

# Affinity-Driven System Design Exploration for Heterogeneous Multiprocessor SoC

Carlo Brandolese, William Fornaciari, *Senior Member, IEEE*,  
Luigi Pomante, Fabio Salice, *Member, IEEE*, and Donatella Sciuto, *Member, IEEE*

**Abstract**—Continuous advances in silicon technology enable the development of complex System-on-Chip as cooperation among Digital Signal Processors (DSPs), General Purpose Processors (GPPs), and specific hardware components. The impact of this choice is not only limited to the target architecture, but also encompasses the overall system specification. It is thus crucial to manage such a complexity using high-level specification languages and a tool chain supporting the designer throughout a set of strategic decisions, such as the identification of a set of possible target architectures, the verification of the correctness of the specification, and the partitioning of the specification onto a set of computational resources. This paper addresses this type of problem by proposing a design flow supporting the system-level design of heterogeneous multiprocessor system-on-chip (MP-SoC), by extracting information from the system description (e.g., SystemC)—statically and in a fast manner—and by providing a set of quantitative measures correlating the type of executor, the functionality, and a timing estimation. Partitioning and architecture selection are built on top of this data and the final analysis of the selected Hardware-Software solution over the identified candidates is finally submitted to a timing verification via simulation. Note that the possibility of actually performing a comprehensive design space exploration, in general, is tightly influenced by the interaction between partitioning/architecture-selection and timing simulation in the design flow; for this reason, the description of this aspect is particularly emphasized in the presentation of the methodology. To show the applicability of the proposed methodology, two relevant case studies are described in the paper.

**Index Terms**—System-on-Chip, embedded systems, multiprocessor systems, codesign, metrics, heterogeneous systems.

## 1 INTRODUCTION

NOWADAYS, many applications, both for research and for commercial use (e.g., management of real-time video streaming [5], video signal processing [6], etc.), are implemented by exploiting the capabilities of several hardware and software executors, possibly integrated onto a single chip. This type of solution, typically called *Multi-Processor System-on-Chip* (MP-SoC), can integrate several processors and a variety of Intellectual Property (IP) blocks, connected via a proper high performance communication network. Depending on the type of application, apart from the specialization of the hardware IP blocks, a proper tailoring of the type of processor can also be envisioned. Such embedded processors can be roughly partitioned in classes like GPPs, microcontrollers, DSPs, ASIPs, etc., varying in terms of cost, throughput, size, power consumption, and level of specialization.

The option of evaluating such a wide range of alternative architectures is one of the new value added features that the designers are looking for whenever a trade-off among aspects like performance, predictability, cost, flexibility, power consumption, etc., has to be achieved [1]. Unfortunately, expanding the design space cannot do anything but

exacerbate the problem of selecting the abstraction level at which comparisons can be made in reasonable times, with an accuracy/confidence ensuring actually optimizing the system.

The typical crucial activities the designer has to face that are not yet totally automated or currently supported by EDA tool suites are those related to the capturing of the characteristics of the possible executors, the extraction of models of the application tailored for the analysis of alternatives and fast evaluation of properties (e.g., timings, area, power, communication bandwidth, etc.), and the selection and comparison among alternatives based on some estimated metrics and figures of merit. The output of this phase should be the identification of the most promising architecture, able to cope, in an optimal manner, with all the constraints and goals of the designer, to be further optimized during synthesis, at a lower abstraction level, by exploiting more consolidated methodologies and tools.

However, the growing diffusion of embedded applications based on these platforms poses numerous challenges to the EDA community to provide methodologies and tools to support the design flow targeted to MP-SoC, even if approaches targeting simpler systems are not yet consolidated on the market. In fact, no assessed general design methodologies are available today and what is still needed is a systematic approach, general enough to fit several application domains, while considering the peculiarity of the system to be designed.

To overcome such problems, a possible solution consists of extending the classical codesign methodologies, defined for single processor applications. This paper presents a part of a more comprehensive work [11] aimed at providing general models, methodologies, and tools to support each

- C. Brandolese, W. Fornaciari, F. Salice, and D. Sciuto are with the Politecnico di Milano-DEI, Piazza L. Da Vinci, 32, 20133 Milano, Italy. E-mail: {brandolese, fornaciari, salice, sciuto}@elet.polimi.it.
- L. Pomante is with the Università di L'Aquila, Facoltà di ingegneria-DEWS, Piazza E. Pontieri 1, 67040 Poggio di Roio (Località Monteluco), Italy. E-mail: pomante@ing.univaq.it.

Manuscript received 31 Dec. 2004; revised 5 Oct. 2005; accepted 22 Nov. 2005; published online 22 Mar. 2006.

For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0431-1204.

step of the codesign flow of *Heterogeneous Multiprocessor System-on-Chip architectures*. In particular, this paper focuses on the definition and validation of an innovative system design exploration strategy—based on specific metrics used to select a system architecture—which associates each system functionality with the most suitable class of processing elements.

The paper is organized as follows: Section 2 sketches the state-of-the-art in MP-SoC codesign research; Section 3 presents the proposed design flow and the related design environment; Section 4 formally introduces the representation model, defined to extract the information necessary to explore the design space with the aid of the innovative metrics defined in Section 5. The proposed system design exploration methodology, based on the defined metrics, is presented in Section 6, while the validation of the methodology is discussed in Section 7, where experimental data are reported for the considered case studies. Section 8 draws some conclusions and outlines future work.

## 2 RELATED WORK

In the past few years, an increasing number of research works have considered the problem of defining a codesign methodology for heterogeneous multiprocessor embedded systems possibly implemented on a single chip. However, the majority of these investigations suffer from a common problem: They adopt, as the entry point, system specifications described with languages too heavily bound to a specific application domain and this limits the applicability of the proposed environments.

For example, the work presented in [10] proposes a codesign approach for the development of real-time systems. The target architecture is based on a mix of microprocessor cores and ASICs connected by a bus and the behavior of the entire system is specified in a high-level, homogeneous description (it is not targeted to any common specification language and it is not executable). The main limitation of this approach is that it requires information about the tasks that are seldom available while designing general systems at a high-level of abstraction. Similar considerations apply for COSYN [7] and COHRA [8]. COSYN presents a cosynthesis algorithm that starts with an embedded-system specification and results in an architecture consisting of hardware and software modules to meet performance, power, and cost goals. The specification is provided by means of a set of annotated acyclic periodic task graphs. COHRA is an extension to the previous one made by the same authors, adding the capability of managing hierarchical task graphs, improving the quality of the results, in particular for large task graphs. However, the task graphs are formalisms suitable only for a restricted range of applications and granularity issues prevent achieving a real system-level approach for complex systems.

In general, all the task-graph-based approaches provide excellent results on small examples, but, unfortunately, each step of the design flow is typically strictly related to the task graph theory and it seems very hard to represent general systems by means of such a specification formalism.

A completely different approach is represented by CMAPS [9]. It tries to shift the effort of the research in the codesign field from system design to requirements analysis. The goal is to design an SoC starting from the application

problem itself, rather than from a detailed behavioral specification. A user can specify a complex application problem by referring to the *elementary problems* in a *Problem Base* and by describing how the selected elementary problems can be composed into the desired application problem. The methodology presented in such a work is a very interesting attempt to raise the abstraction level of the entry-point of a codesign environment. However, it is far from a widespread applicability.

Other approaches [36], [38], [39] focus more specifically on design exploration strategies, but they often rely too much on some architectural aspects (far away from a real system-level description) or on the designer experience. For example, the work presented in [38] addresses performance estimation and architecture exploration issues. It is based on a back-annotation approach in which the analysis of “some” implementations (at RT level) allows the extraction of “all” timing elements needed for performance estimation of “all” feasible implementations. These timing elements are then combined and introduced into the SDL system specification. With this new time-annotated specification, it is possible to predict the performance of all feasible architectures, but some important parameters heavily depend on the designer experience, in particular, the choice of the adequate number of reference implementations to make meaningful estimations. Similar considerations arise for one of the most representative investigations carried out by the TIMA/SLS Lab [36], where a codesign flow for the generation of application-specific multiprocessor architectures for SoC has been defined to extract architectural parameters from a high-level system specification in order to directly instantiate architectural components, such as processors, coprocessors, memory modules, and communication networks. However, the adopted microarchitectural level limits the design space exploration. An attempt to integrate such an approach with a system-level oriented one has been presented in [39], where the proposed generic architecture model is reusable for a large class of applications, but it still presents the need for too many low-level details in the early steps of the design flow.

Beyond specification and design flow, another important feature that limits the applicability of codesign environments is the target architecture considered during the design space exploration. For example, [4] describes the system synthesis techniques available in  $S^3E^2S$ , a CAD environment for the specification, simulation, and synthesis of SoC. In spite of the fact that the  $S^3E^2S$  modeling environment could manage heterogeneous systems, the considered target architecture focuses only on a multiprocessor paradigm that does not consider ASICs or FPGAs.

