Metrics for Design Space Exploration of Heterogeneous Multiprocessor Embedded Systems

Donatella Sciuto, Fabio Salice, Luigi Pomante, William Fornaciari
Politecnico di Milano, DEI, P.zza Leonardo Da Vinci 32, 20133 Milano
sciuto@elet.polimi.it, salice@elet.polimi.it, pomante@cefriel.it, fornacia@elet.polimi.it

ABSTRACT

This paper considers the problem of designing heterogeneous multiprocessor embedded systems. The focus is on a step of the design flow: the definition of innovative metrics for the analysis of the system specification to statically identify the most suitable processing elements class for each system functionality. Experimental results are also included, to show the applicability and effectiveness of the proposed methodology.

Keywords

Heterogeneous Multiprocessor Embedded Systems, Metrics for Hw/Sw Partitioning, System-Level Design.

1. INTRODUCTION

Modern electronic applications consist of a fairly heterogeneous set of components: a mix of analog and digital systems and several software application layers. The hardware can include different microprocessors (general purpose or *DSP*), memories and dedicated *ICs* (ASICs and/or FPGAs) and a set of local connections between the system components, and some interfaces between the system and the environment (sensors, actuators, etc.).
In general, a tradeoff among aspects like performance, predictability, cost, flexibility, architecture distribution, weight, fault tolerance, power consumption, etc. has to be achieved ([1], [3]). Heterogeneous multiprocessor embedded systems have been exploited for the implementation of different applications both for research and for commercial use (e.g. heterogeneous multiprocessor system on chip for the management of real-time video stream [13], single-chip multiprocessor system for video signal processing [14]). Therefore, on one hand, multiprocessor embedded systems seem to be capable to meet the demand of processing power and flexibility of complex applications. On the other hand, such systems are very complex to design and optimize, so that the design methodology plays a major role in determining the success of the products. However, no assessed general design methodologies are available today. To overcome such problems, a possible solution consists in extending the classical co-design methodologies.
In the past few years, a number of research works focused on co-design methodologies for heterogeneous multiprocessor embedded systems ([1], [11], [15], [16], [17], [18]). Such frameworks allow user interaction to exploit the designer experience at system level, where it is still possible to manage the application complexity.

However, what is still needed is a systematic approach general enough to be useful in several application domains, while considering the peculiarity of the system to be designed. In general, an environment to fully support system-level design of heterogeneous multiprocessor systems should encompass the following features:

- homogeneous system-level specification representing the system functionality and the timing constraints;
- analysis of the specification to statically detect the best processing element for each system functionality, and to statically estimate their timing characterization for both hw and sw implementations;
- system-level functional co-simulation to check the functional correctness of the specification and to provide a set of dynamical information on the system behavior (profiling, communication, load, etc.);
- system-level design space exploration composed of two integrated and iterative steps: *partitioning*, i.e. the identification of feasible architectural solutions, (number and type of heterogeneous processing elements); system-level timing *co-simulation* considering heterogeneous multiprocessor architectures and a high-level model for the communication media, to verify the meeting of the timing constraints.

This paper presents a part of a more comprehensive research work [19] aiming at providing general models, methodologies and tools to support each step of the co-design flow of embedded systems based on heterogeneous multiprocessor architectures.
In particular, this paper focuses on a single step of the design flow: the definition and validation of innovative metrics for the analysis of the system specification to statically associate each system functionality wit the most suitable class of processing element.
The paper is organized as follows. Section 2 presents the proposed design flow and the related design environment. Section 3 is the focus of the paper where the proposed metrics are detailed and shown their possible use for codesign. The validation of the methodology is discussed in Section 4, where experimental data are reported for the adopted benchmark set. Section 5 drawn some conclusions and outlines the future work.

2. THE PROPOSED APPROACH

The design flow we are proposing is shown in Figure 1. Concerning the target architecture, the natural way to combine performance, flexibility and effectiveness is to take the best from different *worlds*. By their nature, software implementations on programmable processing elements are preferred to achieve maximum flexibility. Tasks running inefficiently on general-purpose processing elements have to be mapped on specific processors or on dedicated co-processors. However, different application domains (e.g. video, audio, telecom, automotive) have different requirements therefore, an optimal general architecture

does not exist. The optimal solution is found by defining a sort of *template architecture* that can be optimized for the specific characteristics of the application domain.

