

Research-Led Education Case Study: Teaching Experimentation in an Introductory Computer Systems Course

Peter Strazdins,
School of Computer Science,
The Australian National University

February 17, 2009

Abstract

In this paper, we present a case study in *research-led education* (or more precisely *research-based education*) in the course **COMP2300** Introduction to Computer Systems at The Australian National University over the years 2006–2008. Each year, we taught the research skill of experimentation in the form of computer performance evaluation to a cohort of approximately 100 second year undergraduates. The cohort were enrolled in degree programs with varying entry levels.

The research skills involved running computer performance experiments and interpreting their results using the principles of computer design. We found that with sufficient support, these skills could be successfully taught to the majority of students. Approximately 20% of the students however excelled and demonstrated high levels of understanding and creativity by the opportunity presented by research-led education.

1 Context

The course [COMP2300 Introduction to Computer Systems](#) typically has an enrollment of around 100 undergraduate students. The student body comprises of:

- 5% Bachelor of Computing Honours students (UAI ≈ 98 , compulsory course). The [Bachelor of Computer Science](#) was introduced in 2006 as a ‘research-oriented’ degree. Hence, the experiences of these students in this context are of particular interest.
- 25% Bachelor of Software Engineering students (UAI ≈ 85 , compulsory course)
- 15% (combined degree) Engineering students (UAI ≈ 85 , mostly non-compulsory course)
- 30% Bachelor of Information Technology students (UAI ≈ 75 , compulsory course for most majors)
- 25% other degree students, including combined degrees with BIT (UAI ≈ 75 , non-compulsory course for most majors)

In this course, with an assumed background of only basic computer programming, the students were introduced to the ‘internals’ of computers.

2 What was Taught

Experimentation in the form of computer performance evaluation is an important research skill in computer systems. We brought down that skill to the 2000-level undergraduate level in the final assignment for [COMP2300](#) in the years 2006–2008. The memory hierarchy of a modern computer system was studied in these assignments. See [Appendix A](#) for a primer on memory hierarchies.

As well as traditional calculation-style exercises, students were given a sophisticated test program (`cachesim3` in the example below) which could measure memory hierarchy performance in a number of ways. They were given a number of experiments to perform using this test program, and had to infer properties of the memory hierarchy based on the results of their experiments. A typical example would be to progressively increase the size of the data the program accesses and observe and explain the changes in program performance.

Execute the command `cachesim3 -L -k 6 lgC 1 4 2 8`, for $lgC = 15, 16, 17, 18, 19$. Cut-and-paste the output as described above. Comparing your results with Q4, what levels of the memory hierarchy are responsible for each decrease in performance? Briefly explain.

In the above, the results of Q4 determined the relationship between the quantity 2^{lgC} (the amount of data the program accesses) and the capacity of each level of the memory hierarchy. Whenever that amount exceeded a level of memory hierarchy, we would expect performance of the program to drop. A sample answer to the above was:

We see a decrease in performance at $C=2^{17}$, and again at $C=2^{18}$. This, and particularly the latter, is due to L2 cache misses, as $C=2^{17}$

corresponds to the cache size [this is confirmed with the stats from using the -e option].

The role of the TLB capacity is unclear: the expected increase from $C=2^{16}$ to $C=2^{17}$ was noticeable, but the stats from using the -e option indicate that no significant misses are being recorded by the hardware!

```
partch:> cachesim3 -L -k 6 15 1 4 2 8
simulate a 64-way assoc cache of size C with block size 1
for 4 reps cyclically accessing an array of length N=1*C
  for C= 32768, 339.9 cycles/access
  for C= 33280, 349.3 cycles/access
partch:> cachesim3 -L -k 6 16 1 4 2 8
...
```

The number of ‘cycles/access’ indicates the speed of the program. The (remainder of) results for this answer are summarized in the following table:

$2^{\lg C}$	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}
cycles/access (1st)	339	345	431	509	510
cycles/access (2nd)	349	352	433	508	509

As we can see from the above answer, the behavior of a modern computer is complex and does not always follow the broad design principles that the students were given.

A second kind of task would be to optimize the program’s code so that it ran faster than the provided version. This tended to generate a lot of enthusiasm from a sizeable minority of students (and sometimes gave interesting insights into advanced features of computer design).

