Des Autom Embed Syst (2010) 14: 75–103
DOI 10.1007/s10617-010-9051-5

A scheduler synthesis methodology for joint SW/HW design exploration of SoC

Ismail Assayad · Sergio Yovine

Received: 26 April 2009 / Accepted: 3 February 2010 / Published online: 16 March 2010

Abstract The introduction of high-performance applications such as multimedia applications into SoCs led the manufacturers to provide embedded SoCs able to offer an important computing power which makes it possible to answer the increasing requirements of future evolutions of these applications. One of the adopted solutions is the use of multiprocessor SoCs. In this paper, we present a joint SW/HW design exploration methodology for multiprocessor SoCs. The system model relies on transaction-level component-based models for modeling parallel software and multiprocessor hardware. Our proposal comprises two original points. First, we propose a *composable* software-level scheduler *constraints* synthesis technique. Second, we present a *combined* software-level and *exploratory* hardware-level schedulers. The methodology has the advantage of *combining real-time* requirements of software with *effective* exploitation of multiprocessor hardware. We describe and apply the methodology to synthesize a scheduler of a slice-based MPEG-4 video encoder on the multiprocessor Cake SoCs.

Keywords Multiprocessor System-On-Chips (SoCs) · SW/HW design · Scheduling · Exploration · Real-time requirements

1 Introduction

High performance embedded applications such as video compression, HDTV, and packet routing, motivate the use of off-the-shelf, configurable, heterogeneous hardware platforms offering multiple processing units, such as Philips' VIPER [1] and Wasabi/Cake [2] SoCs, and Intel's IXP family of network processors [3]. These architectures provide significant price, performance and flexibility advantages. However, the complexity of such *multiproces-*

I. Assayad (✉)
ENSEM, University of Hassan II Ain Chock, Oasis Casablanca, Morocco
e-mail: i.assayad@ensem.ac.ma

S. Yovine
Departamento de Computacion, Universidad de Buenos Aires, and Researcher at CONICET, Buenos Aires, Argentina
e-mail: syovine@dc.uba.ar

sor embedded systems (MES) makes software programming and analysis difficult, leading to sub-optimal software and hardware performances. Furthermore, as new applications and services are continuously developed, the main challenge is to provide design frameworks capable of supporting accurate, but fast, performance estimation for dimensioning the hardware SoC in order to support integration of future applications [4]. This requires *joint*, rather than separate, software and hardware modeling and simulation at design time. Indeed, an integrated software/hardware design exploration methodology, supported by the appropriate tools, gives system developers means to improve field upgradability and time to market, and therefore lower development costs, of embedded SoCs [5, 6].

In requirement-based system-level design, the set of tasks requirements plays a prominent role in the design of the system. Typically, a designer studies this set of requirements, makes some initial calculations and proposes a system architecture. The effectiveness of this architecture is then to be evaluated and a comparison with alternative architectures is to be made. Architectures are evaluated quantitatively by means of performance estimation. For this, the tasks are scheduled onto the model of hardware and the performance of each tasks-to-hardware scheduling combination is evaluated. The resulting performance numbers would be used by the system designer either (1) to modify the hardware, by adding or removing components, changing their properties (e.g., bandwidth, frequency), etc., (2) to restructurate the tasks, or (3) to use another scheduling, in order to improve overall performances and/or meet the requirements.

In this paper, we present a SW/HW scheduling methodology developed along these lines. The software is composed of hierarchical interdependent tasks which combine local real-time requirements of tasks and performance efficiency of hardware. This methodology is exploratory and is composed of two steps. The first step is the synthesis of a software-level scheduler which defines a partial order on the set of hierarchical tasks. This scheduler is independent from the hardware but take into accounts the interactions with the hardware through data or precedence dependencies. The second step is a mapping step. This step defines task scheduling at the hardware level.

This scheduling methodology is suitable for SW/HW design exploration of multimedia applications. It is successfully used to find correct and efficient implementations of a slice-based MPEG-4 video encoder on a multiprocessor SoC using P-WARE tool.

1.1 Related work

Several techniques have been proposed to address this issue. These techniques are classified into three categories according to their modeling scope, namely *software*, *hardware* and *platform* based approaches; and into two categories according to their modeling method, i.e. *analytical* and *simulation* approaches.

Software-based design approaches for MES, e.g. [7, 8], do not provide means for analyzing the impact of software implementation choices in the performance of hardware *micro-architectures*, such as buses bandwidth or banks conflict. Thus, it becomes impossible to measure the capability of hardware configurations to accommodate future softwares. Hardware-based design approaches, e.g. [9, 10], do not consider software programming and analysis as part of the design flow. Therefore, there is no methodology to evaluate the impact into software performance of changing a hardware configuration parameter. In contrast, platform-based design (PBD) [11] provides the adequate level of abstraction that can be used for addressing this problem.

Most PBD approaches found in the literature are not thought to provide complete solutions for MES performance modeling and analysis.

METROPOLIS [12] provides general-purpose framework in the sense that it does not make any assumption about the functional and timed models of micro-architectures. This has the advantage of broadening the applicability of the framework for modeling concurrency at a generic abstraction level. Nevertheless, this approach does not propose a methodology for modeling and analysis of micro-architectures which resorts to the specific skills of the designer.

The synthesis methodology presented in SHE [13] targets control software on single processor and does not handle multiprocessors. TTL-based approaches [14, 15] does not propose algorithmics for the synthesis of a scheduled software which thus resorts to the designer exploration effort [14, 15], and does not handle software-level design exploration [16].

Analytical models have been proposed in [17, 18] for modeling softwares in the specific domain of packet processing. Reference [19] proposes a design flow allowing analytical reasoning about performance requirements. Reference [20] automatically generates designs of multiprocessor system on chip for FPGA instantiation. These works are however specific for network on chip with guaranteed services, and softwares where only communication requirements on network interfaces are relevant.

1.2 Our approach

In contrast to these approaches, our proposal in this paper is a scheduler synthesis methodology for SW/HW design exploration of multiprocessor SoC. The methodology has the advantage of being exploratory: it provides a software-level scheduler synthesis technique, and an exploration approach for mapping software on hardware and synthesis of a hardware-level scheduler. The synthesized scheduler allows to avoid exploring cases where the temporal requirements of software are violated which simplifies the design exploration for the designer, while the exploratory mapping allows designer to choose an efficient placement with regards the performance of hardware. A formal and compositional synthesis is used for computing each of the software and hardware-level schedulers.

This approach is original and has two novel contributions:

- In the scheduler synthesis, it *combines* both *real-time* software requirements consisting of deadlines, and *performance* efficiency of hardware such as bandwidth consumption, channels size, memory population, etc.
- It is an *exploratory* approach suitable for design space exploration of SoC. In this approach, the synthesized software-level scheduler is *composable* with the hardware-level ones and hence computed only once during all design cycles, while the hardware-level schedulers change according to defined mappings in design loops.

Basics of software/hardware models and a compositional constraint generation technique for concurrent loops are presented in previous work [21–23] and [24] respectively. In this paper we will extend this technique and show how to apply it to the tasks model, and we will focus on presenting the methodology for synthesizing and exploring software-level and hardware-level schedulers by using the technique for constraints generation and P-WARE tool [23] for simulation-driven performance evaluation.

1.3 Outline

The remainder of the paper is structured as follows. Section 2 presents the software and hardware models while Sect. 3 gives a big picture of the synthesis. Section 4 gives the details of the constraints synthesis used for the calculation of the software level scheduler.

Section 5 presents the hardware level scheduler. Section 6 illustrates the methodology on a slice-based MPEG-4 video encoder. Section 7 presents the results obtained for an OC-48 IPv4 forwarding application. Section 8 gives a practical overview of synthesis flow and execution choices. Finally, Sect. 9 presents some conclusions and future work.

2 System model

In this section we present the software and hardware models.

2.1 SW model

A software is composed of a set of simple or hierarchical tasks. A component $\mathcal{A}$ of a task t is a tuple $\langle t, \mathcal{C}, \mathcal{D}\rangle$ where $\mathcal{C}$ is the set of constraints on the begin times of sub-tasks $t1, \ldots, tn$, $\mathcal{D}$ is the SW-to-HW mapping.

The constraints on begin times for a hierarchical task t define the scheduling of its sub-tasks $t1, \ldots, tn$. The execution time of a task may be unknown, or dependent on the mappings. The last case is equivalent to the use of the *or* operator to specify all the mappings.

A SW architecture is a set of tasks components. Sections 4 and 5 explain the methodology for defining the scheduling and mapping constraints of the components.

The software component model is depicted in Fig. 1. The scheduler (C) receives the scheduling requests (?ti.R) from tasks and puts them into a set of waiting tasks. It computes the set of tasks to begin according to the timing constraints and notifies them (!ti.S). During execution, tasks transaction requests (TR) are consumed by hardware components according to defined mapping (D). The scheduler also observes the completion time of tasks, thus when a task completes it notifies the scheduler (?ti.E).

2.2 HW model

Unlike pure functional models, components need accurate models for their traffic and timing changes to obtain timing performance of the hardware. A good example to illustrate this idea is the modeling of a cache. By definition a cache is an implementation to improve the performance of the hardware. It is not required to be included in the model to verify

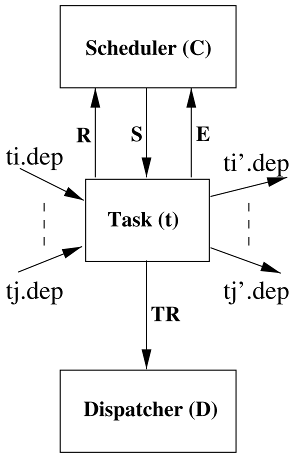


Fig. 1 Software component

the functional correctness of a program. Yet, it is necessary to observe the actual traffic of cache activities on the buses for collecting the timing performance, not only for the cache, but for the overall hardware. In this case, the model of the cache must include not only timing latencies, but also some algorithmics that reflect the cache effect on the data amount generated onto the buses. The same requirements apply to the modeling of other hardware components.

We present here a transaction-level functional and timing model for hardware component.

A transaction request *TR* for a transaction T is a tuple composed of information such as the type (e.g. read, write), stage of transaction (e.g. arbitration), and a data payload. A HW component C is composed of a behavior and an interface $\langle I_i, O_j \rangle$. The behavior is composed of a request arbiter, a transaction controller, the transaction-level timed behavior. I_i is the set of input ports for the requests *TR* of C. O_j is the set of output ports of the requests *TR* of C.

A connection is a tuple $\langle O, I_1, \dots, I_n \rangle$ where O is an output port of the source component, $I_1, \dots, I_n$ are the input ports for the requests *TR* sent from the port O. A component transactor generates and sends requests using blocking and non-blocking calls on input interfaces of destination components through these connections.

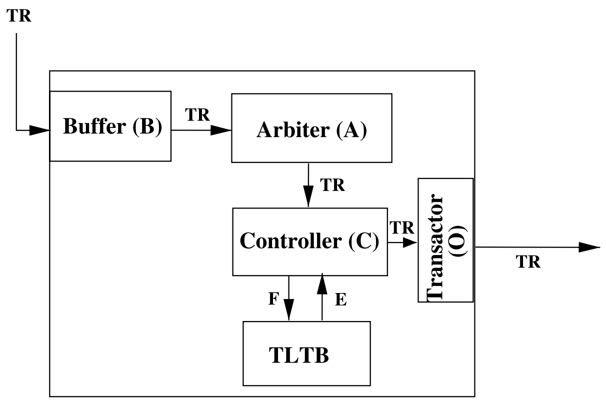
A HW architecture is a pair $\langle \mathcal{C}, \mathcal{B} \rangle$ where

– $\mathcal{C}$ is the set of components.
– $\mathcal{B}$ is the set of connections.

