Computing Communication Cost by Petri Nets for Hardware/Software Codesign

Paulo Maciel *
Edna Barros †
Universidade de Pernambuco
Wolfgang Rosenstiel
Universitaet Tuebingen

April 4, 1997

Abstract

This work presents a method to compute communication cost by applying Petri nets. This cost is being used to guide the hardware/software partitioning in a methodology for hardware/software codesign context, which is being developed.

Petri nets are a family of formalisms sharing basic principles. Although for each purpose or detail level one appropriated formalism have to be chosen from the family, the transformation from one formalism to another could be sound. The use of Petri makes the partitioning method independent on a specific description mechanism. Additionally, Petri net as an intermediate format allows to analyse behavioral properties of the specification and formally to compute performance indices which are used in the partitioning process.

1 Introduction

Hardware/Software codesign is the design of systems comprising two kinds of components: specific application components and general programmable ones. The system functionality is implemented by using a interconnected components set, where these components may be microprocessors, memory, *ASIC* chips, etc. Combined implementation in hardware/software has been more common today. The software is executed in microprocessors, cheap programmable components. In the case where an implementation on a microprocessor does not meet the time constraints, a hardware implementation must be done, i.e. inadequate components to be implemented in software must be implemented in hardware. Although such systems have been designed since hardware and software exist, there is a lack of CAD tools supporting the development of such heterogeneous systems. The progress obtained by the CAD tools at the level of algorithm synthesis, the advance in some key enabling technologies, the increasing diversity and complexity of applications employing embedded systems and the need for decreasing the costs of designing and testing such systems, make techniques for supporting hardware/software codesign an important research topic.

The main tasks when implementing such systems are the choice of the components set (definition of a target architecture) and the partitioning of the description.

An essential aid for hardware/software codesign is the availability of approaches to hardware/software partitioning.

Some partitioning approaches have been proposed by Soininen [11], De Micheli [12], Ernst [10] and Barros [9]. One of the main challenges of approaches for hardware/software partitioning is the increasing number of implementation alternatives to be analysed. Some approaches consider only a subset of alternatives by restricting the analysis to some particular language constructors like loops [10] or delay independent commands [12].

The approach proposed by Barros [6, 9] partitions a description written in *unity* into hardware components and software components by using a clustering algorithm, which considers the distinct implementation alternatives. This work has been adapted for occam [8]and when generating the set of implementation alternatives for a particular process, various features are considered such as: parallelism level, data

*Paulo Maciel have been suported by CAPES. During his stay at Universitaet Tuebingen

†This research has been carried out with suport from KIT Co-operation activity 128

1074-6005/97 $10.00

dependency and multiplicity. By considering a particular implementation alternative as the current one the *clustering* is done, observing criteria like similarity among processes. The analysis of distinct implementation alternatives during the partitioning allows the choice of the best implementation with respect to time constraints and area allocation. However, this approach does not treat *loops* in a general form and communication cost as well.

This paper aims to propose the use of a formal technique to compute communication cost in order to deal with this deficiency. The proposed method uses Petri net as an intermediate format for the partitioning process. Additionally, this work presents an approach to translate programs written in *occam* [56] to Petri net [57, 53, 55, 20].

The use of the Petri nets [16, 17, 18, 19, 27, 1] still allows qualitative analysis of the properties as well as: deadlock freedom, boundedness, safeness, liveness and so on [33, 35, 16, 27, 37, 39, 40, 42, 43, 44, 46, 47, 48, 49].

The next section gives an overview of the hardware/software partitioning approach. The section 3 presents an introduction into Petri nets [1, 57, 50, 51, 52]. The section 4 describes the occam/Petri net translation method. The seciton 5 presents the proposed method to compute communication cost. The section 6 presents an example and finally some conclusions are presented as well as perspectives for future works.

2 The Hardware/Software Partitioning Approach

In this section we describe the partitioning approach used in this work, which takes into account systems described in occam [8]. In order to preserve the semantics of the original description, the partitioning process is done as the application of a set of transformation rules in an occam program. An informal description of some rules applied during the partitioning phase is given in this section. These rules include the splitting of an occam description into a set of communicating simple processes. Once some process is chosen to be implemented in software, the clustering process takes place. During the clustering phase, rules for joining processes in clusters according to their similarity, the minimisation of communication costs and the minimisation of the area-delay cost function are applied.

The clustering process is guided by a time analysis [14] performed in the Petri net model of the processes, which was obtained by the translation method.

The target architecture is specified by the user by using previously defined components stored in a library. The architecture generator provides an graphical interface in which the user connects the components used in the architecture. Each component of the library is formally specified by using high level Petri net. The formal model allows to analyse qualitative aspects and quantitative high level constraints of the proposed architecture.

The use of a family of formalisms (high level Petri nets, P/T net, timed Petri nets etc...) sharing basic principles in a consistent way permits if not a formal qualitative/quantitative analysis and a transformation from one formalism to another, a rigorous transformation and analysis of several aspects observed in the design, which makes easy the interrelation between them [23].

2.1 The Partitioning Phases

The hardware/software partitioning is based on the approach proposed by Barros. This approach was developed initially by considering unity specifications but it can be applied to other description languages and an adaptation for occam is being developed. In order to take into account the target architecture presented earlier, some modifications have been made in the cost functions guiding the clustering process. One of them is the consideration of communication cost due to the message passing.

The main tasks associated with the partitioning approach are depicted in figure 1.