This overview shows that there is a need for a systematic approach to codesign that should be oriented toward the system-level, quite general to be useful in several application domains, based on assumptions that do not limit its applicability, and able to extract and properly consider the relevant features of the system to be designed. This is the goal of the approach presented in the next section.

## 3 THE PROPOSED DESIGN FLOW

The proposed design flow is shown in Fig. 1. The entry point is a *behavioral synthesizable description* (e.g., the *Synopsys SystemC Behavioral Synthesizable Subset* [30], [33]) of the

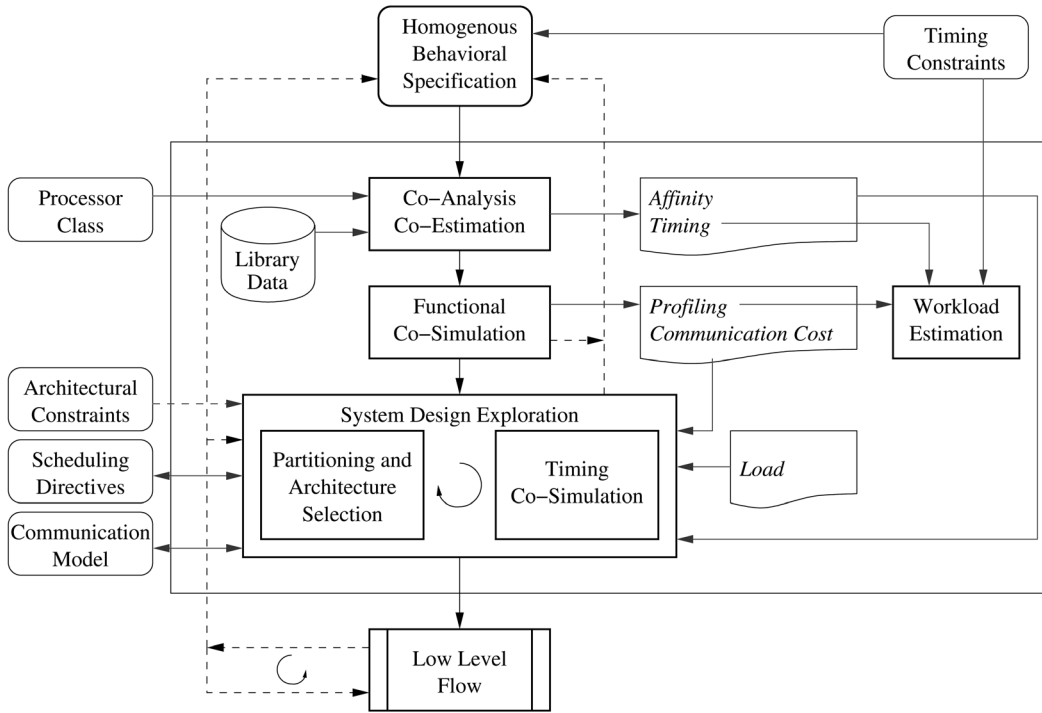


Fig. 1. The proposed high-level flow.

desired system. A procedure-level internal model has been defined to represent the system specification. Such a model, called the *Procedural Interaction Graph* [11], [32], is based on the *Procedural Call Graph* [3], [31]. The procedure-level internal model is able to capture information related to the computational elements present in an imperative, possibly object-oriented, synthesizable behavioral specification along with their relationships.

The first step of the flow aims at extracting as much information as possible on the system by analyzing the specification in a static and fast manner. This step is composed of the *Coanalysis* (Section 5, [11], [12], and *Coestimation* [11] activities. The former one provides a set of data expressing the *Affinity* of the system functionalities toward a set of processing element classes (GPP, DSP, ASIC/FPGA), while the latter provides a set of estimations of the time required, by each class of processing elements, for the execution of each single operation that composes the specification. After the static analysis, the system functionalities are simulated (*Functional Cosimulation* [11]) to verify their correctness with respect to typical input data sets, and important data characterizing the dynamic behavior of the system are extracted: *Profiling* and *Communication Cost*. Combining the data provided by the previous steps (timing and profiling data) with a global time constraint allows the estimation of the *Load* [11] associated with the execution of each system functionality on a 1-GPP SoC (i.e., the worst case). The analysis of such data is useful to evaluate the necessary amount of processor cores, the level of load balancing, and the identification of those procedures that probably need an executor more performing than a general-purpose processor.

Finally, the flow reaches the *System Design Exploration* task that is constituted by two iterative steps: *Partitioning and Architecture Selection* [11], [34] and *Timing Cosimulation* [11]. The partitioning and architecture selection methodol-

ogy is detailed in Section 6 while the timing cosimulation methodology considers the proposed heterogeneous multiprocessor architecture and a high-level model for the communication media in order to evaluate the performance of the SoC by verifying its timing correctness.

#### 4 THE REPRESENTATION MODEL

The entry point of the proposed design flow (Fig. 1) is a behavioral synthesizable description of the desired system (e.g., the Synopsys SystemC Behavioral Synthesizable Subset [30], [33]). A procedure-level internal model, called the *Procedural Interaction Graph* [11], [32], has been defined to represent such a kind of system specification, based on the *Procedural Call Graph* [3], [31].

The procedure-level internal model is able to capture information related to the computational elements present in an imperative, possibly object-oriented, synthesizable behavioral specification and the relationship between these elements.

An intuitive graphical representation of such a graph (see Fig. 2) is obtained by associating a node with each instance of a method (gray nodes are nonblocking, i.e., they can be executed concurrently with their caller) and an arrow for each data transfer (black arrows indicate procedure calls, gray arrows indicate communication/synchronization operations).

The example of Fig. 2 shows a procedure-level internal model where the instance of the *Main* method calls the instances of nonblocking methods *F1*, *F6*, and *F9*. The other instances of the method perform only blocking calls, while instances *F5-F8* and *F6-F9* make data transfers.

Let us formally define the procedural level model.

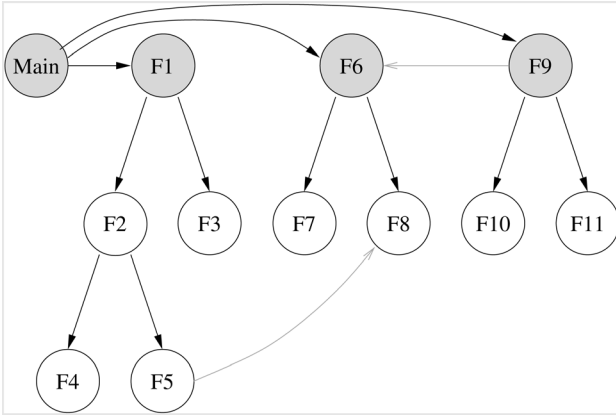


Fig. 2. Internal model graphical representation.

#### 4.1 Procedural Interaction Graph

The Procedural Interaction Graph is a formalism composed of nodes (i.e., instances of methods or procedures) and annotated edges (i.e., method or procedure calls, communication, or synchronization operations) that provide information on the relationships between the nodes and on the data exchanged (e.g., size, profiling, etc.). Such a graph is suitable for representing a coarse grain view of the system that takes into account communications, synchronizations and concurrency issues. The following definitions provide a formalization of the model that is applicable both to imperative object-oriented (OO) specification languages as well as to not object-oriented (NOO) ones. The first two definitions relate to the specification language, while the next ones formalize the components of the Procedural Interaction Graph.

**Definition 1.**  $LN = \{VI, VO, VT\}$  is the adopted specification language, where  $VI$  is the set of valid identifiers of  $LN$ ,  $VO$  is the set of valid operators of  $LN$ , and  $VT$  is the set of valid types of  $LN$ .

**Definition 2.**  $VS$  is the set of valid specifications that can be expressed with  $LN$ .

**Definition 3.**  $m_{i,j} = \langle q, k, c \rangle$  is the method  $j$  of class  $c_i$  (see Definition 5), where  $i, j \in N$ ,  $q \in VI$  is the method name,  $k \in MQ = \{B, NB\}$  is a method qualifier where  $B$  indicates a blocking method while  $NB$  indicates a nonblocking method, and  $c \in VS$  is the method body. Note that the term “method” indicates both a method in the OO sense and a procedure in the NOO one. For NOO languages, all the methods belong to a fictitious class 0.

**Definition 4.**  $d_{i,j} = \langle q, z \rangle$  is the member data  $j$  of the class  $c_i$  (see Definition 5), where  $i, j \in N$ ,  $q \in VI$  is the data name, and  $z \in VT$  is the data type.

