

GEORGIA INSTITUTE OF TECHNOLOGY

College of Computing

CS6290/CS4290 — High-Performance Computer Architecture

Spring 2003

CS6290/CS4290
Homework 2

Issued: January 24, 2003
Due: January 31, 2003

Purpose: This homework covers caching.

Reading: H&P Chapter 5 for caches
[Jouppi90] to be discussed in class Friday, January 24th.
[Dennis65] to be discussed in class Friday, January 31st.

Problems:

1. Read [Dennis65].
2. Cache Organization.
3. Cache Performance.
4. Cache Analysis.
5. Cache Marketing.

Collaboration: (*As in the syllabus*) collaboration on projects and homework in **pairs** is encouraged. If you work in a pair, turn in one write-up with the names of both collaborators. You're welcome to discuss high-level concepts with other groups, but all homework solutions must be worked out and written up separately.

Problem 1: Reading

A: Read [Dennis65] on virtual memory in order to be prepared to discuss it in class on Friday, January 31st. This is quite an old paper and virtual memory systems have generally become *simpler* since 1965. In particular, segmentation is rarely used. What is the key benefit of segmentation? What do current operating systems do to get this benefit without segmentation hardware?

Problem 2: Cache Organization

A cache has a total size of N data words, with B words per block and A -way set associativity. Assume 32-bit data words, 32-bit addresses and that N , B and A are all powers of two.

A: What is the layout of an address in terms of the number and positions of bits used for the tag, index and offset?

B: What is the total number of bits of storage required (data + tags) in the cache hardware?

C: Why are the index bits taken from the “middle” of the address and not from the least significant or most significant bits of the address?

Problem 3: Cache Performance

A: Problem 5.4 in the book

Problem 4: Cache Analysis

The following data shows the time in nanoseconds to perform a read-increment-write on each word of an array (actual code in Appendix A). Each row represents a different size array (total size in bytes) and each column represents a stride (also in bytes).

	stride							
array:	4	8	16	32	64	128	256	512
-----	---	---	---	---	---	---	---	---
512:	11.5	11.9	12.8	13.8	18.8	20.0	40.0	40.0
1024:	11.2	11.4	11.7	12.2	13.1	16.2	20.0	30.0
2048:	11.2	11.2	11.6	11.7	12.8	13.8	17.5	22.5
4096:	11.1	11.2	11.2	11.3	11.7	12.2	13.1	15.0
8192:	11.1	11.2	11.2	11.3	11.5	12.0	12.5	14.4
16384:	17.1	11.2	11.9	11.2	11.3	11.3	11.9	12.2
32768:	18.3	27.0	39.5	38.9	39.0	39.0	39.2	39.4
65536:	19.4	26.7	40.5	42.0	41.6	40.6	39.0	39.1
131072:	18.9	26.0	40.5	41.4	42.1	39.8	39.0	39.0
262144:	20.6	30.2	50.1	68.5	90.9	91.2	90.6	88.8
524288:	29.4	54.2	107.4	210.0	416.7	412.6	410.8	410.5
1048576:	30.1	56.1	110.6	220.2	438.4	436.1	434.4	435.1
2097152:	30.1	55.9	111.0	220.9	439.0	477.9	435.2	435.5
4194304:	30.1	55.7	111.1	220.8	438.9	436.1	435.0	432.2
8388608:	30.0	55.8	111.1	222.9	440.0	436.4	436.3	435.5
16777216:	30.0	55.8	115.8	220.8	439.6	436.2	436.2	433.9

The machine has a two-level cache structure: a split L1 and a unified L2. You can ignore instruction misses (there are none in the body of the experiment).

A: What is the L1-D cache size and block size?

B: What is the L2 cache size and block size?

C: The machine measured has a 360MHz CPU so the basic read-modify-write operation is taking four cycles (about 11.1nS). What is the L1-to-L2 miss penalty in cycles? What is the L2-to-memory miss penalty in cycles? There is obviously some variance in the numbers so pick the most likely answer.

Problem 5: Cache Hacking

Adapted from a problem in Ward & Halstead, Computation Structures, McGraw-Hill, 1990.

You are hired by companies *A*, *B* and *C* to recommend cache designs for their respective machines. Each company's product is a clone of the DLX from the text and executes an identical instruction set. For the sake of variety, you have recommended a different cache to each manufacturer as follows:

A: A one-way (direct-mapped) cache, 4K bytes total size, 16 data bytes per line.

B: A two-way set associative cache, 4K bytes total size, 16 data bytes per line.

C: A four-way set associative cache, 4K bytes total size, 16 data bytes per line.

Note that (for the sake of consistency) you have recommended a cache with a total size of 4K bytes and 16 bytes per cache line. The write policy is write-through and the set-associative caches use an LRU replacement policy among the lines in a set.

Each of the companies is outraged to learn that you have been consulting for the others and you are threatened with multiple lawsuits. To placate your clients, you need to assure each company that its machine now outperforms the others in certain applications. Your challenge is to find three appropriate programs.

Describe three small program fragments. Each should be a loop that would run forever and each should perform best on one of the three machines. For each of your three program fragments, give the following information (example format below):

1. The memory reference string; i.e., the sequence of addresses referenced during the execution of the loop. The sequence is, of course, an infinite cycle; use ellipses as appropriate.
2. For each cache/machine, an indication of whether each reference hits or misses in the cache.
3. The hit ratio for each cache/machine.

Here is an example format (the patterns are nonsensical):

```
ref string:  1  37  42 1025  1  37  42 1025  ...
cache A:    -  H   H   -   -  H   H   -   ... (HR = 0.5)
cache B:    -  H   -   -   -  H   -   -   ... (HR = 0.25)
cache C:    -  -   -   -   -  -   -   -   ... (HR = 0)
```

A: Write three answers in the format above, one that performs best for each cache/machine.

Appendix 1: Cache Measurement Code

```
#define MINARRAYSIZE 512
#define MAXARRAYSIZE (16 * 1024 * 1024)
#define MAXSTRIDE 512
#define NTIMES 10

static int array[MAXARRAYSIZE];

int main(int argc, char *argv[])
{
    int arraysize, stride;

    for (arraysize = MINARRAYSIZE; arraysize <= MAXARRAYSIZE; arraysize *= 2)
    {
        int ntimes = (arraysize < 1024 * 1024) ? 100 : 10;

        printf("%8d:", arraysize);
        for (stride = 4; stride <= MAXSTRIDE; stride *= 2)
        {
            struct timeval stop, start;
            int kount, index;

            for (index = 0; index < arraysize; index += stride)
                array[index / sizeof(int)]++;
            for (index = 0; index < arraysize; index += stride)
                array[index / sizeof(int)]++;
            gettimeofday(&start, NULL);
            for (kount = 0; kount < ntimes; kount++)
                for (index = 0; index < arraysize; index += stride)
                    array[index / sizeof(int)]++;
            gettimeofday(&stop, NULL);
            printf("%6.1f", (diff_nsec(&stop, &start)
                / (double)ntimes
                / ((double)arraysize / (double)stride)));

        }
        printf("\n");
    }
    return(0);
}
```