Computer Architecture
EECE7352
Prof. Kaeli

Homework 4:

PART A.
The Dinero cache simulator is available on the COE 64-bit Linux machines. We will use **dineroIV**. We will use a trace formatted in the traditional din input format (-informat d). You can find more details on using dineroIV by issuing *dineroIV –help on the COE systems*.

Dinero allows you to configure a cache using the command line. The simulator takes as input an address trace. We will use the "din" input format, which is an ASCII file with one LABEL and one ADDRESS per line. The rest of the line is ignored so that it can be used for comments.
LABEL = 0 read data
= 1 write data
= 2 instruction fetch
= 3 escape record (treated as unknown access type)
= 4 escape record (causes cache flush)
0 <= ADDRESS <= ffffffff where the hexadecimal addresses are NOT preceded by "0x."

Here are some examples:
2 0 This is an instruction fetch at hex address 0.
0 1000 This is a data read at hex address 1000.
1 70f60888 This is a data write at hex address 70f60888.
Make sure to use lowercase.

For Part A, you will be generating (by hand) trace files in the above format. After creating a trace file (e.g., trace.txt), you run Dinero as follows:

dineroIV –l1-isize 8K –l1-ibsize 32 –l1-dsize 16K –l1-dbsize 16 -informat d < trace.txt

This will simulate an 8 KB instruction cache with a 32 byte line size and a 16 KB data cache with a 16 byte block size.

Your job is to generate reference streams in Dinero's din format (see description provided above) and produce the following (where you are asked to generate an address stream, submit in your report enough of the trace so that we can understand the pattern you are using – do not submit the full trace since it will only waste disk space):

1. Assume that you have a direct-mapped 8KB instruction cache with a (64B block size). **Generate an address stream** that will touch every cache line once, but no more than once.

2. Assume the same instruction cache organization as in (1), but now the instruction cache is 4-way set associative, with LRU replacement. The total cache space is still 8KB with a 64B block size, but now you have 1/4 the number of indices. **Generate an address stream** that touches every cache index only 7 times, producing 3 misses and 4 hits, but only accesses 3 unique addresses per index.

3. Repeat part 2, but now the cache is 2-way set associative. Produce 5 misses and 5 hits, again with only 3 unique address per index, but produce an interleaving pattern of Miss-Hit-Miss-Hit-Miss-Hit-…….

4. Assume that you have a 2-way set associate 32KB data cache with a (32B block size). **Generate an address stream** that will generate 5 hits and 5 misses. Make sure that your stream includes both loads and stores.

5. Given a partially specified instruction cache organization, **generate one or multiple instruction address reference streams** that can detect the set associativity and the total size of an instruction cache. Assume that you know that the block size is 16B and that the cache size will be no larger than 32KB. Generate the stream(s) and discuss how you figured out the total size and the associativity.

6. Repeat part 4, but now **generate a stream** that will determine the replacement algorithm. Again, discuss how you determined this.

PART B**. (for this part – use the trace provided on Canvas)**

For the next part of this assignment, using the trace provided as input to DineroIV, model an instruction cache and a data cache with a combined total cache space of 8KB (for a split cache, assume an 4KB instruction cache and an 4KB data cache. The block size should be varied (8B, 16B and 32B) and the associativity should be varied (direct-mapped, 2-way and 4-way). Model both split (separate instruction and data caches) and shared (all accesses go to a single cache that holds both instructions and data) caches. There is a total of 18 simulations. No other parameters should be varied. Graph the results you get from these experiments and **discuss in detail** why you see different trends in the graphs. Copy the trace file provided to the COE Linux system to run your study, but please gzip or compress the file when you are not using it.

PART C. **(for this part – use the trace provided on Canvas)**

For the next part of this assignment, use the prefetching capabilities provided in dineroIV (the -Tfetch and -Tpfdist switches). Using the same trace used in part B, study the effects of different data cache prefetching policies and report your results. Attempt to find the best prefetching policy and discuss why you feel this would be the best policy for the given workload. Use the -help switch to learn how to use these switches.

PART D.

Read that the attached HPCA paper on Last-Level Cache design. Then answer the following five questions: i.) Discuss the key points in this paper. ii.) Describe one of the experiments presented and discuss the results. iii.) What did you learn about the workloads studied? vi.) How did cache size benefit or hurt applications with a high degree of data sharing? v.) What is OpenMP, and how do you use this infrastructure to write parallel programs?

PART E.

In this part you will use the allcache.so Pin tool to study the relationship between a program, compiler optimizations, and memory behavior. Select any C program of your choice. You will need source code that you can compile and run on the COE system and can compile with gcc.

You will find the allcache.so Pin tool in the /COEnet/Linux/pin-3.11/source/tools/Memory/obj-intel64 directory. Copy this tool to a directory that you have read/write access to. Then run the tool as follows:

pin –t allcache.so -- Your Program

You will see a large number of memory address values for both the instruction stream and the data stream. Now recompile your program applying different optimization switches, rerun the new binary with Pin, and compare the results. Try out 3 different compiler switch settings that generate a significant change in the memory stream. Plot these results and attempt to explain what trends you are seeing.

Extra Credit: 40 points (added to your quiz grade)

Write your own simulator (you can modify the dineroIV source as a starting point) to implement a column associative cache, as described in the attached paper. Provide a working model that compiles and runs on the COE Linux system and report on the miss rate obtained when modeling an 8KB instruction cache with a 16B block size, that uses a column associative structure. Submit a copy of your code on Canvas in your writeup.