An approach for investigating design and tuning performance of embedded systems

R. GIORGI C. A. PRETE G. PRINA

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

Abstract

Showing the internal behavior of a computer and the didactic path which conducts to the final design of an embedded system is often a difficult task, without adequate tools. Classical approaches may skip details of the underlaying architecture which can be fundamental to meet particular timing or consumption requirements. We propose an approach based on an environment which allows a high level of detail to be simulated, including cache, memory and I/O subsystems, but also permits an immediate feedback to the designer for understanding the behavior of embedded programs and fine tuning of system performance. The designer is guided in this process through simple steps based on a graphical interface and animations.

INTRODUCTION

Computer systems are often too complex to present the main concepts of architecture design in both basic and advanced Computer Engineering courses. Also, many internal events happen inside the chips, and cannot be showed using any kind of probe. Thus, the use of simulators, integrated environments, and graphical tools is a well accepted methodology to explain the internal behavior of such systems [1]. This approach proves particularly important in respect to the emerging co-design technologies for the development of embedded system hardware and software [2]. In that case, the design process can be quite complex, because it has to face a number of various and often conflicting component requirements: cost effective performance through small die size and low power consumption, particularly for mobile systems, but also high processing power (real time applications) [3]. The adoption of slow memory devices suggests using on-chip cache memories to provide the required processing power.

Tuning such a system, and in particular its cache memory, requires the analysis of many details that critically influence system performance, while the designer has to quickly decide which is the best cache configuration tradeoff.

In this paper we present a software environment used in Computer Architecture courses at the University of Pisa and derived from a product developed for VLSI Technology Inc. (San Jose, CA), for ARM-processor [4] based applications, by the same University. Using this environment

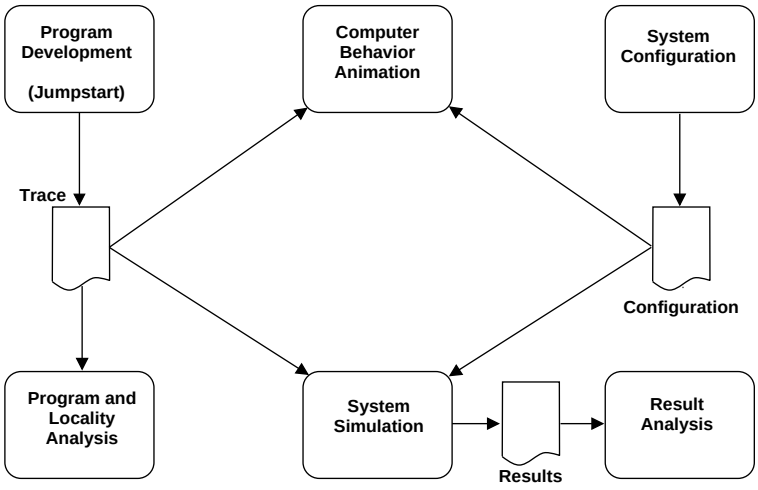

Figure 1. Structure of the programmer workbench.

a designer can effectively design and tune an embedded system, choosing the right configuration of cache (associativity, block-size and capacity), memory (access time), and microprocessor speed, for the program that will run on the system probably for its whole life.

THE PROGRAMMER WORKBENCH

Whilst in traditional programming environments, the designer has to deal simply with program code without caring about the underlaying memory hierarchy, embedded system design workbenches should include facilities to accurately model the system as a whole mix of hardware and software components. In our approach, we find it useful to put several tools beside a traditional software development environment (*Jump-Start*, commercialized by VLSI Technology, Inc.), that complete the environment in the above sense. These tools use the program trace, extracted from the embedded application, to give a feedback to designer regarding some interesting metrics such as the execution time on different, not yet built, prototypes of the embedded system. Thus, the designer can select the appropriate parameter values which allow to meet timing, consumption and cost requirements, as it will be showed in the following. Figure 1 depicts a scheme of the design phases.

In the *Program Development* phase, the user can build an application, debug it and produce a trace file. Applications can be executed and debugged on a dedicated ARM instruction set simulator or loaded in an ARM-based board for native evaluation.

In the *System Configuration* phase, the user defines the system architecture and the features of each component. For each component, the designer specifies the timing, the architecture and the management policy.

The *System Simulation* phase allows the user to plan and perform a single simulation or a performance evaluation *experiment*. An experiment is defined by: i) the trace file; ii) the system configuration; and iii) one or two parameters to be varied and the list of values that those parameters have to assume.

