

# Work in Progress: A Visual Cache Memory Simulator

Danial Neebel<sup>1</sup>, Chris Augeri<sup>2</sup>, Gordon MacMillan<sup>3</sup>, Leemon Baird<sup>3</sup> and Adrian de Freitas<sup>3</sup>

**Abstract** - Cache memory performance analysis is a challenging topic upon first introduction. Students must synthesize a significant amount of computer architecture knowledge, comprehend reasonably complex replacement strategies, and analyze performance. We propose a programming exercise that has students develop a visual cache memory simulator and then use the simulator to analyze several memory reference trace files. Our student learning assessment measured the quality of each team programming exercise solution and each individual's own cache performance analysis. In addition, the final exam has several questions related to cache memory. Early results indicate students achieve a better understanding of cache memory and its impact on performance.

*Index Terms* – Cache Memory, Performance Analysis, Computer Architecture, Programming Project.

## INTRODUCTION

Our students study computer architecture in the spring semester of their sophomore year. These students have declared the computer science major and have completed two computer science courses – *Introduction to Computer Science* and *Computer Programming I*. Students also take *Introductory Digital Systems* from the Electrical Engineering department introducing them to 2's complement arithmetic, basic logic gates, and logic design. Students take their first data structures course, *Computer Programming II*, in conjunction with computer architecture. Since the architecture course is early in the program, most students are still in the process of developing strong programming abilities. This is the first course students develop in a language other than Ada.

The CS program meets the ACM 2001 curriculum guidelines [4], with computer architecture requirements split 30-50-10-5-5 (%) among digital systems, computer architecture, operating systems, networks and compiler courses respectively – more detail is in [1]. The six hours of cache memory material are organized as an overview, direct-mapped cache design and analysis, *m*-way associative design, overall performance analysis, a short introduction to virtual memory and the roles that cache, main memory, and swap space play in the memory hierarchy. Abstraction and in-depth detail are balanced to help students understand each piece without getting bogged down in details.

## PROGRAMMING EXERCISE DESIGN

One of the issues with the many cache simulators often presented is that they either are command-line based or do not

permit large-scale cache analysis [2-3, 5-7]. In this paper we describe a team-based programming exercise (PEX), a cache simulator visualization, to analyze cache performance. We designed the exercise to include a graphical user interface (GUI) with a window showing the cache status. Of course, the simulator must have the ability to produce the requisite hit/miss data for a given trace. Each trace file used contained 1,000,000 memory accesses. Our cache simulator exercise is the last of a four-part PEX series with the first three PEXes and other course information extensively described in [1].

The requirements for the simulator include single-stepping and high-speed analysis of a given memory reference trace file. The simulator must display the individual fields for each memory reference. A visual detailed and summary representation of hits, initialization misses, and collisions completes the requirements. To aid the students with this exercise, we have them work in groups of three, provide a simulator source code template, and have a compiled executable instructor solution available. The template is structured so that students work on the heart of the cache algorithm without having to design a GUI from scratch.

## PROGRAMMING LANGUAGES AND OTHER TOOLS

The primary programming language for the US Air Force Academy computer science program is Ada, the only computer language used in the first two years of the Academy program. Tools such as RAPID [8] and AdaGraph [9] are utilized to make it easy to develop the user interface. Students analyze benchmark programs with provided traces and gather traces from their own code using the Ada debugger and command line pipes. Within their teams, students generate traces of their “best” SPIM and Ada programs from the previous programming exercise. This further increases their exposure to command-line filter facilities early in their academic career. In addition, the Ada trace facility forces students to use a debugger, something they often like to avoid.

Figure 1 on the next page shows the interface and robust functionality of our simulator. Once students have added their own source code, this functionality includes cache analysis, logging, single-step, full speed and hi-speed simulation. The full speed simulation is interruptible at any time and the high speed simulation means that the GUI updates only once after the simulation is finished. Single-stepping is used for debugging / hand-tracing, while the hi-speed mode is used on large files. The user may select associativity, number of sets (cache lines), block size, and word size. The display reports the progress of the simulation as well as the various fields of the address, i.e. tag, set number, block offset, and byte offset.

<sup>1</sup> Danial J. Neebel, Loras College, Dubuque, IA, Danial.Neebel@loras.edu

<sup>2</sup> Christopher J. Augeri, Air Force Institute of Technology, chris.augeri@afit.edu

<sup>3</sup> Gordon B. MacMillan, Leemon C. Baird III, and Adrian de Freitas, United States Air Force Academy

CONCLUSIONS

We have described a cache simulator exercise that visualizes cache behavior in a team environment. We are still in the process of collecting data to see if this project significantly increases a student’s ability to understand cache memories.

