CS61 Section Notes

Week 6 (Fall 2011)

Topics to be covered

- **Memory and Storage Technologies**
  - **DRAM**
  - **Hard Drives**
- **Caches**
  - **Direct Mapped Caches**
  - **Set Associative Caches**

Memory and Storage Technologies

DRAM

A $d \times w$ DRAM contains d supercells, each of which contains w bits. The d supercells are organized into a rectangular array of r rows and c columns (i.e. d=r*c). To access a supercell, we send a row access strobe (RAS), followed by a column access strobe (CAS). This identifies a unique supercell.

Question 1: What are the advantages of sending the address of a supercell in two steps? What are the disadvantages?

Question 2: Given a DRAM with $d \times w$ supercells, how would you chose the dimensions of the rectangular array to minimize the number of address pins needed?

Hard Drives

Question 3: Consider a disk with a rotational rate of 10,000 RPM, an average seek time of 8 ms, and an average of 500 sectors per track. Estimate the average time to read a random sector from disk. Do this by summing the estimates of the seek time, rotational latency, and transfer time.

Question 4: Disk controllers map logical blocks to physical locations on the disk. Suppose that a 1MB file consisting of 512-byte logical blocks is stored on a disk drive with the following characteristics:

Rotational rate:	10,000 RPM
Average seek time	5ms
Average # sectors/track	1000
Surfaces	4
Sector size	512 bytes

Suppose that a program reads all the blocks of this file sequentially, and that the time to position the head over the first block is the average seek time plus the average rotational latency.

4a) What is the best case for mapping logical blocks to disk sectors? Estimate the time required to read the file in this best case scenario.

4b) Suppose that the logical blocks are mapped randomly to disk sectors. Estimate the time required to read the file in this scenario.

Storage trends

Question 5: Given the following trends for the cost of rotating disk storage, estimate the year that you will be able to purchase a petabyte of storage (=10^{15} bytes = 1000 terabytes = 1 million gigabytes) for under $500. (In 2010 a petabyte of storage costs about $300,000).

Year	2000	2005	2010
$/MB	0.01	0.005	0.0003

Caches

Locality

Question 6. What are the two types of locality?

Question 7. Why do we care about locality? In general, how does locality affect program execution time? How does increasing the size of the working set affect locality? What about changing the loop's "stride" distance?

Question 8. Look at the following code.

```
#define N 1000

typedef struct {
   int vel[3];
   int acc[3];
} point;

point p[N];

void clear1(point* p, int n)
{
   int i, j;

   for (i = 0; i < n; i++) {
       for (j=0; j < 3; j++)
                  p[i].vel[j] = 0;
       for (j=0; j < 3; j++)
                  p[i].acc[j] = 0;
   }
}

void clear2(point* p, int n)
{
   int i, j;

   for (i = 0; i < n; i++) {
       for (j=0; j < 3; j++) {
                  p[i].vel[j] = 0;
                  p[i].acc[j] = 0;
       }
   }
}
```

```
void clear3(point* p, int n)
{
   int i, j;

   for (j=0; j < 3; j++) {
       for (i = 0; i < n; i++)
                   p[i].vel[j] = 0;
        for (i = 0; i < n; i++)
                   p[i].acc[j] = 0;
   }
}
```

How would your order these functions from that which displays the most spatial locality to that which displays the least?

Direct-Mapped Caches

Direct-mapped caches have only one cache line per set (*E* = *1*). Consider a cache with *B* = *16* and *S* = *2*. Assume floats are 4 bytes. For simplicity, also assume (incorrectly!) that `x[]` starts at memory address 0x0 and `y[]` at address 0x20 (so the two arrays are adjacent in memory). Consider the following code.

```
float dotprod(float x[8], float y[8])
{
      float sum = 0.0;
      int i;

      for (i = 0; i < 8; ++i)
      sum += x[i] * y[i];
      return sum;
}
```

Question 9. How many bits are used to determine the set to which an address will belong? Which bits of the address are used to determine the set?

Question 10. Step through the access pattern of the function as it executes. Is the cache efficiently used?

Set Associative Caches

The code we just saw causes the cache to *thrash*. A cache *thrashes* when it repeatedly loads and then evicts the same set of cache blocks. We can alleviate this issue with a cache that has more than one cache line per set. A cache with 1 < *E* < *C*/*B* is called an *E*-way set associative cache.

Question 11. Suppose we run the code in Questions 8-9 using a 2-way set associative cache. How will this change the program's use of the cache?

Of course, adding cache lines adds hardware complexity. The cache must check the tag and valid bit of each line in the appropriate set. Since, within a set, any line can contain any block, the cache design must also include a policy for deciding which line to eject when a conflict miss occurs. Many such policies are used, all of which have pros and cons depending on the situation.

Question 12 What are the advantages of a LRU (Least Recently Used) cache eviction policy? The disadvantages? What approaches help overcome these shortcomings?

Question 13 Another possible cache eviction policy is MRU (Most Recently Used): the most recently used line in a set is chosen for eviction. In general, is this a good eviction policy? Is there any situation where this would be a good policy?

Question 14 What are two policies for writing data back to memory and what are the advantages of each?