The set of implementation alternatives is generated during the *Classification* phase by considering distinct degrees of parallelism when implementing the original program. The set of implementation alternatives is represented by a set of class values concerning some features of the program, such as degree of parallelism, pipeline implementation and communication cost.

The choice of some implementation alternative as the current one can be made manually or automatically. When choosing automatically, the alternatives leading to a balanced degree of parallelism among the various statements and minimising the area-delay cost function ($F = \alpha \ln(Area) + \beta \ln(Area)$) will be taken as the current one.

Before the classification to be performed, the original description is split into a set of concurrent simple processes. A simple process can be an assignment, a sequence of one assignment and input and/or output operations, as well as the constructors if, alt, par whose body is a simple process. The split process is performed by applying a set of transformation rules [8] which preserves the semantic of the original description in order to implement the partitioning.

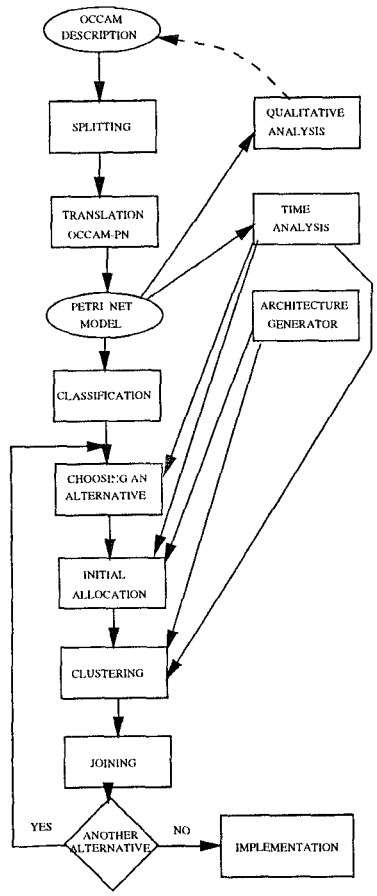

Figure 1: Task Diagram

In the initial allocation phase, the processes are analysed and the best suited processes to be implemented in the software components are chosen considering the communication cost. After that, the allocation the clustering takes place.

The clustering process takes into account only one alternative, for instance, a fully parallel implementation for each simple process presented previously.

The partitioning process is done by using a multistage hierarchical clustering algorithm. The clustering algorithm builds a clustering tree by using a matrix distance. This matrix provides the similarity distance between each two objects. After the construction of the clustering tree, it is cut by a *cutline*. The cutline makes clusters according to a *cutfunction* which observes the possibility of resource sharing and the area-delay cost function.

In the first stage, clusters are built according to the similarity of the process functionality, implementation alternative and communication cost. To build the cluster tree, a metric upon occam constructors [1] and their corresponding implementation alternatives was defined.

In the second stage, a new distance matrix for the clusters (resulting from stage 1) is calculated and from it a new cluster tree is built. The goal of the clustering in the second stage is to keep together processes that should share resources.

After the translation of an occam program into Petri nets, a timing analysis is performed by calculating the minimum execution time of the global process as well as of the set of sub-processes [55] and the communication cost, which are used in the clustering process. The advantage of using Petri net as an intermediate format is the possibility of applying formal techniques for qualitative and quantitative analysis as well as the use of a family of formalism [13] which shares the same basic principles in the architecture specification allowing a sound transformation steps between the different aspects of the design.

As our approach is based on Petri net as intermediate format, in the next section a brief introduction into Petri nets is given.

3 A Brief Introduction into Petri Nets

Petri net is a formal specification technique that allows a graphical, mathematical representation and has powerful methods which allow to perform qualitative and quantitative analysis [27, 17, 18, 16]. In this section a brief introduction into Place/Transitions nets, Deterministic Timed Petri nets and High Level Petri net is given.

3.1 Place/Transition Nets

Place/Transition Nets are bipartite graphs represented by two types of vertices called places (circles) and transitions (rectangles), interconnected by directed arcs.

Place/Transition nets can be defined in terms of matrix as follow:

Definition 3.1 Place Transition Net *is defined as 5-tuple* $N = (P, T, I, O, M_0)$, *where* P *is a finite set of places which represents the state variables,* T *is a set of transitions which represents the set of actions,* $I : P \times T \rightarrow IN$ *is the input matrix which represents the pre-conditions,* $O : P \times T \rightarrow IN$ *is the output matrix which represents the post-conditions and* $M_0 : P \rightarrow IN$ *is the initial marking which represents the initial state.*

[1] In this case, only the occam subset for representing simple processes is considered

The execution of actions is represented by the transition firing.

Definition 3.2 Firing Rule - *One transition t_j is enabled to fire if, only if, all its input places ($p \in P$) has $M(p) \geq I(p,t_j)$. The transition firing changes the marking of the net ($M_0[t_j > M'$). The new marking is obtained as follow: $M'(p) = M_0(p) - I(p,t_j) + O(p,t_j), \forall p \in P$*

Using the matrix representation, the structure of the net is represented by a set of place, a set of transitions, an input matrix (pre-conditions) and an output matrix (pos-conditions). When one transition t fires, the difference between the markings is represented by $O(p,t) - I(p,t), \forall p \in P$. The matrix $C = O - I$ is called incidence matrix. This matrix represents the structure of the net, if the net does not have self-loop.