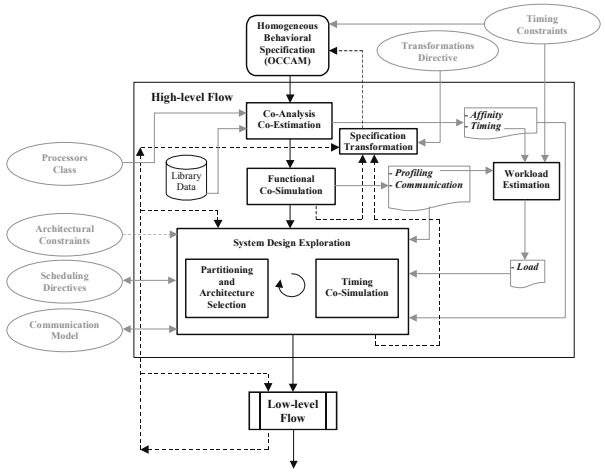

Figure 1. The proposed high-level flow

In the following, the main steps of the mapping of abstract specifications onto such an architecture are presented.

2.1. Co-specification

The entry point is a *SystemC* description [2] of the desired system behavior. This allows the designer not to force the design toward hardware or software during the early stages of the design. A proper procedure-level internal model has been defined to deal with specifications not only expressed in SystemC, thus enforcing the generality of the proposed methodology. The procedure-level internal model is able to capture information related to the computational elements present in an imperative, possibly *object-oriented*, specification and the relationship between them. Such a model, called *Procedural Interaction Graph* (*PIG*) is based on the *Procedural Call Graph* (*PCG*, [5]). Moreover, to enhance generality, the exchange format of the model is based on the VCG format [4], a third party format that can be managed and visualized with open source tools.

2.2. Co-analysis and Co-estimation

The first step of the flow aims at obtaining as much information as possible on the system, by analyzing the specification in a static and fast manner. The goal is twofold: for each system functionality (i.e. every method) the best processing element is statically defined, and a timing characterization for both HW and SW implementations is computed. This step provides a set of data expressing the *affinity* of the functionality towards each possible processing element (GPP, DSP, ASIC/FPGA), and a set of estimations on the *time* needed, to a particular class of processing elements, for the execution of each single operation that composes the specification.

2.3. Functional co-simulation

After the static analysis, the system functionalities are simulated in order to verify their correctness with respect to typical input data sets. This type of simulation is not precise but very fast and allows the designer to easily detect functional errors and anomalous situations (e.g. deadlocks or the presence of dead code). Moreover, it is possible to extract important data characterizing the dynamic behavior of the system: *profiling* and *communication cost* (these information are always related to the behavior of the system in correspondence of typical input data sets).

2.4. Workload estimation

Combining some of the data provided by the previous steps (timing and profiling data) with the timing constraints allows the estimation of the *load* associated with the execution of each procedure on a general purpose processor (*GPP*). The analysis of such data is useful to evaluate the necessary amount of processors, the level of load-balancing, and the identification of those procedures that probably need an executor more performing than a GPP.

2.5. System design exploration

The system design exploration task is constituted by two iterative steps: *partitioning and architecture selection*, and *timing co-simulation*. All the data produced in the previous stages are used to drive the process, together with additional information provided by the designer. Such information expresses the *architectural constraints* (e.g. max number of GPP, max number of DSP, area boundaries for ASIC, etc.), the *scheduling directives* (e.g. procedures priority), and the parameters of the *communication model* (e.g. the number of concurrent communications allowed).

The partitioning methodology explores the design space (it is based on a genetic algorithm) looking for feasible solutions, supporting also the selection of a heterogeneous multiprocessor architecture (which components must be included and how these should be connected) taking into account several issues (degree of affinity, communication cost, workload, physical costs, etc.). It decides the binding between parts of the behavior and the selected components. Architecture selection and partitioning are influenced by performance requirements, implementation cost, and application-specific issues.

The timing co-simulation methodology considers the proposed heterogeneous multiprocessor architecture and a high-level model for the communication media in order to model the system behavior through the behavior of the hardware and software parts. It evaluates the performance of the system by verifying its timing correctness.

2.6. Specification transformation

This step involves restructuring of the specification that can be performed in order to satisfy the design constraints. In particular it is possible to go back in the flow through this step from several points (dotted lines in Figure 1), each of them more costly than the previous one, that is, after the functional co-simulation, after the system design exploration, and after the low-level flow.

