A Petri Net Based Approach For Hardware/Software Partitioning

Fred Cruz Filho, Paulo Maciel, Edna Barros

Abstract— **Co-design methodologies have arisen for supporting the development of digital systems composed of mixed hardware/software. Partitioning is performed by co-design tools for splitting the funcionality of the original system among components. In this paper, a partitioning approach based on a Petri net intermediate format is proposed. The new model will substitute the one currently adopted in PISH co-design system. The main advantages of such model are having more precise metrics and becoming independent of the specification language.**

Keywords— **Hardware/software co-design, Petri nets, metrics, data-dependency analysis**

I. Introduction

Most modern electronic systems are composed by both dedicated elements (hardware) and programmable elements (software). The choice between a hardware or a software implementation may be obvious because of some aspects related to the nature of the element (e.g. sensors/actuators, AD/DA converters, etc). But when these intrinsic aspects are not important, that choice is mainly guided by the trade-off among achieving the time constraints (hardware is faster) and reducing the implementation cost (software is cheaper).

An approach known as hardware/software co-design proposes the cooperative design of such mixed systems as a whole, departing from a unique behavioral specification (Figure 1). It has shown to be advantageous when compared to the traditional approach. When designing such mixed hardware/software systems, the analysis of design alternatives and the decision of where to implement each part of system, either in hardware or in software, are very important tasks. The process of deciding where to implement each part is known as *partitioning*.

The estimation of quality metrics permits design space exploration and may guide the decision of the implementation media of a system's parts [1]. Such metrics are calculated at system level, or rather, without real implementation. Estimations also speed up a system's design and permit the analysis of design constraints, providing a quick feedback for design decisions.

Nowadays, many co-design tools use high-level languages as its input format [2]. Some co-design tools also use an internal format for modeling the system's functionality [3], [4]. Different formalisms are considered for internal representation of these systems, as such as finite state machines, control/data flow graphs and, recently, Petri nets.

In this paper, a partitioning approach based on Petri net internal format is presented. Additionally, a short description of the proposed Petri net model for the internal representing of the system's functionality is exhibited. Furthermore, a set of methods for feature capturing, exploring design space and choosing implementation alternatives are shown. The combined application of such methods performs the hardware/software partitioning in the PISH co-design system.

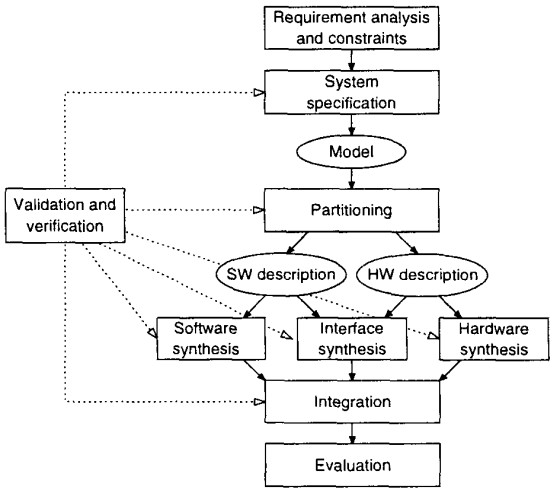

Fig. 1. Workflow of a co-design methodology

This paper is organized as follows: Section II summarizes some works related to the hardware/software partitioning problem. Section III brings an overview of the PISH co-design methodology. After that, the Petri net model is presented on Section IV. In the following, Section V shows the partitioning approach which is being proposed. Section VI depicts the application of such methodology by considering a real-world example. Finally, Section VII summarize some results and presents some ways for improving the proposed approach.

II. Related work

Co-design tools are still incipient and, in the most part, they are under-development versions usable only by its developers for academic purposes. Despite this disadvantage, some published works descript how these tools carry out the partitioning process.

The VULCAN II system [2] translates a specification written in *Hardware C* into an internal format. Execution time is estimated (when possible) and the original description is divided into three groups. Two of them are com-

Centro de Informática — Universidade Federal de Pernambuco
E-Mail: {fmcf,prmm,ensb}@cin.ufpe.br

0-7695-1333-6/01 $10.00 © 2001 IEEE

posed of those objects where execution time is undefined due to data dependencies. The objects with internal dependencies are moved to the software component and those with external dependencies are moved to the hardware component. The remaining objects (with defined duration) are initially moved to a hardware component (hardware-oriented approach). Finally, a *graph min-cut* algorithm chooses a group of objects which can be moved to the software component, minimizing the hardware area without increasing the communication cost between the hardware and the software partitions.