The full component blocks semantics is given in [22, 23]. The principal component details are also given in what follows.

The hardware components model is depicted in Fig. 2. It is composed of five blocks whose interactions are fixed. The buffer stores input TR, and it signals if the component accepts or refuses new TR. The arbiter is to be instantiated with the arbitration policy inside the buffer. Both the buffer and the arbiter may incur, i.e. may be annotated by, latencies. The controller is to be instantiated with a control policy in charge of deciding which and when to launch a transaction and resolving conflicts between operations: an operation starts execution when it receives a "fire" message from the controller, and notifies the latter and the next operation to start with the completion of the operation by sending the end message and the start message, respectively. The transactor sends outputs to recipient's input buffers. It is also used to control transaction granularity, i.e., size of physical data on which transactions operate in the real micro-architecture (e.g. data of the bus). The transaction-level timed behavior (TLTB) is to be instantiated with the set of transaction behaviors.

Fig. 2 Hardware component



When the controller receives a transaction request Ti from the arbiter, it decides which transaction of TLTB to fire and when. This decision making is hardware specific. If the transaction is enabled it immediately fires it, otherwise, the firing is delayed until the transaction becomes enabled. The controller also resolves inter-transaction conflicts discussed before. When a transaction completes, the controller is notified and sends results to the transactor. The latter outputs results, i.e. generates `TR` to output buffers or to controller of target component.

As aforementioned, the interactions between blocks are fixed but their behaviors are described by instantiating the automata: functions associated to some states of these automata are to be implemented (e.g. arbitration function of the arbiter, operation latencies of the TLTB, etc.). To return to the cache example, this model allows the designer to accurately describe the cache data access latencies, its algorithmics, and its generated traffic on the buses through the instantiations of the TLTB, controller and transactor, respectively.

It is worth noticing that In this work, hardware component models are used for simulation purposes in the hardware-level scheduler synthesis step: for a given architecture configuration, these component-based TLM simulations will allow for evaluating the efficiency, with regards the hardware performance, of a set of tasks mapping choices.

3 Big picture of the synthesis

3.1 Synthesis steps and flow

We present here a big picture introducing a view of the context of our scheduler synthesis approach. We consider an architecture and an application running on it. The application is the software and its environment, while the architecture is the underlying execution hardware. The application model only[1] takes into account environment execution times and periodic activations, and software and environment interactions due to data and control dependencies.

Then, the step "constraint synthesis" derives a software-level scheduler of this application and constraints on the parameters which must be satisfied at runtime, by any parallel mapping of software which will be defined later. After that, we look for system implementations, e.g. mappings on processors, and compute corresponding hardware-level schedulers which must satisfy the latter constraints. This is done by considering several classes of parallel implementations (e.g., data parallel, task parallel and hybrid implementations).[2]

The different parts of the context and the flow of the scheduler synthesis are depicted in Fig. 3.

3.2 Kind of equations used in the synthesis

Before describing the constraints synthesis in details, we show here on a very simple example what kind of equations are necessary for the synthesis with less formulations first. This is composed of two tasks which are two loops with data dependencies (Fig. 4). e4 and e9 denote the computations end times. Numbers beside states are the execution times.

[1]That is, the model abstracts away from hardware-dependent issues, such as conflicts between software and environment communication due to concurrent bus and memory accesses. An automatic target specific profiling of user code is outside the scope of this paper and considered as an input. Thus timing properties attached to (computation and communication) tasks should take them into account, otherwise the inaccurate timing will be identified by simulation and back-annotated on the model.

[2]This exploration is currently manual.

Fig. 3 Simplified design flow

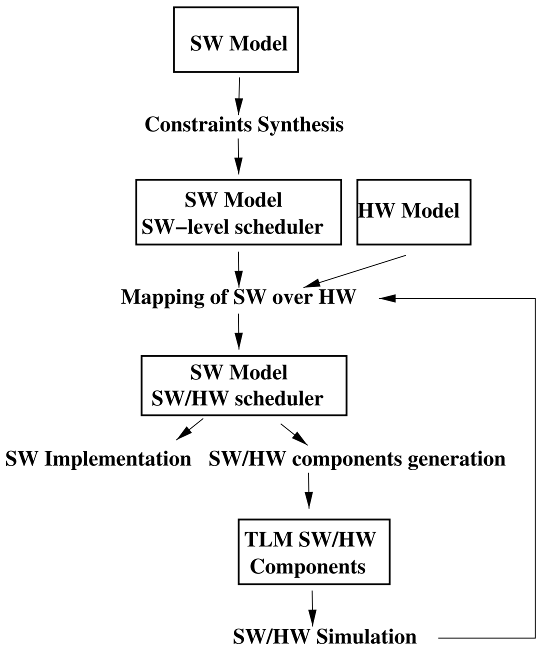
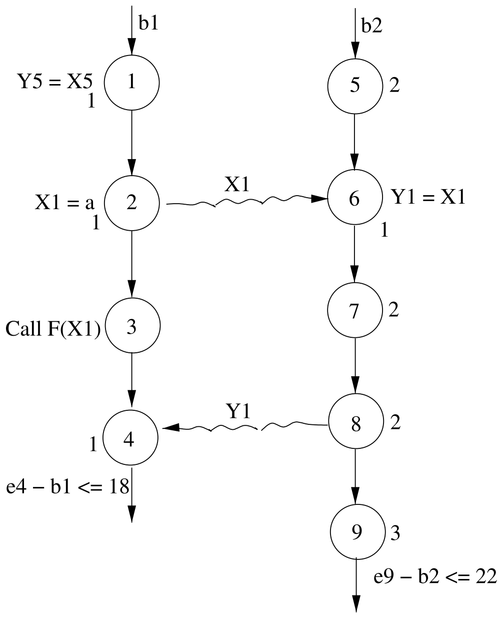


Fig. 4 Simple tasks automata



Without loss of generality, control points of tasks are considered to be on the first computations of the loops. It is possible to use parallel tasks when more control points are needed. In the latter case, computations are dispatched on several automata instead of one.

Therefore, the scheduler to be synthesized will start execution of tasks according to some constraints on b1 and b2, the begin times of computations 1 and 2 as shown in Fig. 4.

In the process of computing a scheduler we synthesize a set of constraints. These constraints are compositional which allows for incremental synthesis. As will be seen in details in the paper, we will take advantage from this feature to incrementally compute and compose a software-level scheduler with hardware-level schedulers in two steps considering both real-time and performance objectives, respectively.

Concretely, for Fig. 4, we may synthesize the software-level scheduler for the two tasks and compose it with the software-level scheduler of the sub-tasks of $F(X)$. Then, after deciding the tasks-mapping we can compute the hardware-level scheduler by adding the mapping constraints. These compositions are detailed in this paper for the general case.

Let us now focus here on the main kind of equations which might be calculated for the synthesis in both steps: the backward propagation equations and interaction equations.

The backward propagation Equations are the equations resulting from the propagation of the requirements starting from the last states of the automata. Since the control points are considered to be on the beginning of the first computations, the equations of the first kind are the following:

$$\Psi_1 = \begin{array}{l} \sigma_9 \leq 22 \wedge \sigma_9 = \sigma_8 + 3 \\ \wedge\, \sigma_8 \leq 19 \wedge \sigma_8 = \sigma_7 + 2 \\ \wedge\, \sigma_7 \leq 18 \wedge \sigma_7 = \sigma_6 + 2 \\ \wedge\, \sigma_6 \leq 16 \wedge \sigma_6 = \sigma_5 + w_6 + 1 \\ \wedge\, \sigma_5 \leq 15 \wedge \sigma_5 = 2 \end{array} = \begin{array}{l} \sigma_9 = \sigma_8 + 3 \\ \wedge\, \sigma_8 = \sigma_7 + 2 \\ \wedge\, \sigma_7 = \sigma_6 + 2 \\ \wedge\, w_6 \leq 16 - 1 - \sigma_5 \leq 13 \wedge \sigma_6 = \sigma_5 + w_6 + 1 \\ \wedge\, \sigma_5 = 2 \end{array}$$

$$\Psi_1 = \text{Constraint} \wedge \text{Invariant} = w_6 \leq 13 \wedge I_1 = W_1 \wedge I_1$$

Where σ_i and w_i are the relative end time, and waiting time of automaton state i respectively. Similarly, for second automaton we obtain equations of the form:

$$\Psi_2 = \sigma_3 + w_4 \leq 17 \wedge I_2 \;=\; W_2 \wedge I_2$$

Interaction equations The second kind of equations are equations that characterize the interactions of automata. These equations will link waiting time variables to (scheduler) control points. If we assume that waiting times meet the W's constraints, then these equations characterize the subset of interactions which satisfy the real-time requirements. For the simple example above the equations of the dependencies-based interactions are the following four: the two first equations for one interaction and the two second equations for the other interaction, $\pi_{12} = -\pi_{21}$ denotes $b1 - b2$.

$$\begin{array}{ll} \pi_{21} + \sigma_5 < \sigma_2 & \wedge\, w_6 = \sigma_2 - \pi_{21} + \sigma_5 \\ \pi_{21} + \sigma_5 >= \sigma_2 & \wedge\, w_6 = 0 \\ \pi_{12} + \sigma_3 < \sigma_8 = 6 + w_6 & \wedge\, w_4 = 6 + w_6 - \pi_{12} - \sigma_3 \\ \pi_{12} + \sigma_3 \geq \sigma_8 = 6 + w_6 & \wedge\, w_4 = 0 \end{array}$$

How to compute constraints? Finally, to compute the constraints on $b1$, $b2$ and σ_3, the execution time of computation 3, we replace, in the last two interactions, w_6 with its values given in the first two interactions. Hence, for each interaction we get equations with one waiting time variable. Lastly, by simple transitivity from the latter obtained equations and the previous equations $W1$ and $W2$, we deduce a disjunction of four constraints on $b1, b2, \delta_3$:

$$\begin{array}{l}
\delta_3 \in [0, 8[\quad \wedge \quad +\pi_{12} \geq 2 \quad \wedge \quad \pi_{12} \leq 13 \\
\delta_3 + \pi_{12} \leq 4 \quad \wedge \quad 2 + \pi_{12} \in [-11, 0] \\
\delta_3 \geq 8 \quad \wedge \quad 2 + \pi_{12} - 2 \geq 0 \quad \wedge \quad \pi_{12} \geq 13 \\
\delta_3 + \pi_{12} \geq 4 \quad \wedge \quad \pi_{12} \leq 0
\end{array}$$

Notice that for the hardware-level scheduler, some additional (mapping) interactions may be added such as the sequentialization of two initially parallel tasks. They are expressed in the same way as the dependency-based interactions.

4 Constraints synthesis

In the following, we show how to apply the constraints synthesis technique presented in [24] to hierarchical tasks subject to deadlines for synthesizing a software-level scheduler, independently from hardware. This scheduler consists of a set of constraints[3] on the begin-times of each task executions such that the timing requirements of all tasks are satisfied.

4.1 Description

The technique presented in [24] computes constraints for concurrent *loops*. Loops bodies are described as inter-dependant timed automata[4] whose states correspond to computations.

Requirements A task has a local deadline as a requirement on its end time. For instance, deadline of a *while* task is the deadline of each iteration of loop starting from the begin time of this iteration.

Output The technique produces constraints on the begin times of tasks and constraints on the execution times of tasks whose execution times are unknown. These constraints are solved using integer linear programs *ILP*.