3. METRICS FOR CO-ANALYSIS

Co-analysis 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 best processing element for the execution of each system functionality. The proposed analysis provides a set of data expressing the *affinity* of a functionality towards a type of processing element (*GPP*, *DSP*, *ASIC/FPGA*). For this, several subtasks should be performed: an architectural analysis of the existing 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; the definition of a set of metrics able to provide meaningful indications useful to make design choices.

3.1. Characterization of executors

A first characterization discriminates between *processor-like* and *ASIC-like* executors. The former is equipped with a more complex control unit so it is quite independent in the retrieval and management of data. The latter is more suitable in a *co-processing architecture*, i.e. where it acts as a co-processor for a main device, performing only specific tasks.

Such preliminary considerations represent a first guideline towards an effective association between functionality and category of executors. This section introduces the analysis to detect main exploitable architectural features of the executors considered above.

3.1.1. GPP architectural features

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

3.1.2. DSP architectural features

Digital Signal Processor (*DSP*) have been tailored 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 [8]. For example, typical DSP operations are represented by regular (i.e. repetitive) computations on fixed length arrays (e.g. filtering). 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*).

3.1.3. ASIC-like devices architectural features

. *Application Specific Integrated Circuits* (*ASIC*) are developed for specific applications. They are generally high performing but their design and development costs are very high so they are affordable only for high production volumes. *Field Programmable Devices* (*FPD*) are arrays of logic blocks with programmable interconnections that define the performed functionality. They represent a tradeoff between processors and ASICs with respect to performance, flexibility, and cost (e.g. *Field Programmable Gate Array* [6]).

We worked on the identification of a set of features that allow an early selection of the functionalities able to exploit *ASIC-like* devices. The most relevant features are the following.

A mismatch between application data-path requirements and those presented by the processor data-path could lead to inefficient use of processor resources (*non-standard data-path*).

Therefore, ASIC-like devices are more suitable to perform *bit manipulation operations* (shifting, Boolean operators, etc.). Finally, repeated operations of similar types on large regular data sets are an ideal candidate for ASIC-like implementations. Regularity in operations imposes less demand on the control unit complexity better exploiting the available resources.

3.2. The proposed approach: rationale

Considering the architectural features previously identified 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 matching. Finally, these metrics are properly combined in order to build a global metric (the *affinity*) able to suggest the best processing elements for the execution of each system functionality.

Definition: the *Affinity* (Am)

The affinity $A_m=[AGPP_m\ ADSP_m\ AHW_m]$ of a method 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 towards 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. For example, in [7] 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 [9], the authors create a methodology that fully characterizes any algorithm with respect to the elements of its structure that affect its implementation. Such methodologies are based on the definition of seventeen properties that are gathered into groups (e.g. *size*, *concurrency*, *temporality*, *spatial locality*, *regularity*, *cyclic properties*, etc.). The identified groups are meaningful, however only a few of them are supported by an effective quantification approach, and when such a support is provided, the metrics defined are strictly bounded to high-level synthesis issues (as an example, the methodology is used to estimate the implementation area of a custom ASIC).

A co-design oriented work is instead the one presented in [10] where the concept of *hardware/software repelling* is used to drive a hw/sw partitioning algorithm. The approach is based on the analysis of the system functionalities, detecting a set of features that suggest a repelling of certain functionalities towards a certain type of implementation. Unfortunately, the work considers only one kind of software executor, and the set of features considered isn't clearly defined.

Finally, [11] represents the work more similar to the one presented in this paper. In fact, it 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 doesn't consider dedicated hardware devices (it considers only GPPs, *microcontrollers*, and DSPs), and works with a too coarse granularity level (whole classes and not single methods), and poorly defines the metrics to be used within the methodology.

3.2.1. Model and methodology

The classification of the specification is based primarily on the data involved in the execution of a functionality and on its structural properties. Moreover, several properties oriented to particular classes of executors are considered. In the following, a set of metrics is defined providing a model for the classification of the specification. The functional and structural features considered in the affinity are described in detail.

Data oriented metrics

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

Definition: 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 ratio between the number of declarations of *t* type with respect to the total number of declarations made in *m*.

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: 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 statement and the total number of lines.

