Ad-hoc HW/SW Architectures for DBMSs: a Co-Design Approach

L. POMANTE, P. DI FELICE
Università degli Studi dell'Aquila-Facoltà di Ingegneria, 67040 Poggio di Roio (AQ), ITALY

Abstract: - This work presents a co-design methodology for the definition of ad-hoc HW/SW architectures for modern DBMSs. The main goals of such a methodology are: to analyze the DBMS specification in order to identify the operators that could benefit from ad-hoc executors, to explore the solutions design space, and to define the architecture that optimize the relevant design metrics (e.g. performance, power, cost, etc.) depending on the reference target (e.g. handheld PC, DB server, smartcard, etc.). A meta-example is presented to give the flavour on how such a metodology could work when supported by the proper toolchain.

Key-Words: - DBMS, Co-Design, Database, Embedded Systems, HW/SW Architectures

1 Introduction

In the last years, database technology has become fundamental in several application domains (e.g. *Geographic Information Systems*, *Bioinformatics Information Systems*, *Multimedia Information Systems*, etc.). Very often, such domains involve the heavy use of application-specific data-types (e.g. spatial geometries, protein structures, audio-video data, etc.), and related operators, that are generally very different from the ones that traditional DBMSs have coped within the past (which historically handled simple data types like numbers and alphanumeric characters).

These custom data types and their associated operators present new challenges to the designer of DBMSs because they upset the traditional implementation guidelines (e.g. [4]) that focused on the optimization of the I/O cost as the dominant one. Instead, it has been recently realized that the computational cost of some application-specific operators can be orders of magnitude higher than the associated I/O cost. For example, in the GIS domain [2], it has been shown that in spatial selections for point geometries, the I/O cost is the dominant one, but for more complex geometries (e.g. polygons), both costs are significant.

The primary consequence of this kind of "revolution" is the emerging need of a specific support for the classical DBMSs in order to efficiently manage such complex data. So, in order to provide such a support several attempts have been made in the recent years to add some kind of "accelerators" ([1][3][12][13]).

Actually, the idea of a dedicated *DB machine* is very old and periodically re-proposed in the past ([5][6][7]). However, the technological limitations of the related epochs have always "pushed-back" the idea of a computer architecture dedicated to DBMS execution.

Nowadays, with the availability of cost-effective FPGA (*Field-Programmable Gate Array*) and application-specific processors (e.g. GPU, *Graphical Processing Unit*) the idea of providing ad-hoc support to DBMS machines is more than an opportunity and it is gaining (again) a lot of consideration in the research community ([8][9][10][11]). Recent approaches mainly focus on the exploitation of GPU ([1][3][13]) and partially on FPGA ([12]).

However, such works lack of generality because they target the acceleration of specific operators by means of specific executor classes selected according to the experience of the designer. Moreover, it is worth noting that acceleration is not the only issue that should be considered when referring to modern DBMS applications. In fact, with the increasing diffusion of portable devices to support the pervasive-ubiquitous computing paradigm, different factors (i.e. energy-consumption, reliability, cost, etc…) should be taken into account not only for the application-specific scenarios depicted above but also for other ones (e.g. portable DBMSs with standard data types and operators [14][15], resource-limited flash-based smartcard DBMSs [16]).

In essence, what is still lacking is a general methodology supporting the designer to take implementation choices suitable to exploit the unique features of each class of executors (*General Purpose Processors*, GPU, FPGA, etc.), while keeping also in consideration the other relevant aspects (i.e. communication and storage mechanisms), with the goal of optimizing relevant factors.

To fill such a gap, this work follows an *hw/sw co-design approach*, suitable to support the designer in the selection of those operators that could benefit from an ad-hoc executor, identifying also the system architecture (i.e. number and class of executors) appropriate to reach such a goal. The choice of a co-design approach arises because a lot of issues that emerge in the considerations presented above are very similar to those historically coped with the design and implementation of hw/sw embedded systems (see Section 2). Such a similarity has been already identified in the past [19] but the main goal was to show the effectiveness of the approach with reference to a specific algorithm without delineating a general methodology. The definition of such a methodology is the main objective of this work that is structured as follows.