4.2 Application to tasks

4.2.1 Description

The technique presented in [24] computes constraints for concurrent real-time *loops*. Loops bodies are described as inter-dependant timed automata whose states correspond to computations. The following are the key ideas of how the technique is applied to timed automata representing loops bodies.

– For each (body of) loop, *propagate* the requirement and express it as a constraint on the waiting time (w's) of loop computations and unknown execution times variables (δ's). The result of this propagation is the propagation relation denoted by $\Psi(w, \delta)$.

[3]These constraints define a partial order whereas the total order is defined by the mapping step.

[4]The motivations for timed automata models are their ability to express timing constraints between events very naturally, and the growing interest in their applicability to both qualitative and quantitative synthesis.

– For each (body of) loop, compute a constraint on begin times (b's) of loop iterations which guarantees the constraint $\Psi(w, \delta)$. To do that, the waiting time of a loop computation is *expressed* by a relation between the begin time of that computation and the computations it depends on. This interaction relation is denoted $\Phi(b, w, \delta)$. Then using the constraint on that waiting time given by $\Psi(w, \delta)$, and the relation $\Phi(b, w, \delta)$, constraints are *inferred* on variables b and δ. These scheduler constraints are denoted $\Delta(b, \delta)$.

The constraints $\Delta(b, \delta)$ are consistent by construction. Indeed, Φ characterizes the set of interactions consistent with the waiting times objects of Ψ.

4.2.2 *Preliminary example*

We describe in this example the main steps of the constraints synthesis. We use two automata A_P and A_Q depicted in Fig. 5 to illustrate the technique. The indexed arrows indicate dependencies between iterations of computations at begins and ends of arrows.

We denote by σ_x the relative end time of the execution of x. For example $\sigma_{p_3} = e_{p_3} - b_{p_1}, \sigma_{q4} = e_{q4} - b_{q_1}$. We denote by $b_{\mathit{first}(A)}$ the begin time of the execution of the first computation of automaton A. We denote by $\mathit{Prd}(q)$ the set of computations in other automata than q's automaton and on which q is dependant.

Computing Ψ_i This step computes a set of constraints denoted W_i on the waiting times variables, w's variables, for each automaton separately, of the form $X_i \Leftarrow W_i \wedge I_i$ where X_i is the result of the backward constraints propagation on the automaton A_i starting from the *last* state, W_i is the constraint on the w's variables of A_i and I_i is an invariant of A_i. Notice that W_i contains only the w's. The I_i formula may contain the w's b's and e's. Computations of X_P, W_P and I_P for automaton A_P are explained in Table 1. A_P has no conditional branch hence W_P is the sufficient and necessary condition on waiting times, i.e., $X_p \Leftrightarrow W_P \wedge I_P$. Notice that in Table 1, b's and e's are contained in σ's. Constraints propagation process for A_Q is done similarly.

Computing Φ_i It's a relation which links uncontrollable variables, i.e., the w's of A_i, and control points, i.e., the activation times $b_{\mathit{first}(A_i)}$. This relation characterizes the possible interactions of A_i with its environment. For instance, A_Q has two execution paths corresponding

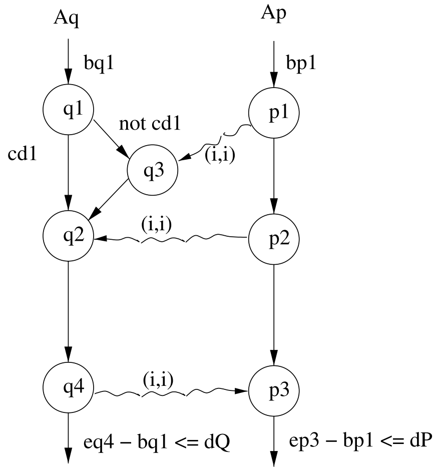


Fig. 5 Concurrent automata A_P and A_Q corresponding to two loops bodies

Table 1 Details of the constraints propagation process on P

Constraints propagation on P:

$$X_P \equiv \begin{array}{l} (\sigma_{p_3} \le d_P \wedge \sigma_{p_3} = \sigma_{p_2} + \delta_{p_3} + w_{p_3}) \\ \wedge (\sigma_{p_2} \le d_P - \delta_{p_3} - w_{p_3} \wedge \sigma_{p_2} = \sigma_{p_1} + \delta_{p_2}) \\ (\sigma_{p_1} \le d_P - \delta_{p_3} - w_{p_3} - \delta_{p_2} \wedge \sigma_{p_1} = \delta_{p_3}) \end{array}$$

X_P can be expressed as a conjunction of an invariant of P, I_P, and a constraint on w's, W_P, as follows:

$$\begin{aligned} X_P &\equiv (\sigma_{p_1} \le d_P - \delta_{p_3} - w_{p_3} - \delta_{p_2}) \wedge I_P \\ &\equiv (w_{p_3} \le d_P - \delta_{p_3} - \delta_{p_2} - \delta_{p_1}) \wedge I_P \end{aligned}$$

$$I_P \equiv \begin{pmatrix} (\sigma_{p_3} = \sigma_{p_2} + \delta_{p_3} + w_{p_3}) \\ \wedge (\sigma_{p_2} = \sigma_{p_1} + \delta_{p_2}) \\ (\sigma_{p_1} = \delta_{p_3}) \end{pmatrix}$$

$$W_P \equiv (w_{p_3} \le d_P - \delta_{p_3} - \delta_{p_2} - \delta_{p_1})$$

Finally, we obtain Ψ_P. Notice that since P does not contain a branch, W_P is a necessary and sufficient property on w's, that is, no need to compute the weakest precondition:

$$X_P \Leftrightarrow W_P \wedge I_P = \Psi_P$$

to the conditions $cond_1 = cd_1$ and $cond_2 = \overline{cd}_1$. This causes two possible interactions of A_P with A_Q, that is, $\Phi_Q^{cond_1}(w_{q_2}, b_{q_1}, b_{p_1})$ when $cond_1$ is true and $\Phi_Q^{cond_2}(w_{q3}, w_{q2}, b_{q_1}, b_{p_1})$ when $cond_2$ is true, and two interactions of A_P with A_Q, that is, $\Phi_P^{cond_1}(w_{p_3}, b_{q_1}, b_{p_1})$ when $cond_1$ is true and $\Phi_P^{cond_2}(w_{p_3}, b_{q_1}, b_{p_1})$ when $cond_2$ is true. $\Phi_i^{cond_j}$ is computed as follows: for each state q of the automaton A with a non empty set $Prd(q)$, and which is executed when $cond_j$ holds, we compute $\phi(w_q)$: $w_q = \max_{q' \in Prd(q)} w_{q'}$ where $w_{q'}$ satisfies relation:

$$\begin{pmatrix} (\sigma_{q'} + b_{first(A)} > \sigma_q + b_{first(A')}) \\ \wedge \\ (w_{q'} = 0) \end{pmatrix} \vee \begin{pmatrix} (\sigma_{q'} + b_{first(A)} \le \sigma_q + b_{first(A')}) \\ \wedge \begin{pmatrix} w_{q'} = \sigma_q - \sigma_{q'} \\ + b_{first(A')} \\ - b_{first(A)} \end{pmatrix} \end{pmatrix}$$

In the relation above, A' is the automaton to which q' belongs and σ_q is the relative end time of q when condition $cond_j$ holds. Expression w_q may contain some unknown waiting times variables in which case they are recursively computed in function of appropriate control points under $cond_j$. Finally, $\Phi_i = \bigvee_j \Phi_i^{cond_j}$. The details of this calculation process are also illustrated on example of Fig. 5: relations $\Phi_P^{cd_1}$ and $\Phi_P^{\overline{cd}_1}$, $\Phi_Q^{cd_1}$ and $\Phi_Q^{\overline{cd}_1}$ are given in

Table 2 Interactions of Q

Sub-interactions of Q with P when the branch condition cd_1 holds:

$$\Phi_Q^{cd_1} \equiv \left(\vee \begin{array}{l} (\pi_{q_1 p_1} + \sigma_{q_1} > \sigma_{p_2} \wedge w_{q_2} = 0) \\ (\pi_{q_1 p_1} + \sigma_{q_1} \leq \sigma_{p_2} \wedge w_{q_2} = \pi_{q_1 p_1} + \delta_{p_1} + \delta_{p_2} - \delta_{q_1}) \end{array} \right)$$

Sub-interactions of Q with P when cd_1 is false:

$$\Phi_Q^{\overline{cd}_1} \equiv \left(\wedge \begin{array}{l} \left(\vee \begin{array}{l} (\pi_{q_1 p_1} + \sigma_{q_3} > \sigma_{p_2} \wedge w_{q_2} = 0) \\ (\pi_{q_1 p_1} + \sigma_{q_3} \leq \sigma_{p_2} \wedge w_{q_2} = \pi_{q_1 p_1} + \delta_{p_1} + \delta_{p_2} - \delta_{q_1} - \delta_{q_3} - w_{q_3}) \end{array} \right) \\ \left(\vee \begin{array}{l} (\pi_{q_1 p_1} + \sigma_{q_1} > \sigma_{p_1} \wedge w_{q_3} = 0) \\ (\pi_{q_1 p_1} + \sigma_{q_1} \leq \sigma_{p_1} \wedge w_{q_3} = \pi_{q_1 p_1} + \delta_{p_1} - \delta_{q_1}) \end{array} \right) \end{array} \right)$$

Table 3 Interactions of P and the global interactions of both P and Q

Sub-interactions of P with Q when the branch condition cd_1 holds:

$$\Phi_P^{cd_1} \equiv \left(\vee \begin{array}{l} (\pi_{q_1 p_1} + \sigma_{q_4} < \sigma_{p_2} \wedge w_{p_3} = 0) \\ (\pi_{q_1 p_1} + \sigma_{q_4} \geq \sigma_{p_2} \wedge w_{p_3} = -\pi_{q_1 p_1} + \delta_{q_1} + \delta_{q_2} + \delta_{q_4} + w_{q_2} - \delta_{p_1} - \delta_{p_2}) \end{array} \right)$$

$$\Phi^{cd_1} \equiv \left(\vee \begin{array}{l} (\pi_{q_1 p_1} + \sigma_{q_4} < \sigma_{p_2} \wedge w_{p_3} = 0) \\ (\pi_{q_1 p_1} + \sigma_{q_4} \geq \sigma_{p_2} \wedge \pi_{q_1 p_1} + \sigma_{q_1} > \sigma_{p_2} \wedge w_{p_3} = -\pi_{q_1 p_1} + \delta_{q_1} + \delta_{q_2} + \delta_{q_4} - \delta_{p_1} - \delta_{p_2}) \\ (\pi_{q_1 p_1} + \sigma_{q_4} \geq \sigma_{p_2} \wedge \pi_{q_1 p_1} + \sigma_{q_1} \leq \sigma_{p_2} \wedge w_{p_3} = \delta_{q_2} + \delta_{q_4}) \end{array} \right)$$