The value of such a metric is increased by variations in the execution flow due to decision points (i.e. loops and branches), therefore a linear sequence of instructions has zero control flow complexity.

Definition: 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. Moreover, high LR_m values indicate the possibility of exploiting a spatially limited computational unit by means of a compact implementation and a strong component reuse.

DSP oriented metrics

The goal is to identify functionalities suitable to be executed by a *DSP* by considering those issues that exploit the most relevant architectural features of such executor class: *Circular Buffering*, *MAC* operations, and *Super Harvard architecture*.

For the circular buffering, the goal is to identify subsets of the specification that access a linear data structure (one-dimensional array, row or column of bi-dimensional array). The use of a circular buffer is identified, more or less explicitly, by portions of code that try to shift an array of one or more positions.

Definition: 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 *v* is a vector (or a row/column of a matrix), and *K* is a constant value.

Definition: 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 $v[K]=f(v[i])$ or $q=f(v[i])$ and the total number of lines, where *v* is a vector (or a row/column of a matrix), *K* is a constant value, and $f(v[i])$ is a generic expression that involves $v[i]$.

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: 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 $s_1=s_1+s_x \cdot s_y$ and the total number of lines.

Definition: 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 $s_1=s_1+s_x \cdot s_y$ and the total number of lines.

For the concurrent memory access, the goal is to identify subsets of the specification able to exploit concurrent memory accesses to instructions and data, as provided by the *Super Harvard* architectures [8].

Definition: 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 following structure $v[i]\ op\ w[i]$ or $q\ op\ w[i]$ and the total number of lines, where *v* and *w* are vectors, and *op* is an operator different from =.

Definition: Weak Harvard Degree (WHD_m)

For each method *m*, WHD_m is the ratio between the number of source lines that contain, outside a loop, expressions such as $v[i]\ op\ w[i]$ or $q\ op\ w[i]$ and the total number of lines, where *v* and *w* are vectors, and *op* is an operator different from =.

GPP oriented metrics

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

Definition: Conditional Ratio (CR_m)

The Conditional Ratio of a method m is CR=CFC–LR where CFCm is the Control Flow Complexity and LRm is the Loop Ratio.

Definition: I/O Ratio (IORm)

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: Structure Ratio (STR_m)

For each method *m*, the *Structure Ratio* is the ratio between the number of structures declared and the total number of declarations.

ASIC-like oriented metrics

The goal is to identify regular functionalities that significantly rely on operations that involve bit manipulation. Therefore, in addition to some of the previously defined concepts (i.e. *LR*, and DR_m for the type *bit*) the following metric is defined.

Definition: 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 possible executor. Such a global metric, called *affinity* is operatively defined in the following.

The affinity

The affinity of a functionality can be expressed by a normalization function applied to a linear combination of the metrics, with weights that depend on the considered executor class. Intuitively, the affinity towards a *GPP* executor depends primarily on: the *I/O Ratio*, the *Conditional Ratio*, the *Structure Ratio*, and the number of declared variables of *GPP* compatible type. The affinity towards a *DSP* executor primarily depends on: the *degrees* of *circularity*, *Harvard*, and *MAC*, the *Loop Ratio*, and the number of declared variables of *DSP* compatible built-in type. The affinity towards an *ASIC-like* executor depends on: the *Loop Ratio*, the *Bit Manipulation Ratio*, and the number of variables of bit type. Therefore, it is possible to evaluate the affinity for each method *m* as follows:

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

where:

$$A_m = \left[A_{GPP_m}\ A_{DSP_m}\ A_{HW_m}\right]$$

$$C_m = \left[SCD_m \; WCD_m \; SHD_m \; WHD_m \; SMD_m \; WMD_m \; IOR_m \; CR_m \; LR_m \; BMR_m \; DR_{bit}^m \sum_{z=char,string} DR_z^m \sum_{z=int|real} DR_z^m \; STR_m \right]$$

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

The weights of the matrix *W* are set to 1 when the associated metric is meaningful for a given executor class, 0 otherwise. In this way, the affinity represents the sum of all the contributions determined by each relevant metric. Since such a sum could be greater than one, a function should be applied to obtain values in the [0, 1] interval allowing a direct comparison between affinity values related to different executors.