A third kind of task (bonus question) was to propose their own experiments and gain bonus marks. For example, in 2008 a student investigated the Mobile Pentium II processor in which the caches could be enabled or disabled, and could have different operational modes. He modified the test program accordingly and devised experiments to test the effect of each setting. He followed this up by investigating the performance of an image analysis program. There were also a number of highly enthusiastic postings on the course forum related to this question. For example, an excerpt from one posting is as follows

> Also I believe the core architecture has some pretty aggressive prefetching support.

There is a hardware prefetcher in Northwood too, though the algo is different. The P4 hardware prefetcher is actually hilarious, it was the subject of my bonus mark experimentation. Works well when it works but it’s easily confused. You can get some huge performance fluctuations depending on the access pattern (quite apart from those due to normal cache usage).

In the 2006 assignment, there was also a place for students to write comments on their experience of completing this part. The assignment from this year was an extension of research recorded in a [conference paper](#) that we previously co-authored.

The 2008 assignment document can be found from:

<http://cs.anu.edu.au/student/comp2300-2008/assign/A3.php>

The test program used was actually a simulator program for the memory hierarchy itself! This presented further opportunities to introduce research-led education. The supporting tutorial exercise involved predicting the behavior of cache memory with a random replacement policy¹. While the questions seemed simple, the correct answers for some were not obvious and required an advanced application of mathematics to solve. A number of such questions were posed to students early in the 2008 semester as a non-assessable challenge (The Random Cache\$ for Beer Challenge!).

3 Evaluation

In 2006, 98 students (out of 120) attempted this part of the assignment with an average mark of 10.6 / 20. This can be compared with an overall average assignment mark for the course of 65%. The low student performance can be attributed to the following factors:

- it was huge challenge to even the strong students, in that they needed to think deeper and in a different way about computers than before. About 1/4 of the students excelled and enjoyed the experience.
- only the stronger and more organized students were excited. The other students generally made a half-hearted attempt at the last moment (most were still struggling with the first part of the assignment by the submission date).
- many of the students felt lost, and found the experience frustrating. The main opportunities for support were at the tutorials, but this was not used. Some motivating background was given at a lecture, but there was no attempt to teach how similar problems could be worked through.

The above observations came from reading a sample of a third of the experience survey question in the assignment, plus our observations as lecturer and tutor.

From the teacher's point of view, it was a lot of work. This was mainly in setting up the infrastructure so (naive) students could perform the experiments and get reliable results, and in working out and providing the background knowledge needed. Note that the infrastructure is specific to the computer being studied; with a new (state-of-the-art) computer being studied in each of the subsequent years, setting up the assignment remained a lot of work. The unexpected problems encountered also resulted in the delayed release of the assignment document (in all 3 years!).

We concluded in 2006 that more work, support and care are required if this kind of teaching can be made successful to a student body of largely non-elite students at the early 2000-level.

Accordingly, with the assignment in 2007, we introduced some tutorial support in treating the underlying concepts, and made the assignment submittable on its own. This resulted in improving the submission rate to 90% and the average mark to 20.0 / 32, indicating the above problems can be to an extent addressed. However, it should be noted that there were still a number of 'half-hearted' submissions which pulled down the average mark. These in part can be attributed to the submission date being late in the semester and the overall weighting of the assignment being relatively small.

¹See Question 5 in <http://cs.anu.edu.au/student/comp2300-2008/labs/TuteLab09.php>.

We provided slightly more support in 2008, with the concepts being introduced earlier in lectures. Also, as mentioned above, the assignment theme for the semester was a memory hierarchy simulator, where the concepts were reinforced to the students implementing such a program. Despite this, the results were similar to 2007, the average mark was 20.4/32 with an 85% submission rate. As in previous years, the distribution of marks was roughly bimodal. For the ‘bonus’ questions, 8% of the students answered the ‘propose your own experiment’ question; 9% attempted the advanced code optimization question.

An analysis of the students’ open-ended questionnaire from 2008 gave no direct references to this assignment. There were however three responses that the assignments were particularly interesting, and the same number that the assignments were too hard.

3.1 Comments from the BCS Students (2006)

We interviewed 4 BCS students in September 2006 as part of the requirements for the [Graduate Certificate in Higher Education](#). The interviews were on their views of research-led education and their degree.

When asked what examples of what could be termed research-based education they had encountered so far (they had all encountered by this stage the courses COMP1130/40² and COMP2300), three cited the COMP2300 memory hierarchy assignment as the main example, but all mentioned COMP1130/40 to a small extent fitted such a criterion.