The next section provides some background information about the hw/sw co-design discipline. Then, Section 3 presents the hw/sw co-design methodology proposed for the ad-hoc implementation of DBMS operators and, finally, Section 4 concerns conclusions and the future work.

2 HW/SW Co-Design

HW/SW co-design (i.e. *hw/sw concurrent design*) [17][18] is a relatively young discipline that has lead out to several methodological approaches devoted to support an embedded systems designer during the exploration of the system design space keeping a unified view of the problem instead of the typical separate design approach between the hardware and software parts.

The interest in co-design research has been steadily increasing from the beginning of the 1990s. Initially, the dominating research issue was the partitioning of a system description into an ASIC (*Application Specific Integrated Circuit*) part and a software part to be executed on a tightly coupled processor. Later, an increasing number of research works has considered the problem of defining a co-design methodology for heterogeneous multiprocessor embedded systems (e.g. [20][21][22]).

Such modern approaches consider, as the entry point, a (often executable) system-level specification expressed by means of languages similar to common programming languages (e.g. C, C++, Java, etc…). In general, starting from such a specification, they allow the identification of the (sub)optimal heterogeneous multiprocessor hw/sw architecture and the partitioning/allocation of the functionalities of the system on such an architecture that optimizes a given cost function.

From a general point of view, a modern co-design methodology can be decomposed into several steps (Figure 1). For the purposes of this work it is important to analyze with some details only the high-level ones.

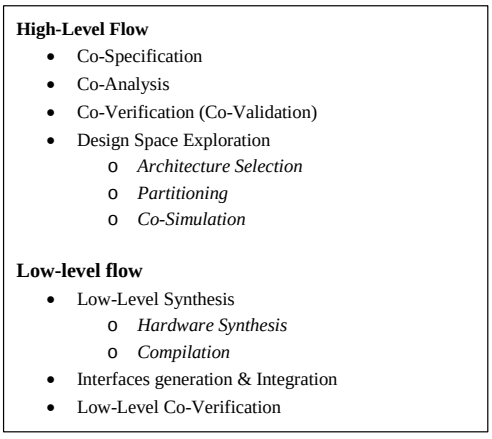

Figure 1. Modern co-design methodology

During the co-specification step the requirements are translated from an informal language into a formal (or semi-formal) description of the system functionalities. An abstract homogeneous behavioral (i.e. algorithmic) description is given for the complete heterogeneous system, regardless of the target architecture that will be chosen and how, later, the different parts will be implemented.

Next, in the co-analysis step, several techniques allow early estimation of the final implementation characteristics (e.g. performance, power consumption, etc.). Such analysis methods are necessary to allow a comparison between different implementation candidates in the early steps of the methodology.

Then, the functional correctness of the system is verified: the specification is simulated to check its behavior with respect to representative test-benches.

The step calledis “design space exploration” could be decomposed (at least logically) in three interacting tasks: *partitioning, architecture selection* and *co-simulation*. When deciding on the implementation, the designer needs to choose the components to include and how them should be connected in the hardware architecture. It must also be decided which parts of the behavior should be implemented on which of the selected components. The first of these activities is called architecture selection and the second is known as partitioning. Architecture selection and partitioning are influenced by performance requirements, implementation cost, reconfigurability, and application-specific issues. Co-simulation evaluates the system behavior from a functional point of view or a timing point of view, in order to validate either the specification or the performed partitioning.