The adopted normalization function is the *arctangent* one because it is limited to the interval $[-\pi/2, \pi/2]$ when *x* varies from $-\infty$ to ∞ so. So, to normalize the affinity in the interval [0, 1] it should be scaled of a $\pi/2$ factor. Moreover, to take into account that a value of 1 for a single relevant metric means a strong matching between the functionality and the executor a proper coefficient is multiplied to the *x* in order to obtain an affinity equal to *0.9* in correspondence with *x=1*. Finally, to better discriminate between low and high affinity values, a quadratic form is introduced, leading to the following normalization function:

$$f(x) = \frac{atan\left(2\pi x^2\right)}{\pi/2}$$

The function *f(x)*, when applied to $W \cdot C_m^T$, provides affinity values that are directly comparable and therefore it can be used to select the best executors class for each functionality.

4. METHODOLOGY VALIDATION

In order to support the presented co-analysis methodology, and to validate the methodology itself, a tool has been developed and integrated in the tool suite supporting the design flow of Figure 1.

Due the wide diffusion of *C* language (especially in the *DSP* field), a meaningful validation has been setup based on a *C* test suite. A tool has been developed and integrated with a C/C++ code analyzer (*GENOA*, [12]). The tool computes the affinity values for each system functionality that are then provided to the system design exploration tools.

The adopted benchmark suite is composed of 311 procedures; each one of them representing a specific functionality. A subset of these procedures (i.e. 100) has been selected from applications oriented to digital signal processing and, therefore, they represent a valid sample of the main functionalities involved in these applications (e.g. *Fast Fourier Transform*, filtering, convolutions, etc.). The other procedures are representative of a general set that contains functionalities related to the field of coding, string manipulation, common operations (e.g. *sorting*) and parts of videogames. During the validation process, the values of the metrics previously defined have been collected, and the affinity value of each functionality has been evaluated in the normalized form.

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). A_{DSP} for the DSP applications is fairly larger than the other affinity values and the A_{DSP} values evaluated 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 (the three average values in Table 1 are nearly the same) those procedures that exploit some features associated with the *ASIC* executor class.

Table 1. Affinity average values

	Average (on the whole suite)	**Average** (only the *DSP aplications*)	**Average** (all but the *DSP applications*)
A_{GPP}	0,46	0,38	0,49
A_{DSP}	0,25	0,57	0,10
A_{HW}	0,27	0,28	0,27

4.1. Metrics and Partitioning

To give the flavor on how the methodology can actually work on real designs and to show its effectiveness, an example of co-analysis and system-level partitioning is here reported.

The considered application consists of 52 methods and its *Procedure Interaction Graph* is represented in Figure 2. The target architecture is composed of an unconstrained number of *GPP*, *DSP* and *FPGA*. Starting from an annotated VCG (with affinity, load and communication cost data), the partitioning tool builds its procedure-level internal model. The next action of the tool has been the cost function minimization based on a genetic algorithms strategy.

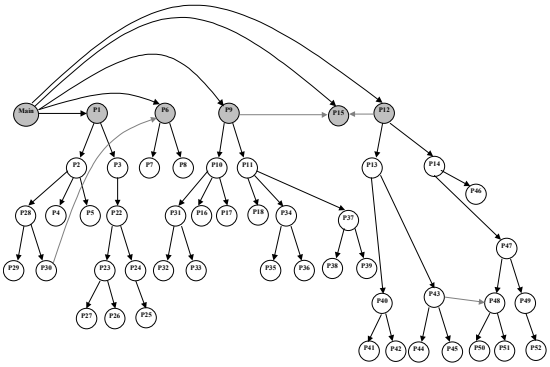

Figure 2. Procedure Interaction Graph

Table 2 shows the affinity values of each procedure. The load depends on the imposed timing constraints and the relative physical costs are 1 for a GPP, 1.2 for a DSP and 2 for a FPGA.

The goal of this validation is to check the behavior of the partitioning tool for different timing constraints and different cost function weights in order to highlight, in particular, the role of the affinity values. Different timing constraints have been imposed on the execution time of the whole application in the following way: with respect to a T_{REF} (evaluated by simulation for a single *GPP* system) in different experiments the constraints have been 90% T_{REF}, and 50% T_{REF}. The latter constraint aims at forcing the partitioning tool to exploit the concurrency by increasing the number of executors.