One of the future extensions involves an additional visualization showing a mapping from the memory address to the cache address with the line color indicating a hit or miss as shown in Figure 3 below. This implementation is done in the Java programming language using an open-source chart and plot package. We are developing additional visualizations, but they are not presented here due to space limitations.

Another extension is a user help file that explains the program, the functions of the various buttons, displays, etc. Students would expand the help file to demonstrate their understanding and promote other users’ understanding.



FIGURE 1  
USER INTERFACE DURING SIMULATION

Figure 2 below shows the cache memory map visualization in mid-simulation. Various colors are used to promote understanding, e.g., gray indicates not initialized, green signals a hit, and red marks a collision.



FIGURE 2  
CACHE MEMORY VISUALIZATION (2-WAY SET ASSOCIATIVE)

Since the students have only just been introduced to GUI programming in their concurrently taken data structures course, we provided them a GUI template that they modified. While the internal cache analysis piece was similar to previous offerings, the GUI component was new, and hopefully more instructional. The GUI version identified each memory field (tag, block offset) and the most recent result (hit, collision, etc.) for each block, along with the overall cache performance. For extra credit, the student teams could incorporate additional GUI components of their own design into the simulator.

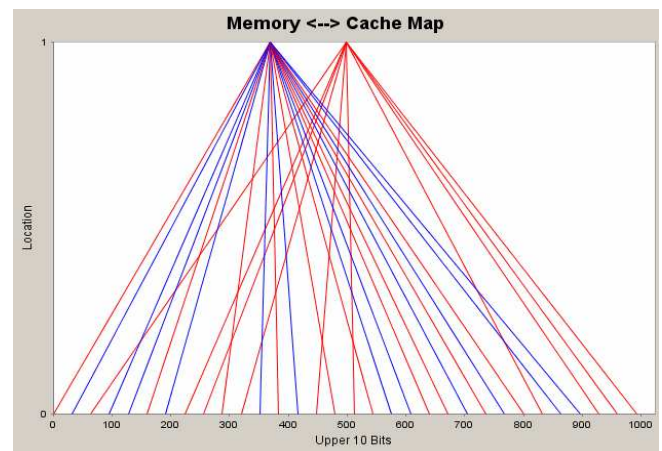


FIGURE 3  
MEMORY - CACHE LOCATION MAPPING VISUALIZATION

ACKNOWLEDGMENT

The authors gratefully acknowledge all the cadets enrolled in CS 351 over the past three years. We thank Steve Hadfield and Barry Fagin for encouraging us to write this paper.

REFERENCES

- [1] Augeri C., Neebel, D., Baird, L., and de Freitas A., “UAV Communications: Integrating a Real-World Scenario with Computer Architecture”, *Frontiers in Education Conference*, Oct. 2005.
- [2] Rotithor, H.G., "On the Effective Use of a Cache Memory Simulator in a Computer Architecture Course", *IEEE Trans. on Education*, Vol 38, No 4, November 1995, pp. 357-360.
- [3] Grünbacher, H., “Teaching Computer Architecture /Organisation Using Simulators”, *Frontiers In Education Conference*, 1998, pp. 1107-1112.
- [4] Computing Curricula 2001 Task Force, *ACM Computing Curricula Guidelines*, ACM, <http://www.computer.org/education/cc2001/>, 2001.
- [5] Edler, J. and Hill, M., *Dinero IV Trace-Driven Uniprocessor Cache Simulator*, <http://www.cs.wisc.edu/~markhill/DineroIV/>, 2003.
- [6] Shivakumar, P. and Jouppi, N., *CACTI 3.2*, <http://www.research.compaq.com/wrl/people/jouppi/CACTI.html>, 2005.
- [7] Null, L. and Rao, K., “CAMERA: introducing memory concepts via visualization”, *SIGCSE Bulletin*, Jan 2005, Vol. 37, No. 1, pgs. 96-100.
- [8] Carlisle, M., “A Truly Implementation Independent GUI Development Tool”, *SIGAda Letters*, Vol XIX, No. 3, Sept. 1999, pp. 47-52.
- [9] Carlisle, M, et al, *AdaGraph*, USAFA, [http://www.usafa.af.mil/dfcs/bios/mcc\\_html/ada\\_stuff.html](http://www.usafa.af.mil/dfcs/bios/mcc_html/ada_stuff.html), 2001.