Comments on the COMP2300 memory hierarchy assignment include it was much more interesting than a standard assignment (that is “on a well-trodden path”), and very challenging in that a deeper understanding had to be built up in order to answer the problems. Their approach to learning in such a situation was different in they had to actively seek way of building their understanding on the topic first. One of these said such an exercise would have been improved if provided with more resources; this was seconded by the student who did not cite this as an example, who also said it was important to review the solution after the assignment was handed in. This student also echoed the comment made by the others that research-based education is potentially more engaging and satisfying (but only if you could arrive at a satisfactory solution).

4 Conclusions

It is possible to successfully teach the research skills involved in experimentation in the area of computer systems to a second year undergraduates. This was even in the case where the student cohort is moderately large and not highly selective.

However, to do this successfully requires some effort for support in both lectures and tutorials, as the underlying concepts in this case are advanced and often subtle. The majority of students were then able to successfully complete the central tasks of performing experiments and explaining the results according to the theory (or, in a few cases, identifying where the results do not match the theory). These students all learned to think about computers in a different way from this experience. About 20% of the students, being conceptually stronger and more curious, however, excelled from the experience, showing great levels of creativity and understanding. In 2008, this also led to the opportunity to pose related research-level problems to the students as a further (non-assessable) challenge.

²This was taught as a ‘research-seminar’ course in 2006.

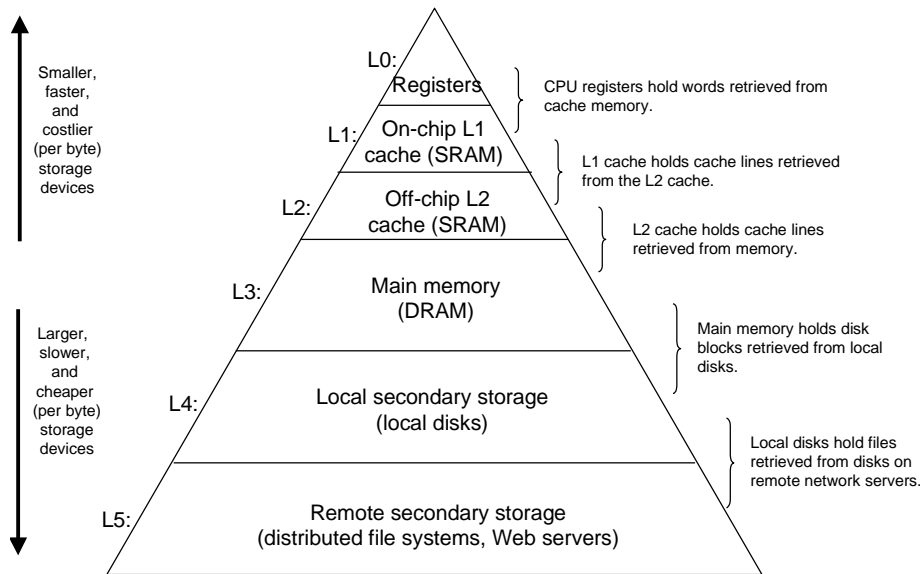


Figure 1: Schematic of Memory Hierarchy, courtesy of *Computer Systems: A Programmer's Perspective*, by Randal E Bryant and David O'Hallaron, Prentice-Hall 2003.

While each year, this essay in research-led education required a very significant amount of teacher effort and care to set up, we feel it has been very worthwhile, as it was both enriching for both the teacher and students.

A Background: What is a Memory Hierarchy?

In a computer, instead of a single level of memory, there are several levels of memory (a *memory hierarchy*) due to the existence of many different mediums for the storage of data with a *trade-off* between speed and capacity.

i.e. fast memory (mediums) tend to be small; large memories tend to be slow.

In the hierarchy, we store recently accessed data from main memory in smaller, faster memory closer to / inside the Central Processing Unit (CPU). This is illustrated in Figure 1. In particular, between the main memory and the CPU there are typically two levels of *cache memory*. Provided a computer program's recently accessed data fits within these upper levels, there will not be the performance penalty of having to (frequently) accessing the next lower level.

The performance of programs running on a modern computer is thus critically affected by how well it can take advantage of the memory hierarchy. In particular, if the size of data that the program accesses increases beyond the size of (say the level 2 cache memory), we expect its performance to decrease.