Cache Memory Design for Embedded Systems Based on Program Locality Analysis

Roberto Giorgi   Cosimo Antonio Prete   Gianpaolo Prina

Dipartimento di Ingegneria della Informazione
Università di Pisa - Italy
{giorgi,prete,prina}@iet.unipi.it

Abstract

Cache memory design in embedded systems can take advantage from the analysis of the software that runs on that system, which usually remains the same for its whole life. Programs can be characterized, in respect of the memory hierarchy, using locality analysis. We propose an environment which permits to analyze the locality of a program and the effects on the target system performance. The student can thus figure out the best tradeoff between costs and performance for cache, memory and timings exploring different system configurations. A fully graphical interface permits to observe the program behavior from many points of view: locality surface, working set evolution, performance metrics. The tool is currently used as a teaching tool at our University and it is distributed as part of a commercial development environment for embedded systems.

1 Introduction

In this paper we present how a designer can exploit the knowledge of the program running on an embedded system for choosing the best configuration of cache (associativity, block-size and capacity), memory (access time and dimension), microprocessor and speed, for that program. The approach makes use of a software environment used in Computer Architecture courses at the University of Pisa. We will show that the student can extract from the program much significant information to carry out the actual design activity of embedded application oriented systems [2]. The environment has been made up by our University and is integrated in a toolkit (JumpStart) distributed by VLSI Technology, Inc., for the design of ARM-based applications. (ARM [6] is a 32-bit microprocessor designed by ARM Ltd. and largely used in embedded products. It uses RISC technology and a fully-static design approach to obtain both high performance and very low power consumption.)

2 The Didactic Environment

Actual computer systems and/or commercial design tools are generally not suitable to be used as didactic tools and to present the basic concepts of architecture design in both basic and advanced Computer Engineering courses. Their structure is often too complex and usually prevents the detection of all the events occurring in the activity of the machine; moreover, the high number and frequency of these events may require a too expensive acquisition system, or also, several events are not directly observable, since they occur within the chips. As a result, one cannot generally obtain an accurate, step-by-step observation of the internal events occurring in a system. From the designer perspective, tuning an embedded systems and in particular its cache memory to achieve lower cost and higher performance, is a difficult task. The designer has to rapidly find out if a cache is necessary, and if so, to choose the optimal values for its parameters.

All these reasons encouraged us to develop a new environment which could combine the different needs of the potential users.

As a mere didactic tool, the environment offers the student a wide range of opportunities to investigate the structure and the behavior of a cache memory, starting from the basic concepts and definitions up to a relatively complex level of depth. The concept of program locality is particularly emphasized, since it is one of the critical issues in this computer architecture branch.

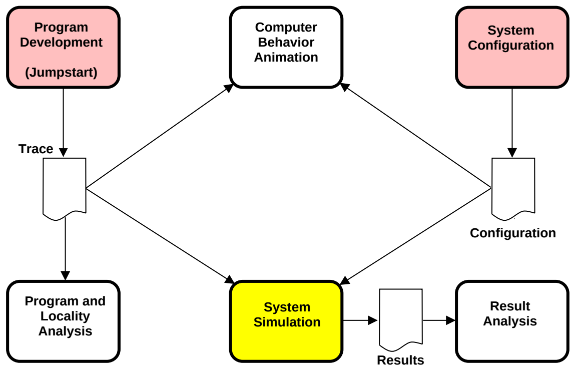

Program Development (Jumpstart)
Computer Behavior Animation
System Configuration
Trace
Configuration
Program and Locality Analysis
System Simulation
Results
Result Analysis

Figure 1. Structure of the software environment.

As a design tool, the environment allows to project embedded applications within a proper software development environment (JumpStart), and to perform a trace-driven simulation [5, 4] and performance evaluation of the system.

Possible paths for approaching the design and tuning of target system are showed in figure 1, both from designer and student point of view.

3 The Program Locality Analysis

In a didactic approach to computer architecture, one of the key concepts that the student has to deal with is program locality and how it can influence the choice of a proper cache structure in order to maximize the system performance.

In the *Program Development* phase, the user can build an application, debug it and produce a trace file to perform a detailed program locality analysis (getting number of unique blocks, locality surface, spatial locality).

A quantitative approach to locality analysis was first proposed by Archibald *et al.* with the use of the *locality surface*. They proposed a 3D graph where reference stride and distance are the base axes. Due to limited space we don't introduce details on this topic, which can be found in [3]. We highlight the fact that the designer can produce graphs like the one in Figure 2, in which we can observe information about locality features like *sequentiality* due typically to code fetching, *striding* typical of numerical algorithms such as matrix operations, *temporality* and *loops*.

