An Educational Environment for Designing and Performance Tuning of Embedded Systems

Roberto Giorgi and Cosimo Antonio Prete

Dipartimento di Ingegneria della Informazione
Università di Pisa, Via Diotisalvi 2, I-56126, Italy

I. Introduction

TEACHING how to design and tune an embedded system is a not-easy task, since the student has to learn the many trade-offs that lead to the final system project.

A typical design path starts from the definition of the hardware/software requirements needed to implement the specified function through the embedded system [1]. After that, the designer usually has a prototype program and a prototype hardware configuration that has to be tuned in order to meet product requirements concerning power consumption, cost or speed. The tuning of the system relies on a good knowledge about the memory hierarchy behavior and the program locality effects on that hierarchy.

For example, low power consumption and low cost requirements suggest the adoption of an off-chip slow memory, so that designers often turn to on-chip cache memories to both provide high processing power and allow using large slow main memory. In this case, the designer may wonder if a cache memory is necessary, and if so how to choose an optimal configuration both in performance and cost.

Existing tools to accomplish this task are often too complex, or do not stress the basic steps in the design path. These steps should also be learned during first training sessions of students. All these reasons motivated us to develop a new tool (Csim2) which could combine the different needs of student and the actual designer.

II. The Csim2 environment

This environment allows the student to learn about the concepts of system architecture, performance tuning, program locality, and cache structure, while analyzing the behavior of the program that has to be tied with the embedded system.

The student is actively involved in making authentic choices that affect the target system, such as changing the architecture parameters and analyzing the immediate response of that system to the user actions; otherwise, elaborate graphics and simulations may result not effective.

As a design tool, Csim2 permits: i) to design an embedded application within a proper software development environment; ii) to explore different strategies for the memory hierarchy, including different level of caching, split caches, write-buffering, by means of a graphical design tool; iii) to carry out the performance evaluation, in order to choose the system configuration which can guarantee the best performance for the target application. Focusing on these simple crucial steps of system design, we foster the student attitude to select appropriate memory hierarchy and tune the influencing parameters of an embedded system.

Second level optimizations include the investigation of more sophisticated techniques like *selective cacheing* to cache only particular memory areas, *cache locking* to leave some data in cache thus avoiding replacement, *scratch memory* which is a small on-chip memory for allocating frequently used data, *code mapping* for reducing conflict cache-misses. All these techniques are supported by adequate tools that help the designer to select the best strategy for the application code and data [2]. Etherogeneous multicore architecture is also permitted, thus allowing the designer to try combinations of processors and DSPs.

To help the students, Csim2 has a very friendly point-and-click graphical interface, by which the teacher can easily show and discuss, using practical examples, the basic concepts of cache architecture and behavior (Figure 1).

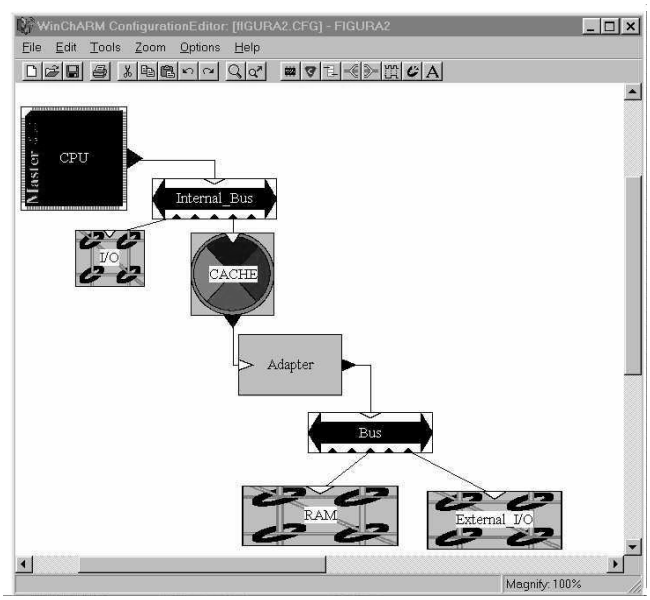

Fig. 1. Embedded system configuration example. The system includes an ARM core, an internal I/O device, a cache memory, a bus adapter, and an external RAM and an external I/O device. The configuration is produced by means of the configuration editor of the environment.

The system design and tuning procedure can be organized in the following four phases, as shown in Figure 2.

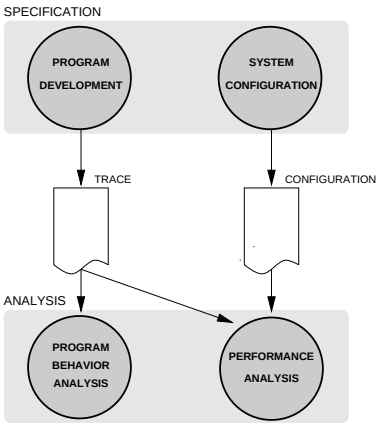

Fig. 2. Structure of a Csim2 session.