Sub-interactions of P with Q when cd_1 is false:

$$\Phi_P^{\overline{cd}_1} \equiv \left(\vee \begin{array}{l} (\pi_{q_1 p_1} + \sigma_{q_4} < \sigma_{p_2} \wedge w_{p_3} = 0) \\ (\pi_{q_1 p_1} + \sigma_{q_4} \geq \sigma_{p_2} \wedge w_{p_3} = -\pi_{q_1 p_1} + \sum_i \delta_{q_i} + w_{q_3} + w_{q_2} - \delta_{p_1} - \delta_{p_2}) \end{array} \right)$$

$$\Phi^{\overline{cd}_1} \equiv \left(\vee \begin{array}{l} (\pi_{q_1 p_1} + \sigma_{q_4} < \sigma_{p_2} \wedge w_{p_3} = 0) \\ (\pi_{q_1 p_1} + \sigma_{q_4} \geq \sigma_{p_2} \wedge \pi_{q_1 p_1} + \sigma_{q_1} > \sigma_{p_1} \wedge \pi_{q_1 p_1} + \sigma_{q_3} > \sigma_{p_2} \wedge w_{p_3} = -\pi_{q_1 p_1} + \sum_i \delta_{q_i} - \delta_{p_1} - \delta_{p_2}) \\ (\pi_{q_1 p_1} + \sigma_{q_4} \geq \sigma_{p_2} \wedge \pi_{q_1 p_1} + \sigma_{q_1} \leq \sigma_{p_1} \wedge \pi_{q_1 p_1} + \sigma_{q_3} > \sigma_{p_2} \wedge w_{p_3} = \delta_{q_2} + \delta_{q_3} + \delta_{q_4} - \delta_{p_2}) \\ (\pi_{q_1 p_1} + \sigma_{q_4} \geq \sigma_{p_2} \wedge \pi_{q_1 p_1} + \sigma_{q_1} > \sigma_{p_1} \wedge \pi_{q_1 p_1} + \sigma_{q_3} \leq \sigma_{p_2} \wedge w_{p_3} = \delta_{q_2} + \delta_{q_4}) \\ (\pi_{q_1 p_1} + \sigma_{q_4} \geq \sigma_{p_2} \wedge \pi_{q_1 p_1} + \sigma_{q_1} \leq \sigma_{p_1} \wedge \pi_{q_1 p_1} + \sigma_{q_3} \leq \sigma_{p_2} \wedge w_{p_3} = \pi_{q_1 p_1} - \delta_{q_1} + \delta_{q_2} + \delta_{q_4} + \delta_{p_1}) \end{array} \right)$$

Table 3 for P and Table 2 respectively where symbol $\pi_{q_1 p_1}$ denotes $b_{q_1} - b_{p_1}$ and $\Phi_{cd_1} = \Phi_P^{cd_1} \wedge \Phi_Q^{cd_1}$. and $\Phi_{\overline{cd}_1} = \Phi_P^{\overline{cd}_1} \wedge \Phi_Q^{\overline{cd}_1}$.

Execution constraints Execution constraints of automaton A_i, named $\Delta_i(b_{first(A_i)}, \langle b_{first(A_j)} \rangle_{j \in Env(A_i)})$, are obtained by mean of quantifiers $w's$ elimination. For conditional choices $cond_j$ we compute conjunctions over execution combination choices: $\Delta_i = \bigwedge_{C_j} \exists w \; (\Phi_j(\langle b_{first(A_k)} \rangle) \wedge \Psi_i(\langle w \rangle))$. Finally, for (non-conditional) execution choices we compute disjunctions over execution combination choices: $\Delta_i = \bigvee_{C_k} \Delta_{i_k}$. On the example of A_P and A_Q and when considering unit execution times for computations, i.e., $\delta_{q_i} = 1$, and deadlines equal to 10 units, $d_P = d_Q = 10$, the two generated constraints after removing redundancy are $(b_{q_1} + 2 \geq b_{p_1})$ and $(b_{p_1} + 5 \geq b_{q_1})$.

4.2.3 Application to tasks

We describe how to apply the technique to tasks. For this, we show the timed automata models of tasks on which it is applied on.

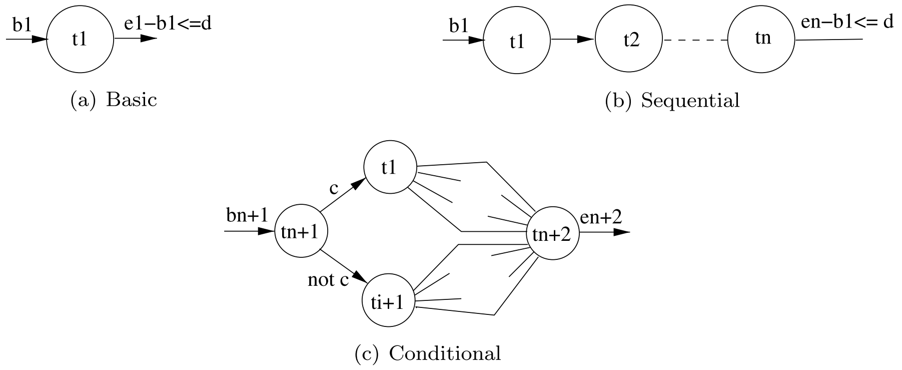


Fig. 6 Tasks to automata

Basic blocks, sequentials and conditionals Applying it to basic and sequential nodes is straight forward and is done as for loop bodies: general forms of corresponding timed models are shown in Figs. 6(a), 6(b), 6(c).

Loops The timed models of *while*, *for* and dependency-free *forall* tasks are generated for the corresponding loops bodies and the synthesis is done in a similar way for all of them. As for inter-dependent *forall* nodes, they are handled in the same way as the parallel task *par*.

Or Task *Or* describes a choice between several sub-tasks computations. As much timed model as number of choice are generated. Therefore, this technique consists of generating constraints in one of the following forms:

– $\bigwedge \Delta(b_{ti}, \delta_j)$
– $\bigvee_{C_k} (\bigwedge \Delta(b_{ti}, \delta_j) \bigwedge\{b_t = \bot, t \notin C_k\})$

$\bigwedge$ and $\bigvee$ denote conjunctions and disjunctions, respectively. The set of variables δ_j is the set of unknown execution times; each set C_k is a set of choices (for *Or* tasks); $\bigwedge \Delta(b_{ti}, \delta_j) \bigwedge\{b_t = \bot, t \notin C_k\}$ are the corresponding constraints where $b_t = \bot, t \notin C_k$ indicate that computations of each task t which does not belong to C_k are not executed. The software scheduler is thus either a solution of an integer linear system, *IL*, or a set of pairs of integer linear systems with the set of corresponding choices, $\langle IL_k, C_k \rangle$.

Parallels Some parallel tasks may be removed since all top level tasks are implicitly parallel,[5] and imbrication of multiple parallel task may be transformed to a single parallel. Therefore, after performing the multiple to single parallel task transformation if any, the synthesis technique for a parallel task is done as follows:

– (a) Apply the synthesis while unexpanding the parallel task, i.e., while the parallel task is treated as a basic block;
– (b) Apply recursively the synthesis on sub-tasks of parallel task by considering the possible constraints on that task in (a) as a requirement. The composition of the obtained constraints and the ones of (a) is the conjunction of the two.

This will be illustrated by the example given in Sect. 5.

[5] Top level tasks behave as if they were combined by a parallel task.

4.2.4 Complexity

For a given combination of sub-tasks choices, C_i:

– We denote by k_i the number of disjoints sets of dependencies or hyper-dependencies, such that each set contains all the dependencies or hyper-dependencies with a same source number, say it is equal to n_s, and a same target number, say it is equal to n_d.
– For each set D_j above, we denote l_j its cardinality.

Thus, the number of generated constraints for the combination C_i is at most the product of the number of generated constraints for each set D_j. Since there are k_i such set, this number is

$$\prod_{j=1}^{k_i} n_d(n_s+1)^{l_j}$$

For instance, 5 and 3 linear constraints are generated for the Fig. 5 for conditions $cond_1$ and $cond_2$, respectively, as shown in 3. By summing over the combinations choices, we obtain a bound on the total generated linear constraints:

$$\sum_{i=1}^{I}\prod_{j=1}^{k_i} n_d(n_s+1)^{l_j}$$

Where I is the number of tasks *Or* multiplied by the number of their execution choices.

Each constraint may have at most N $w's$ variables each of whom may depend on less than N $w's$ variables; therefore $w's$ elimination is performed in at most N^2 operations, where N is the tasks number.

Finally, the overall complexity is dominated by

$$N^2 \times I \times \prod_{j=1}^{k} n_d(n_s+1)^{l_j}$$

Where k is the maximal number, over all possible execution choices, of sets of dependencies or hyper-dependencies whose, as explained before, sources number is equal to n_s and target number is equal to n_d, i.e., $k = \max_{i=1}^{I} k_i$.

We notice that the latter complexity is determined by the following factors:

1. The sum of the multiplications of the choice tasks number by their number of choices (from their sub-tasks), I.
2. Exponents of the number of inter-tasks dependencies. Indeed the relation Φ is potentially disjunctive. Hollow times due to the latencies for synchronization of the tasks computations may intervene in the calculation of the relation Φ. This depends on end times and begin times of one or more source tasks computations and their target computations, respectively. Thus the number of generated systems of linear constraints is determined by a product over these dependencies as previously explained.
3. The quadratic number of tasks.

By choosing a small partition, the synthesis complexity may be reduced, Such partition may be a set of (a) hierarchical tasks whose sub-tasks have the same constraints, or (b) hierarchical tasks whose sub-tasks are ignored in the scheduler synthesis and thus whose

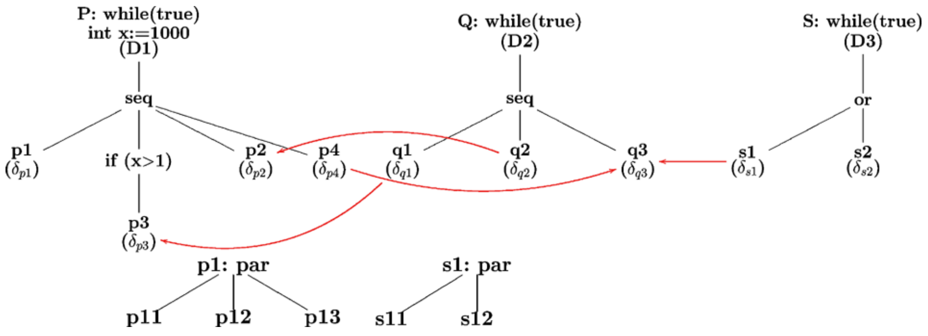


Fig. 7 Three tasks P, Q and S and expanded views for sub-tasks $p1$ and $s1$

scheduler is not computed. As an example if sub-tasks $(t1, \ldots, tn)$ of T are not involved in any dependency with any other task, then begin times of tasks $(t1, \ldots, tn)$ depend only on begin time of T. Hence, sub-tasks $(t1, \ldots, tn)$ may be put inside the same element in the partition. This element may be T or any other containing task verifying this condition.

The example of Fig. 7 contains twelve simple tasks. Partitions are $\{P, Q, S\}$ $\{p_{11}, p_{12}, p_{13}, Q, S\}$ … $\{P, Q, s_1, s_2\}$. The maximal partition $\mathcal{P}$, i.e., the partition with maximal cardinality, for this set of tasks when applying partitioning (a) is composed of eleven tasks, $\mathcal{P} = \{p_{11}, p_{12}, p_{13}, p_3, p_2, p_4, q_1, q_2, q_3, s_1, s_2\}$ because only sub-tasks s_{11}, s_{12} are not subject to any dependency and hence the synthesis results in them having the same constraints as s_1. For simplicity, the dependencies between sub-tasks of $p1$ are not presented in this figure.

4.3 Example

Let us consider the example of Fig. 7 with deadlines data $D1 = 10$, $D2 = 10$, and $D3 = 5$; and following worst case execution times: $\delta_{p_2} = 1$, $\delta_{p_3} = 1$, $\delta_{p_4} = 1$, $\delta_{q_1} = 1$, $\delta_{q_2} = 4$, $\delta_{q_3} = 1$, $\delta_{s_1} = 1$, $\delta_{s_2} = 1$; while WCET of p_1 is unknown.