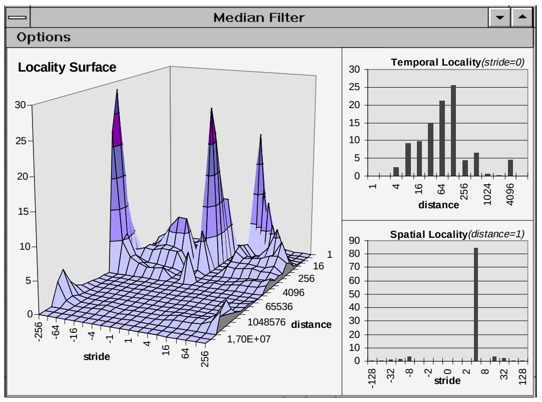
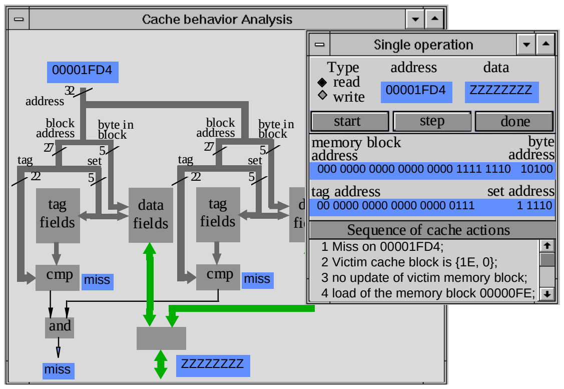


Figure 2. The locality surface of a program.

Another graph which gives insights of program behavior is the one that shows the number of unique blocks for a given number i of references. These blocks only cause misses in an infinite cache, the number of such blocks delineates a low bound for the miss ratio. This kind of graph shows also the evolution of the working set as the program runs.

The next step in a didactic path is the *System Configuration* phase in which the user find out how the presence of a cache memory can exploit program locality to improve the system performance. All cache parameters like replacement policy, timing, sizes can be selected. In the *System Simulation* phase, simulations are then carried out and in the *Result Analysis* line charts and bar diagrams for miss ratio, execution time and other metrics of interest can be examined.



Type: ◆ read ◇ write
address: 00001FD4
data: ZZZZZZZZ

memory block address: 000 0000 0000 0000 0000 1111 1110
byte address: 10100

tag address: 00 0000 0000 0000 0000 0111
set address: 1 1110

Sequence of cache actions

```
1 Miss on 00001FD4;
2 Victim cache block is {1E, 0};
3 no update of victim memory block;
4 load of the memory block 00000FE;
```

Figure 3. Step-by step operation.

Finally, in the *Cache Behavior Analysis* environment can be employed to perform the step-by-step simulation and review what is happening in great detail (Figure 3).

4 The Approach to Design

In the design of embedded system architectures, a key point is the optimization of each component, which should fit as better as possible with the behavioral features of the application for which the whole system is designed.

The designer should provide the timing for read and write operations of the ARM core. A specific window shows the ARM timing plot, derived from the data book, in order to aid the designer in finding the proper values. The simulator models a generic bus, which is capable to accommodate the typical memory operations. The designer has to specify the data bus width and the time for each type of bus operation. Then he specifies the features of memory and I/O devices.

If this system simulation shows that, without cache memory, the application takes too much time to execute the program, the designer can add cache memory, define the cache parameters and again execute a parametric simulation to find out an optimal choice. The designer can now try different solutions for the main memory by executing simulations in which the varied parameter is the RAM bank access time.

Another graph can show the silicon area cost of the chosen configuration by using a metric proposed in [1].

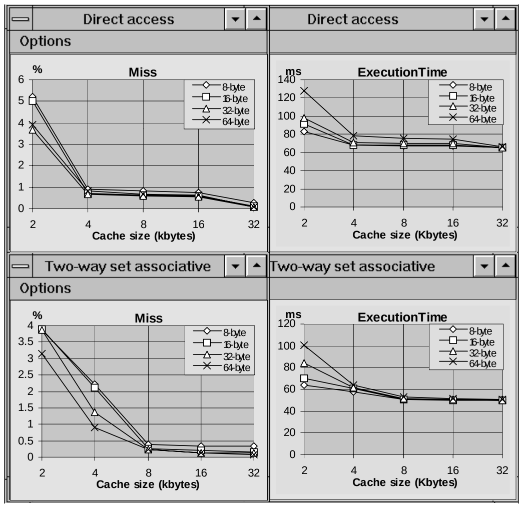


Figure 4. Miss ratio and execution time.

Writes and miss ratios affect system performance in a different way. A lower miss ratio magnifies the influence of write operations on global performance. In this case, the choice of an optimal update policy becomes critical. An example of output metrics for an audio filtering program is shown in Figure 4.

Acknowledgments

This work was supported by the Ministero della Università e della Ricerca Scientifica e Tecnologica (MURST), Italy and by VLSI. We wish to thank Francesco Lazzarini that took part in the development of the cache simulator, Angelo Rappelli that contributed in developing the graphical interface for the Microsoft Windows version, Massimiliano Panico that conducted performance evaluations of the case studies.

References

[1] M. J. Flynn. *Computer Architecture, Pipelined and Parallel Processor Design*. Jones and Bartlett Publishers, 1995.

[2] D. D. Gajski and F. Vahid. Specification and design of embedded software-hardware systems. *IEEE Design & Test of Computers*, 12(1), Spring 1995.

[3] K. Grimsrud, J.Archibald, R. Frost, and B. Nelson. Locality as a visualization tool. *IEEE Transaction on Computers*, 45(11):1319–1326, Nov. 1996.

[4] F. Lazzarini, C. A. Prete, and M. Graziano. Tuning the configuration of a cache memory for embedded systems. to appear in IEEE Micro.

[5] C. A. Prete, G. Prina, and L. Ricciardi. A trace-driven simulator for performance evaluation of cache-based multiprocessor system. *IEEE Transaction on Parallel and Distributed Systems*, 6(9):915–929, Sept. 1995.

[6] A. V. Someren and C. Atack. *The ARM RISC Chip, A Programmer's Guide*. Addison-Wesley, 1993.