Definition 3.3 Incidence Matrix - *Let a net $N = (P,T,I,O)$. The incidence matrix represents the relation $C : P \times T \rightarrow Z, \forall p \in P$ defined by: $C(p,t) = O(p,t) - I(p,t), \forall p \in P$.*

One net that has self-loop, may be represented by the incidence matrix if the self-loop is refined using dummy pair [27].

The state equation describes the behaviour of the system, as well as allows to analyse properties of the models.

$$M'(p) = M_0(p) + C.\overline{s},$$

where $\overline{s} = s(t_0)^T, s(t_1)^T, ..., s(t_n)^T$ is called *Parikh vector.*

3.1.1 Analysis

This sections describes shortly the Petri net analysis methods. These methods are used to analyse behavioural and structural properties. The first method is a graph-based and it builds on the reachability graph (reachability tree). The reachability graph is initial marking dependent and so it is used to analyse behavioural properties. The second method is based on the state equation. This method allows to analyse structural properties and necessary or sufficient conditions for behavioural properties in general Petri nets. The third method is based on reductions laws. In this method, a set of reductions rules are applied to the net looking for the size reduction of the model. The set of rules must preserve characteristics such as liveness and boundedness of the systems.

In order to build the reachability tree is necessary to enumerate all the possible reachable markings from the initial marking. It is quite possible that the tree could grow indefinitely. In this case the system is unbounded. Even for bounded systems, the main problem in the using of reachability tree is the high computational complexity even if some interesting techniques are used such as: reduced graphs, graph symmetries, symbolic graph etc [25, 26].

One approach used to keep the size of the tree finite to unbounded systems is called coverability tree [17]. In this method one special symbol "w" is used to represent the number of tokens which can be stored largely in the places. The use of that symbol implies in lose of information, but this method still allows to analyse properties such as: coverability, boundedness, safeness and conservation.

The analysis method based on matrix algebra has some advantages over the reachability graph method [24, 27, 19]. The advantage is the existence of simple linear algebraic equations that aid in determining properties of the nets.

The main problem of this method is that it gives only necessary or sufficient condition to the analysis of properties when it is applied to general Petri nets. This method is particularly interesting if it is applied to some subclasses of Petri nets such as: marked graph and state machine, where it aids to analyse necessary and sufficient conditions.

The analysis methods of large dimensions nets is not a trivial problem, therefore methods which aids the transformation of the models preserving the properties of the original models has been proposed [27, 28]

The reduction laws based methods provides a set of transformation rules which reduces the size of the models preserving the properties. However, it is possible that for a given system and some set of rules, the reduction can not be completed.

In the pragmatic point of view, it is fair to suggest that a better, more efficient and more comprehensive analysis can be done by a cooperative use of these techniques. Nevertheless, necessary and sufficient conditions can be obtained by applying the matrix algebra for some subclasses of Petri nets.

3.2 High Level Nets

The term *High Level Net* is usually applied for the whole class of nets. In most of the approaches, the main characteristics are of a high level net are the fact that it can be unfolded into a *low level net* and that the tokens represent data items as opposed to boolean conditions as in the elementary nets systems [5, 50]. Structured tokens permits the concise representation of complex systems, such as data base and flexible manufacturing systems.

Large system models are only comprehensible if the hierarchy concepts are applied. In Petri net context, these concepts represent the transition and place refinements. Under the theoretical view point, hierarchy is only a graphical convenience and it does not provide any computational power in relation with non-hierarchical models [50].

The hierarchy concept permits the inspection of the systems considering levels of details and re-use of the sub-models. One high level object (place or transition) is represented as a sub-net (page) which allows a structured description taking into account abstraction levels.

4 The *Occam* - Petri Net Translation Method

The *occam*-Petri-net translation method [55] receives an *occam* description and translates it into Petri-nets. The *occam* programming language is derived from *CSP* [54], and allows the specification of parallel algorithms as a set of concurrent processes. An *occam* program is constructed by using primitive processes and process combiners.

The *occam* subset is described in *BNF* format:

$$P ::= SKIP \;||\; STOP \;||\; x := e$$

$$||\; ch?x \;||\; ch!e$$
$$||\; IF(c_1p_1, ..., c_np_n) \;||ALT(g_1p_1, ..., g_np_n)$$
$$||SEQ(p_1, ..., p_n)||PAR(p_1, ..., p_n)$$

Occam programs are constructed by combining primitive processes. The simplest *occam* processes are the assignment, the input action, the output action, the *skip* process and the *stop* process.

This section presents the translation method from a subset of *occam* process into Petri net by using an example. The Petri net models obtained represent both control and data flow. Due to the space restrictions only two primitive process as well as one combiner will be described in this section

4.1 Assignment

The assignment primitive process assigns an expression evaluation to a variable ($var := exp$). For instance, lets consider : $x := y + 1$. The figure 2 presents a Petri net model.

For convenience, each part (control and data) are represented as a separated model, however places, which interconnect both models, are represented as guards attached to the transitions, that is, these guards are places interconnected to the transitions by input and output arcs. The figure 2 shows both models.

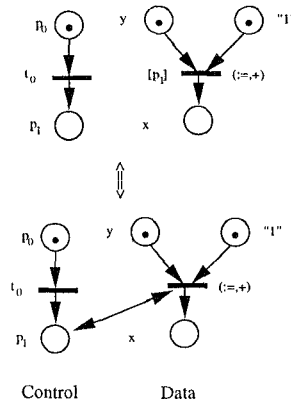

Figure 2: Assignment

4.2 Primitive Communication Process: Input and Output

Occam processes can send and receive message synchronously through channels by using input (?) and output (!) operations. When a process receives a value through a channel, this value is assigned to a variable.