**Definition 5.**  $c_i = \langle q, MT_i, DT_i \rangle$  is the class  $i$ , where  $i \in N$ ,  $q \in VI$  is the class name,  $MT_i$  is the set of the class methods, and  $DT_i$  is the set of the class member data. For NOO languages, the only class with methods is a fictitious one (class 0). Therefore,  $MT_0$  is the set of all the methods, and  $DT_0$  is the set of global variables. Other classes  $c_i$  can exist if and only if  $MT_i = \emptyset$  (i.e., they are typically called structures).

**Definition 6.**  $CL$  is the set of classes  $c_i$  declared in the specification, i.e.,  $CL = \bigcup_i c_i$ .

**Definition 7.**  $MT$  is the set of methods associated with all the classes, i.e.,  $MT = \bigcup_i MT_i$ . For NOO languages,  $MT \equiv MT_0$  is the set of all the procedures.

**Definition 8.**  $o_k^i \in VI$ , where  $i, k \in N$ , is the instance  $k$  of a class  $c_i$ .

**Definition 9.**  $OB$  is the set of all the instances of all the classes, i.e.,  $OB = \bigcup_{i,k} o_k^i$ .

**Definition 10.**  $f_{k,j}^i$ , where  $i, j, k \in N$ , is the instance of the method  $m_{i,j}$  of the instance  $k$  of class  $c_i$ . For NOO languages, that allow multiple instances of a method (i.e., procedure),  $f_{k,j}^i$  is the instance  $k$  of the method  $m_{0,j}$ .

**Definition 11.**  $MI$  is the set of all the method instances in the specification, i.e.,  $MI = \bigcup_{i,j,k} f_{k,j}^i$ .

**Definition 12.**  $t_i = \langle s, w, d, g, f \rangle$  is the data transfer  $i$ , where  $s, d \in MI$ , with  $s \neq d$ , are the source and the destination method instances of the data transfer, respectively,  $w \in TQ = \{MC, CS\}$  is a data transfer qualifier where  $MC$  indicates a method call (i.e., a procedure call) while  $CS$  indicates a communication (or synchronization) operation,  $g \in N$  is the exchanged data size in bits, and  $f \in Z$  is the average number of times that the data exchange occurs.

**Definition 13.**  $DT$  is the set of all data transfers of the specification, i.e.,  $DT = \bigcup_i t_i$ .

**Definition 14.**  $G = \langle MI, DT \rangle$  is the graph representing a given specification, where the graph nodes are the instances of methods and the graph edges are data transfers.

## 5 METRICS FOR COANALYSIS

The coanalysis activity aims at obtaining as much information as possible about the system by statically analyzing the specification. The goal of this step is to statically detect the most suitable class of processing elements for the execution of each system functionality. To this purpose, several subtasks are required [11], [12]: an architectural analysis of the available processing elements to determine their relevant features, the definition of a set of patterns able to identify subsets of the specification that could exploit the identified architectural features, and the definition of a set of metrics able to provide meaningful indications to drive design choices.

### 5.1 Characterization of Executors

This section describes the analysis performed to detect the most relevant exploitable architectural features of the identified executors classes.

#### 5.1.1 GPP Architectural Features

General Purpose Processors (GPP) have been designed to be useful in several contexts and, so, it is difficult to detect particular features that strongly identify a GPP-suitable application. They are typically adopted as control elements, I/O manager, for general computations and in complex systems that use an operating system.

#### 5.1.2 DSP Architectural Features

Digital Signal Processors (DSP) are dedicated to digital signal processing applications and, so, they present a loss of generality with respect to GPP and a higher cost, but they

provide a better performance in the execution of a particular set of instructions [24]. The architectural features included in a DSP allow concurrent loading of multiple operands, concurrent execution of sums and multiplications, fast management of loops, and fast access to sequential memory space (e.g., array).

### 5.1.3 ASIC-Like Devices Architectural Features

Application Specific Integrated Circuits (ASIC) are developed for specific applications. They generally exhibit high performance, but their design and development costs are so relevant that they are only affordable for high volumes. Field Programmable Devices (FPD) are arrays of logic blocks with programmable interconnections to shape the desired functionality. Such devices are a trade-off between processors and ASICs with respect to performance, flexibility, and cost (e.g., Field Programmable Gate Arrays [25]).

We worked on the identification of a set of features that allow an early selection of the functionalities able to exploit ASIC-like devices. Since a mismatch between application data-path requirements and the characteristics of the processor data-path could lead to inefficient use of processor resources (nonstandard data-path), the ASIC-like devices are more suitable to perform bit manipulation operations (shifting, Boolean operators, etc.). Moreover, repeated operations of similar types on large regular data sets are also ideal candidates for ASIC-like implementations. Regularity in operations imposes less demand on the control unit complexity, better exploiting the available resources.

It is worth noting that the presented methodology could be easily extended in order to consider new technologies provided by research and the market (e.g., Tensilica Configurable Processors [35]). This can be obtained by defining new specific metrics to highlight relevant architectural features and used to identify design choice.

## 5.2 The Proposed Approach

Considering the architectural features previously described, it is possible to define a set of patterns able to identify subsets of the specification that match some executor features and a set of metrics that quantify such a matching. Finally, these metrics are combined to build a global metric (the *Affinity*) that suggests the most suitable processing elements for the execution of a given functionality.

**Definition 15 (The Affinity ( $A_m$ )).** The affinity  $A_m = [A_{GPPm} A_{DSPm} A_{HWm}]$  of a method (i.e., a procedure)  $m$  is a triplet of values in the interval  $[0, 1]$  that provides a quantification of the matching between the structural and functional features of the functionality implemented by the method and the architectural features for each one of the considered executor classes (i.e., GPP, DSP, ASIC/FPGA).

An affinity of 1 toward an executor class indicates a perfect matching, while a 0 affinity indicates no matching at all.

With respect to previous attempts to perform similar analysis, the proposed one is more general and accurate since it provides a specific quantification of the matching through the affinity measure. For example, in [26], the efficiency of GPPs and FPGAs is evaluated only with respect to the exploitation of the available area evaluating the *spatial efficiency* of a device. In [27], the authors create a methodology that fully characterizes any algorithm with

respect to the elements of its structure that affect its implementation. The identified elements are meaningful; however, only a few of them are supported by an effective quantification approach and the metrics defined are strictly bounded to high-level synthesis issues (e.g., area estimation of a custom ASIC implementation). A codesign oriented work is instead the one presented in [28], where the concept of *hardware/software repelling* is used to drive a hardware/software partitioning algorithm. The approach is based on the analysis of the system functionalities; it detects a set of features that suggest the *repelling* of certain functionalities toward a certain type of implementation. Unfortunately, the work considers only one type of software executor and the set of features considered is not clearly defined. Finally, the work in [4] considers multiprocessor systems synthesis, starting from an object-oriented specification, and it analyzes subsets of such a specification in order to detect features that allow marking them as *control dominated*, *data transformation dominated*, or *memory access dominated*. However, it does not consider dedicated hardware devices, works with a too coarse granularity level, and poorly defines the metrics to be used within the methodology.

Let us describe the metrics defined to compose the affinity.

### 5.2.1 Data Oriented Metrics

The goal of this metric is to take into account the type of data involved in the execution of a given functionality.

**Definition 16 (Data Ratio ( $DR_{m,t}$ )).** For each method  $m$  and for each allowed type  $t$  (e.g., *int*, *float*, etc.),  $DR_{m,t}$  is defined as the fraction of declarations of type  $t$  with respect to the total number of declarations made in  $m$ .

### 5.2.2 Structural Metrics

The goal of these metrics is to identify the structural properties of a functionality focusing on the analysis of the control flow complexity.

**Definition 17 (Control Flow Complexity ( $CFC_m$ )).** For each method  $m$ ,  $CFC_m$  is defined as the ratio between the number of source lines that contains loop or branch statements and the total number of lines.

**Definition 18 (Loop Ratio ( $LR_m$ )).** For each method  $m$ ,  $LR_m$  is defined as the ratio between the number of source lines that contain loop statements and the total number of lines.

Such a metric allows discriminating between computational and control oriented functionalities.

### 5.2.3 DSP Oriented Metrics

The goal is to identify functionalities suitable to be executed by a DSP by considering those characteristics that exploit the most relevant architectural features of such executor class: *Circular Buffering*, *MAC operations*, and *Super Harvard Architecture* [24]. For the circular buffering, the goal is to identify subsets of the specification accessing a linear data structure (1D-array, row or column of 2D-array) to shift it one or more positions.

**Definition 19 (Strong Circularity Degree ( $SCD_m$ )).** For each method  $m$ ,  $SCD_m$  is the ratio between the number of source lines that contain expressions of the form  $v[i] = v[i \pm K]$  and

the total number of lines, where  $\mathbf{v}$  is a vector (or a row/column of a matrix), and  $K$  is a constant.