The COSYMA system [4] translates a specification written in C* into an internal format based on CDFG[1]. In the following, a simulation/profiling module extracts performance information of the system. All objects are initially allocated on the software partition (software-oriented approach). After that, an iterative optimization algorithm (*simulated annealing*) is applied. The objective here is moving objects to the hardware partition in order to achieve the time constraints of the system. Finally, the resulting components are translated into languages which can be compiled/synthesized. Such output is analyzed for extracting the real computing time and performing a validation of the system.

The LYCOS system [3] takes a specification (in C or VHDL) and translates into an internal format based on CDFG[1]. After that, estimators compute all necessary information about the objects, summarizes them in a metric and generates a linear integer programming problem. Finally, such problem is resolved and the solution that minimizes a cost function is adopted as the final partitioning. In LYCOS, only time constraints are analyzed for validating the generated system.

In the PISH system, the partition method emphasizes the correctness of the generated system. The partitioning approach considered in PISH uses program rewriting rules, which divides the original specification in communicant objects. Such rules follow a formal strategy which is proved to be correct-per-construction [5], [6]. An overview of the partitioning in the PISH system will be presented in the next section.

The main disadvantage of the PISH when compared to the other co-design tools is the lack of an internal format, appropriated to metric computing and estimation. On the other hand, the partitioning approach of the PISH system provides a formalism that guarantees the correctness of the generated system, without the necessity of verifying the generated implementation.

Petri nets were considered for the intermediate format of PISH system mainly because they are appropriated for modeling parallelism, which is not well represented by other models, as such as finite state machines (e.g. due to state explosion problems). Furthermore, a set of methods for computing metrics necessary to the partitioning process was developed over a Petri net model [7], wich represents the control-flow of a system. The PISH system considers occam as its specification language and, in the same work, it was provided a method for translating between occam and Petri nets.

During this work, a Petri net model for representing the system's data-flow was developed, as well as two method for computing data dependencies between objects. Besides, a set of methods for capturing attributes of objects and choosing implementation alternatives were developed, too. After the next sections, the reader should be familiar to the current state of the PISH co-design system.

III. The PISH co-design system: an overview

The PISH system is being developed in our research group and it implements all those steps depicted in Figure 1. A subset of the occam language [8] is used as specification language. occam is a language that allows the specification of concurrent systems, whose components communicate each other through synchronous channels. Based on the algebraic properties of the occam language, it is possible to apply rewriting rules for performing transformations on the specification. A set of rewriting rules for occam were proposed by [6] and it was proved that those rules preserve the semantics of the original specification.

The partitioning phase of PISH system is composed of four steps: *splitting*, *classification*, *clustering* and *joining*. The workflow of such tasks is depicted in Figure 2. The partitioning approach in the PISH co-design system is based on an orthogonality principle, where the functional aspects of the partitioning can be dissociated from the non-functional ones [6].

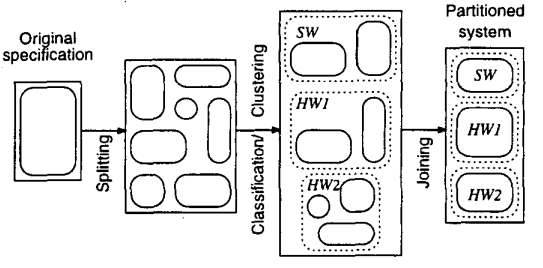

Fig. 2. Partitioning phase in PISH system

Splitting: In the splitting phase, algebraic rules are exhaustively applied to the original specification until it reaches a *normal form*, which is composed of a set of *simple processes* that communicate each other [6].
Classification: In the classification phase, a set of implementation alternatives is generated for each process. One of them is chosen according to the designer experience or by an automatic method [9], [10].
Clustering: In the clustering phase, processes are combined in order to achieve a composition that minimizes a cost function, which takes into account area, delay and communication [9], [10].
Joining: In the joining phase, another set of algebraic rules are exhaustively applied to the final specification in order to reflect the organization proposed by the clustering phase.