The figure 3.b gives a net representing the input and the output primitive processes of the example in the figure 3.a. The synchronous communication is correctly represented by the net. The reader should observe that the communication action is represented by the transition t_0 and it is only fired when both the sender and the receiver processes are ready, which are represented by tokens in the places p_0 and p_1. When an value is sent by an output primitive process through ch_1, it is received and assigned to the variable x, being represented in the net by the data part of the model. Observe that the transition t_1 can only be fired when the places p_2 and p_3 have tokens. After that, both processes become enabled to execute the next actions.

4.3 Parallelism

The combiner *Par* is one of most powerful of the *occam* language. It permits the concurrent process execution. The combined processes are executed simultaneously and only finish when every one has finished.

The Figure 4.a shows a program containing three processes E, K and S. The figure 4.b shows a net that represents the control of this program.

One token in the place p_0 enables the transition t_0. Firing this transition, the tokens are removed from the input place (p_0 and one token is stored in the output places (p_1, p_2 and p_3). This marking enables the concurrent firings of the transitions E, K and S. After the firing of these transitions, one token is stored in

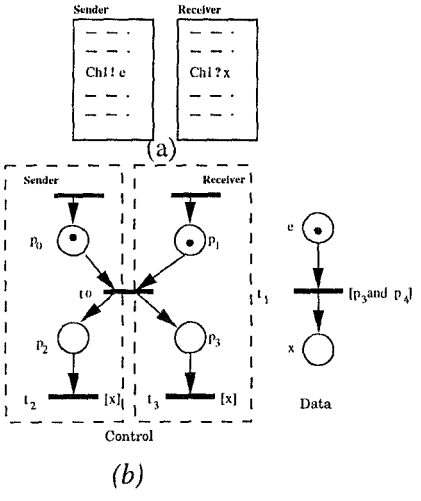

Figure 3: Communication

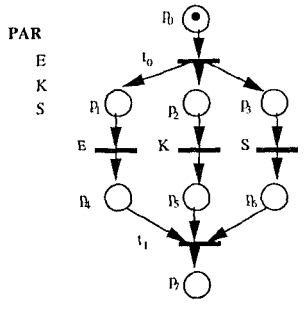

Figure 4: Parallel

the places p_4, p_5 and p_6, respectively. This marking allows to fires the transition t_1, which represents the end of the parallel execution.

5 Computing Communication Cost

This section presents the method proposed to compute communication cost between process by using Petri nets. The communication cost related to a process depend on two metrics: the number of transfered bit by each communication action and the execution nunber that action.

Considering that we are dealing with behavioral descriptions that are translated into Petri nets, we already have defined, in each communication action, the number of transfered bits in each communication actions execution.

Definition 5.1 Number of Transfered Bits by Communication Action - $Nb_b : nbc \rightarrow IN$, *where* $\#Nb_b = T$ *and* T *is the set of transitions. Each component (nbc), associated to a transition that represents a communication action, defines the number of transfered bits by the respective communication action, otherwise is zero.*

However, we have to define a method to compute the communication action execution number related to each process, the communication cost for each process, the description communication cost, the communication cost between two sets of process and finally to compute the normalized communication cost.

The communication cost for each process (CC_{pi}) is the product of the number of transfered bits by communication action (Nb_c) and the communication execution number (NC_{p_i}).

Definition 5.2 Communication Cost for each Process - *Let* Nb_c *the number of transfered bits by communication action and* ${NC_{p_i}}^T$ *a vector that represents communication execution number. The communication cost for each process* p_i *is defined by* $CC_{p_i} = Nb_c \times {NC_{p_i}}^T$.

The used approach to compute the Communication Execution Number (NC_{p_i}) for the processes p_i is preeseted in the following. However first, we present some definitions and theorems in Petri net theory which are important for the proposed approach [27].

A Petri net N is said repetitive if there is a marking and a firing sequence from this marking such that every transition occurs infinitely often. More formally:

Definition 5.3 - Repetitive net: *Let* $N = (R; M_0)$ *a marked Petri net and firing sequence* s. N *is said to be repetitive if there exist a sequence* s *such that* $M_0[s > M_i$ *every transition* $t_i \in T$ *fires infinitely often in* s.

Theorem 5.1 *A Petri net* N *is repetitive if, and only if, there exist a vector* X *of positive intergers such that* $C \cdot X \geq 0, \quad X \neq 0$.

A Petri net is said to be consistent if there is a marking and a firing sequence from this marking back to the same marking such that every transition occurs at least once. More formally:

Definition 5.4 - Consistent net: *Let* $N = (R; M_0)$ *a marked Petri net and firing sequence* s. N *is said to be consistent if there exist a sequence* s *such that* $M_0[s > M_0$ *every transition* $t_i \in T$ *fires at least once in* s.

Theorem 5.2 *A Petri net N is consistent if, and only if, there exist a vector X of positive intergers such that $C \cdot X = 0, \quad X \neq 0$.*

The proofs of such theorems can be found in [27].

