COMP25212 – Laboratory Exercise 1
Measuring Cache Performance

Duration: 1 x two-hour lab session.

Resources: Any computer with a C compiler. This script is intended for those using Linux, but if you wish to use Windows, (or Solaris, FreeBSD or RiscOS) you are welcome to do so. But beware – some of the library calls may be different, and you'll need to fix them yourself.

AIMS

To investigate, by running a series of tests, the performance of the cache memory system of the computer.

LEARNING OUTCOMES

- To understand how to use microbenchmarks to investigate particular aspects of computer performance.
- To understand how large differences in performance can result from relatively small changes in workloads.
- To learn how to plot and interpret results and the importance of using an appropriate axis scale.

INTRODUCTION

In this exercise, you will use a real program to measure how the lab computer performs under a variety of memory workload scenarios. This lab has a common deadline for all groups, which you can find on Blackboard. The deadline is, in general, at the end of the week following the lab session (on the Friday at 18:00). This is so that you can get help in the lab and then have some time to complete the exercise. Remember that you must use "submit" in the usual way, through Gitlab, to show that you completed your work by the deadline. All labwork MUST be marked in the following session unless there is a good excuse or it will be zeroed. This is to avoid tedious marking sessions at the end of the course as well as for you to remember the work you did, which is needed for marking as the TAs are instructed to go in detail over your work.

PREPARATION

All of the course labs are in a single Gitlab project where each student works within an individual fork. You will have received an email notification for accessing the repository for your labwork in this course. Once cloned, each lab exercise can be found within its respective directory. For instance for this first exercise, you will find all relevant files (cachetesta.c and answer.txt) in the "ex1" sub-directory.

Please refer to the UG Handbook, section "Developing and submitting with Gitlab" for further information on setting up Git, cloning the repository and finally submitting your work.

*** N.B. You must write your answers for parts 3, 6, 7, 8, 9 and 10 in the answer.txt file or else they will not be considered answered. Feel free to use the writing space below to draft your answers, but make sure you also type them in the file, commit it and push it to Gitlab properly.**

PART 1 – (1 mark)

Compile the file (e.g. make CFLAGS=-O2 cachetesta).
Run it (e.g. ./cachetesta).

Record the final line of output here: ______________________

PART 2 – (1 mark)

Edit the first declaration in the program so that nStructs takes the value one million. Compile the program and run it again.

Record the final line of output here: ______________________

PART 3* – (3 marks)

Why are the times reported in PART 1 and PART 2 different?

____

____

PART 4 – (1 marks)

Record the information reported about the CPU (in /proc/cpuinfo on Linux):

Vendor_id: ______________________

Cpu family: ______________________

Model: ______________________

Model name: ______________________

Cache size: ______________________

PART 5 – (4 marks)

Investigate the effect of varying nStructs on the performance of the cachetest program, paying particular attention to the regions in which performance changes quickly with nStructs. Plot your results with an appropriate scale using the attached grid papers (linear and logarithmic).

PART 6* – (4 marks)

Explain how cache size reported in PART 4 relates to the values of nStruct at which performance changes quickly.

PART 7* – (3 x 2 marks)

Identify three regions of your graph with different performance characteristics and explain each region:

In region 1:

In region 2:

In region 3:

Optional Extras:

PART 8* – (5 marks)

The cachetest program as written simply tests memory reads over different cache working sets. Modify the program so that every iteration also writes into the structure and repeat the performance measurement. How does the characteristic vary? Why?

PART 9* – (4 marks)

The cachetest program as written uses reads to a structure of a certain number of bytes in width. What happens if we double the size of the structure? What happens if we halve its size?

PART 10* – (10 marks)

How could we enhance the cachetest program to determine whether a processor cache is direct mapped, set associative, or fully associative?