**Definition 20 (Weak Circularity Degree ( $WCD_m$ )).** For each method  $m$ ,  $WCD_m$  is the ratio between the number of source lines that contain expressions of the form  $\mathbf{v}[K] = f(\mathbf{v}[i])$  or  $\mathbf{q} = f(\mathbf{v}[i])$  and the total number of lines, where  $\mathbf{v}$  is a vector (or a row/column of a matrix),  $K$  is a constant, and  $f(\mathbf{v}[i])$  is an expression involving  $\mathbf{v}[i]$ .

For the MACs, the goal is to identify subsets of the specification that express a particular mix of operations (i.e., a sum and a multiplication) that a DSP can perform concurrently.

**Definition 21 (Strong MAC Degree ( $SMD_m$ )).** For each method  $m$ ,  $SMD_m$  is the ratio between the number of source lines inside a loop that contain expressions of the form  $\mathbf{s1} = \mathbf{s1} + \mathbf{sx} * \mathbf{sy}$  and the total number of lines.

**Definition 22 (Weak MAC Degree ( $WMD_m$ )).** For each method  $m$ ,  $WMD_m$  is the ratio between the number of source lines that contain, outside a loop, expressions of the form  $\mathbf{s1} = \mathbf{s1} + \mathbf{sx} * \mathbf{sy}$  and the total number of lines.

For concurrent memory accesses, the goal is to identify subsets of the specification able to exploit concurrent memory accesses to instructions and data capability [24].

**Definition 23 (Strong Harvard Degree ( $SHD_m$ )).** For each method  $m$ ,  $SHD_m$  is the ratio between the number of source lines that contain, inside a loop, expressions with the structure  $\mathbf{v}[i] \langle op \rangle \mathbf{w}[i]$  or  $\mathbf{q} \langle op \rangle \mathbf{w}[i]$  and the total number of lines, where  $\mathbf{v}$  and  $\mathbf{w}$  are vectors, and  $\langle op \rangle$  is an operator different from the assignment one.

**Definition 24 (Weak Harvard Degree ( $WHD_m$ )).** For each method  $m$ ,  $SHD_m$  is the ratio between the number of source lines that contain, outside a loop, expressions with the structure  $\mathbf{v}[i] \langle op \rangle \mathbf{w}[i]$  or  $\mathbf{q} \langle op \rangle \mathbf{w}[i]$  and the total number of lines, where  $\mathbf{v}$  and  $\mathbf{w}$  are vectors, and  $\langle op \rangle$  is an operator different from the assignment one.

#### 5.2.4 GPP Oriented Metrics

The goal is to identify functionality that significantly relies on operations that involve conditional dependent control flows, complex data structures and complex I/O management.

**Definition 25 (Conditional Ratio ( $CR_m$ )).** For each method  $m$ ,  $CR_m$  is equal to  $CFC_m - LR_m$ , where  $CFC_m$  is the Control Flow Complexity and  $LR_m$  is the Loop Ratio.

**Definition 26 (I/O Ratio ( $IOR_m$ )).** For each instance of method,  $IOR_m$  is the ratio between the number of source lines that contain I/O operations (e.g., read, write, etc.) and the total number of lines.

**Definition 27 (Structure Ratio ( $STR_m$ )).** For each method  $m$ , the  $STR_m$  is the ratio between the number of structures declared and the total number of declarations.

#### 5.2.5 ASIC-Like Oriented Metrics

The goal is to identify regular functionalities that significantly rely on bit-manipulation operations. Therefore, the following metric is defined:

**Definition 28 (Bit Manipulation Rate ( $BMR_m$ )).** For each method  $m$ ,  $BMR_m$  is the ratio between the number of source lines that contain bit-manipulation operations (e.g., and, or, xor, etc.) and the total number of lines.

The information gathered by means of the metrics previously defined is organized in a global metric that allows a straightforward characterization of a functionality with respect to each considered executor class. Such a global metric, called Affinity, is operatively defined as follows.

#### 5.2.6 The Affinity Metric

The Affinity of each method  $m$  can be expressed by a normalization function applied to a linear combination of the metrics, with weights that depend on the considered executor class, as follows:

$$A_m^T = f(W \cdot C_m^T),$$

where

$$A_m = [A_{GPPm} \quad A_{DSPm} \quad A_{HWM}],$$

$$W = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix},$$

and  $C_m$  is a 14-element row vector collecting, in this order, the costs  $SCD_m$ ,  $WCD_m$ ,  $SHD_m$ ,  $WHD_m$ ,  $SMD_m$ ,  $WMD_m$ ,  $IOR_m$ ,  $CR_m$ ,  $LR_m$ ,  $BMR_m$ ,  $DR_{bit}^m$ ,  $\sum_{z \in \{char, string\}} DR_z^m$ ,  $\sum_{z \in \{int, read\}} DR_z^m$ , and  $STR_m$ . The weights of the matrix  $W$  are set to 1 when the associated metric is meaningful for a given executor class, to 0 otherwise. In this way, the affinity represents the sum of all the contributions determined by each relevant metric normalized by means of the arctangent function scaled of a  $\pi/2$  factor. Finally, to better discriminate between low and high affinity values, a quadratic form is introduced, leading to the following normalization function:

$$f(x) = \frac{\arctan(2\pi x^2)}{\pi/2}.$$

#### 5.3 Validation

A meaningful validation process has been set up based on a C test suite. A tool has been developed and integrated with a C/C++ code analyzer (GENOA [29]). The adopted benchmark suite is composed of 311 procedures, each representing a specific functionality. A subset of these procedures (about 100) has been selected from applications oriented to digital signal processing. The remaining procedures are representative of a generic set and contain functions such as encoding/decoding, string manipulation, and several common operations. During the validation process, the values of the metrics previously defined have been collected to evaluate the affinity.

Interesting considerations can be made by analyzing the averages of the affinity values on the whole test suite, for the DSP applications, and for the others (see Table 1). The affinity  $A_{DSP}$  for the DSP applications is fairly large with respect to the other affinity values and with respect to the

TABLE 1  
Affinity Average Values per Application Class

	Whole set	Only DSP	All but DSP
$A_{GPP}$	0,46	0,38	0,49
$A_{DSP}$	0,25	0,57	0,10
$A_{HW}$	0,27	0,28	0,27

$A_{DSP}$  values obtained for the other application cases. It is worth noting that  $A_{GPP}$  has the largest average of the whole set, revealing the general purpose nature of the related executors class, while the  $A_{HW}$  indicates, in general, those procedures that exploit some features associated with the ASIC executor class.

## 6 SYSTEM-LEVEL PARTITIONING

When the functionalities of the system should be targeted for a specific implementation, the related task is called system-level partitioning. However, the design space that must be explored when considering all the parameters associated with a multiprocessor architecture may easily become unmanageable, thus making an exhaustive analysis of all the possible solutions impossible: Efficient heuristics are needed to manage the task complexity.

Existing tools dealing with multiprocessor embedded systems codesign start from a heterogeneous specification where the partitioning strongly relies on the experience of the designer [13], [14] or provide a partitioning methodology targeted only to particular applications [37], [40]. Some approaches focus mainly on communication issues [15], [16], while a general approach that considers both communications and different executors is presented in [17], but the *area-delay curves* considered in such a work are impractical. The approaches closer to our proposal are [18] and [19]. They perform a clustering of the behavioral specification, a task classification based on area and time, and a design space exploration by means of transformations. However, they are targeted to single processor systems only.

### 6.1 The Proposed Partitioning Approach

The system partitioning methodology proposed here [11], [34] is composed of two phases: a *clustering* of the system functionalities to achieve a preallocation and a *refinement* phase, driven by a genetic algorithm and a global cost function. The resulting design space exploration strategy works with a variable-granularity to increase efficiency in allocating the functionality and the cost function includes four main components: affinity, communication, load, and economical cost.

#### 6.1.1 Metrics

The comprehensiveness of the parameters composing the cost function is crucial to drive the design space exploration. In the following, the proposed metrics are detailed.

**Affinity Index.** Given a solution, it is very important to evaluate the matching between the properties of the functionalities and the processing elements on which they

have been allocated. To this purpose, the *Affinity Index* ( $I_A$ ) of a solution has been defined as follows:

$$I_A = \frac{\sum_{m \in MI} \sum_{e \in \{GPP, DSP, HW\}} x_{e,m} (1 - A_{e,m})}{|MI|},$$

where  $x_{e,m}$ , i.e.,  $x_{GPPm}$ ,  $x_{DSPm}$ ,  $x_{HWm}$ , are 1 or 0, respectively, if the method instance  $m$  (of the behavioral, possibly OO specification) is allocated or not to the associated type of executor;  $A_{GPPm}$ ,  $A_{DSPm}$ ,  $A_{HWm}$  are the affinities of an instance of method;  $|MI|$  is the number of instances of a method. The Affinity Index falls in the range  $[0, 1]$  and values closer to 0 indicate a perfect match between functionalities and executors.