The Communication Execution Number (NC_{p_i}) is a vector, where each component (nc_{pi}), associated to a transition that represents a communication action in the process p_i, is the execution number related to the respective communication action, otherwise, that is, the component vector associated to the transition which does not represent the communication action in the process pi, is zero. More formally:

Definition 5.5 Communication Execution Number - *$NC_{p_i} : nc_{pi} \rightarrow IN$, where $\#NC_{p_i} = T$ and T is the set of transitions. Each component $nc_{pi} = max(nc_{p_i X_k})$, $\forall X_k$, where X_k is a vector of positive intergers such that either $C \cdot X = 0$ or $C \cdot X \geq 0$.*

Definition 5.6 *$NC_{p_i X_k} : nc_{pi X_k} \rightarrow IN$, where $\#NC_{p_i X_k} = T$ and T is the set of transitions.*

Each vector X_k is the minimum suport which can be obtained by solving either $C \cdot X = 0$ (in this case X_k are minimum t-invariants) or $C \cdot X \geq 0$. The execution number related to a communication action a_i (represented by a transition t_i) is the respective value obtained in the component x_i for the correspondent X_k. The other components, which do not represent communication actions in the process p_i, are represented by zero value.

According to the results obtained in the qualitative analysis, it is possible to choose methods to compute the communication execution number (NC_{p_i}) regarding to the complexity of the used method.

Whether the net (system) is consitent, first we have to compute the minimum t-invariants then the NC_{p_i}, $\forall p_i \in D$ are obtained. However, if the net is not consistent, but it is repetitive, first the minimal support to X_k, which are obtained by using X in the system $C \cdot X \geq 0$, *where* $X \neq 0$, has to be obtained then the NC_{p_i}, $\forall p_i \in D$ are computed. In the case of the net do not be repetitive and if it is possible to transform it into a repetive or consistent net by inserting one transition t_f such that $I(t_f) = \{p_f\}$ and $O(t_f) = \{p_0\}$, we apply the same method to compute X and then to obtain the NC_{p_i}, $\forall p_i \in D$. These places (p_0 and p_f) are well defined, because one token in the place p_0 (*initial place*) enables the execution of the process and when one token arrives in the place p_f (*final place*), it means that the execution already had finished. Otherwise, if it is not possible to transform the net into a repetive or consistent one, although this system seems do not have interesting properties and even so the designer do not intend to modify it, we can compute the X and then NC_{p_i} by using either the reachability graph or by solving the system $C \cdot X = M_{final} - M_0$, where M_{final} and M_0 are the final and the initial markings, respectively. However, the reader has to remenber that, the state equation could provides spurious solutions for some Petri nets sub-classes [23].

Theorem 5.3 *Let N a consistent net and X_k a minimum t-invariant in the net. Considering every minimum t-invariant in the net ($\forall X_k \in N$) the maximum value obtained for each component vector is the minimum transition firing number for each transition.*

Proof:
Supose a Petri net is consistent. Then there exist an $X \neq 0$ such that $M_0 = M_0 + C \cdot X$. The vectors X can be obtained by using the minimum t-invariants (X_k) and vice-versa. If the vector X is obtained by taking the maximum value of each component between every minimum t-invariants X_k, so each X component is the minimum transition firing number for each transition.

Theorem 5.4 *Let N a repetitive net and X_k a minimum support in the net which can be obtained by using X which solves the equation $C \cdot X \geq 0$. Considering every minimum support X_k in the net, the maximum value obtained for each component vector is the minimum transition firing number for each transition.*

Proof:
Supose a Petri net is repetitive. Then there exist an $X \neq 0$ such that $M \geq M_0 + C \cdot X$. The vector X can be obtained by using the minimum supports (X_k) and vice-versa. If the vector X is obtained by taking the maximum value of each component between every minimum supports X_k, so each X component is the minimum transition firing number for each transition.

The communication cost between two sets of processes p_i and p_j is the product of the communication action execution number between the processes for each comunication and the number of transfered bits by each communication action. More formally:

Definition 5.7 Communication Cost between Processes - *Let Nb_c the number of transfered bits by*

communication action and $NC_{bp_ip_j}$ a vector that represents communication execution number between the process p_i and p_j. The communication cost between processes is defined by $CC_{bp_ipj} = Nb_c \times {NC_{bp_ip_j}}^T$.

The communication execution number between two set of process p_i and p_j is represented by a vector, where each vector component, associated to a transition which represents a communication action between both processes, defines the execution number related to the respective action.

Definition 5.8 Communication Execution Number Between two set of Process - $NC_{bp_ip_j} : nc_{bp_ip_j} \rightarrow IN$, *where* $\#NC_{bp_ip_j} = T$ *and* $nc_{bp_ip_j} = min(nc_{p_i}, nc_{p_j})$.

The behavioral description communication cost is represented by summation of communication cost between each pair of processes in the description. More formaly:

Definition 5.9 Description Communication Cost - $CC_D = \sum_{\forall(p_i,p_j)\in D} CC_{bp_ip_j}$

We have to define two kinds of normalization: local normalization and global normalization [4]. The global normalized communication cost between two processes is defined by the communication cost between both processes diveded by the communication cost for the whole description.

Definition 5.10 Global Normalized Communication Cost - *Let CC_{bpipj} the communication cost between the processes p_i and p_j, and CC_D the communication cost in whole behavioral description. The global normalized communication cost is defined by* $NCC_{bp_ip_j} = \frac{CC_{bp_ip_j}}{CC_D}$.

The local normalized communication cost between two process is defined by the communication cost between both processes diveded by the summation of the communication cost for each process.