The weight of the affinity is variable and it assumes different values during several experiments in order to enforce at each step weights of the affinity index in the cost function. For each value of the affinity weight, Table 3 and Table 4 report the iteration (each iteration works with a finer-granularity) that has found the minimum value for the considered cost function and the related timing simulation result.

Table 3 shows the results for the constraint 90% T_{REF}. Such a choice enforces the presence of an architecture with more than one executor in order to reduce the execution time and, in fact, the timing constraint is always largely met. With lower affinity weights (e.g. 0 and 2), the partitioning does not consider *DSPs*

executors, while with the weights 3 and 4, the affinity becomes an important factor and a DSP is introduced. Theses solutions provide acceptable simulated times and physical cost. When the affinity weight is too high, the tool considers the affinity more than other factors and then, even if the simulated time is the best one, the physical cost increases and the load index indicates an unbalancing that indicates a possible under-load of the resources. In this example, the timing constraint is not considered as a big issue.

Table 2. Affinity values

Procedure	Affinity			Procedure	Affinity		
	GPP	DSP	HW		GPP	DSP	HW
P1	0.612	0.605	0.397	**P27**	0.799	0.447	0.492
P2	0.425	0.649	0.388	**P28**	0.580	0840	0.403
P3	0.535	0.640	0.392	**P29**	0.460	0.840	0.403
P4	0.259	0.772	0.398	**P30**	0.559	0.772	0.398
P5	0.297	0.748	0.396	**P31**	0.868	0.046	0.688
P6	0.543	0.551	0.468	**P32**	0.597	0.053	0.911
P7	0.658	0.062	0.894	**P33**	0.997	0.053	0.211
P8	0.997	0.053	0.925	**P34**	0.643	0.571	0.416
P9	0.586	0.619	0.394	**P35**	0.580	0.620	0.405
P10	0.548	0.640	0.391	**P36**	0.544	0.653	0.396
P11	0.351	0.637	0.393	**P37**	0.608	0.595	0.406
P12	0.623	0.577	0.413	**P38**	0.520	0.659	0.388
P13	0.553	0.626	0.397	**P39**	0.553	0.634	0.390
P14	0.424	0.648	0.388	**P40**	0.604	0.911	0.409
P15	0.524	0.648	0.388	**P41**	0.471	0.810	0.401
P16	0.997	0.053	0.915	**P42**	0.352	0.748	0.396
P17	0.612	0.596	0.412	**P43**	0.459	0.772	0.388
P18	0.587	0.607	0.405	**P44**	0.259	0.772	0.398
P19	0.616	0.609	0.397	**P45**	0.574	0.651	0.533
P20	0.725	0.519	0.428	**P46**	0.384	0.648	0.388
P21	0.565	0.635	0.398	**P47**	0.524	0.628	0.368
P22	0.997	0.053	0.550	**P48**	0.234	0.608	0.378
P23	0.658	0.053	0.694	**P49**	0.987	0.063	0.921
P24	0.799	0.447	0.492	**P50**	0.992	0.092	0.890
P25	0.494	0.648	0.388	**P51**	0.968	0.085	0.787
P26	0.384	0.648	0.388	**P52**	0.989	0.099	0.833

Table 3. Timing constraint: 90% T_{REF}

w_A	Iteration	I_C	I_{LSW}	I_A	Architecture			Simulated Time
					GPP	DSP	FPGA	
0	6	0.003	0.004	0.501	1	0	1	66% T_{REF}
2	6	0.003	0.121	0.395	2	0	0	69% T_{REF}
3	9	0.020	0.121	0.390	1	1	0	60% T_{REF}
4	5	0.004	0.126	0.384	1	1	0	58% T_{REF}
7	9	0.032	0.277	0.254	2	1	0	55% T_{REF}

Table 4 shows the results for the constraint 50% T_{REF}. The heavy constraint forces an architecture with several executors. The timing constraint is always met except in the case when the affinity index is not taken into account (i.e. w_A=0). With affinity weights from 2 to 4, the partitioning provides good solutions.