**Load Indexes.** For each solution, it is very important to consider the balancing of the workload over the available processing elements. For this reason, two indexes are considered to distinguish software from hardware. The Load Index for software executors ( $I_{LSW}$ ) is evaluated as follows:

$$I_{LSW} = \frac{1}{m} \sum_{j=1}^m \frac{1}{n_j L_{SW}} \sum_{i=1}^{n_j} |l_{i,j} - L_{SW}|,$$

where  $m$  is the number of software executors present in the solution (GPP or DSP),  $n_j$  is the number of method instances allocated on the software executor  $j$ ,  $l_{i,j}$  is the estimated load [11] of the  $i$ th functionality on the software executor  $j$ , and  $L_{SW}$  is the ideal load. It is theoretically 100 percent, but it is generally set to about 70 percent (a typical value that allows considering the presence of a possible operating system [20]). This index varies in the interval  $[0, 1]$  and captures the average of the differences between measured and ideal load. Values closer to 0 indicate that each processing element presents a load similar to the ideal situation.

The Load Index for hardware executors ( $I_{LHW}$ ) avoids the overloading of hardware devices, i.e., the allocation of a number of functionalities whose implementation exceeds the available gates:

$$I_{LHW} = \frac{1}{m} \sum_{j=1}^m \frac{s_j}{\sum_{i=1}^{n_j} g_{i,j}},$$

where

$$s_j = \begin{cases} 0 & \sum_{i=1}^{n_j} g_{i,j} \leq G_j \\ \sum_{i=1}^{n_j} g_{i,j} - G_j & \sum_{i=1}^{n_j} g_{i,j} > G_j \end{cases}$$

and where  $m$  is the number of hardware devices (ASIC or FPGA),  $n_j$  is the number of method instances allocated on the hardware executor  $j$ ,  $g_{i,j}$  is the number of gates needed to implement the  $i$ th functionality on the executor  $j$ , and  $G_j$  is the maximum number of gates on the hardware executor  $j$ . The load index values for hardware range in the  $[0, 1]$  interval and a value of 0 indicates that no hardware executors are overloaded.

**Communication Index.** The information on communication cost, gathered during the functional cosimulation, allows the evaluation of the exchanged data size due to the interactions between functionalities allocated on different

processing elements. An associated metric, called Communication Index ( $I_C$ ), has been defined:

$$I_C = \frac{\sum_{i \in MI} \sum_{j \neq i \in MI} com_{i,j}}{\sum_{i \in MI} \sum_{j \neq i \in MI} b_{i,j} c_{i,j}},$$

where

$$com_{i,j} = \begin{cases} 0 & E_i = E_j \\ b_{i,j} c_{i,j} & E_i \neq E_j \end{cases}$$

and where  $MI$  is the set of instances of a method,  $b_{i,j}$  is the size of the data exchanged between  $i \in MI$  and  $j \in MI$ ,  $c_{i,j}$  is the number of interactions between  $i \in MI$  and  $j \in MI$ , and  $E_i$  and  $E_j$  are, respectively, the executors of  $i \in MI$  and  $j \in MI$ . The communication index values belong to the interval  $[0, 1]$ . Values closer to 0 indicate that the communication between functionalities allocated on different executors are negligible with respect to the total number of communications.

**Physical Cost Index.** Another important parameter to be considered when evaluating a possible solution to the partitioning problem is the physical cost of the components present in the proposed architecture. The Physical Cost Index ( $I_\$$ ) is evaluated as follows:

$$I_\$ = \frac{1}{|MI| \$_{MAX}} \sum_{j \in E} \$_j,$$

where  $|MI|$  is the number of instances of a method,  $E$  is the set of the executors present in the proposed architecture,  $\$_j$  is the physical cost of the executor  $j \in E$ , and  $\$_{MAX}$  is the cost of the most expensive executor in  $E$ . The Physical Cost Index values belong to the interval  $[0, 1]$  and represent the cost of the proposed solution with respect to the most expensive one. Values closer to 0 indicate a cheaper solution.

### 6.1.2 Cost Function

By combining the metrics described above, it is possible to build a cost function that allows a comparison between different solutions identifying the one better representing a trade-off between the different parameters. In fact, the affinity and load parameters tend to separate the functionalities in order to balance the load and exploit the processing elements features, while communications and physical costs tend to keep together such functionalities to minimize the number of processing elements. These considerations can be taken into account through a linear combination of the indexes (other approaches are still under investigation), thus obtaining the following cost function expression:

$$CF = w_A I_A + w_{Lsw} I_{Lsw} + w_{Lhw} I_{Lhw} + w_C I_C + w_\$ I_\$,$$

where  $w_A$ ,  $w_{Lsw}$ ,  $w_{Lhw}$ ,  $w_C$ , and  $w_\$$  are the weights associated with the different parameters, thus providing the possibility of customizing their importance.

### 6.1.3 Methodology

The proposed methodology explores the design space by using a genetic algorithm. It works with a variable coarse-grain granularity where a functionality, or a group of them,

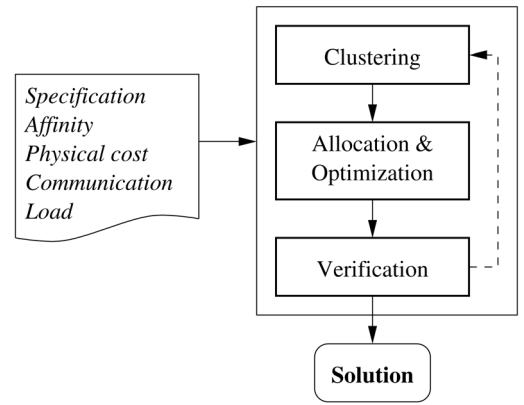


Fig. 3. Partitioning methodology steps.

is allocated to the same processing element. The methodology is composed of several steps that are iteratively repeated, working each time with a finer granularity, until a solution to the problem is found. Fig. 3 shows the steps of the algorithm, detailed in the following.

**Clustering.** The clustering phase represents the starting point of the methodology. Its purpose is to select groups of functionalities, bound by the *caller-callee* relationship, to be considered as clusters, i.e., not to be further decomposed in the following steps of the same iteration of the methodology.

If a solution is not found, a more detailed analysis of the elements composing the clusters is required. Since each cluster is allocated on the same processing element, the former solutions tend to be the cheapest, while the latter tend to better exploit the characterization of each single functionality with respect to the type and number of executors.

**Allocation and optimization.** This step allocates the clusters provided by the previous steps to different processing elements by exploring the design space in search of the allocation that minimizes the cost function previously defined. This step is based on a genetic algorithm where each individual of the considered population represents a possible architecture/allocation item. The structure of the individual is represented by an entry for each cluster considered during the current iteration (see Fig. 4): Each cluster is associated with a type of processing element and an instance of it (the maximum number of instances allowed for each type of processing element is specified by the designer).

The initial population is randomly generated, while, during the evolution of the population, the algorithm performs the optimizations that minimize the cost function following the classical rules of genetic approaches [21].

**Verification.** The architecture/allocation solution provided by the previous step is verified by means of the *Timing Simulation* (such a simulation methodology, derived from [11], [41], [42], also contributes to the definition of the interconnection network and the scheduling policy). If such a combination is not a valid solution to the partitioning problem (i.e., it does not meet the timing constraints), the whole process is repeated, starting from a finer granularity clustering.



Cluster 1	Cluster 2	Cluster 3	...	Cluster $n$
GPP	HW	GPP	...	DSP
1	1	3	...	2

Fig. 4. Individual structure.

## 7 METHODOLOGY VALIDATION

To support the proposed system design methodology, a set of tools has been developed and integrated in the design flow of Fig. 1. A tool to compute the affinity values has been implemented by means of a C/C++ code analyzer (GENOA [29]). The data are then provided to the system design exploration tool, called EMuP (*Embedded Multiprocessor Partitioning*), that has been developed following two different goals: to integrate the tool in the proposed environment and to provide an easy portability toward different ones. EMuP has been developed in C++ and it is based on a library of classes for genetic algorithms (GALIB [22]). The portability-enabler feature is the input format accepted by the tool: It is based on the VCG format [2], a third party format that can be managed and visualized with third party open source tools. To give the flavor of how the methodology actually works on real designs and to show its effectiveness, two examples of system-level partitioning are reported in the following.