Definition 5.11 Local Normalized Communication Cost - *Let CC_{bpipj} the communication cost between the processes p_i and p_j, and CC_{p_i} and CC_{p_j} the communication cost for the process p_i and p_j, respectively. The local normalized communication cost is defined by* $LCC_{bp_ip_j} = CC_{bp_ip_j}/(CC_{p_i} + CC_{p_j})$.

The algorithm to compute the global normalized communication cost is given following:

1. To compute the communication execution number for each process (NC_{p_i})
2. To compute the communication cost for each process p_i ($CC_{p_i} = Nb_c \times {NC_{p_i}}^T$)
3. To compute the communication execution number between two set of Process for all pair of processes in the description ($NC_{bp_ip_j}$).
4. To compute the communication cost between each pair of processes ($CC_{bp_ip_j} = Nb_c \times {NC_{bp_ip_j}}^T$)
5. To compute the description communication cost ($CC_D = \sum_{\forall(p_i,p_j)\in D} CC_{bp_ip_j}$)
6. To compute the global normalized communication cost for each pair of processes ($NCC_{bp_ip_j} = \frac{CC_{bp_ip_j}}{CC_D}$).

The algorithm to compute the local normalized communication cost is given following:

1. To compute the communication execution number for each process (NC_{p_i})
2. To compute the communication cost for each process p_i ($CC_{p_i} = Nb_c \times {NC_{p_i}}^T$)
3. To compute the communication execution number between two set of Process for all pair of processes in the description ($NC_{bp_ip_j}$).
4. To compute the communication cost between each pair of processes ($CC_{bp_ipj} = Nb_c \times {NC_{bp_ip_j}}^T$)
5. To compute the local normalized communication cost for each pair of processes ($LCC_{bp_ip_j} = CC_{bp_ip_j}/(CC_{p_i} + CC_{p_j})$).

6 An Example

This section shows the use of the proposed method applied to an example. The method is applied to a behavioral description written in occam which was translated into Petri nets by using translation method proposed in the section 4. The example implements the convolution function given by

$$y_i = \sum_{j=1}^{n} x_{i-j} \times \omega_j \times \alpha_i, 1 \leq i \leq 2n-1$$

Where ω_j is $\omega_{j+1} = b \times x_1 \times \omega_j$ and $\alpha_i = c_i + d_i$.

$$c_i = \begin{cases} x_i & \text{if } x_i \geq 0 \\ \frac{x_i}{2} & \text{if } x_i < 0 \end{cases}$$

$$d_i = \begin{cases} x_{i+1} & \text{if } x_{i+1} \geq 0 \\ \frac{x_{i+1}}{2} & \text{if } x_{i+1} < 0 \end{cases}$$

The *occam* program implementing this function is described by a set of concurrent processes:

```
CHAN OF INT P1.P4, P2.P4, P4.P3 :

CHAN OF [5] INT P3.P4 :

PAR

  INT c:                                    Process P1

  SEQ i=0 FOR 2

      IF(x[i]>=0 c:=x[i], x[i]<0 c:=x[i]/2)

      p1.p4 ! c

  INT d:                                    Process P2

  SEQ i=0 FOR 2

    IF(x[i]>=0 d:=x[i+1], x[i]<0 d:=x[i+1]/2)

    p2.p4 ! d

  INT w:                                    Process P3

  SEQ i=0 FOR 2

    p4.p3 ? w

    PAR j=0 FOR 4

      e[j]:=x[5*(i/(j+((j+1)/(i+1))))+(j-i)]*w

    p3.p4 ! e

  INT c,d:                                  Process P4

  [5] INT e :

  SEQ i=0 FOR 2

    p4.p3 ! w

    PAR

      p1.p4 ? c

      p2.p4 ? d

      p3.p4 ? e

  PAR

    w:=k*e[i]

    PAR j = 0 FOR 4

      y[j] := y[j] + e[j] * (c + d)
```

The obtained communication costs are:

Process	CC_{p_i}
1	96
2	96
3	192
4	384

Processes	$CC_{bp_ip_j}$
p_1p_2	0
p_1p_3	0
p_1p_4	96
p_2p_3	0
p_2p_4	96
p_3p_4	192
Description	384

Processes	$NCC_{bp_ip_j}$
p_1p_2	0
p_1p_3	0
p_1p_4	0.25
p_2p_3	0
p_2p_4	0.25
p_3p_4	0.5

Processes	$LCC_{bp_ip_j}$
p_1p_2	0
p_1p_3	0
p_1p_4	0.2
p_2p_3	0
p_2p_4	0.2
p_3p_4	0.3333

7 Conclusion

This work has presented a method to compute the communication cost as well as the use of Petri nets as an intermediate format in the context of hardware/software codesign. An approach of *occam*-Petri nets translation, the main tasks of our partitioning approach were presented as well. The translation scheme

and the use of the proposed algorithm to compute communication cost were illustrated by a case study.

Due to the large size of the real applications, it is essential to use an intermediate format, which is powerful in the process modelling, property analysis and performance analysis, as well as to deal with hierarchies. Additionally, Petri nets can also be used to compute cycle time of the descriptions and the target architecture. This feature is very interesting, since the same family of formalism is used for modelling the target architecture and the partitioned system.

As future works, we intend to use Petri net to compute a mutual exclusion and load balance metric in order for use it to performe the initial allocation and the partition process as well.

References

[1] A.A.Desrochers, R.Y.Al-Jaar *Applications of Petri Nets in Manufacturing Systems* IEEE Press. 1995.