In the *Program Development* phase, the user builds an application, debugs it and records the program behavior in a file (*program trace*). Applications can be executed and debugged on a dedicated ARM instruction set simulator or loaded in an ARM CPU-based board for a native evaluation. Once that the application has been developed, the user can generate the trace file by simply pointing-and-clicking while the program is running in emulator mode.

In the *System Configuration* phase, the user defines the system architecture and the features of each component. The user first draws the schematic of the system architecture at functional level. The system may include the following components: an ARM core, cache memory that can be combined in different levels or in split instruction/data architecture, a system bus, memory banks, and a number of I/O devices (Figure 1). For each component, the designer has to specify the timing and the other custom parameters.

The *Program Behavior Analysis* phase allows the student to perform two types of program analysis. The first one uses traditional program statistics such as the percentages of data/code, read/write accesses. The second one regards program locality. An accurate knowledge of locality features plays a crucial role in understanding cache concepts. The locality graphs include: the number of unique blocks, the locality surface [3], and the spatial locality (Figure 3).

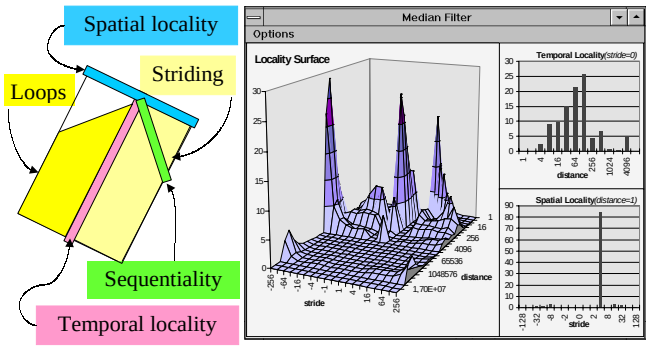

Fig. 3. The locality surface for median filter program. From this surface, the designer can derive information about locality features like *sequentiality*, *striding*, *temporality*, and *loops*.

The *Performance Analysis* phase allows the student to plan, perform a single simulation or a performance evaluation *experiment*, and analyze the results. An experiment is defined by: i) the trace file; ii) the system configuration; and iii) the varying parameters (one or two). The results consist of: *global system performance* (execution time, lost time in waiting, and word transfer ratio); *cache behavior* (miss, code miss, data miss, read miss, data read miss and data write miss ratios and cumulative cold misses); and *bus traffic* (occupation rate, number of read-block operations, number of write operations for write-through cache models and number of update-block operations for copy-back cache models).

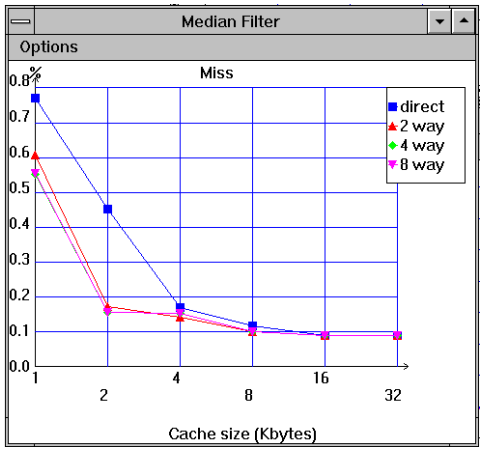

Fig. 4. Miss percentage for median filter program. For this program, a 2-Kbyte 2-way set associative cache is recommended since it supplies the best balance between cost and performance.

As an example of output statistics, figure 4 shows the miss percentage of the median filter program as a function of the cache size and degree of associativity. We notice that a 2-way set associative cache is a convenient configuration for 1-KByte and 2-KByte caches and also that a 4-way cache produces quite the same result. For a 4-Kbyte or grater cache size a direct-access cache can be adopted without any miss rate degradation. At a more detailed level of analysis, the student is called to search, for a given cache structure, the cache and block sizes which can provide the best results in terms of global performance. Again a new graph can be produced (like the one in Figure 5) to find the best value for the cache block size.

III. An Example of Performance Tuning

In the design of embedded systems, a key point is the optimization of each component, which needs to meet, as better as possible, the specific application for which the whole system is designed. We are going to present an example of design training path, and we will show how Csim2 can help a student to find out the optimal cache and system configuration for a specific application. We consider the `cjpeg` program, a `jpeg` image compression/decompression tool which is frequently used in commercial embedded systems.

First, the student should wonder the following question: for a 20 MHz ARM running an application using the `cjpeg` program, is it necessary to add a cache memory in order to achieve the required performance? We suppose that the product requires that the image compression be completed in less than 400ms. First, the student traces the execution of the `cjpeg` program while compressing an image. The source image is a 101-KByte bitmap consisting of 227x149 pixels. Since performance may vary with the program input, the student has to choose the input leading to worst case performance.