We obtain the set of constraints C of Fig. 8 where $x = b_{p_1} - b_{q_1}$ and $y = b_{s_1} - b_{q_1}$ are time distances between tasks p_1 q_1 and s_1.

How to compute these constraints? The constraints shown in Fig. 8 are obtained in the same way described in examples of Sects. 3.2 and 4.2.2. That is:

1. We first compute the constraints propagation equations,
2. Then we compute the interaction equations, and
3. Finally, by variables substitution and transitivity of waiting times constraints we deduce the shown set of constraints on δ_{p_1}, b_{p_1}, b_{q_1} and b_{s_1}.

As an example, when s_2 is executed ($b_{s_1} = \perp$), and when the scheduling of q_1 and p_1 verify $x = b_{q_1} - b_{p_1} \le 2$ which means that time distance between hierarchical tasks Q and P is smaller than or equal to 2, then the constraint of execution time of p_1 is: $\delta_{p_1} = \max(\delta_{p_{11}}, \delta_{p_{12}}, \delta_{p_{13}}) \in [0, 6]$.

When s_2 is executed

$$\begin{cases}
(1)\ \delta_{p_1} >= 5 + x \wedge \delta_{p_1} <= 6 \\
(2)\ \delta_{p_1} >= 4 + x \wedge \delta_{p_1} <= 5 \wedge x \leq 2 \\
(3)\ \delta_{p_1} >= 1 + x \wedge \delta_{p_1} <= 4 \wedge x \leq 2 \\
(4)\ \delta_{p_1} >= 0 \qquad \wedge \delta_{p_1} <= 1
\end{cases}$$

When s_1 is executed

$$\begin{cases}
(5)\ \ \delta_{p_1} >= x - y - 2 \wedge \delta_{p_1} >= x + 5 \wedge \delta_{p_1} <= x + 6 \wedge \delta_{p_1} <= 7 \\
(6)\ \ \delta_{p_1} >= x - y - 2 \wedge \delta_{p_1} >= x + 4 \wedge \delta_{p_1} <= x + 5 \wedge \delta_{p_1} <= 7 \wedge x <= 2 \\
\qquad \wedge\, y >= -7 \\
(7)\ \ \delta_{p_1} >= x - y - 2 \wedge \delta_{p_1} >= x + 4 \wedge \delta_{p_1} <= x + 5 \wedge \delta_{p_1} <= 7 \wedge x <= 2 \\
\qquad \wedge\, y >= -8 \wedge y <= -6 \\
(8)\ \ \delta_{p_1} >= x - y - 2 \wedge \delta_{p_1} >= x + 4 \wedge \delta_{p_1} <= x + 5 \wedge \delta_{p_1} <= 7 \wedge x <= 2 \\
\qquad \wedge\, y >= -7 \wedge y <= -4 \\
(9)\ \ \delta_{p_1} >= x - y - 2 \wedge \delta_{p_1} <= x + 5 \wedge \delta_{p_1} <= 7 \wedge y >= -8 \wedge y <= -4 \\
(10)\ \delta_{p_1} >= x - y - 2 \wedge \delta_{p_1} >= x + 4 \wedge \delta_{p_1} <= x + 5 \wedge \delta_{p_1} <= 7 \wedge x <= 2 \\
\qquad \wedge\, y >= -8 \wedge y <= -6 \\
(11)\ \delta_{p_1} >= x + 1 \wedge \delta_{p_1} <= x + 4 \wedge x <= 2 \wedge y >= -6 \\
(12)\ \delta_{p_1} >= x + 1 \wedge \delta_{p_1} <= x + 4 \wedge x <= 2 \wedge y >= -7 \wedge y <= -6 \\
(13)\ \delta_{p_1} >= x + 1 \wedge \delta_{p_1} <= x + 4 \wedge x <= 2 \wedge y >= -8 \wedge y <= -7 \\
(14)\ \delta_{p_1} <= x + 1 \wedge \delta_{p_1} <= 5 \wedge x <= 2 \wedge y >= -6 \\
(15)\ \delta_{p_1} >= x + 1 \wedge \delta_{p_1} <= 5 \wedge x <= 2 \wedge y >= -7 \wedge y <= -6 \\
(16)\ \delta_{p_1} >= x + 1 \wedge \delta_{p_1} <= 5 \wedge x <= 2 \wedge y >= -8 \wedge y <= -7
\end{cases}$$

Fig. 8 The sets of constraints for the example of Fig. 7

5 Hardware-level scheduler

The set of synthesized constraints in Sect. 4 constitutes a set S of linear constraints systems over the begin times and execution times of tasks. To compute the hardware-level scheduler, we first define the software-to-hardware mapping functions of the components. Then either these systems remain unchanged in the case where tasks are mapped on different (groups) of processors, and are composed with mapping constraints; or the interaction constraints are resynthesized and systems are recomputed:

– Either the systems remain unchanged in the case where tasks are mapped to different groups of processors. In this case, the running hardware platform does not limit the tasks parallelism and may offer more. Therefore some tasks may be expanded to a par (or forall) combinations to exploit these groups. And the systems have to composed with the computed corresponding hardware-level constraints, in a recursive way as explained earlier.
– Either the interaction constraints have to be synthesized again. It is done in the same way but on a new timed model which considers the limitation of platform parallelism reflected by the defined mapping. For instance, the mapping may add new dependencies between tasks or transform some parallel combinations to sequential ones (dependency free forall to for, forall with dependent iterations to sequence of for, or par to seq).

Then, the final solutions are obtained by choosing the early begin times of tasks. For that we add the objective function $\min \sum_i b_{t_i}$.

Table 4 Software and hardware level schedulers for the example of Fig. 7

	case (a)	case (b)
Software-level scheduler	S	
Hardware-level scheduler	$\begin{cases} \min\ f \\ S_{HW} \end{cases}$	$\begin{cases} \min f \\ b_{p_1} + D1 > b_{q_1} \\ S \end{cases}$ $\begin{cases} \min f \\ b_{q_1} + D2 > b_{p_1} \\ S \end{cases}$

As an example, let us consider the example of Fig. 7, with the following three cases. In the first case, the mapping on three processors one for each task P, Q and S. This case does not require additional mapping constraints. However for the second case only two processors are used, one for P and Q, and one for S, the mapping constraints have to be composed with the software-level scheduler. The constraint Φ_{HW} on end times and begin times of P and Q, expressing that P and Q share one processor, is a disjunction:

$$b_{p_1} + \delta_{p_1} + \delta_{p_2} + \delta_{p_3} + w_{p_3} + w_{p_2} > b_{q_1} \quad \bigvee \quad b_{q_1} + \delta_{q_2} + \delta_{q_3} + w_{q_3} > b_{p_1}$$

In this case the constraints of the interaction relation Φ is obtained by composition and becomes $S \wedge \Phi_{HW}$ which, let us say, result in the hardware-level scheduler S_{HW}. Then the solutions for the hardware-level scheduler are obtained as before by taking the early begin times and are given in Table 4(a). In this table we denoted f the expression $\sum_i b_{q_i} + \sum_j b_{p_j} + \sum_k b_{s_k}$.

The last case is also when only two processors are used, one for P and Q, and one for S, but P and Q keep the processor during a duration equal to their deadline, respectively. In this case the hardware-level constraints Φ_{HW} do not include waiting time variables and are:

$$b_{p_1} + D1 > b_{q_1} \bigvee b_{q_1} + D2 > b_{p_1}$$

Therefore the hardware-level scheduler is obtained by composition and becomes $S \wedge \Phi_{HW}$ as shown in Table 4(b).

5.1 Efficiency

After the definition of the software-to-hardware mapping function and calculation of the final scheduler, system performance are estimated to evaluate its efficiency. For ease of presentation, we present here this evaluation for the hierarchical sub-task p_1. The sub-tasks of p_1 are depicted in Listing 1.

The hardware architecture properties are depicted in Table 5.

Writing, reading, addition and multiplication latencies are equal to two cycles. In the opposite of the two first transactions, addition and multiplication must be executed on different cycles. References [22, 23] give detailed presentations on the modeling of hardware architectures.

As the sub-tasks of p_1 are not a part of the initial partition, their scheduler is not synthesized. However, the synthesis is compositional and can be applied on the sub-tasks by taking the constraint on p_1 as the execution time deadline to obtain the constraints S_{p1}. Then the

```
p1: par
    p11: while(i>0)  x=add(x0,read(&M[i]));
    p12: while(i>0)  dec(i);  write(&M[i],x);
    p13: while(i>0)  write(&z,mult(x,read(&c)));
```

Listing 1 Sub-tasks of p1

Table 5 Hardware properties

TR	read	write	add	mult	dec
T latencies	2	2	2	2	1
Conflictual *T*	–	–	mult	add	–
Parallel *T*	write, dec	read, dec	read, write	–	read, write

Table 6 Mappings and performance results

Scheduling	P1 Mapping	P2 Mapping	δ_{p_1}	Rate
Sequential	p11, p12, p13	–	6	<0.17
Pipeline	p11, p12	p13	6	0.25

hardware-level scheduler is: $S' = S \wedge S_{p1}$. In this example, it is necessary to take into account latencies due to the cyclic dependencies, not shown in the listing, between tasks, i.e., those due to the task reading the variable i and the task which decrements it on the one hand, and the reading/writing of variables M on the other hand.

Finally, P-WARE [22, 23, 25] is used to evaluate the performance of global scheduling of the system (rate, bandwidth, execution time, etc.). A sequential scheduling has an execution time of six cycles and give a rate of one output every six cycles, whereas pipeline scheduling on two processors P1 and P2 produces a better rate for the same execution time as shown in Table 6.

6 Video encoding SoC

6.1 Description

We present here the results obtained using our methodology for synthesizing a software scheduler for an MPEG-4 video encoder. The video encoder tasks are shown on Figs. 9 and 10. This encoder is based on the one presented in [26] and supports slicing.

The video encoding SoC functions as follows: first of all the images of the camera CAM are sent to the user interface HI with the format of the camera, that is, RGB. Then the user interface performs certain operations, like zooming on the image, before transmitting it to the display DISP (Fig. 10). In parallel to displaying, the other tasks, the format converter FC and a video encoder VE, runs on the processors. They convert then encode the 640 × 480 images. FC is in charge of transforming the images to the entry format of VE, that is, YUV, before transmitting them to VE so that they are encoded in an MPEG-4 format on the cake multiprocessor hardware [2] shown in Fig. 11.

The objective for this application is to use our methodology for finding an implementation which meets the application requirements with a good bandwidth consumption on Cake

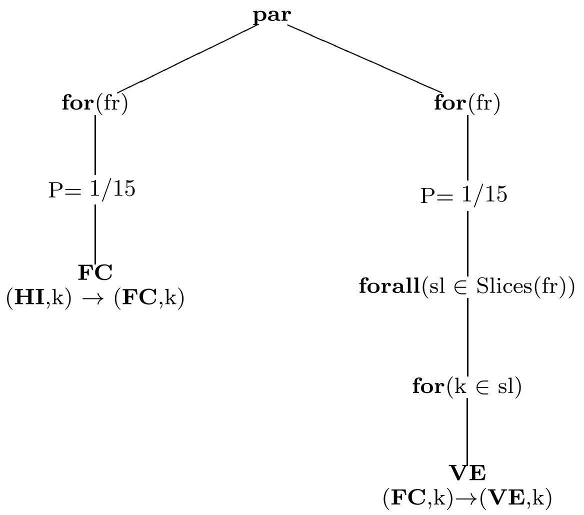


Fig. 9 Two simple and hierarchical sub-tasks of the encoder FC and VE. *fr* is a frame and *sl* is a slice of macroblocks

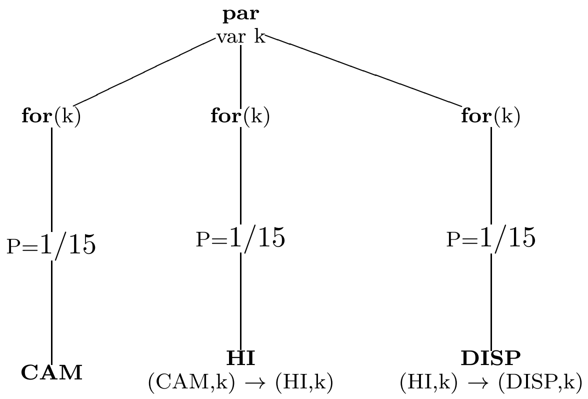


Fig. 10 Three simple tasks of period *P* interacting with the encoder

SoCs. Degree of freedom is the type of software parallelization (data and task-level parallelism) and the number of processors in each of the interconnected Cake SoCs (Fig. 11).