### 7.1 Case Study 1

The first case study considers a *synthetic* application (it is a composition of the programs belonging to the test suite described in Section 5.3) that consists of 52 methods and its Procedure Interaction Graph is represented in Fig. 5. The target architecture is composed of an unconstrained number of GPP, DSP, and FPGA.

In the following, the application of each step of the proposed flow (Fig. 1) is briefly presented. The coanalysis step aims at statically detecting the most suitable processing element for the execution. The next step estimates the

performance of the system. In this example, the performed analysis for the software implementation is targeted to the Intel 486T GX [23] processor. The hardware timing characterization has been performed by considering the LSI Logic 10k technology library with a 150 MHz system clock. The *functional simulation* represents the checkpoint for the functional correctness of the system specification. Moreover, it allows the evaluation of different important measures that characterize the system: profiling, dynamic communication cost, and load estimation. The *load estimation* task requires the performance estimation data provided by the coestimation step and a timing simulation. Such a simulation is performed by considering a target architecture composed of one general purpose processor that executes all the procedures, thus producing a reference time  $T_{REF}$ . The average  $T_{REF}$  has been approximately  $280 \cdot 10^3$  clock cycles for the considered architecture. By imposing, as timing constraint to the system, an execution time of  $0.4 \cdot T_{REF}$ , it has been possible to estimate the load provided by each procedure to a GPP executing the application with such a constraint. Finally, to represent the information needed for the design space exploration in a compact and general way, a VCG file [2], representing the procedural-level model of the system annotated with affinity data, the dynamic communication costs, and the estimated load, is provided to the simulator. Hence, the system design exploration is divided into two iterative steps: *partitioning and architecture selection* and *timing cosimulation*. For the partitioning, the weights of the cost function have been set as follows:

$$I_A = I_{Lsw} = I_{Lhw} = 4.0 \quad I_C = I_s = 2.0,$$

while the ideal load  $L_{SW}$  considered in the load index has been set to 70 percent (according to [20]). For the timing cosimulation, the timing constraint imposed on the execution time of the whole application, as described previously, is 40 percent of  $T_{REF}$ . The default policy scheduling is a

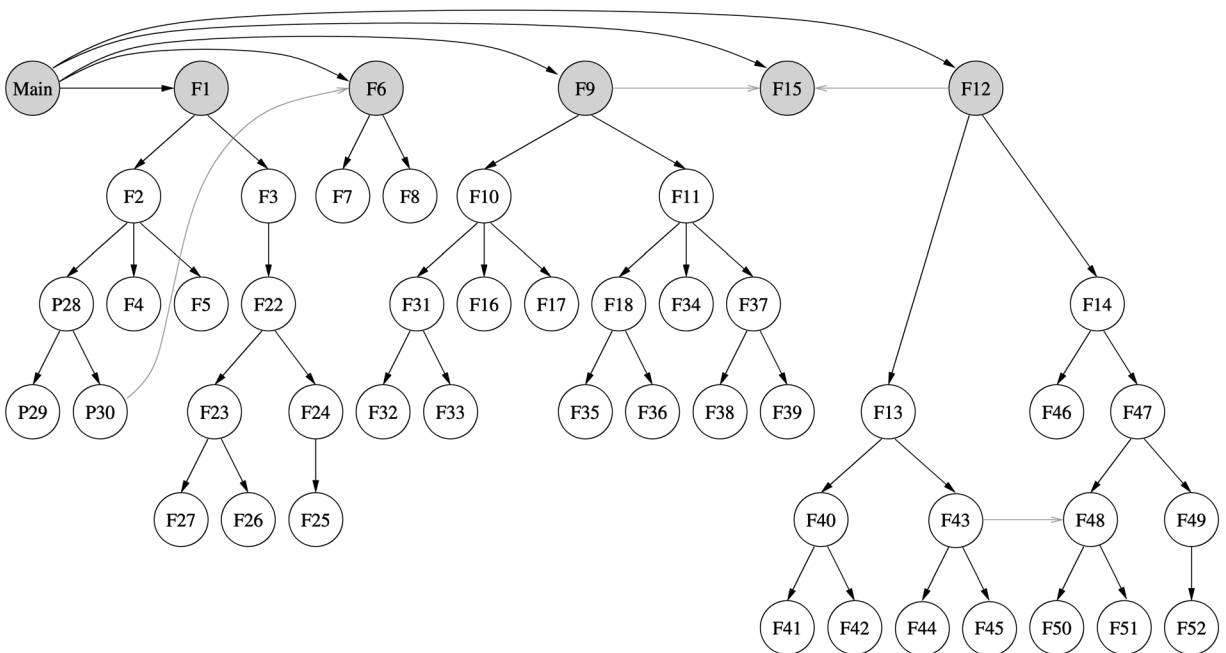


Fig. 5. Procedure Interaction Graph.

TABLE 2  
Design Space Exploration

Iter.	$I_C$	$I_{LSW}$	$I_A$	$I_S$	Architecture			Simulated time	
					GPP	DSP	FPGA	$n_{ave} = 1$	$n_{ave} = 2$
0	0.000	0.340	0.390	0.276	0	1	0	$0.89T_{REF}$	$0.89T_{REF}$
1	0.320	0.040	0.394	0.334	3	0	0	$0.51T_{REF}$	$0.45T_{REF}$
3	0.012	0.124	0.386	0.377	2	1	0	$0.42T_{REF}$	$0.41T_{REF}$
7	0.006	0.023	0.388	0.377	2	1	0	$0.40T_{REF}$	$0.38T_{REF}$

round-robin one and the parameters of the communication model (i.e., the number of allowed concurrent communications and number of hops [11]) are initially  $n_{ave} = 1$  and  $h_{ave} = 0$  (i.e., a single bus). The results obtained are shown in Table 2.

Iteration 0 considers only one cluster (i.e., only one executor) and the affinity metrics suggest the choice of a DSP. However, one executor does not meet the timing constraints. The next iteration found a new minimum for the considered cost function. By simulating such a solution, the simulated time is  $0.51 \cdot T_{REF}$ . The simulator can also provide a measure related to the average number of concurrent communications: In this case, such a number is 1.6 and, therefore, a new simulation is performed with  $n_{ave} = 2$ . However, the results obtained are not successful. Another cost function minimum is obtained during iteration 3 (each iteration works with a finer granularity), but also in this case the timing constraint is not satisfied. Finally, iteration 7 reaches the solution: The proposed architecture/allocation (with both the nave values) satisfies the imposed simulated time.

To perform the design space exploration, the partitioning tool, implemented in C++ and executed on a Pentium III (700 MHz, 640 MBytes RAM), has processed a population of 700 individuals for 3,500 generations in less than 20 minutes (comprehensive of four cosimulations performed on a SUN UltraSparc2 running at 256 MHz and equipped with 128 MBytes RAM).

## 7.2 Case Study 2

The second case study considers a real application that manages digital signatures in MPEG encoded images. The application is described by means of 21 methods and its Procedure Interaction Graph is shown in Fig. 6. The target architecture is composed of an unconstrained number of GPPs, DSPs, and FPGAs.

As in the previous case study, each step of the proposed flow (Fig. 1) has been applied to the specification. The

average  $T_{REF}$  has been, for the considered architecture, of approximately  $300 \cdot 10^6$  clock cycles and the time constraint has been set to  $0.5 \cdot T_{REF}$ . The cost function weights, the ideal load, and the simulator configuration have been kept equal to the previous examples. The results obtained during the design space exploration are shown in Table 3.

Iteration 0 considers only one cluster (i.e., only one executor) and the affinity metrics suggest the choice of a DSP. However, one executor does not meet the timing constraints. Iteration 1 found a new minimum for the considered cost function. The simulated time for such a solution is  $0.71 \cdot T_{REF}$ . The same considerations apply in the following of the case study. Another cost function minimum is obtained during iteration 6 (each iteration works with a finer granularity), but also in this case the timing constraint is not satisfied. Iteration 8 identifies a solution: The proposed architecture/allocation with  $n_{ave} = 2$  (e.g., a crossbar switch) reaches the imposed simulated time; however, the margin is considered too low and, therefore, a new iteration is performed. The last iteration, changing only the allocation of the procedures on the same architecture, finds a better solution that is considered acceptable. To perform such a design space exploration, the partitioning tool, implemented in C++ and executed on a Pentium III (700 MHz, 640 MBytes RAM), has processed a population of 500 individuals for 3,500 generations in a time less than 35 minutes (comprehensive of five cosimulations performed on a SUN UltraSparc2 running at 256 MHz and equipped with 128 MBytes RAM).