[1] Control/Data Flow Graph

Furthermore, it tries to remove the communication introduced by the splitting phase where it is unnecessary [6].

The splitting and joining tasks use algebraic rules for transforming the original specification. Hence, the verification of the partitioning correctness is straightforward from the transforming model of such phases. The formal verification of the partitioning phase can be considered innovative when compared to the other related works, which does not consider such aspect. Another innovative aspect of PISH approach to the partitioning is the orthogonality between functional and non-functional aspects of the system.

The classification and clustering phases are responsible for rearranging those simple processes. Such rearrangement is guided by estimates computed over the specification and also data dependency information. The estimates were extracted directly from the occam specification, in earlier versions of the PISH system.

However, many problems may arise from this approach. One of them is the dependency of the specification language. occam is not an industrial standard, despite its desirable and powerful algebraic properties. Hence, estimates computed over such syntactic model are not sufficiently accurate for guiding the partitioning.

Methods for computing area, delay, communication cost, mutual exclusion degree, similarity, etc, were proposed by [7]. Such metrics are computed over a Petri net model and an occam/Petri net translation method was also proposed in the same work. In order to complete the set of analyzed properties, another model was developed with an associated method for computing data dependencies [11]. Both models and metrics were considered for proposing an intermediate formalism for the classification and clustering phases in the PISH co-design system [10].

IV. A Petri net model for concurrent systems

Petri nets are a family of mathematical formalisms that model concurrent systems by implicit token-flow in a net, providing a sort of methods for qualitative and quantitative analysis [12]. Place/transition nets are composed by two kind of nodes: places, passive components which can hold zero or more tokens, and transitions, active components that consume and produce tokens. Such nodes are connected by directed arcs, which determine the token flow as well as enumerates how many tokens are consumed/produced by the following/preceding transition.

A. Data-flow model

Dependency constraints guarantee the fact that the program variables are defined and used in the correct order, for producing correct results. Transformations applied on programs must consider the dependency analysis for producing a new code that performs the same behavior and has the same properties of the previous code.

The data-flow model represents the many sequences of operations that produce, modify or consume data. The translation is performed in order to achieve the greatest possible parallel degree between instructions in the same process, once the real instructions ordering is represented in the control-flow model (Section IV-B). Such model allows the representation of the data flow that is independent from the original specification language. Due to space restrictions, only a brief description of the model will be shown. A complete explanation of the model can be found in [10].

Places: Each place represents a value, either a literal, a variable or the result of some expression.
Variables references are modeled by using duplication and renaming strategy. Every time a variable is read, the place representing its value is duplicated: one of these places is directed to the operation which is reading the variable and the other remains for future references. When writing to a variable, the place representing its value is attached to a transition which is responsible for destroying the previous value and a new place is added for representing the new instance of the variable.
Transitions: Transitions are used to represent two distinct groups of constructors. The first of them is composed by operators (arithmetic, logic, relational, binary and boolean). The second group is composed by structural constructors of the specification language.
Tokens: Tokens model the availability of some value in the place.
However, the presence of a token in a place does not model particular values, but only the fact that some value assigned to that place is available.

The data flow model represents each process as a net which has a starting and an ending place. The start place is the pre-condition of the first transition of the process and the end place is the post-condition of its last transition. Hence, determining if a process can be fully executed is the same as determining if the marking that holds a token in the end place is reachable from some initial marking. A formal proof of the data-dependency analysis can be found on [10].

B. Control-flow model

The control-flow model uses a timed Petri net [13] for representing the behavior of the specification. The role performed by each net component is described in the following:

Transitions: Each transition represents one operation of the specification. Information related to the kind of operation, type, variables, constants, etc.
Places: Places are used to represent instructions as well as the instruction ordering, giving the pre and post conditions for each operation.

The model is used for performing qualitative and quantitative analysis. Such analysis is based on both structural and behavioral properties of Petri nets. Metrics computed by using the control-flow net model include execution time, area estimation (control and data path), mutual exclusion degree, communication cost, etc [7].