The *Results Analysis* phase yields line charts and bar diagrams to show: global system performance (execution time, lost

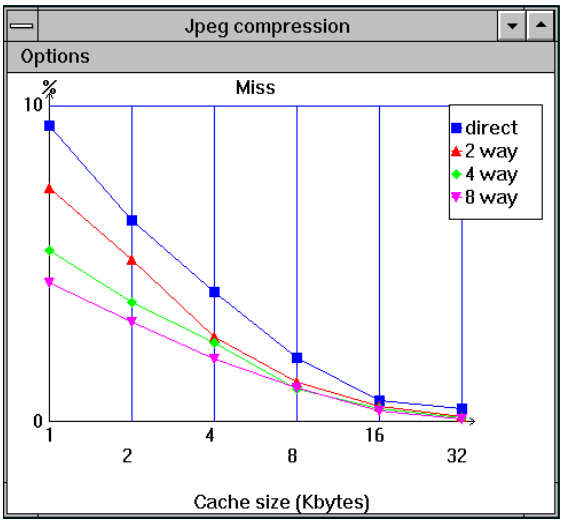

Figure 2. Miss percentage for the jpeg compression program (`cjpeg`).

time in waiting, and word transfer ratio); cache behavior (miss (Figure 2), 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).

For mere didactic intents, the *Computer Behavior Animation* phase shows an example of how a computer works. For example, in the case of cache memory, the student can execute a single memory operation and examine in detail the sequence of actions necessary for the cache controller to carry out the requested operation, or execute a specified sequence of operations and examine cache status and contents after the execution (Figure 3).

Finally, the *Program and Locality Analysis* phase allows the student to perform two types of trace analysis. The first one uses traditional program statistics such as the percentages of data/code, read/write accesses. The second one encompasses program locality. The locality statistics produced include: the number of unique blocks, the temporal locality, and the spatial locality (Figure 4). Analyzing these figures the student can understand the program behavior, and thus fine-tune the cache and main memory subsystems.

A DIDACTIC CASE STUDY

One of the key concepts that the designer has to deal with is program locality [5]. Typical modules designed to boost speed are prefetching, write-buffers, cache. Anyway, the cache is often the most critical component in achieving higher performance. The student can thus use the *Computer Behavior Animation* to view how this component behaves and the *Program and Locality Analysis* to understand how a cache can influence the performance.

As an example, for a 2-KByte, 2-way set associative cache with a 32-Byte block size), let us suppose that the student selects a read operation on location with address $(00001FD4)_{16}$. The following events

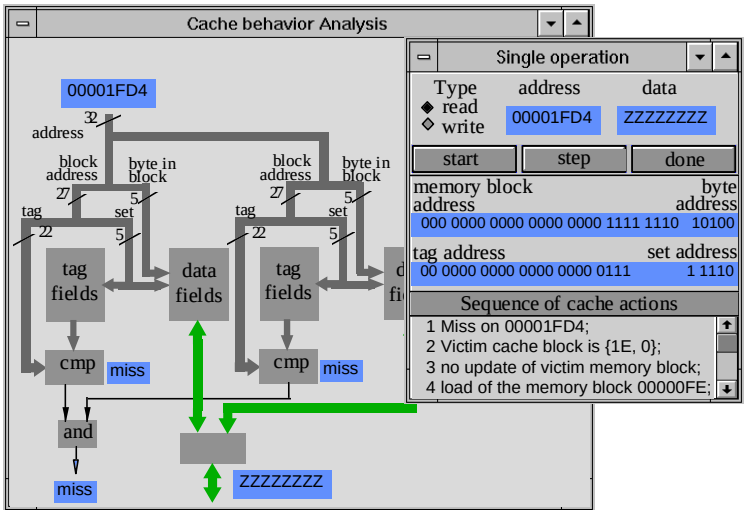

Figure 3. Step-by-step operation.

and actions are evidenced (Figure 3): i) the memory block $(00000FE)_{16}$ is not present in cache memory (miss condition), ii) the victim cache block to be replaced is the block 0 in the set $(1E)_{16}$; iii) it is not necessary to update the main memory block because the copy is not modified; iv) the cache loads the memory block $(00000FE)_{16}$ and so on. The window shows also the structure of the selected cache, the information about the operation being executed and how the cache uses a specified memory address. In our example, the address $(00001FD4)_{16}$ is split in $(000007)_{16}$ used as tag field; (compared with the tag field of both blocks of set $(1E)_{16}$); in $(1E)_{16}$ as set field (used as set address), and finally, $(14)_{16}$ used as byte offset in the block.

Then, the student can analyze locality features of the program that he has to develop, and examine how the presence of a cache memory can exploit program locality. Figure 2 shows the miss percentage of an application based on a `jpeg` image compression/decompression tool [6], a software package that is frequently used in commercial embedded systems. Global performance of a given system is strictly dependent on the specific application being considered: using different workloads may lead to different results. Figure 2 also shows that, for the `cjpeg` program, a 4-way set associative cache is recommended since it supplies the best balance between cost and performance. At a more detailed level of analysis, the student is called to search, for a given cache structure, cache and block sizes which can provide the best results in terms of global performance and cost. Again a new graph can be produced (Figure 5) to find the best value for the cache block size.

A CASE STUDY OF ACTUAL DESIGN

In this case the key point is the optimization of each component, which needs to fit as better as possible the behavioral features of the specific application for which the whole system is designed.

Let us suppose that the program requires an image compression to be completed in less than 1s and we wish to find out whether or not cache is necessary.

