Homework 9 – CS 261

Link to download all hw9 files

https://uofi.box.com/s/43ptvaq6tpwdcyfscr3eudumlxr8kxdh

For this homework, you will be using the cache simulator to compare the cache performance of two sorting algorithms: Exchange Sort and Quicksort. You can work in groups of at most 2 people. If you do work in a group, make a single submission, and add your partner(s) to the gradescope submission.

Files You Will Need to Submit

1. Qsort.cc
2. Xsort.cc
3. PDF version of hw9_template.xlsx

Students are advised to understand cache and related topics mentioned in the cache_summary.pdf thoroughly.

1) The hw9.zip file attached provides the cache simulator and files needed to complete the homework. Download and copy this file to systemsX.cs.uic.edu. Connect to systemsX using SSH and extract the file using the unzip command.

   unzip hw9.zip

The extracted files should be contained within the hw9 directory. The files within this directory should include cache, sort.cc, xsort.cc, qsort.cc, inst_compact.h, inst_legible.h, inst_none.h, histogram.cc, blocking.cc and the hw9_template.xlsx file.

2) For this homework, we're going to look at sort.cc, qsort.cc, and xsort.cc files. The program sort.cc uses either Exchange sort or Quicksort to sort a list of 10,000 random numbers. The program qsort.cc contains the algorithm for Quicksort whereas xsort.cc contains the algorithm for Exchange sort. Read through the source code and check that you understand what the program is doing.

3) Your task is to add the cache instrumentation (INST_R(array_item) or INST_W(array_item) to the sorting functions within qsort.cc and xsort.cc to identify all the list item access.

   INST_R and INST_W are two macros that are used to provide the addresses of the data being accessed. The macro is included using the preprocessing directive #include using one of the following files (each file provides different functionality).

inst_legible.h Defines the macros to display the addresses to the console for you to view.

inst_none.h Defines the macros to do nothing.

inst_compact.h Defines the macros to output the address for use by the cache simulator.

You can also look at histogram.cc or blocking.cc to see how the addresses for the data values are sent to the simulator using instrumentation (INST_R and INST_W). Here's an example.

Your code before instrumentation:

```
temp = list[i];

list[j] = temp;
```

Your code after instrumentation:

```
INST_R(list[i]); //add INST_R before you need to look at a value

temp = list[i];

list[j] = temp;

INST_W(list[j]); // add INST_W only after actual write takes place
```

4) Explore the cache simulator. The cache simulator provides a text and graphics mode to visualize cache usage. The cache simulator binary is included in the zip file and is named cache. Examine the options available by running the help

./cache-h

IMPORTANT: If you receive an error of "permission denied", run following command to fix this error**: chmod +x cache**

Follow the steps to use a cache simulator after you complete adding instrumentation (calls to INST_R and INST_W) to xsort.cc and qsort.cc.

a. Edit sort.cc to include "inst_compact.h" instead of "inst_none.h"

b. In sort.cc call either XSort or QSort function (the one that you are experimenting) and comment the other one.

c. In hw9 directory, compile sort.cc using:

g++ sort.cc

d. Run the below command to ensure the output of cache is shown correctly in the terminal (disables automatic margins to ensure text is not wrapped incorrectly).

tput rmam

e. Run cache simulator using some given configuration of cache size and block size (Eg: 4KB cache size having a block size of 2 words)

./a.out | . /cache -s 4k -b 2

5) For the Exchange sort algorithm (call XSort in sort.cc and compile), experiment with different direct-mapped caches: try cache sizes in the range 4 KBytes to 64 KBytes, for example, {4 KB, 8KB, 16KB, 32KB and 64KB} and number of words per block in the range one to eight words say {1, 2, 4 and 8}. In each case, record the cache access statistics and attempt to understand the observed miss rates. Note that exchange sorting a list of 10,000 items takes some time: you'll probably want to interrupt each simulation (by pressing ctrl-c) after, say, 100,000 accesses, and record the statistics up to that point if your simulation takes too long. Next, edit sort.cc to enable Quicksort in place of Exchange sort, recompile and repeat the above experiments.

6) Submissions should be made to the gradescope under the link
    a. Homework 9 – Part 1.
        i. For submission of Part 1 – upload the exported pdf file from hw9_template.xlsx detailing the results of your experiments for each cache size, block size, and sorting algorithm.
        ii. You should be able to discuss the miss rates that you were seeing and comment on any trends that you are able to discern. You should use the attached hw9_template.xlsx file as a template to fill in the results as well as the discussions of your experiments and export it as a PDF.
    b. Homework 9 – Part 2 (Coding Part)".
        i. Submit your instrumented xsort.cc and qsort.cc code as well as the pdf report from Part 1.
        ii. The files xsort.cc, qsort.cc and the exported pdf file from hw9_template.xlsx should all be zipped and uploaded as hw9.zip
        iii. The coding part will be autograded to check the correctness. Please make sure your instrumented qsort.cc and xsort.cc pass correctness tests in the gradescope autograder to ensure the results you get in the experiments are valid.

Grading:
20 Points - Correct code
20 Points - Experiment with sorting algorithm, cache size, block size
10 Points - discussion