3 The Proposed Methodology

The *all-sw implementation* of a DBMS could be considered as a system-level executable specification to be used as entry point for a co-design methodology. At the best of our knowledge, it does not exist a co-design methodology tailored to the peculiarity of DBMSs. Therefore, a *DBMS co-design methodology* (based on an existing methodology for the co-design of heterogeneous multiprocessor embedded systems [22] that follows the general schema depicted in Section 2) is presented in the following, as the main result of this work, by taking into account both the considered target architecture and the proposed co-design flow. Finally, a meta-example is provided in order to clarify how such a methodology could work when fully supported by a proper toolchain.

3.1 The target architecture

The reference methodology [22] is oriented to the co-design of embedded systems based on a heterogeneous multiprocessor architecture in which various kinds of communication links interconnect processing elements

and memories. Such an architecture may consist of GPP, *Digital-Signal Processors* (DSP), *Application-Specific [Instructions] Processors* (ASP or ASIP), ASIC, FPGA, and memory modules, properly interconnected by some kind of network topology (point-to-point, bus, multiple busses, mesh, etc. [23]) to perform application-specific functions.
However, an optimal general architecture does not exist but a viable solution could be found by defining a sort of *template architecture* to be optimized for the specific characteristics of the application domain.
In this work, where the focus is on ad-hoc executors (i.e. co-processors) for DBMS operators, it seems natural to consider a *GPP-centric* template architecture (i.e. an architecture in which there is a main GPP that executes the *core* of the DBMS and that demands tasks to co-processors while mastering also the bus) with an heterogeneous *multi-co-processors* support that shares memory with DMA (*Direct Memory Access*) capabilities and have an associated amount of local memory sufficient to store the executed code and the local data. Considering a number of co-processors in the order of ten, it is possible to limit the interconnection media to a shared bus so depicting the template architecture shown in Figure 2. Such an architecture, that could be considered viable both for desktop and portable devices, is considered as the reference one in the proposed methodology.

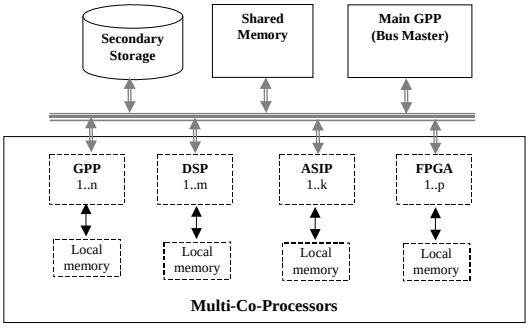

Figure 2. The reference template architecture

3.2 The co-design flow

Figure 3 defines the co-design flow oriented to the ad-hoc implementation of DBMS operators. Such a flow is detailed in the following.

3.2.1 DBMS specification

As stated before, the *all-sw implementation* of a DBMS could be considered as a system-level executable specification to be used as entry point in the co-design flow. However, with respect to traditional co-design methodologies, there are some issues to be considered carefully:

- The source code of the all-sw implementation of the DBMS could be not available. In this work, it is considered to be available, at least, the source code related to the *selected operators* to be implemented ad-hoc.
- The all-sw implementation will be probably not synthesizable (i.e. not suitable to be automatically translated into a format compatible with the target executors class) so, some manual (or semi-automatic) refinement steps could be required.