6.2 Constraints synthesis

Let us now see, on this application, how the step of constraints synthesis of our methodology functions. The synthesis takes into account the requirements of the application, that is, the periods of task activations, and hardware and software tasks interactions.

These interactions consist of dependencies on data stored in memory buffers as depicted in Fig. 12. The timing requirements for the input model are the periods of the loops (1/15) for the software tasks FC, and VE; and 1/30, which is half of the periods, for the hardware tasks CAM, HI, and DISP.

The synthesis procedure derives a scheduler Fig. 13 and a constraint on the execution times of the software (5). In these figures, begin and execution times of a task X are indicated by notations b_X and δ_X, respectively.

As an example, in this scheduler, the first constraint stipulates that the interface must begin its execution after the beginning of the camera by one thirtieth of seconds.

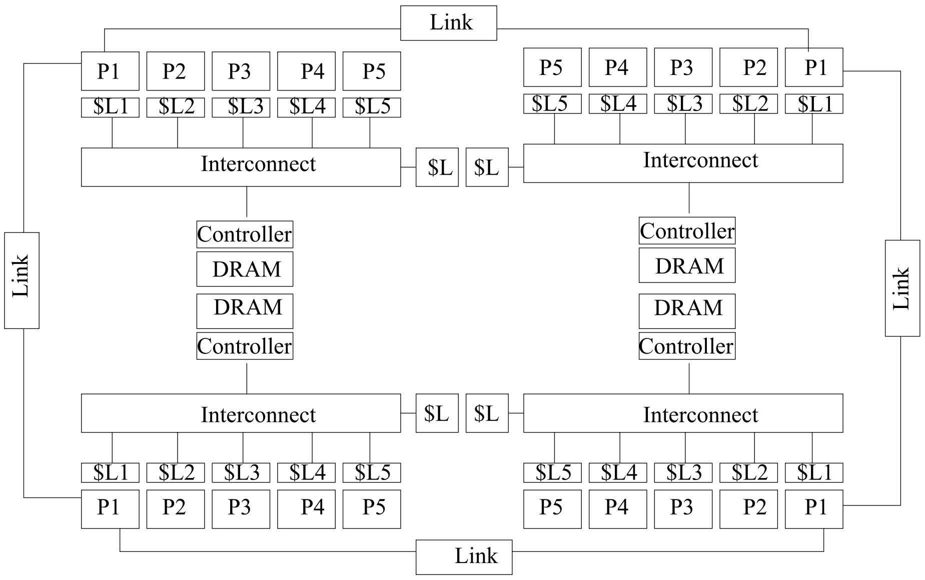


Fig. 11 4-tiles cake configuration

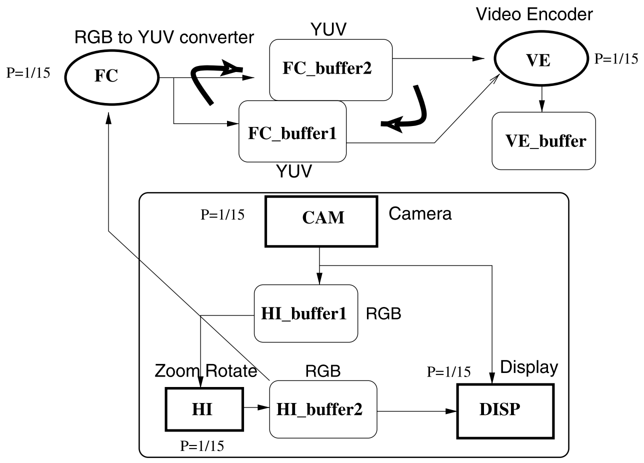


Fig. 12 Software and hardware environment interactions

On the other hand, the synthesis also derives a constraint on the execution times of FC and VE which were regarded as parameters in the model:

$$(5) \quad \delta_{\mathrm{FC}} + \delta_{\mathrm{VE}} \leq \frac{1}{15} \quad \text{Constraint on exec times of VE and FC to be met}$$

This constraint fixes the condition under which the scheduler respects the requirements of the model. Equation (5) is the execution times constraint.

$$\begin{cases} (1)\ b_{\mathrm{HI}} = b_{\mathrm{CAM}} + \frac{1}{30} & \text{HI activated } \frac{1}{30} \text{ after CAM} \\ (2)\ b_{\mathrm{DISP}} = b_{\mathrm{HI}} + \frac{1}{30} & \text{DISP activated } \frac{1}{30} \text{ after HI} \\ (3)\ b_{\mathrm{FC}} = b_{\mathrm{HI}} + \frac{1}{30} & \text{FC activated } \frac{1}{30} \text{ after HI (software scheduler)} \\ (4)\ b_{\mathrm{VE}} = b_{\mathrm{FC}} + \frac{1}{30} & \text{VE activated } \frac{1}{30} \text{ after FC (software scheduler)} \end{cases}$$

Fig. 13 Synthesized scheduler constraints

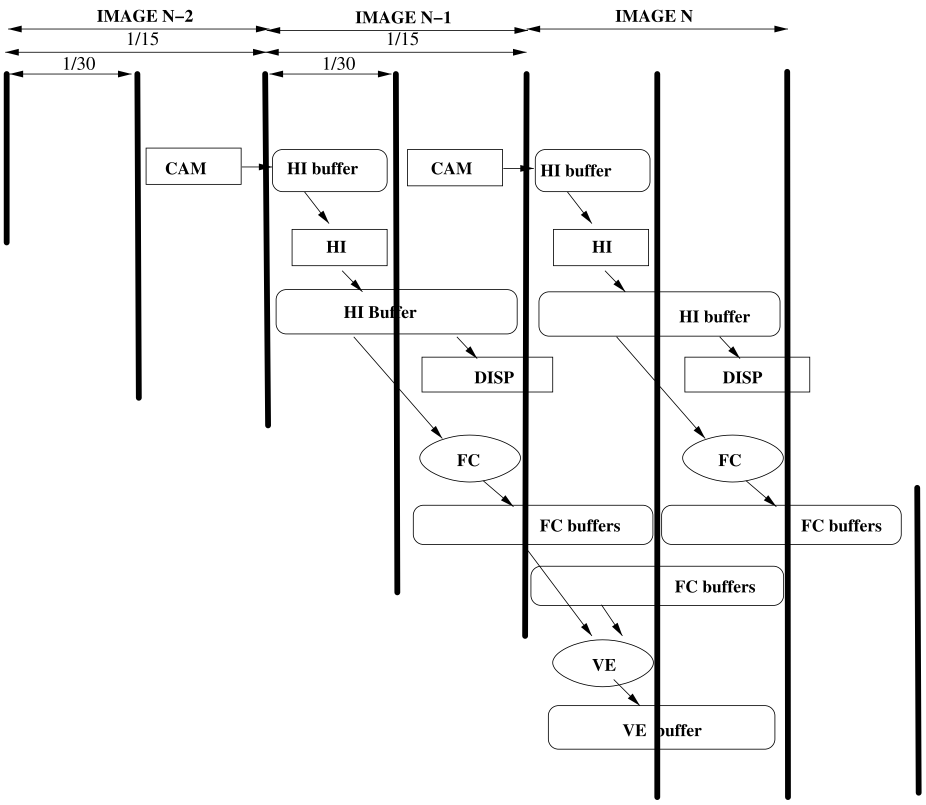


Fig. 14 A feasible scheduling of the application

Let us assume that the execution time of FC is:

$$\delta_{\mathrm{FC}} \leq \frac{2}{3} \times \frac{1}{25}$$

From this constraint, it follows that a deadline on the execution time of the encoder is:

$$\delta_{\mathrm{VE}} \leq \frac{1}{25}$$

Figure 14 shows a scheduling compatible with this set of constraints.

VE is parallelized on the available processors on the Cake SoCs while FC is sequentially run on one of the processors. According the software model VE and FC run in mutual

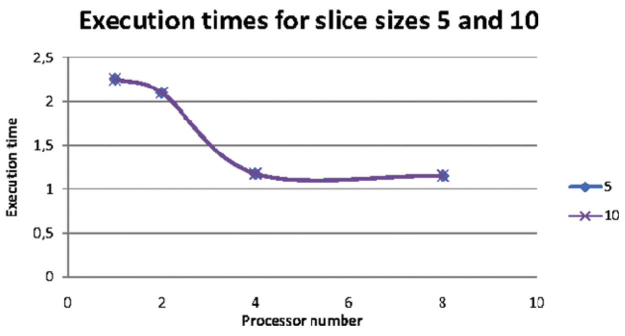


Fig. 15 Execution times for parallel mappings of tasks

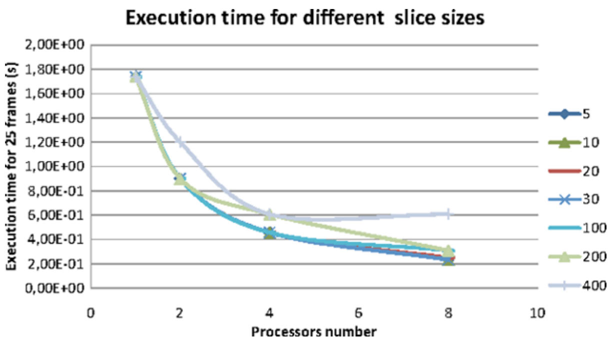


Fig. 16 Execution times for parallel mappings of data slices

exclusion. Moreover we chosed not to expand VE tasks when computing the hardware-level scheduler in order to accelerate mappings exploration. Therefore, the hardware-level scheduler does not introduce new interaction constraints on VE or FC. For the mappings below, it is equal to (3), (4) with constraint (5).

Per tasks parallel mappings consist of mapping one task on one processor and mapping (groups of) different tasks on different processors. Figure 15 shows that these mappings violate the constraint $\delta_{\mathrm{VE}} \leq \frac{1}{25}$ in the case where slice sizes are five or ten macroblocks. The other higher sizes have very similar and overlapping performance results not shown in the figure.

Slice-level parallel mappings consist of mapping independent units composed of one or several slices on different processors. Corresponding threads have the same constraints as VE. Figure 16 shows the execution time results. These mappings provide better results than preceding ones. Indeed, they allow satisfying the scheduler constraint starting from two processors except for slices whose sizes are equal or bigger than four hundred macroblocks.

The data rate and available bandwidths per slice-level mappings are depicted in Fig. 17 where SLS denotes the slices size, i.e., number of macroblocks per slice. These experiments show that the available bandwidth are higher for slices of size one hundred or higher. Also, the data rates are almost the same which indicate a negligible variation in the size of the produced bit stream.

The gain in execution time when using more than four processors is not significant, while the obtained available bandwidths are better for bigger slices. Therefore, the slice-level mapping using one hundred macroblock slices on four processors is the implementation which meets the software encoding requirements and which is the most efficient in term of used bandwidth and produced data rate.