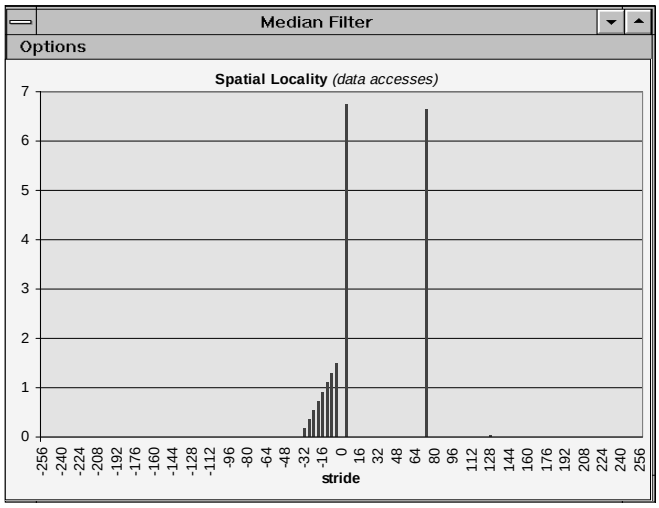

Figure 4. Spatial locality in the data area.

The designer defines the system configuration including a 20 MHz ARM core [4], a system bus, a 1-MByte memory DRAM bank, a 128-KByte memory PROM bank and a memory-mapped graphical I/O device. He also needs to specify the timing and architecture of each component. In order to obtain a low power-consumption solution, a slow system bus and slow memory devices should be selected. Also, it is up to the designer to select the input parameters of the embedded programs in such a way that the worst timing case is examined.

Choosing a memory having 150ns access time and a 250ns PROM a simulation shows that, without cache memory, the application takes up 2.898s to execute the program. Thus, the addition of a cache memory is necessary to meet time requirements.

The designer can now execute a parametric simulation in order to search the optimal cache configuration. Figure 5 shows 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; the second one is a more complex copy-back, two-way set associative cache. 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. So, the student can select a configuration that best meets cost-effectiveness and performance requirements (execution time $\leq$ 1s). For example, an optimal choice is a 16-KByte, write-through, direct access cache with 16-Byte block size.

The designer can find out the cheapest main memory that meets time requirements, by executing simulations where the varying parameter is the DRAM access time. Simulation shows that the memory delay can be 180ns (30ns more than the initial constraint) without loosing the required performance.

CONCLUSIONS

The growing demand for embedded products requires highly sophisticated computing functions at low cost and low consumption. Designers must select the most efficient system configuration in order to

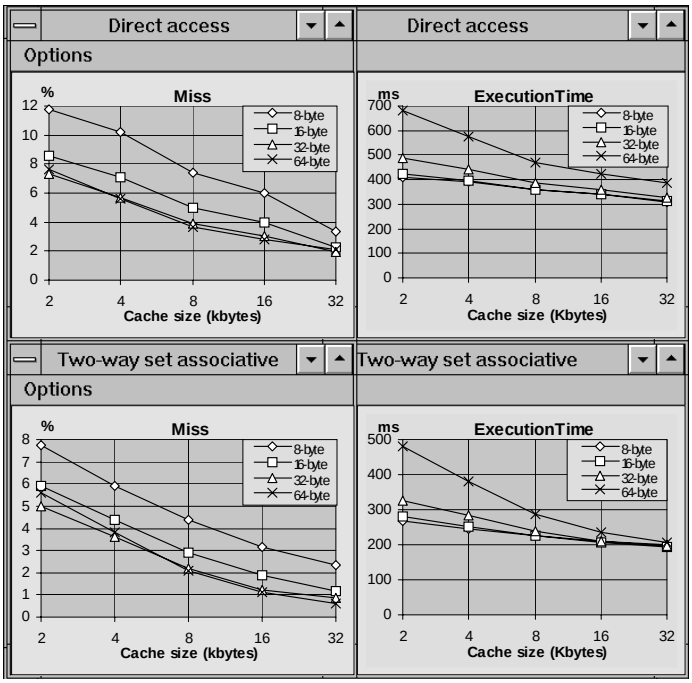

Figure 5. Miss ratio and time execution for the jpeg compression program.

resolve complex, even conflicting, requirements for low-power/high-speed and component cost. This makes accurate and reliable system simulation and performance analysis crucial. We have presented an approach based on a trace-driven system simulator that can help students in the design activity of cache memory to be employed in ARM-based embedded systems. By means of practical examples, we have shown how the student can successfully use the tool in a typical didactic path.

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 performed performance evaluations of the case studies.

REFERENCES

[1] C. A. Prete, “Cachesim: A graphical software environment to support the teaching of computer system with cache memories,” in *Proceedings of 7th SEI Conf. on Software Engineering Education, Springer-Verlag*, Jan. 1994.

[2] 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.

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

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

[5] K. Grimsrud, J.Archibald, R. Frost, and B. Nelson, “Locality as a visualization tool,” *IEEE Transaction on Computers*, vol. 45, pp. 1319–1326, Nov. 1996.

[6] G. K. Wallace, “The jpeg still picture compression standard,” *Communications of the ACM*, vol. 34, pp. 30–44, Apr. 1991.