V. HW/SW PARTITIONING OF A PETRI NET MODEL

Among the steps contained on the partitioning phase of the PISH system, the classification and clustering are performed considering the Petri net model. Such approach allows achieving the desired orthogonality between partitioning and verification, as proposed by [6]. Formal transformations are applied to the specification, generating a group of parallel processes. For their turn, processes are rearranged considering non-functional aspects. Finally, another set of transformations are applied for removing those constructions which were inserted on the specification by the splitting strategy, but not used at all.

A. Classification

The classification phase is divided into two steps: class capturing and class choice. The first is responsible for identifying all possible implementation alternatives for each process, considering its attributes.

Communication: Based on the data-dependency information, the attribute determines if the process is independent or should be implemented as a server, a client/server or a client of the remaining.

Sequential replication: Based on the control-flow net structure, the attribute determines if nested sequential loops can be implemented as a pipeline (for increasing performance) or should be kept in sequence (for decreasing area cost).

Parallel replication: The attribute is also based on the structure of the control-flow net. It is responsible for determining if sets of nested parallel processes can be implemented sequentially or partially sequential (for decreasing area cost), or it should be kept in parallel (for increasing performance).

Multiple assignment: The attribute is based on some of the information relative to the transition and determines whether multiple expressions should be evaluated in sequence (sharing funcional units) or should be evaluated in parallel.

In a second step, once the possible implementation alternatives have been assigned for each attribute, the classification algorithm will choose the one which will be actually implemented. Such choice can be either manual or automatic. If the automatic choice is selected, the system will suggest an implementation alternative which has a high and balanced parallelism degree with a low area cost. For reaching the objective, the algorithm computes an objective function which is dependent on the area and the delay. Area and delay estimations are computed over the control-flow model. Hence, the algorithm tries to equalize the parallelism degree of all processes, without having the objective function being greater than a pre determined treshold.

B. Clustering

Once an implementation alternative has been chosen for each process, the clustering phase takes place. The objective of clustering phase is to find a configuration of processes that have a low area and communication cost and a high performance. PISH system uses an hierarchical multistage clustering [14] approach, as proposed on [9].

On such method, objects are grouped according to a closeness metric. First, the distances between each pair of objects are computed. In the following, the pair of objects which are closer one another are removed and combined generating a new object. The new object is reinserted in the objects set and the process is repeated until only one object remains.

Parallel to the object combining, a tree structure (named clustering tree) is constructed. Initially, all objects are single nodes on level 0 (zero) of the tree. Each time an object pair is combined in a level l, their nodes are linked to a new node, which is marked as level $l + 1$. At the end of the process, the clustering tree is produced (as shown in Figure 3, from left to right hand side).

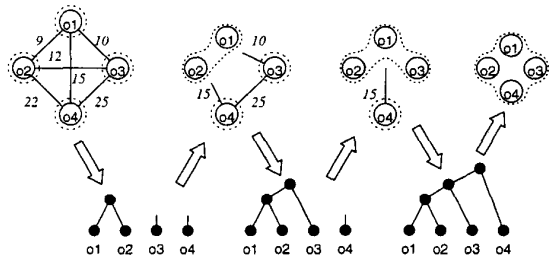

Fig. 3. Construction of the clustering tree

When the clustering tree construction is finished, the objective function is computed for each branch from the lower to the top level. The tree is cut in a level where the objective function is minimum, generating a new group of objects (Figure 4).

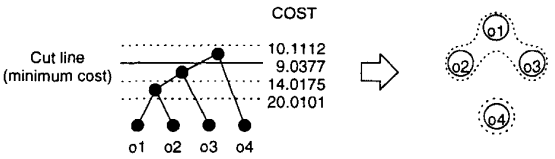

Fig. 4. Cutting of the clustering tree

The metrics estimated over the control-flow net and the data-dependency analysis results are considered for computing the distances for the clustering algorithm. Due to the complexity of the closeness metrics, the clustering phase is performed in two stages. Each one of the phases considers different closeness metrics and objective functions for cutting the tree.

In the first stage, similarity between processes is used as closeness metric. Two implementation alternatives are generated for each pair of objects in the tree: sequential and parallel. In the case of parallel composition, the control-flow model is modified to reflect the change by adding the net elements that correspond to a parallel composition of processes. In the other case, the control-flow model is modified also by the addition of a place between the ending place of a process and the starting place of the other. Be-

fore generating alternatives, one of the processes is chosen for software implementation, which is always combined sequentially.

