COMP7300 Fall 2021 HW05

Libo Sun lzs0101@auburn.edu
Compute Science and Software Engineering
Auburn University, Auburn, AL, U.S.

1. **What is the function of a replacement algorithm? Do you need a replacement algorithm for direct mapping?**

   - Once the cache has been filled, when a new block is brought into the cache, one of the existing blocks must be replaced. To achieve high speed,an algorithm must be implemented in hardware.
   - For direct mapping, there is only one possible line for any particular block, and no choice is possible.

2. **Define the two write policies in about ten sentences total. Discuss the relative merits of the policies.**

   - **Write through**: Using this technique, all write operations are made to main memory as well as to the cache, ensuring that main memory is always valid. Any other processor–cache module can monitor traffic to main memory to maintain consistency within its own cache. This is the **simplest policy**, while the main disadvantage of this technique is that it generates substantial memory traffic and may create a bottleneck.
   - **Write back**: With write back, updates are made only in the cache. When an update occurs, a dirty bit, or use bit, associated with the line is set. Then, when a block is replaced, it is written back to main memory if and only if the dirty bit is set. It **minimizes memory writes**, but the problem with write back is that portions of main memory are invalid, and hence accesses by I/O modules can be allowed only through the cache. This makes for complex circuitry and a potential bottleneck.

3. **Describe a simple way of implementing an LRU replacement algorithm in a 4-way set associative cache.**

   - Dividing the four lines into three pairs, say three LRU bits, B0, B1, and B2.
   - Divide these four lines into two pairs of lines, pair L01 comprising lines L0 and L1, and pair L23 comprising lines L2 and L3.
   - These are all set to 0 when the cache is flushed, and are updated on every cache hit or replacement. Call the four lines in the set, L0, L1, L2, and L3.
   - Let bit B0 indicate whether pair L01 or L23 was last accessed. That is, if either L0 or L1 is accessed, B0 is set to 1, while if either L2 or L3 is accessed, B0 is set to 0.
   - Let bit B1 indicate which line in pair L01 is accessed. That is, if L0 is accessed, B1 is set to 1, else B1 is set to 0.

- Similarly, bit B2 indicates which line in pair L23 is accessed. That is, if L2 is accessed, B2 is set to 1, else B2 is set to 0.
- When all lines in a set are in use (all Valid bits are 1), the replacement mechanism works as follows. If B0 = = 0, a line in pair L01 is to be replaced, else the line is in L23. If the line is in pair L01 and B1 = = 0, L0 is to be replaced, else L1 is to be replaced. If the line is in pair L23 and B2 = = 0, L2 is to be replaced, else L3 is to be replaced.

4. **What is the impact of the design decision of line size on hit ratio? Discuss the various approaches of cache coherency in a multiprocessor system (in about 10 sentences). Discuss the relative merits of unified vs. split (code/data) cache design.**

   - **Line size**: Another design element is the line size. When a block of data is retrieved and placed in the cache, not only the desired word but also some number of adjacent words are retrieved. As the block size increases from very small to larger sizes, the hit ratio will at first increase because of the principle of locality, which states that data in the vicinity of a referenced word are likely to be referenced in the near future. As the block size increases, more useful data are brought into the cache. The hit ratio will begin to decrease, however, as the block becomes even bigger and the probability of using the newly fetched information becomes less than the probability of reusing the information that has to be replaced. Two specific effects come into play:
     - Larger blocks reduce the number of blocks that fit into a cache. Because each block fetch overwrites older cache contents, a small number of blocks results in data being overwritten shortly after they are fetched.
     - As a block becomes larger, each additional word is farther from the requested word and therefore less likely to be needed in the near future.
   - **Cache Coherency**: If data in one cache are altered, this invalidates not only the corresponding word in main memory, but also that same word in other caches (if any other cache happens to have that same word). Even if a write-through policy is used, the other caches may contain invalid data. A system that prevents this problem is said to maintain cache coherency. Possible approaches to cache coherency include the following:
     - **Bus watching with write through**: Each cache controller monitors the address lines to detect write operations to memory by other bus masters. If another master writes to a location in shared memory that also resides in the cache memory, the cache controller invalidates that cache entry. This strategy depends on the use of a write-through policy by all cache controllers.
     - **Hardware transparency**: Additional hardware is used to ensure that all updates to main memory via cache are reflected in all caches. Thus, if one processor modifies a word in its cache, this update is written to main memory. In addition, any matching words in other caches are similarly updated.
     - **Noncacheable memory**: Only a portion of main memory is shared by more than one processor, and this is designated as noncacheable. In such a system, all accesses to shared memory are cache misses, because the shared memory is never copied into the cache. The noncacheable memory can be identified using chip-select logic or high-address bits.
   - **unified versus split caches**
     - There are two potential advantages of a unified cache:
       * For a given cache size, a unified cache has a higher hit rate than split caches because it balances the load between instruction and data fetches automatically. That is, if

an execution pattern involves many more instruction fetches than data fetches, then the cache will tend to fill up with instructions, and if an execution pattern involves relatively more data fetches, the opposite