## 8 CONCLUDING REMARKS AND FUTURE WORK

A comprehensive codesign environment for heterogeneous multiprocessor SoC has been developed to support the designer from the specification phase at a high level of abstraction to the definition of the target architecture and the identification of the best hw/sw partitioning. In particular, the information provided by the coanalysis step allows an effective design space exploration, based on two main tasks: partitioning and architecture selection and timing cosimulation. The partitioning methodology is based on an initial clustering and on a heuristic optimization algorithm supported by a proper set of metrics. Such an algorithm provides solutions that consist of an architecture and the binding between parts of the system behavior and the selected components of the architecture. Such solutions are then validated by means of a system-level cosimulation strategy that considers the presence of multiple executors and a

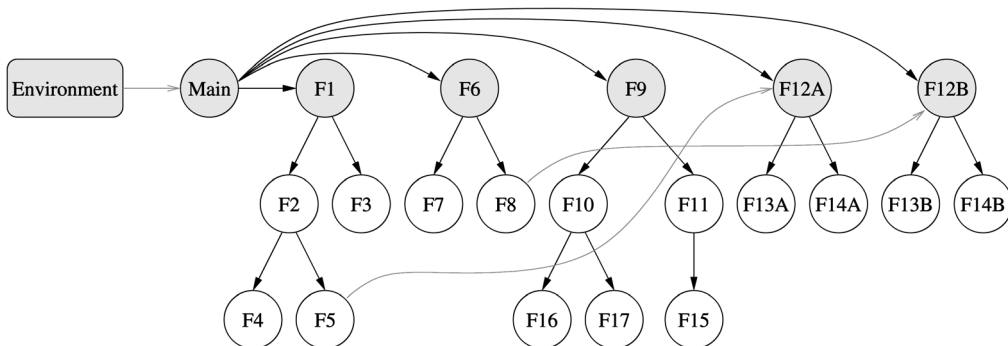


Fig. 6. Procedure Interaction Graph.

TABLE 3  
Design Space Exploration

Iter.	$I_C$	$I_{LSW}$	$I_A$	$I_g$	Architecture			Simulated time	
					GPP	DSP	FPGA	$n_{ave} = 1$	$n_{ave} = 2$
0	0.000	0.241	0.292	0.477	0	1	0	$0.92T_{REF}$	$0.92T_{REF}$
1	0.120	0.140	0.344	0.655	0	2	0	$0.71T_{REF}$	$0.65T_{REF}$
6	0.201	0.640	0.376	0.698	1	1	1	$0.58T_{REF}$	$0.52T_{REF}$
8	0.146	0.730	0.398	0.684	1	2	0	$0.54T_{REF}$	$0.49T_{REF}$
9	0.138	0.720	0.401	0.684	1	2	0	$0.50T_{REF}$	$0.47T_{REF}$

high-level model for the communications. The experimental results obtained so far are encouraging and justify current efforts to improve both methodologies and tools.

Currently, the presented methodology suffers from some limitations that have been considered as part of future activities targeting the improvement of the results quality. In particular, the “Affinity” and other metrics will be improved by considering not only static effects, but also dynamic effects by instrumenting the code in order to identify the real contribution of loop, calls, data structures, etc. However, a static approach is faster than a dynamic method to cope with large specifications. Moreover, this approach was further encouraged by some experimental results, which provided satisfactory results.

Future activities will also be focused on considering compiler optimization (constants propagation, dead code elimination, loop splitting, loop tiling, etc.), architectural aspects (pipelining, stalls, cache miss/hit, superscalarity, etc.), and logic synthesis optimization. To deal with these aspects, we will explore the results produced by methodologies developed for power modeling [43], [44], [45], [46]. These latter methodologies, extended to the metrics presented in this paper, make possible a more accurate estimation and partitioning at the cost of implementing a more sophisticated analysis.

## REFERENCES

- [1] T.A.C.M. Claasen, “High Speed: Not the Only Way to Exploit the Intrinsic Computational Power of Silicon,” *Proc. IEEE Int’l Solid-State Circuits Conf. (ISSCC 1999), Digest of Technical Papers*, pp. 22-25, 1999.
- [2] G. Sander, R. Tamassia, and I.G. Tollis, “Graph Layout through the VCG Tool,” *Proc. DIMACS Int’l Workshop (GD ’94)*, 1994.
- [3] J. Choi, M. Burke, and P. Carini, “Efficient Flow-Sensitive Interprocedural Computation of Pointer-Induced Aliases and Side Effects,” *Proc. 20th Ann. ACM Symp. Principles of Programming Languages*, pp. 233-245, 1993.
- [4] L. Carro, M. Kreutz, F.R. Wagner, and M. Oyamada, “System Synthesis for Multiprocessor Embedded Applications,” *Proc. Design, Automation and Test in Europe Conf. and Exhibition*, pp. 697-702, 2000.
- [5] M.T.J. Strik, A.H. Timmer, J.L. Van Meerbergen, and G. VanRootselaar, “Heterogeneous Multiprocessor for the Management of Real-Time Video and Graphics Streams,” *IEEE J. Solid-State Circuits*, vol. 35, no. 11, pp. 1722-1731, Nov. 2000.
- [6] J. Hilgenstock, K. Herrmann, S. Moch, and P. Pirsch, “A Single-Chip Video Signal Processing System with Embedded DRAM,” *Proc. IEEE Workshop Signal Processing Systems (SiPS 2000)*, pp. 23-32, 2000.
- [7] B.P. Dave, G. Lakshminarayana, and N.K. Jha, “COSYN: Hardware-Software Co-Synthesis of Heterogeneous Distributed Embedded Systems,” *IEEE Trans. Very Large Scale Integrated Systems*, vol. 7, no. 1, pp. 92-104, Mar. 1999.
- [8] B.P. Dave and N.K. Jha, “COHRA: Hardware-Software Cosynthesis of Hierarchical Heterogeneous Distributed Embedded Systems,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 10, pp. 900-919, Oct. 1998.
- [9] P.A. Hsiung, “CMAPS: A Cosynthesis Methodology for Application-Oriented General-Purpose Parallel Systems,” *ACM Trans. Design Automation of Electronic Systems*, vol. 5, no. 1, pp. 58-81, Jan. 2000.
- [10] J. Axelsson, “Towards System-Level Analysis and Synthesis of Distributed Real-Time Systems,” *Proc. Fifth Int’l Conf. Information Systems Analysis and Synthesis*, vol. 5, pp. 40-46, 1999.
- [11] L. Pomante, “System-Level Co-Design of Heterogeneous Multiprocessor Embedded Systems,” PhD thesis, DEI, Politecnico di Milano, 2002.
- [12] L. Pomante, W. Fornaciari, D. Sciuto, and F. Salice, “Metrics for Design Space Exploration of Heterogeneous Multiprocessor Embedded Systems,” *IEEE Proc. 10th Workshop Hardware/Software Codesign*, May 2002.
- [13] G.F. Marchioro, J.M. Daveau, and A.A. Jerraya, “Transformational Partitioning for Co-Design of Multiprocessor Systems,” *IEEE Proc. Int’l Conf. Computer Aided Design*, 1997.
- [14] M. Srivastava and R. Brodersen, “Siera: A Unified Framework for Rapid-Prototyping of System-Level Hardware and Software,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, pp. 676-693, June 1995.
- [15] S. Agrawal and R.K. Gupta, “Data-Flow Assisted Behavioral Partitioning for Embedded Systems,” *IEEE Proc. 34th Design Automation Conf.*, pp. 709-712, 1997.
- [16] P.V. Knudsen and J. Madsen, “Graph Based Communication Analysis for Hardware/Software Codesign,” *IEEE Proc. Seventh Workshop Hardware/Software Codesign*, pp. 131-135, 1999.
- [17] J.M. Chang and M. Pedram, “Codex-dp: Co-Design of Communicating Systems Using Dynamic Programming,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 7, pp. 732-744, July 2000.
- [18] J.I. Hidalgo and J. Lanchares, “Functional Partitioning for Hardware-Software Codesign Using Genetic Algorithms,” *Proc. 23rd EUROMICRO Conf.*, pp. 631-638, 1997.
- [19] J.K. Adams and D.E. Thomas, “Multiple-Process Behavioral Synthesis for Mixed Hardware-Software Systems,” *IEEE Proc. Eighth Intl Symp. System Synthesis*, pp. 10-15, 1995.
- [20] C.L. Liu and J.W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment,” *J. ACM*, vol. 20, no. 1, pp. 37-53, 1973.
- [21] M. Mitchell, *An Introduction to Genetic Algorithms*. MIT Press, 1996.
- [22] GALIB, <http://lancet.mit.edu/ga/>, year?
- [23] Intel, “Embedded Ultra Low Power Intel 486T GX Processor Datasheet,” <http://developer.intel.com/design/intarch/DATASHTS/272755.htm>, year?
- [24] T. Cooper, “Taming the SHARC,” technical report, Ixthos Inc., 2000.
- [25] S. Broen, R. Francis, J. Rose, and Z. Vranesic, *Field Programmable Gate Array*. Boston: Kluwer Academic, 1992.
- [26] A. DeHon, “Reconfigurable Architectures for General-Purpose Computing,” Technical Report No. 1586, MIT-AI Laboratory, 1996.
- [27] L. Guerra, M. Potkonjak, and M. Rabaey, “System-Level Design Guidance Using Algorithm Properties,” *J. VLSI Signal Processing*, vol. VII, pp. 73-82, 1994.
- [28] A. Kavalade and A. Lee, “A Global Criticality/Local Phase Driven Algorithm for the Constrained Hardware/Software Partitioning Problem,” *Proc. IEEE CODES/CASHE*, pp. 42-48, 1994.
- [29] P. Devanbu, “GENOA: A Customizable, Language- and Front-End Independent Code Analyzer,” *Proc. Int’l Conf. Software Eng. (ICSE)*, 1992.
- [30] <http://www.systemc.org>, year?
- [31] L. Choi and P.-C. Yew, “Compiler Analysis for Cache Coherence: Interprocedural Array Data-Flow Analysis and Its Impact on Cache Performance,” *IEEE Trans. Parallel and Distributed Systems*, vol. 11, no. 9, pp. 879-896, Sept. 2000.
- [32] F. Salice, W. Fornaciari, L. Pomante, and D. Sciuto, “An Internal Representation Model for System-Level Co-Design of Heterogeneous Multiprocessor Embedded System,” *Proc. Forum Specification and Design Languages*, pp. 669-679, Sept. 2003.
- [33] <http://www.synopsys.com>, year?
- [34] L. DelVecchio, W. Fornaciari, L. Pomante, and F. Salice, “Partitioning of Embedded Applications onto Heterogeneous Multiprocessor Architectures,” *Proc. ACM Symp. Applied Computing*, pp. 661-665, Mar. 2003.
- [35] <http://www.tensilica.com>, year?