[2] R. Gupta *A Framework for Interative Analysis of Timing Constraints in Embedded Systems* Proceedings of the Fourth Codes/CASHE, pp 44-51. IEEE Computer Society. March, 1996.

[3] W.Hardt, W.Rosenstiel *Speed-Up Estimation for HW/SW-Systems* Proceedings of the Fourth Codes/CASHE, pp 36-43. IEEE Computer Society. March, 1996.

[4] F. Vahid, D. D. Gajski *Closeness Metrics for Systems Level Functional Partitioning* Proceedings of the EURO-DAC'95, pp 328-333. IEEE Computer Society. September, 1995.

[5] H. J. Genrich. *Predicate/Transition Nets.* Lecture Notes in Computer Science, part I, vol-254, p. 207-247, Springer-Verlag, Edited by G. Rozenberg 1987.

[6] E. Barros *Hardware/Software Partitioning using UNITY* .Universität Tübingen 1993.

[7] E. Barros and W. Rosenstiel *A Clustering Approach to Support Hardware/Software Partitioning* Computer Aided Software/Hardware Engineering, edited by Jerzy Rozenblit and Klaus Buchenrieder, IEEE Press.

[8] E. Barros and A. Sampaio *Towards Provably Correct Hardware/Software Partitioning Using Occam* Proceedings of the third International Workshop on Hardware/Software Codesign Codes/CASHE94, IEEE Computer Society. September, 1994.

[9] E. Barros and X. Xiong and W. Rosenstiel *Hardware/Software Partitioning with UNITY* Handouts of International Workshop on Hardware-Software Co-design. 1993.

[10] R. Ernst and J. Henkel *Hardware-Software Codesign of Embedded Controllers Based on Hardware Extraction* Handouts of the International Workshop on Hardware-Software Co-Design. October, 1992.

[11] J. P. Soininen and M. Sipola and K. Tiensyjä *SW/HW Partitioning of Real-Time Embedded Systems* Microprogramming and Microprocessing, vol 27, pp 239-244. 1989.

[12] R. Gupta and G. De Micheli *System-level Synthesis Using Re-programmable Components* Microprogramming and Microprocessing, vol 27, pp 239-244. 1989.

[13] C.Carreras, J.C.López, M.L. López, C.Delgado-Kloos, N. Martínez, L. Sánchez *A Co-Design Methodology Based on Formal Specification and High Level Estimation* Proceedings of EDAC, pp 2-7. 1996.

[14] F.Rose, T.Carpenter, S.Kumar, J. Shackleton, T. Steeves *A Model for the Coanalysis of Hardware and Software Architecture* Proceedings of the Fourth Codes/CASHE, pp 94-103. IEEE Computer Society. March, 1996.

[15] R.Gupta, G.De Micheli *Constrained Software Generation for Hardware-Software Systems* Proceedings of the Fourth Codes/CASHE, pp 56-63. IEEE Computer Society. September, 1995.

[16] G.W. Brams. *Réseaux de Petri: Théorie et Pratique, tome 1 and 2.* Masson Editions, 1983.

[17] J. L. Peterson. *Petri Nets an Introduction.* Pretence-Hall, Inc, 1981.

[18] W. Reisig. *Petri Nets: An Introduction.* Springer-Verlag, 1982.

[19] T. Murata. *State Equation, Contrallability, and Maximal of Petri Nets.* IEEE Trans. on Automatic Control, 1977.

[20] O.Botti, F.Cindio *From Basic to Timed Net Models of Occam: an Application to Program Placement.* PNPM, pp 216-221, 1991.

[21] P. R. M. Maciel, T. M. P. Medeiros, L. C. Albuquerque, J. F. B. Castro. *Uso de Redes de Petri Temporizadas para o Diagnóstico de Falhas em Sistema Digitais.* VI Simpósio de Computadores Tolerantes a Falhas, p. 181-200, Canela, RS, Brazil, 1995.(portuguese)

[22] A.Marsan, G.Balbo, G.Conte, A.Bobbio, G.Chiola, A.Cumani *The Effect of Execution Poliies on the Semantic and Analysis of Stochastic Petri Nets.* IEEE TSE, Vol 15, n7, pp 832-846, 1989.

[23] M.Silva, E.Teruel *Petri Nets for the Design and Operation of Manufacturing Systems.* CIMAT'96, 1996.

[24] M.Silva, E.Teruel *Analysis of Autonomous Petri Nets with a Bulk Services and Arrivals.* 11^{th} International Conference on Analysis and Optimization of Systems. Discret Event Systems, Vol 199 of Lecture Notes in Control and Information Science, pp 131-143, 1994.

[25] A.Valmari. *Stubborn Sets for Reduced State Space Generation.* Advanced in Petri Nets, vol 483, Lecture Notes in Computer Science, Springer Verlag, Edited by G. Rozenberg, pp 491-515, 1991.

[26] A.Valmari. *Compositional State Space Generation.* Advanced in Petri Nets, vol 674, Lecture Notes in Computer Science, Springer Verlag, Edited by G. Rozenberg, pp 427-457, 1993.

[27] T. Murata. *Petri Nets: Properties, Analysis and Aplications.* Proceding of The IEEE, 1989.

[28] G.Berthelot. *Checking Properties of Nets Using Transformations.* Advanced in Petri Nets, vol 222, Lecture Notes in Computer Science, Springer Verlag, Edited by G. Rozenberg, pp 19-40, 1986.