  * Only one cache needs to be designed and implemented.
- The key advantage of the split cache design is that it eliminates contention for the cache between the instruction fetch/decode unit and the execution unit. This is important in any design that relies on the pipelining of instructions. Typically, the processor will fetch instructions ahead of time and fill a buffer, or pipeline, with instructions to be executed.

5. **Define spatial and temporal locality. Consider the following code:**

```
for (i=0; i < 50; i++)
    for (j =0; j < 30; j++)
       a[i] = a[i] * j;
```

- Give an example of spatial locality in the above code.
  - the reference string contains several closely spaced references to adjacent addresses.
  - References to $a[i]$ (sequences in for (i=0; i < 50; i++)) are a good example of this, as one can assume (most of the time) that $a[0]$ and $a[1]$ will be next to each other in memory.
- Give an example of temporal locality in the above code.
  - The principle of Temporal locality states that two instructions reference the same location within a relatively short timeframe.
  - In the loop:

```
for (j =0; j < 30; j++)
   a[i] = a[i] * j;
```

  - $a[i]$ and j are referenced frequently, references to j and $a[i]$ are temporally local. References to i are also temporally local, because i is referenced every time $a[i]$ is.

6. Consider a memory system with the following parameters:

- T_c = 5 ns, C_c = \$ 10^{-6}/bit
- T_m = 50 ns, C_m = \$ 10^{-7}/bit

(a) What is the cost of 4 GB of main memory?

- Given, C_m = \$ 10^{-7}/bit and $4GB$ Size
- $4GB = 4 \times 2^{30} \times 8\, bits = 32 \times 2^{30}\, bits \approx 32 \times 10^9\, bits$
- $C_{4GB} \approx 32 \times 10^9 \times 10^{-7} = \$3200$

(b) What is the cost of 4 GB of memory using cache only technology?

- Given, C_n = \$ 10^{-6}/bit and $4GB$ Size
- $4GB = 4 \times 2^{30} \times 8\, bits = 32 \times 2^{30}\, bits \approx 32 \times 10^9\, bits$
- $C_{4GB} \approx 32 \times 10^9 \times 10^{-6} = \$32000$

(c) If the effective access time is 20% greater than cache access time, what is the hit ratio H?

- Given T_c = 5 ns and T_m = 50 ns
- Given 20%, hence ,$T = T_c \times (1 + 20\%) = 6\, ns$

- $T = T_c \times H + (1 - H) \times (T_c + T_m) = 5H + (1 - H) \times 55 = 55 - 50H = 6\,ns$
- $H = 98\%$

7. **A computer has cache, main memory and a disk used for virtual memory. If a referenced word is in the cache, 15 ns are required to access it. If it is in the main memory, but not in the cache 60 ns are needed to load into the cache, and then the reference is started again. If the word is not in the main memory, 10 ms are required to fetch the word from the disk, followed by 70 ns to copy it to the cache, and then the reference is started again. The cache hit ratio is 0.95 and the main memory hit ratio is 0.6. What is the average time in ns required to access a referenced word on this system?**

   - Given, Cache access time: $T_c = 15\,ns$
   - Given, Memory access time: $T_m = 60 + 15 = 75\,ns$
   - Given, Disk access time: $T_d = 10000000 + 70 + 15 = 10000085\,ns$
   - Given, Cache Hit ratio: $H_c = 0.95$
   - Given, Memory Hit ratio: $H_m = 0.6$
   - Average Memory access time:
     $T_{avgM} = H_m \times T_m + (1 - H_m) \times T_d$
     $= 0.6 \times 75 + 0.4 \times 10000085 = 4000079\,ns$
   - Average Access time:
     $T_{avg} = H_c \times T_c + (1 - H_c) \times T_{avgM}$
     $= 0.95 \times 15 + 0.05 \times 4000079$
     $= 200018.2\,ns$