- [36] D. Lyonnard, Y. Sungjoo, A. Baghdadi, and A.A. Jerraya, "Automatic Generation of Application-Specific Architectures for Heterogeneous Multiprocessor System-on-Chip," *IEEE Proc. Design Automation Conf.*, pp. 518-523, 2001.
- [37] Y. Fei and N.K. Jha, "Functional Partitioning for Low Power Distributed Systems of Systems-on-a-Chip," *IEEE Proc. Asia and South Pacific Design Automation Conf. (ASP-DAC)*, pp. 274-281, 2002.
- [38] 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. Software Eng.*, vol. 28, no. 9, pp. 822-831, 2002.
- [39] A. Baghdadi, D. Lyonnard, N.E. Zergainoh, and A.A. Jerraya, "An Efficient Architecture Model for Systematic Design of Application-Specific Multiprocessor SoC," *IEEE Proc. Design, Automation and Test in Europe*, pp. 55-62, 2001.
- [40] M. Auguin, L. Capella, F. Cuesta, and E. Gresset, "CODEF: A System Level Design Space Exploration Tool," *Proc. IEEE Int'l Conf. Acoustics, Speech, and Signal Processing*, vol. 2, pp. 1145-1148, 2002.
- [41] D. Sciuto, F. Salice, W. Fornaciari, and L. Pomante, "Hw/Sw Cosimulation for Fast Design Space Exploration of Multiprocessor Embedded Systems," *Canadian J. Electrical and Computer Eng.*, vol. 26, nos. 3/4, pp. 135-140, 2001.
- [42] P. Taddei and A. Tornatore, "Sched\_PA: A Scheduler in SystemC," master's thesis, Univ. of Illinois at Chicago, 2003.
- [43] C. Brandolese, F. Salice, W. Fornaciari, and D. Sciuto, "Static Power Modeling of 32-Bit Microprocessors," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 11, pp. 1306-1316, Nov. 2002.
- [44] G. Beltrame, C. Brandolese, W. Fornaciari, F. Salice, D. Sciuto, and V. Trianni, "Modeling Assembly Instruction Timing in Superscalar Architectures," *Proc. IEEE Int'l Symp. System Synthesis*, pp. 132-137, 2002.
- [45] G. Beltrame, C. Brandolese, W. Fornaciari, F. Salice, D. Sciuto, and V. Trianni, "Dynamic Modeling of Inter-Instruction Effects for Execution Time Estimation," *Proc. IEEE Int'l Symp. System Synthesis*, pp. 136-141, 2001.
- [46] C. Brandolese, W. Fornaciari, and F. Salice, "An Area Estimation Methodology for FPGA Based Designs at System-Level," *Proc. IEEE Design Automation Conf.*, pp. 129-132, 2004.



**Carlo Brandolese** received the degree in electronic engineering in 1995 from the Politecnico di Milano, Italy. He worked until 1997 at Central R&D Labs of the Italian telecom company Italtel as CAD engineer, in which position he was responsible for FPGA design flows and methodologies. In 1998, he received the MS degree in information technology from Cefriel (Politecnico di Milano) working on system-level design and codesign issues. Finally, in 2001, he received the PhD degree in information technology and design automation from the Politecnico di Milano with a dissertation on the analysis and optimization of power consumption of heterogeneous embedded system. Since 1998, he has been a consultant at Cefriel in the Embedded System Design Unit and, since 2003, he has been an assistant professor at the Politecnico di Milano. His current interests range from low-power design, software power analysis and optimization, system-level design methodologies, and codesign.



of Appreciation from the IEEE Circuits and Systems Society. Currently, his main effort is in the field of design automation for embedded systems, reconfigurable computing, hardware/software codesign, and low-power system-level analysis/design both for software and hardware. He is a senior member of the IEEE.



His activities mainly focus in the area of analysis and codesign of heterogeneous multiprocessor embedded systems.



ologies of self-checking VLSI systems, hardware/software codesign and, low-power system design. He has published more than 80 papers on CAD for VLSI. He received the best paper awards during IEEE-IJCNN'92, IEEE-ICONIP'95, and IEEE-ICCD'98. He is a member of the IEEE and the Computer Society.



is a member of different program committees of EDA conferences: DAC, DATE, CODES/CASHE, ISSS, DFT, FDL, a member of the executive committee of ICCAD and DATE, and an associate editor of the *IEEE Transactions on Computers*, *Design Automation of Embedded Systems*, and the *Journal of System Architecture*. She was a guest editor of a special issue of *IEEE Design and Test* in 2000. Her research interests cover mainly the following areas: methodologies for codesign of embedded systems, including system verification, design for low power consumption, HW/SW partitioning, and test generation techniques.

**William Fornaciari** received the laurea and PhD degrees in electronic engineering from the Politecnico di Milano, Italy. Currently, he is an associate professor at the Politecnico di Milano and also supervises the Embedded Systems Design (ESD) Unit at the Cefriel research center in Milano. He has published more than 100 papers and received the best paper awards during IEEE-IJCNN'92, IEEE-ICONIP'95, and IEEE-ICCD'98, and, in 1996, the Certification

**Luigi Pomante** received the degree in computer science engineering from the Politecnico di Milano, Italy, in 1998, the MS degree in information technology from Cefriel (a research center collaborating with the Politecnico di Milano), Milano, in 1999, with a thesis on codesign, and the PhD degree in information technology in 2001. He was a researcher at Cefriel from 2001 to 2005 and now he is a researcher at DEWS (a research center of the University of L'Aquila, Italy).

**Fabio Salice** received the degree in electronic engineering and the PhD degree in electronics and communications, both from the Politecnico di Milano, Italy. Since 2002, he has been an associate professor in the Department of Electronics and Information (DEI) at the Politecnico di Milano and, since 1995, he has also been a consultant researcher at Cefriel in the Embedded System Design Unit. His research interests include design and synthesis methodologies of self-checking VLSI systems, hardware/software codesign and, low-power system design. He has published more than 80 papers on CAD for VLSI. He received the best paper awards during IEEE-IJCNN'92, IEEE-ICONIP'95, and IEEE-ICCD'98. He is a member of the IEEE and the Computer Society.

**Donatella Sciuto** received the Laurea in electronic engineering in 1984 and earned the PhD degree in electrical and computer engineering in 1988 from the University of Colorado, Boulder. She was an assistant professor at the University of Brescia, Dipartimento di Elettronica per l'Automazione until 1992. She is currently a full professor in the Dipartimento di Elettronica e Informazione at the Politecnico di Milano, Italy. She is a member the IEEE, IFIP 10.5, EDAA. She

► For more information on this or any other computing topic, please visit our Digital Library at [www.computer.org/publications/dlib](http://www.computer.org/publications/dlib).