[29] Ramchandani *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets.* Techinical Report n^{120}, laboratory for Computer Science, MIT, Cambridge, MA. 1974.

[30] G. Balbo, G. Chiola, S.c. Bruell, P. Cheng. *An Example of Modeling and Evaluation of Concurrent Program Using Colored Stochastic Petri Nets: Lamports's Fast Mutual Exclusion Algorithm.* IEEE Transaction on Parallel and Distributed Systems , 1992.

[31] K. Bilínsk, M. Adamski, J.M. Saul, E.L. Dagless. *Parallel Controller Synthesis from a Petri Net Specification.* Proceedings EURODAC-94 , 1994.

[32] G. Dohmen. *Petri Nets as Intermediate Representation Between VHDL and Symbolic Transition Systems.* Proceedings EURODAC-94 , 1994.

[33] J. Esparza. *Reduction and Synthesis of Live and Bounded Free Choice Petri Nets.* Information and Computation , 1994.

[34] M. Hack. *Analysis of Production Schemata by Petri Nets.* Master's Thesis, Departament of Eletrical Engineering, Massachusetts Institute of Technology Massachusetts, Febuary, 1972.

[35] F. Commoner. *Deadlock in Petri Nets.* Applied Data Research Inc. Massachusetts, 1972.

[36] J. Esparza, M. Nielsen. *Decidability Issues for Petri Nets.* Gesellschaft für Informatik, 1994.

[37] C. Rackoff. *The Covering and Boundedness Problem for Vector Addition Systems.* Theorical Computer Science, vol. 6, p. 223-231. 1978.

[38] R. Karp, R. Miller. *Parallel Program Schemata.* Journal of Computer and System Science, vol. 3, N. 4, p. 167-195. 1969.

[39] R. Lipton. *The Reachability Problem Requires Exponential Space.* Research Report 62, Departament of Computer Science, Yale University, 1976.

[40] G. S. Sacerdote, R. L. Tenney. *The Decidibility of the Reachability Problem for Vector Addition System.* 9th Annual Sympossium on Theory of Computing, p. 61-76.

[41] E. W. Mayr. *Persistence of Vector Replacement System is Decidable.* Acta Informatica 15, p. 309-318. 1981. Boulder, 1977.

[42] S. R. Kosaraju. *Decidibility of Reachability in Vector Addition Systems.* 14th Annual ACM Symposium on Theory of Computing, p. 267-281. San Francisco, USA, 1982.

[43] J. L. Lambert. *Vector Addition Systems and Semi-linearity.* SIAM Journal of Computing, 1994.

[44] D. Frutos, C. Johne. *Decidability of Home States in Place Transition Systems.* 14th Internal Report, Dpto. Informatica y Automatica, Univ. Complutense de Madrid, 1986.

[45] E. Cardoza, R. J. Lipton, A. R. Meyer. *Exponential Space Complete Problems for Petri Nets and Commutative Semigroups.* 8th Symposium on Theory of Computing, p. 50-54. 1976

[46] M. H. T. Hack. *Decidability Questions for Petri Nets.* PhD Thesis, MIT, 1976.

[47] A. Cheng, J. Esparza, J. Palsberg. *Complexity Results for 1-safe Nets.* 13th Conference on Foundations of Software Technology and Theorical Computer Science, Bombay, 1993.

[48] J. Grabowsky. *The Decidability of Persistence for Vector Addition Systems.* Information Processing Letters 11, vol-1, p. 20-23, 1980. 76.block Boulder, 1977.

[49] H. Müller. *On the Reachability Problem for Persistent Vector Replacement Systems.* Computing Supplements, vol-3, p. 89-104, 1981.

[50] K. Jensen. *Coloured Petri Nets: A High Level Language for System Design and Analysis.* Lecture Notes in Computer Science, vol-483, p. 342-416, 1990.

[51] K. Jensen, P. Huber, R. M. Shapiro. *Hierarchies in Coloured Petri Nets.* Lecture Notes in Computer Science, vol-483, p. 313-341, Springer-Verlag, Edited by G. Rozenberg 1990.

[52] G. Dittrich. *Modeling of Complex Systems Using Hierarchical Petri Nets.* Codesign - Computer-Aided Software/Hardware Engineering, p. 128-144, IEEE Press, Edited by J. Rozenblit, K. Buchenrieder 1995. ck Springer-Verlag, Edited by G. Rozenberg 1990.

[53] P. R. M. Maciel, E. N. S. Barros. *Captura de Requisitos Temporais Usando Redes de Petri para o Particionamento de Hardware/Software.* IX Simpósio Brasileiro de Concepção de Circuitos Integrados, p. 383-396, Recife, PE, Brazil, 1996.

[54] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall International, 1985.

[55] P. R. M. Maciel, E. N. S. Barros. *Capturing Time Constraints by Using Petri Nets in the Context of Hardware/Software Codesign.* a ser publicado no 7th IEEE International

Workshop on Rapid System Prototyping, Porto Caras, Thessaloniki, Grécia, 1996.

[56] G. Jones. *Programming in OCCAM.* C.A.R. Hoare Series Editor, Prentice-Hall International Series in Computer Science, 1987.

[57] P.R.M.Maciel, R.D.Lins, P.R.F.Cunha. *Uma Introdução às Redes de Petri e Aplicações.* Book published in the 11^{th} Escola de Computação. Campinas, Brazil. July, 1996. (portuguese)