Then, the student defines the system configuration including a 20 MHz ARM core, a system bus, a 1-MByte memory DRAM bank, a 128-KByte memory PROM bank, and a memory-mapped graphical I/O device. The student also specifies the timing of each component. In order to obtain a low cost and a low power-consumption solution, a slow system bus and slow memory devices are selected.

With the system configuration just examined, a simulation shows that, without cache memory, the system cannot meet the execution time requirement. The addition of a cache memory proves to be necessary, therefore, to meet the time requirement. In this case the execution time falls below 700ms (Figure 5).

The student can now execute a parametric simulation in order to search the optimal cache configuration. Figure 5 shows also the miss ratio and the execution time versus block size (from 8 to 64 Bytes) and cache size (from 2 to 32 KBytes) for two cache configurations. The first cache is a simple write-through, direct access cache without a write buffer; the second one is a more complex copy-back, two-way set associative cache with a two-entry write buffer. The cache uses the LRU technique as replacement policy.

The designer can observe that, in both configurations, execution time and miss ratio exhibit different values and behaviors. In this way the student can select the configuration that best meets cost-effectiveness and performance requirements (execution time $\leq$ 400ms). For example, an optimal choice is a 8-KByte, write-through, direct access cache with 32-Byte block size without a write buffer.

Now, the student can also answer the question: for the selected cache configuration, which is the cheapest main memory meeting the time requirement? The designer can thus execute simulations having the RAM bank access time as parameter. The simulation permits to find out that the memory bank delay can be increased by no more than 30ns with respect to the values specified in the configuration (not shown here).

Of course, if the student uses the same design path for a different embedded application, he/she may find out a different cache and system configuration.

Csim2 was designed at our University [4]. The full version is part of a toolkit (JumpStart) distributed by VLSI Technology, Inc., for the design of ARM-based applications. (ARM [5], 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.)

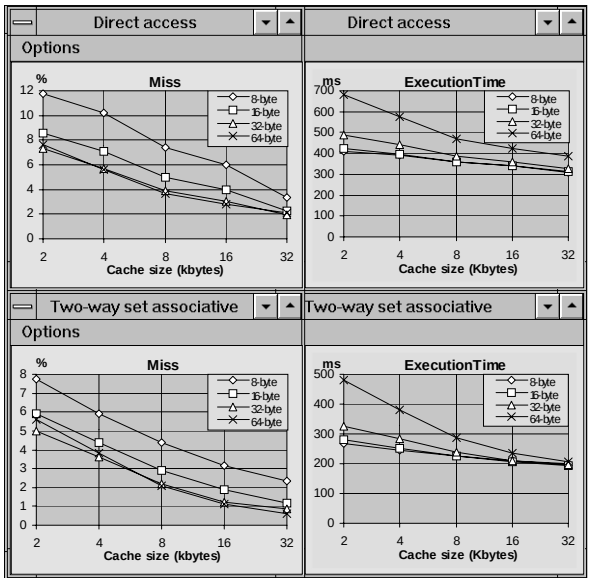

Fig. 5. Miss ratio and time execution for `cjpeg` program. The graphs permits to decide whether the cache memory is necessary and to find out the optimal system configuration meeting the requirements.

We believe that this educational environment based on a trace-driven system simulator can help students in the design activity of cache memory to be employed in embedded systems. The growing demand for embedded products requires highly sophisticated computing functions. Designers must select the most efficient cache/system configuration in order to resolve complex – even conflicting – requirements for low-power/high-speed and component cost. This makes accurate and reliable system/cache memory simulation and performance analysis crucial.

References

[1] D. D. Gajski and F. Vahid, "Specification and design of embedded software-hardware systems," *IEEE Design & Test of Computers*, vol. 12, no. 1, Spring 1995.

[2] S. Lorenzini, G. Luculli, and C. A. Prete, "A fast procedure placement algorithm for optimal cache use," in *Proc. MELECON'98, Tel Aviv, Israel*, May 1998, pp. 1279–1284.

[3] K. Grimsrud, J.Archibald, R. Frost, and B. Nelson, "Locality as a visualization tool," *IEEE Trans. Computers*, vol. 45, no. 11, pp. 1319–1326, Nov. 1996.

[4] Cosimo Antonio Prete, Marco Graziano, and Francesco Lazzarini, "The ChARM tool for tuning embedded systems: Selecting and tuning system configurations to meet cost, performance, and power consumption requirements," *IEEE Micro*, vol. 17, no. 4, pp. 67–76, July/Aug. 1997.

[5] D. Jaggar, "Arm architecture and systems," *IEEE Micro*, pp. 9–11, July 1997.