Next, the area and execution time are estimated for computing the objective function. The algorihtm chooses the alternative which has the lower cost. The objective function takes into account the area cost, execution time and resource sharing. Parallel composition of objects is assigned to an infinite cost. Hence, the clustering tree is cut below the first level where the parallel composition is chosen.

In the second stage, the clustering tree is constructed considering a communication cost metric. The objective function for cutting the tree is the communication cost. Finally, the clustering process generates N objects, where 1 is implemented by software and $N-1$ by hardware.

VI. Case study

This section presents the results obtained for the partitioning of an ATM switch. ATM is a set of protocols for providing several kinds of services (telephony, data transfer, audio/video on-demand, etc) over a common media (a digital network). ATM uses connection oriented peer-to-peer services. Hence, partners should negotiate the communication parameters before establishing it.

These parameters also need to be accepted by every intermediate component (ATM switches). The objective of the ATM switch is to direct information from source to destination. However, some policy should be applied to the information stream to guarantee that negotiated parameters are being followed.

One of the modules of a switch is responsible for the load balancing of the connection. The load balancing algorithm verifies if the connection obeys the negotiated parameters. If affirmative, the data is forwarded with or without changes in parameters. Otherwise, the data is discarded for preserving the other connections.

The occam code for the ATM switch is 150 lines long, and supposes the existence of a host which maintains a routing table containing the parameters of the active connections. The *leaky bucket* algorithm is considered for load balancing. After the splitting phase, the description is divided into 9 processes:

- P1 receives a packet in a connection and reads the table row corresponding to that connection.
- P2 computes the load balancing and verifies if the packet is inside acceptable limits.
- P3 is responsible for taking the decision about the destination of the packet (*discard*, *forward* or *modify parameters and forward*).
- P4 forwards the packet to the next switch.
- P5 updates the host table.
- P6 to P9 are controller processes, created by the splitting phase and are not considered for partitioning.

In the following, the output of splitting phase is translated to the Petri net models. The control-flow net is composed of 113 places and 106 transitions and the data-flow net, 1940 places and 1150 transitions.

TABLE I
Chosen alternatives for each process

	Comm	RepSeq	RepPar	Mult
P_1	S	–	–	–
P_2	C/S	–	Parallel	–
P_3	C/S	–	–	–
P_4	C	–	–	–
P_5	C/S	–	–	–

After that, the data-dependency analysis is carried out. It has shown that: P1 does not depend on any process; P2 depends on P1; P3 depends on P1 and P2; and P4, P5 depend on P1, P2 and P3.

Next, the classification phase takes place and the object features are extracted both from the data-dependency analysis results and the control-flow model. The algorithm has chosen the alternatives depicted in Table I for each attribute.

Before getting into the clustering phase, all the necessary metrics are computed. In the first stage of clustering, the similarity metric considers the sharing of the same functional units by similar objects. After the first stage of clustering, process P3 is chosen for software implementation, considering its low internal communication cost. The clustering tree is constructed and it is cut in the second level (the first level where resource sharing is not possible), after grouping P4 and P5 in sequence (;). The new group of objects is formed by four processes P1, P2, P3 and P4;P5. In the second stage, the clustering tree is constructed and cut again on the second level (l1) as shown on Figure 5.

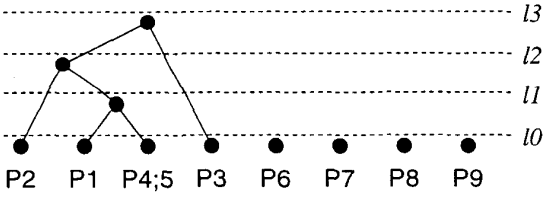

Fig. 5. Second stage clustering tree

The generated system is composed by a software component P3 and two hardware components: P2 and P1||(P4;P5), which is the parallel composition of P1 and the sequential composition of P4 and P5. A new occam specification is generated and sent to the joining phase, with annotations that will reflect the desired implementation. The joining algorithm will apply rules for serializing processes P4 and P5, removing the communication between them.