Table 4. Timing constraint: 50% T_{REF}

w_A	Iteration	I_C	I_{LSW}	I_A	Architecture			Simulated Time
					GPP	DSP	FPGA	
0	10	0.320	0.040	0.394	3	0	0	51% T_{REF}
2	9	0.006	0.024	0.386	2	1	0	42% T_{REF}
3	9	0.007	0.024	0.385	2	1	0	42% T_{REF}
4	9	0.011	0.126	0.392	2	1	0	43% T_{REF}
7	10	0.220	0.060	0.340	1	2	0	45% T_{REF}

Finally, as in the previous case, an affinity weight too high drives to solutions that do not consider properly the other aspects: in this case, communication issues cause a worst simulated time. The results show how the partitioning tool is able to perform an effective design space exploration, while the affinity represents a useful indicator that allows the selection of an architecture tailored to the features of the specification.

5. REFERENCES

[1] J. Axelsson. Analysis and Synthesis of Heterogeneous Real-Time Systems. Ph.D. thesis No. 502, 1997. Department of Computer and Information Science, Linköping University, Sweden.

[2] SystemC Home Page, http://www.systemc.org, 1999.

[3] T.A.C.M. Claasen. High speed: not the only way to exploit the intrinsic computational power of silicon. Solid-State Circuits Conference, 1999. Digest of Technical Papers. ISSCC, 1999. IEEE International, 1999 Page(s): 22 –25.

[4] G. Sander, *R. Tamassia, I. G. Tollis.* Graph Layout through the VCG Tool. In *Proceedings of DIMACS Int. Workshop GD'94.*

[5] J. Choi, M. Burke, P. Carini. Efficient flow-sensitive inter-procedural computation of pointer-induced aliases and side effects. In proceedings of the 20th Annual ACM Symposium on Principles of Programming languages, pp. 233-245, January 1993.

[6] S. Broen, R. Francis, J. Rose, Z. Vranesic. Field Programmable Gate Array. Kluwer Academics, Boston, 1992.

[7] A. De Hon. Reconfigurable Architectures for General-Purpose computing. Technical Report 1586, MIT-AI Laboratoty, 1996.

[8] T. Cooper. Taming the SHARC. Tech. report, Ixthos Inc., 2000.

[9] L. Guerra, M. Potkonjak, M. Rabaey. System-level design guidance using algorithm properties. J. VLSI Signal Processing, VII, 1994, [Workshop on] , 1994 Page(s): 73 –82.

[10] A. Kavalade, A. Lee. A Global Criticality/Local Phase Driven Algorithm for the Constrained Hardware/Software Partitioning Problem. Codes/CASHE '94, France, Sept. 22-24, 1994, pp 42-48.

[11] L. Carro, M. Kreutz, F.R. Wagner, M. Oyamada. System synthesis for multiprocessor embedded applications. Design, Automation and Test in Europe Conference 2000. Page(s): 697 –702.

[12] P. Devanbu. GENOA: A Customizable, Language- and Front-end Independent Code Analyzer. In Proceedings of ICSE '92, 1992.

[13] M.T.J. Strik, A.H. Timmer, J.L. Van Meerbergen, G. Van Rootselaar. Heterogeneous multiprocessor for the management of real-time video and graphics streams. Solid-State Circuits, IEEE Journal of, Volume: 35, Issue: 11 , Nov. 2000.

[14] J. Hilgenstock, K. Herrmann, S. Moch, P. Pirsch. A single-chip video signal processing system with embedded DRAM. Signal Processing Systems, 2000. SiPS 2000, IEEE Workshop on, 2000.

[15] B.P. Dave, G. Lakshminarayana, N.K. Jha. COSYN: Hardware-software co-synthesis of heterogeneous distributed embedded systems. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, Volume: 7 Issue: 1 , March 1999 Page(s).

[16] B.P. Dave, N.K. Jha. COHRA: hardware-software cosynthesis of hierarchical heterogeneous distributed embedded systems. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on , Volume: 17 Issue: 10 , Oct. 1998 pp: 900 –919.

[17] P.A. Hsiung. CMAPS: A Cosynthesis Methodology for Application-Oriented General-Purpose Parallel Systems. ACM Transactions on Design Automation of Electronic Systems. Vol. 5, n. 1, pp.58-81, Jan. 2000.

[18] J. Axelsson. Towards System-Level Analysis and Synthesis of Distributed Real-Time Systems. In Proc. 5th International Conference on Information Systems Analysis and Synthesis, Vol. 5, pp. 40-46, Orlando, July 31-August 4, 1999.

[19] L. Pomante. "System-Level Co-Design of Heterogeneous Multiprocessor Embedded Systems". Ph.D thesis, 2002, DEI, Politecnico di Milano, Italy.