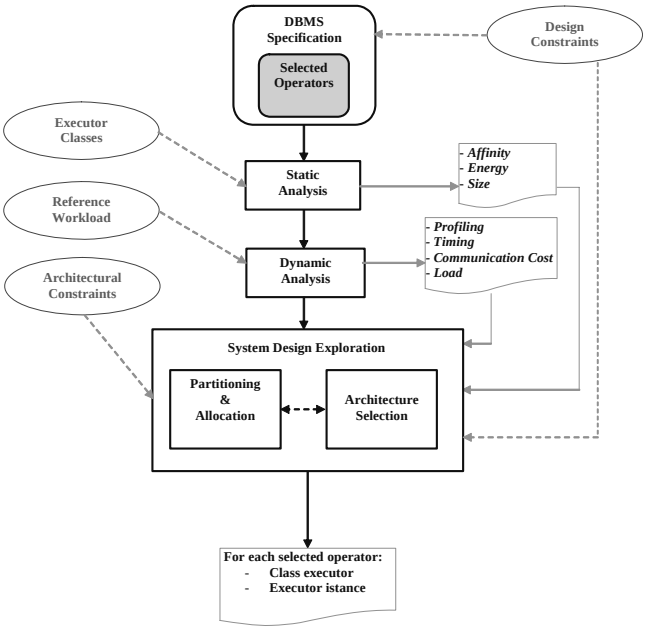

Figure 3. The DBMS co-design flow

3.2.2 Static analysis

The *Static Analysis* step provides quantitative information (i.e. metrics) useful to take system-level decisions such as architecture selection, partitioning and allocation. The underlying idea is that the relevant metrics of a final design can be related to the properties of the specification itself. Therefore, the core of this step involves the identification and evaluation of functional and structural properties of the specification, which could affect its implementation on different architectural platforms.
Given the system specifications and the involved *Executor Classes* (e.g. GPP, DSP, FPGA, etc…) to be considered in the design, this step provides the following output information for each selected operator:

Affinity

The *Affinity* metric [22] is defined as a triplet of values in the interval [0, 1] that provides a quantification of the matching between the structural and functional features of a function (i.e. a DBMS operator) and the architectural features of each one of three executor classes (i.e. GPP, DSP, FPGA). In the DBMS context, the original Affinity metric should be extended to consider some ASIP executors class. In particular it should consider also GPU and NPU (*Network Processor Unit*) classes [8][9].

Energy
The *Energy* metric [25][26] provides a set of estimations of the energy required by HW and SW executors for the execution of each single statement that composes the specification of the selected operators. Such information, combined with the profiling, is very important to optimize the energy consumption of the whole system.

Size
The *Size* metric [27] provides a set of estimations/evaluations of the memory space (i.e. KB for SW and FPGA cells for HW) required by the data structures and each single statement that composes the specification of the selected operators. Such information is very important in order to avoid overloading of the system resources.

3.2.3 Dynamic analysis

The *Dynamic Analysis* provides different information useful to take system-level decisions such as architecture selection, partitioning and allocation. The underlying idea is that the relevant metrics of a final design can be related to the execution of the specification itself when fed with meaningful input data set.
A fundamental task associated with this step consists in the identification of a *Reference Workload*. Such a task could be performed with two different goals: to optimize the implementation with respect to a target DB schema, and to optimize the implementation generally.
In the first case, the target DB schema, and a meaningful set of typical queries on it, represent the reference workload in order to stress the DBMS in a specific manner. In the second one the analysis should be performed using a set of meaningful and typical DB schemas each one with related queries. Generally, more specific is the target application more appreciable are the optimization results.
Once defined the reference workload, by means of the dynamic analysis, performed during the execution of such a workload, it is possible to extract the following information:

Profiling
By means of the *Profiling* [22] it is possible to evaluate the number of executions of each function (i.e. procedures, methods, etc…) involved in the all-sw implementation of the selected operators. Moreover, for each such a function, it is possible to evaluate the number of executions of each statement composing it.

Timing
The *Timing* metric [24], applied to the proposed DMBS co-design flow, should provide a set of evaluations of the time required by a GPP for the execution of selected operators.

Communication Cost
The *Communication Cost* [22] represents the amount of data exchanged between the functions involved in the all-sw implementation of the selected operators, and between such functions and the rest of the system (i.e. the DBMS). This cost is very important in order to determine which functions should be allocated on the same co-processor and which ones should be executed directly by the main GPP in order to limit the cost of the whole communications.

Load Estimation
The *Load Estimation* [22], combining some of the data provided by the previous steps of the design flow (i.e. profiling and timing) with the designer imposed timing constraints allows the estimation of the load that each functions will impose to a GPP that should execute it. The extraction of these data is an important task that allows the evaluation of the number of needed processing elements and the identification of those functions that will probably need an executor different than a GPP.