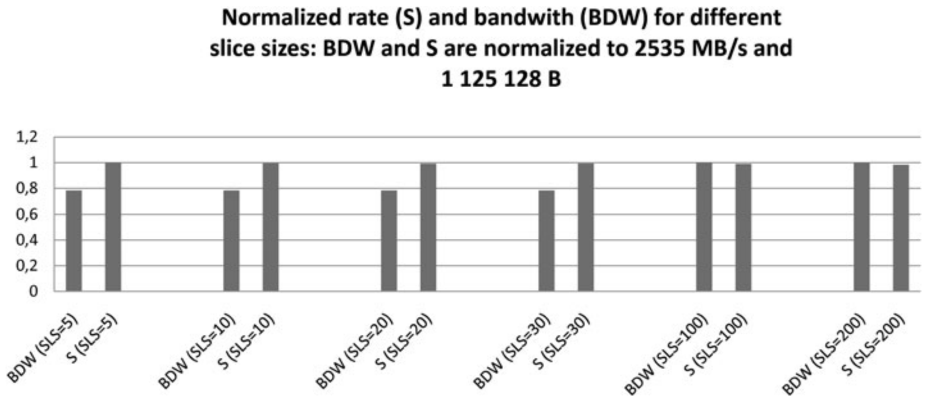


Fig. 17 Hardware bandwidth (BDW) and data rate (S) for different slice-level parallel mappings

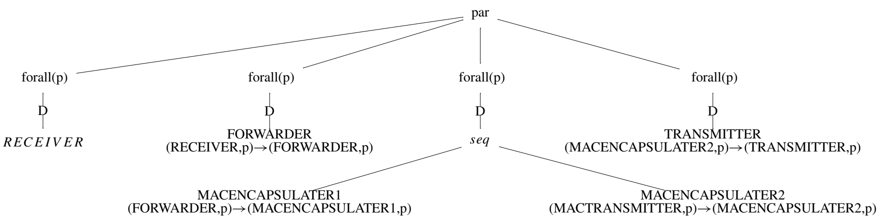

par
forall(p)
D
RECEIVER
forall(p)
D
FORWARDER
(RECEIVER,p)→(FORWARDER,p)
forall(p)
D
seq
MACENCAPSULATER1
(FORWARDER,p)→(MACENCAPSULATER1,p)
MACENCAPSULATER2
(MACTRANSMITTER,p)→(MACENCAPSULATER2,p)
forall(p)
D
TRANSMITTER
(MACENCAPSULATER2,p)→(TRANSMITTER,p)

Fig. 18 IPv4 Data plane tasks

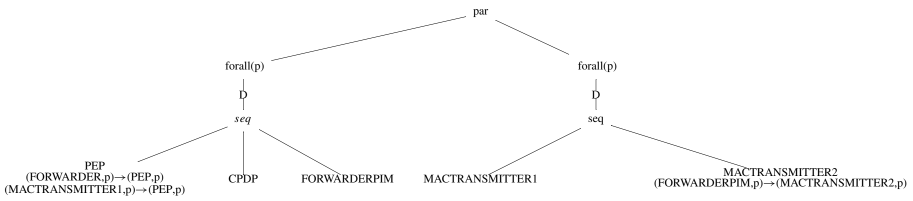


Fig. 19 IPv4 Control plane tasks

The scheduler synthesis approach is very fast with the constraint synthesis for the encoder application taking some seconds, and the simulation speed for design exploration reaching 490000 processor cycles per second at average without exceeding 644000 cycles per second in other cases.

7 IPv4

We show the results of scheduling an IP Version 4 (IPv4) packet forwarder application with three levels lookup [27] on the IXP2800 NP [28] with IXP channels sizing. The target OC-48 data rate for the forwarder application is at least 2.5 Gbs for 53 bytes cells.

Software tasks are depicted in Figs. 18 and 19. To meet the OC-48 requirement above, *forall* tasks have an execution deadline D of 236 cycles.

$$\begin{cases}
(1)\ b_{\text{FORWARDER}} >= b_{\text{RECEIVER}} + \delta_{\text{RECEIVER}} \\
(2)\ b_{\text{FORWARDERPEP}} >= b_{\text{RECEIVER}} + \delta_{\text{RECEIVER}} + \delta_{\text{FORWARDER}} + \delta_{\text{MACENCAPSULATER1}} \\
\qquad\qquad + \delta_{\text{MACTRANSMITTER1}} \\
(3)\ b_{\text{FORWARDERPEP}} - b_{\text{MACTRANSMITTER}} <= 131 - \delta_{\text{MACTRANSMITTER2}} \\
(4)\ b_{\text{MACENCAPSULATER}} >= b_{\text{RECEIVER}} + \delta_{\text{RECEIVER}} + \delta_{\text{FORWARDER}} \\
(5)\ b_{\text{MACTRANSMITTER}} >= b_{\text{RECEIVER}} + \delta_{\text{RECEIVER}} + \delta_{\text{FORWARDER}} + \delta_{\text{MACENCAPSULATER1}} \\
(6)\ b_{\text{MACENCAPSULATER}} - b_{\text{RECEIVER}} \\
\quad <= 2\delta_{\text{RECEIVER}} + 2\delta_{\text{FORWARDER}} + 2\delta_{\text{MACENCAPSULATER}} + 2\delta_{\text{MACTRANSMITTER}}
\end{cases}$$

Fig. 20 Software-level scheduler constraints

$$\begin{cases}
(1)\ b_{\text{FORWARDER}} = b_{\text{RECEIVER}} + \delta_{\text{RECEIVER}} \\
(2)\ b_{\text{FORWARDERPEP}} = b_{\text{RECEIVER}} + \delta_{\text{RECEIVER}} + \delta_{\text{FORWARDER}} + \delta_{\text{MACENCAPSULATER1}} \\
\qquad\qquad + \delta_{\text{MACTRANSMITTER1}} \\
(3)\ b_{\text{MACENCAPSULATER}} = b_{\text{RECEIVER}} + \delta_{\text{RECEIVER}} + \delta_{\text{FORWARDER}} \\
(4)\ b_{\text{MACTRANSMITTER}} = b_{\text{RECEIVER}} + \delta_{\text{RECEIVER}} + \delta_{\text{FORWARDER}} + \delta_{\text{MACENCAPSULATER1}} \\
(5)\ \delta_{\text{MACTRANSMITTER1}} + \delta_{\text{MACTRANSMITTER2}} <= 131
\end{cases}$$

Fig. 21 IPv4 IXP Scheduler

The computed software-level scheduler constraints are given in Fig. 20, with FORWARDERPEP, CPDP and FWPIM having worst case execution times of 49, 28 and 42 respectively.

We considered tasks pipelining with multiprocessing at stage-level. A mapping strategy which is shown to achieve high rates consists of mapping tasks operating on data content of packets (Fig. 18) on distinct microengines while assigning more of them to heavy tasks as follows: 4 microengines for task FORWARDER, 2 for TRANSMITTER, 1 for others, and using the eight parallel threads per microengine. Computation tasks (Fig. 19) are mapped on XSCALE using proper thread for MACTRANSMITTER.

The corresponding scheduler constraints are given in Fig. 21.

The forwarding application is able to meet OC-48 line rates with more than 3 Gb/s throughput as shown in the simulation results in Fig. 22. The best channels sizing is 12 banks: 3 channels with 4 banks in each, resulting in up to 5.7 Gb/s.

8 Practical overview

We have presented the scheduler synthesis methodology and its global context. So far we have left aside some system-level practical aspects, most important of which are specifics of tasks executions, synthesis flow and scheduler execution choices.

8.1 Tasks executions

P-WARE is a modeling and simulation framework that allows designer to specify P-WARE software and hardware components at transaction level according to the previously presented models and evaluate their joint performance.

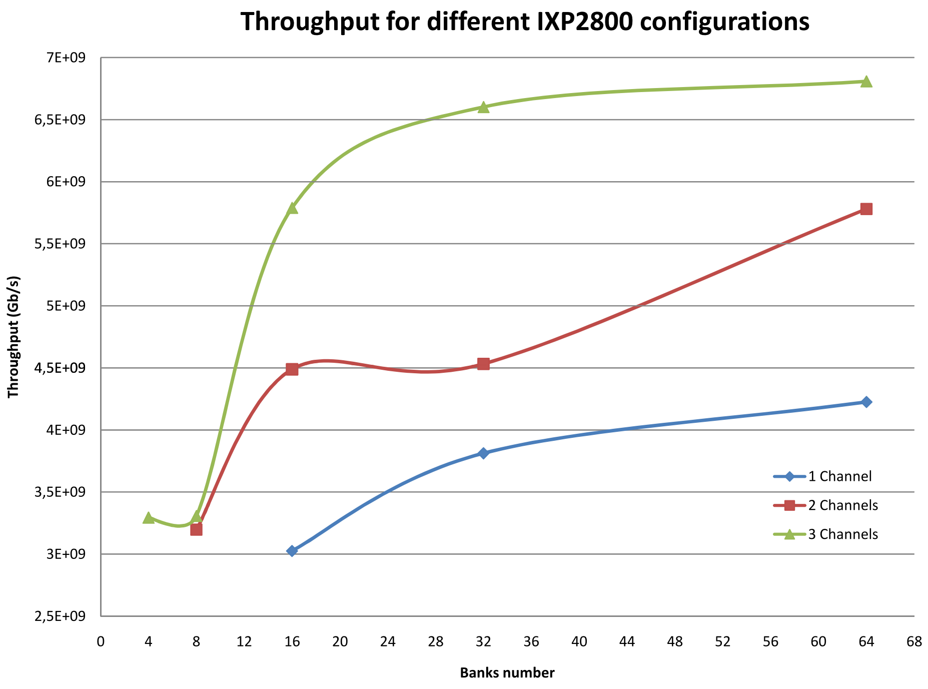


Fig. 22 IPv4 throughput in function of IXP2800 configurations

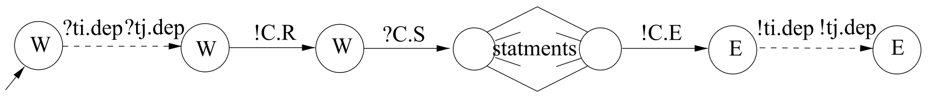


Fig. 23 Tasks interactions

In a practical point of view, to be ready to run onto a given platform, tasks need to be instrumented with supplementary information which is aimed at the hardware platform. This information defines the executions and relates to the sequentialization, parallelization and evaluation of assignments on hardware. Figure 23 depicts the interactions model that is chosen and added to P-WARE tasks in order to define its executions.

A P-WARE task *t* behavior is depicted in Fig. 23. Once all preceding tasks are completed (**?ti.tdep**), and *t* is scheduled by the scheduler (**?C.S**), the task transaction requests are sent to hardware dispatcher.

8.2 Synthesis flow

As depicted in Fig. 24, the idea is as follows. First, the code generation and tasks-to-automata transformation chains generate two outputs, namely the application code and the automata-based timed model of it, both generated from the system model.

The former, rather than calling platform primitives directly, calls a platform-dependent function, implemented on top of the target platform, that performs the appropriate scheduling which is responsible for ensuring that timing constraints are met. For this, the application provides some reflective information such as the task identifier and its current state.

