222. Consider a direct-mapped cache with 16 cache lines, indexed 0 to 15, where each cache line encompasses 32 words.

Consider a two-dimensional, 32×32 array of words a. This array is laid out in memory so that $a[0, 0]$ is next to $a[0, 1]$, and so on. Assume the cache is initially empty, but that $a[0, 0]$ maps to the first word of cache line 0.

Consider the following *column-first* traversal:

```
int sum = 0;
for (int i = 0; i < 32; i++) {
  for (int j = 0; j < 32; j++) {
    sum += a[i,j];      // 2nd dim changes fastest
  }
}
```

and the following *row-first* traversal:

```
int sum = 0;
for (int i = 0; i < 32; i++) {
  for (int j = 0; j < 32; j++) {
    sum += a[j,i];      // 1st dim changes fastest
  }
}
```

Compare the number of cache misses produced by the two traversals, assuming the oldest cache line is evicted first.

Solution. Note that `a[i,j]` and `a[i+1,j]` are 32 words apart, and are therefore mapped to different cache lines. The column-first traversal takes a cache miss on every reference. The first 16 references find empty cache lines, the next 16 references evict the first 16, for a total of 1024 cache misses. By contract the row-first traversal moves through memory in sequence, reading each cache line once, for a total of 32 cache misses.

Exercise 223. In the MESI cache-coherence protocol, what is the advantage of
distinguishing between exclusive and modified modes?

What is the advantage of distinguishing between exclusive and shared modes?

Solution. Evicting a modified cache line requires writing the data back to memory, while an exclusive cache line can simply be discarded.

Modifying a shared cache line requires invalidating that line in other caches, while modifying an exclusive cache line does not.

Exercise 224. Implement the test-and-set and test-and-test-and-set locks shown in Figs. **??** and **??**, test their relative performance on a multiprocessor, and analyze the results.

Solution. This exercise does not need a solution.