3.2.4 System design exploration

Finally, the flow reaches the *System Design Exploration* step that is constituted of two interacting tasks: *Partitioning & Allocation* and *Architecture Selection*.
All the data produced in the previous steps of the flow are used to guide the process, together with additional information provided by the designer. Such information expresses the *Architectural Constraints* (e.g. max number of GPP, cells limitation for FPGA, etc.).
Such a step [22] explores the design space to identify feasible solutions, supporting also the selection of the final architecture by suggesting the type and number of executors that should be included. It takes into account several issues while trying to identify a system implementation that optimizes a cost function composed of a suitable combination of the metrics described above.
The output of this step is the allocation of the selected DBMS operators on the proposed architectural components. More in detail, for each selected operator it will be suggested an executor (type and instance number). So, the set of the suggested executors provides a specialization of the template architecture considered above.

3.3 A meta-example

To give the flavour of how the methodology could actually work a meta-example on ad-hoc implementation of DBMS operators is reported in the following.
Let *S* be the specification of a DBMS, and {*Op1*, *Op2*, *Op3*} three operators to consider for an ad-hoc implementation (i.e. the *selected operators*). Let each operator be implemented in sw by means of a unique

C++ method {$Op1_m$, $Op2_m$, $Op3_m$} whose source code is available.

Let the design constraint be oriented exclusively to performance issues, that is, to accelerate selected operators in order to obtain an execution time of each one of 50% less than the all-sw execution on a single GPP architecture.

Considering only two executor classes (i.e. GPP and FPGA), the *Static Analysis* applied to the three methods {$Op1_m$, $Op2_m$, $Op3_m$} provides the quantitative information in Table 1.

	Affinity		**Size**	
	GPP	*FPGA*	*SW (KB)*	*HW (FPGA Cells)*
$Op1_m$	0.62	0.24	458	122
$Op2_m$	0.43	0.49	356	145
$Op3_m$	0.44	0.87	812	213

Table 1. Static analysis results

It is worth noting that the affinity metric (the only relevant for performance issues) suggests that $Op3_m$ is very suitable for a FPGA implementation, $Op1_m$ is better executed by a GPP, and an equivalence is indicated for $Op2_m$.

Proceeding to the next step of the flow (Figure 3), the first task for a meaningful *Dynamic Analysis* is the definition of the reference workload. For this purpose, it is possible to consider a single DB schema designed for a specific application and a set (e.g. 100) of diferent queries representative of a typical DB utilization (the implementation will then be optimized specifically for such a DB).

During the execution of the selected queries the following average values about the methods have been collected: profiling, timing and load estimation, and communication cost.

	Profiling *(#executions/query)*	**Timing** *(second/#executions)*	**Load Estimation** *(50% exe time reduction)*
$Op1_m$	2.3	0.8	0.21
$Op2_m$	0.8	1.2	0.09
$Op3_m$	3.3	1.9	0.34

Table 2. Dynamic analysis results: profiling, timing and load estimation

The results (Table 2) show that $Op3_m$ is the more used and computationally intensive operator, while (Table 3) $Op1_m$ is the one that exchanges more data with the DBMS core. It is worth noting (Table 3) that the operators are independent one from each other and so they don't exchange any data.

(#bytes/execution)	***DBMS-Core***	$Op1_m$	$Op1_m$	$Op1_m$
DBMS-Core	0	112	32	66
$Op1_m$	112	0	0	0
$Op2_m$	32	0	0	0
$Op3_m$	66	*0*	0	0

Table 3. Dynamic analysis results: communication cost

The final step of the proposed flow is the *System Design Exploration*. Such a step takes as input all the information provided by the previous step plus some *Architectural Constraints* by the designer. Let such constraints be set to a maximum of one FPGA and one GPP (other than the main one).