A hardware-only version of the ATM switch occupies 9962 logic blocks [15]. The implementation obtained in the partitioing uses 5330 logic blocks. Hence, the mixed implementation achieve the time constraints and have an area cost which is 53% of the hardware approach.

VII. Conclusions & future works

In this paper, a new approach for the partitioning phase of the PISH system was presented. Such approach con-

siders an intermediate model based on Petri nets. The Petri net model is used for extracting information about implementation alternatives for each attribute taken into account by the partitioning algorithm. Furthermore, estimates computed over such model are used for computing closeness metrics of the clustering algorithm.

In earlier versions, all analysis were performed directly over the syntax tree of the specification language, which led to inaccurate estimates besides dependence on the specification language. The development of such intermediate format has produced a version of PISH system whose classification and clustering phases are fully developed over the Petri net model. Therefore, the capturing of all non-functional aspects of the system being partitioned is performed independently of the specification language.

Finally, the method was applied to a real application, an ATM switch. The partitioning has generated 3 components, 2 of them implemented in hardware and the remaining in software.

Area estimation does not consider the register allocation, which leads to a superestimated number of registers and underestimated number of multiplexers. Hence, a natural extension to the work is to use the data-flow model for computing estimates about register allocation, which will increase the precision of the area estimates.

REFERENCES

[1] D. D. Gajski, F. Vahid, S. Narayan, and J. Gong, *Specification and Design of Embedded Hardware-Software Systems*, Prentice Hall, 1994.

[2] R. Gupta and G. DeMicheli, "Hardware/software co-systhesis for digital systems," *IEEE Design & Test of Computers*, pp. 29–41, 1993.

[3] J. Madsen, J. Grode, P. V. Knudsen, M. E. Peterson, and A. Haxthausen, "LYCOS: the lyngby co-synthesis system," *Design Automation for Embeded Systems*, vol. 2, no. 2, pp. 195–236, 1997.

[4] A. Österling, T. Benner, R. Ernst, D. Herrmann, T. Scholz, and W. Ye, "The COSYMA system," in *Hardware/Software Co-Design: Principles and Practice*, pp. 263–281. Kluwer Academic Publishers, Amsterdan, 1997.

[5] E. Barros and A. Sampaio, "Towards provable correct hardware/software partitioning using OCCAM," in *Proc. of 3rd Intl. Workshop on Hardware/Software Co-Design*, Los Alamitos, 1994, pp. 210–217, IEEE.

[6] L. Silva, *An Algebraic Approach to Hardware/Software Partitioning*, Ph.D. thesis, Universidade Federal de Pernambuco, Brazil, Jul. 2000.

[7] P. Maciel, *Petri Net Based Estimators for Hardware/Software Co-Design*, Ph.D. thesis, Universidade Federal de Pernambuco, Brazil, Dec. 1999.

[8] Dick Pountain and David May, *A Tutorial Introduction to occam Programming*, BSP Professional Books, Oxford, UK, 1987.

[9] E. Barros, *Hardware/Software Partitioning Using UNITY*, Ph.D. thesis, Tübingen Universität, Tübingen, Germany, Jul. 1993.

[10] F. Cruz, Filho, "Particionamento em hardware/software usando redes de petri," M.S. thesis, Universidade Federal de Pernambuco, Brazil, Dec.2000.

[11] F. Cruz, Filho, P. Maciel, and E. Barros, "Using Petri nets for data-dependency analysis," in *Proc. of Intl. Conference on Systems, Man and Cybernetics*, USA, Oct. 8-11 2000, pp. 2998–3003.

[12] T. Murata, "Petri Nets: Properties, analysis and applications," in *Proc. of IEEE*, 1989.

[13] W. M. Zubereck, "Timed Petri Nets definitions, properties and applications," in *Microelectronic and Reliability*, vol. 31, pp. 627–644. 1991.

[14] E. Lagnese and D Thomas, "Architectural partitioning for system level synthesis of integrated circuits," in *IEEE Transactions on Computer-Aided Design*. Jul. 1991.

[15] J. Lima, *Um Controlados Microprogramável para Comutadores ATM*, Ph.D. thesis, Universidade Federal da Paraíba, Campina Grande, PB, Brazil, Jun.1999.