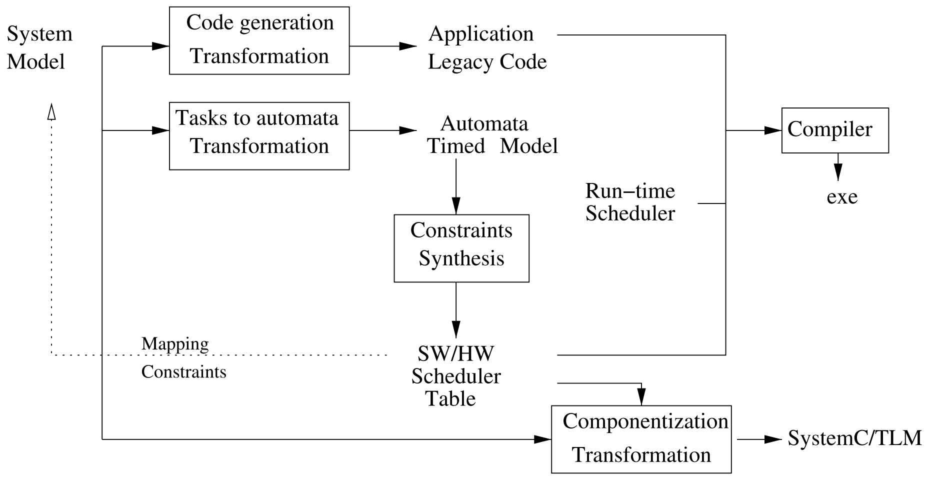


Fig. 24 Synthesis flow

The timed model is fed into the scheduler synthesis, which generates the tasks scheduling table, if a scheduler exists. This table contains tasks constraints and will be used to determine for each task with which the scheduler is called, what actions need to be taken accordingly. Such actions consist in, typically, a sequence of waitings, for either variable conditions corresponding to starts or an terminations of tasks executions, or for timeouts.

Second, a componentization transformation[6] is in charge of producing SystemC/TLM simulation and performance evaluation program from the system model and the synthesized scheduler constraints.

P-WARE TLM components and programs are written in SystemC. P-WARE components observers provide during a simulation for a given observation time interval various performance metrics. These performance metrics are tasks *waiting times*, tasks *execution times*, components *available bandwidth*, *conflicts* and *output rates*. A component idle time is the time during which the arbiter, the controller and the TLTB are all idle. The per TLTB-transaction available bandwidth are computed similarly by taking into account the idle time of the transaction only. The component conflicts and output rate are the average buffer size and number of issued transactions, respectively, over the observation interval.

8.3 Scheduler executions

The schematic view of the application and platform run-time is shown in Fig. 25.

Once a thread[7] starts to execute, it calls the scheduler to assure that it can safely continue. We have implemented the schedulers on SystemC/TLM with non preemptive execution of threads. Once the scheduler has been synthesized using the timed model, it has to be implemented and integrated with the code generated. The generated code consists of two parts,

[6]Transformations operate successive source-to-source transformations defined in handy way using xsl stylesheets. This allows for easier model/chain extensibility but requires software and hardware models to be described by XSD schemas. Hence, in the source, all basic blocks are described as function calls, and all hardware behaviors are described as input/output interfaces; while the actual software functions and hardware behaviors were defined as external legacy or P-Ware code, respectively.

[7]In current implementation one posix/SystemC thread corresponds to one task.

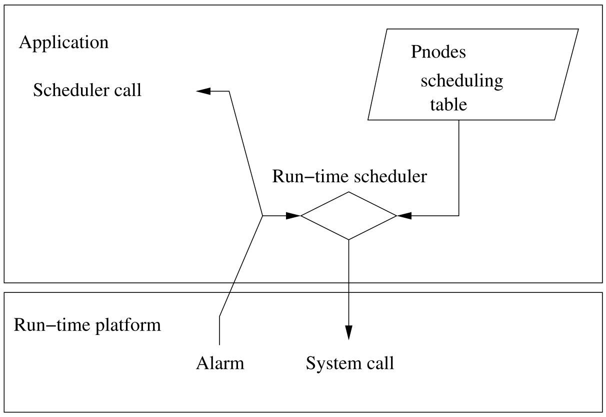


Fig. 25 Schematic architecture and the run-time flow

namely, the application code and the synthesized scheduler. The application code is instrumented to call the application-level scheduler functions before an application thread begins execution and after it terminates.

We also studied scheduler implementation on top of eCos. Scheduler functions are executed in the caller execution flow and use mutexes, condition variables, thread timers, standard posix lock, unlock, wait, notify routines, and non preemption. Three priorities are however used to ensure that a notifying thread is not preempted by another thread by assigning the lowest to notified ones. Also, a thread execution performing a timed wait, except for periods, gets the highest priority so that it has the chance to be executed as soon as it timeouts. Timed waits, for a period or a certain amount of time, are treated by the runtime library.

9 Conclusion

We have proposed an original exploratory methodology for scheduler synthesis for embedded softwares modeled as hierarchical interdependent tasks subject to real-time deadlines. This methodology allows calculating a set of linear constraints systems which define the software level scheduler. Then, different mappings of tasks and their sub-tasks in the hardware architecture define the hardware level scheduler for the software. Finally, the performance evaluation of software and hardware for these mappings allows comparing the efficiency of different hardware-level scheduling mappings.

This approach is original and has two novel contributions: (a) It combines software real-time requirements with hardware performance optimization objectives. (b) It synthesizes composable software-level and hardware-level schedulers which makes the hardware-level scheduler computation incremental and suitable for design exploration of SoCs in particular.

A tool chain supporting the methodology and producing P-WARE simulation programs has been developed. P-WARE has been used to validate our approach on a slice-based MPEG-4 encoder on a multiprocessor SoC. We have shown that a slice-level parallel implementation of the encoder using slices of one hundred macroblocks for 640×480 frames satisfies the encoding rate and produce efficient hardware performance on four processors. The methodology has also been successfully applied to an OC-48 packet forwarder application.

Currently we are working on the automatization of the methodology. The designer effort will focus on system modeling using high level XML-based format of SW/HW models. The scheduler synthesis is automated which make it possible to offer a considerable gain in time of exploration cycles and in time of implantation [25].

References

1. Dutta S, Jensen R, Rieckmann A (2001) Viper: A multiprocessor SOC for advanced set-top box and digital TV systems. IEEE Des Test Comput 18(5):21–31
2. Stravers P, Hoogerbrugge J (2001) Homogeneous multiprocessing and the future of silicon design paradigms. In: International symposium on VLSI technology, systems, and applications (VLSI-TAS), pp 184–187
3. Adiletta M, Rosenbluth M, Bernstein D, Wolrich G, Wilkinson H (2002) The next generation of Intel IXP network processors. In: INTEL Technology Journal, vol 6, Intel Communications Group, Intel Corporation, Aug 2002
4. Moonen A, van den Berg R, Bekooij M, Bhullar H, van Meerbergen J (2005) A multi-core architecture for in-car digital entertainment. In: Proceedings of the GSPx conference
5. Clements PC, Northrop L (2001) Software product lines: Practices and patterns. Addison-Wesley, Reading
6. Meyer MH, Lehnerd AP (1997) The power of product platforms: building value and cost leadership. Free Press, New York
7. Paulin PG, Pilkington C, Langevin M, Bensoudane E, Nicolescu G (2004) Parallel programming models for a multi-processor soc platform applied to high-speed traffic management. In: CODES+ISSS '04: Proceedings of the 2nd IEEE/ACM/IFIP international conference on hardware/software codesign and system synthesis. ACM Press, New York, pp 48–53
8. Cornea R, Dutt N, Gupta R, Krueger I, Nicolau A, Schmidt D, Shukla S (2003) Forge: A framework for optimization of distributed embedded systems software. In: IPDPS '03: Proceedings of the 17th international symposium on parallel and distributed processing. IEEE Computer Society, Washington, p 208.1
9. Cesario W, Baghdadi A, Gauthier L, Lyonnard D, Nicolescu G, Paviot Y, Yoo S, Jerraya AA, Diaz-Nava M (2002) Component-based design approach for multicore socs. In: DAC '02: Proceedings of the 39th conference on design automation. ACM Press, New York, pp 789–794
10. Jalabert A, Murali S, Benini L, Micheli GD (2004) Pipescompiler: A tool for instantiating application specific networks on chip. In: DATE, pp 884–889
11. Sangiovanni-Vincentelli A (2002) Defining platform-based design. EEdesign, EETimes
12. Balarin F, Watanabe Y, Hsieh H, Lavagno L, Passerone C, Sangiovanni-Vincentelli A (2003) Metropolis: An integrated electronic system design environment. Computer 36(4):45–52
13. Theelen BD, Florescu O, Geilen M, Huang J, van der Putten PHA, Voeten J (2007) Software/hardware engineering with the parallel object-oriented specification language. In: 5th ACM & IEEE international conference on formal methods and models for co-design (MEMOCODE 2007), May 30–June 1st, Nice, France, 2007, pp 139–148
14. Tibboel W, Reyes V, Klompstra M, Alders D (2007) System-level design flow based on a functional reference for hw and sw. In: DAC '07: Proceedings of the 44th annual conference on design automation. ACM Press, New York, pp 23–28
15. van der Wolf P, de Kock E, Henriksson T, Kruijtzer W, Essink G (2004) Design and programming of embedded multiprocessors: an interface-centric approach. In: CODES+ISSS '04: Proceedings of the 2nd IEEE/ACM/IFIP international conference on hardware/software codesign and system synthesis. ACM Press, New York, pp 206–217
16. Reyes V, Kruijtzer W, Bautista T, Alkadi G, Núñez A (2006) A unified system-level modeling and simulation environment for mpsoc design: Mpeg-4 decoder case study. In: DATE '06: Proceedings of the conference on design, automation and test in Europe, 3001 Leuven, Belgium, Belgium: European Design and Automation Association, pp 474–479
17. Thiele L, Chakraborty S, Gries M, Kinzli S (2002) Design space exploration of network processor architectures. HPCA Workshop, February 2002
18. Gries M, Kulkarni C, Sauer C, Keutzer K (2003) Comparing analytical modeling with simulation for network processors: A case study. In: DATE
19. Goossens K, Dielissen J, Gangwal OP, González Pestana S, Rădulescu A, Rijpkema E (2005) A design flow for application-specific networks on chip with guaranteed performance to accelerate SOC design and verification. In: DATE'05. IEEE CS, Washington, pp 1182–1187

20. Kumar A, Hansson A, Huisken J, Corporaal H (2007) Interactive presentation: an fpga design flow for reconfigurable network-based multi-processor systems on chip. In: DATE '07: Proceedings of the conference on design, automation and test in Europe. ACM Press, New York, pp 117–122
21. Assayad I, Bertin V, Defaut F-X, Gerner P, Quévreux O, Yovine S (2005) JAHUEL: A formal framework for software synthesis. In: ICFEM'05
22. Assayad I, Yovine S (2006) System platform simulation model applied to multiprocessor video encoding. In: IEEE symposium on industrial embedded systems
23. Assayad I, Yovine S (2007) P-WARE: A precise and scalable component-based simulation tool for embedded multiprocessor industrial applications. In: EUROMICRO DSD
24. Assayad I, Yovine S (2005) Compositional constraints generation for concurrent real time loops with interdependent iterations. In: I2CS'05. LNCS. Springer, Berlin
25. Assayad I, Yovine S (2007) Modelling and exploration environment for application specific multiprocessor systems. In: HASE '07. IEEE CS, Washington, pp 433–434
26. Assayad I, Gerner P, Yovine S, Bertin V (2005) Modelling, analysis and implementation of an on-line video encoder. In: DFMA'05. IEEE Computer Society, Washington
27. Requirements for ip version 4 routers, United States, June 1995
28. IXP2800, Network processor hardware reference manual, http://www.intel.com/design/network/manuals/278882.htm