Collecting all the gathered information in a proper annotated graph (*Procedural Interaction Graph* [22]), it is possible to exploit a proper tool (*EmuP* [22]) to perform the design space exploration. Defining a cost function with equal weights for affinity, communication cost and load, such a tool provides the following result (Table 4): the DBMS-Core and two methods are allocated on GPP#0 and one method on FPGA#0, for a total of 1 GPP and 1 FPGA.

	DBMS-Core	$Op1_m$	$Op2_m$	$Op3_m$
Executor Class	GPP	GPP	GPP	FPGA
#instance	0	0	0	0

Table 4. Suggested partitioning/allocation/architecture

The suggested system architecture, derived from the reference template, is shown in Figure 4. Due to affinity and load considerations, $Op3_m$ has been allocated on a FPGA, while $Op1_m$ (mainly for the communication cost) and $Op2_m$ (mainly for the low imposed load) have been kept together with the *DBMS Core* on the main GPP.

This step ends the high-level co-design flow providing fundamental information to approach the low-level one (Figure 1) to reach a physical implementation of the system.

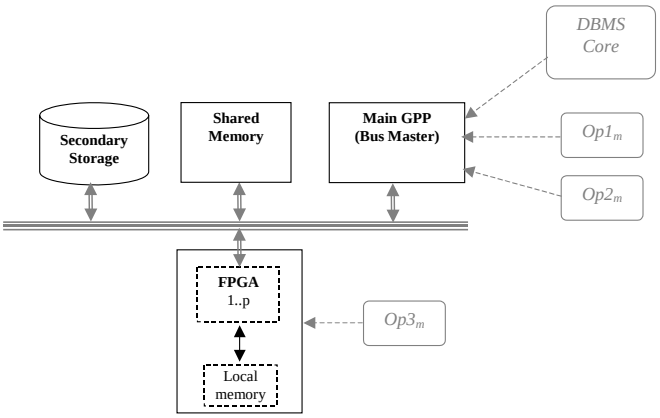

Figure 4. The suggested system architecture

4 Conclusion and Future work

This work has presented a co-design methodology for the definition of ad-hoc HW/SW architectures for DBMSs. In detail, the main goals of such a methodology are to analyze the DBMS specifications in order to identify the DBMS operators that could benefit from an ad-hoc executor and to define the architecture that optimize the relevant design aspects depending on the reference target.

Future work will be devoted to enhancements of the presented methodology (e.g. to consider different components of a DBMS other than operators), the development of a related toolchain, and the design flow validation by means of real-world case studies.

References:

[1] N. Bandit, C. Sun, D. Agawal, and A. El Abbadi. *Hardware Acceleration in Commercial Databases: A Case Study of Spatial Operations*. Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004.

[2] R. K. Kothuri and S. Ravada. *Efficient processing of large spatial queries using interior approximation*. In Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases (SSTD'01), pages 404–421. ACM Press, 2001.

[3] C. Sun, D. Agrawal, and A. El Abbadi. *Hardware acceleration for spatial selections and joins*. In Proceedings of the ACM SIGMOD international conference on on Management of data, pages 455–466. ACM Press, 2003.

[4] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database System Implementation*. Chapter 4. Prentice Hall, 1999.

[5] D. K. Hsiao. *Advanced Database Machine Architectures*. Prentice Hall, Englewood Cliffs, N.J., 1983, pp. 1-18.

[6] E. Ozkarahan. *Database Machine and Database Management*. Prentice Hall, Vol. 10, 1986, pp. 319-337.

[7] C. Lee, S. Y. Su, and H. Lam. *Algorithms for Sorting and Sort-Based Database Operations Using a Special-Function Unit*. Database Machines and Knowledge Base Machines. Kluwer Academic Publishers, 1988, pp. 103-116.

[8] A. Ailamaki. *Database Architectures for New Hardware*. Proceedings of the 21st International Conference on Data Engineering (ICDE 2005).

[9] A. Ailamaki, N. K. Govindaraju, and D. Manocha. *Query Co-Processing on Commodity Hardware*. Proceedings of the 22nd International Conference on Data Engineering (ICDE'06).

[10] DAMON 2005, www.cs.cmu.edu/~damon2005/

[11] DAMON 2006, www.cs.cmu.edu/~damon2006/

[12] K. T. Leung1, M. Ercegovac, and R. R. Muntz. *Exploiting Reconfigurable FPGA for Parallel Query Processing in Computation Intensive Data Mining Applications*. In UC MICRO Technical Report, Feb. 1999.

[13] K. E. Hoff III, A. Zaferakis, M. Lin, and D. Manocha. *Fast and simple 2d geometric proximity queries using graphics hardware*. In Proceedings of the Symposium on Interactive 3D Graphics, pages 145–148. ACM Press, 2001.

[14] C. Bobineau, L. Bouganim, P. Pucheral, and P. Valduriez. *PicoDBMS: Scaling down Database Techniques for the Smartcard*. Proceedings of the 26th International Conference on Very Large Databases, Cairo, Egypt, 2000.

[15] C. Bolchini, C. Curino, M. Giorgetta, A. Giusti, A. Miele, F. A. Schreiber, and L. Tanca. *PoLiDBMS: Design and Prototype Implementation of a DBMS for Portable Devices*. Proc. SEBD'04, June 2004, pp. 166-177.

[16] C.Bolchini, F. Salice, F. A. Schreiber, L. Tanca. *Logical and Physical Design Issues for Smart Card Databases*. ACM Transactions on Information Systems, Vol. 21, No. 3, July 2003, Pages 254–285.

[17] W. H. Wolf. *Hardware-Software Co-design of Embedded Systems*. Proceedings of the IEEE. Vol. 82, NO. 7., July 1994.

[18] G. De Micheli, and R. Gupta. *Hardware/Software Co-Design*. Proceedings of the IEEE, Vol 85., No. 3, March 1997, pp349-365.

[19] W. M. Badawy, A. Kumar and M. A. Bayoumi. *A Co-design Based High-Performance Real-Time GIS Systems*. IEEE Workshop on Signal Processing Systems (SiPS), Taipei, Taiwan, Oct. 20-22, 1999, pages 410-419.

[20] A. Baghdadi, N.E. Zergainoh, W.O. Cesario and A.A. Jerraya. *Combining a performance estimation methodology with a hardware/software codesign flow supporting multiprocessor systems*. IEEE Trans. on Software Engineering, vol. 28, no. 9, 2002, pp. 822-831.

[21] B.P. Dave, G. Lakshminarayana and N.K. Jha. *COSYN: Hardware-software co-synthesis of heterogeneous distributed embedded systems*. IEEE Trans. on Very Large Scale Integration Systems, vol. 7, no. 1, March 1999, pp. 92-104.

[22] C. Brandolese, W. Fornaciari, L. Pomante. F. Salice, and D. Sciuto. *Affinity-Driven System Design Exploration for Heterogeneous Multiprocessor SoC*. IEEE Transactions on Computers, vol. 55, no. 5, May 2006.

[23] D. Sima. *Advanced computer architectures*. Addison-Wesley, 1997.

[24] C. Carraras et al. *A Co-Design Methodology Based On Formal Specification And High-Level Estimation*. Proc. of IEEE Codes/CASHE'96, Pittsburgh, Pennsylvania, 1996.

[25] C. Brandolese, F. Salice, W. Fornaciari, and D. Sciuto. *Static power modeling of 32-bit microprocessors*. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on Volume 21, Issue 11, Nov. 2002 Page(s):1306 – 1316.

[26] C. Brandolese. *A co-design approach to software power estimation for embedded systems*. Ph.D. thesis, 2000. DEI, Politecnico di Milano, Italy.

[27] C. Brandolese, W. Fornaciari, and F. Salice. *An area estimation methodology for FPGA based designs at systemc-level*. Design Automation Conference, 2004. Proceedings. 41st, 2004 Page(s):129 – 132.