Design of an Optimal Loosely Coupled Heterogeneous Multiprocessor System

Armin Bender

Faculty for Mathematics and Computer Science (Prof. Dr. W. Grass)
University of Passau, D-94030 Passau, Germany
e-mail: bender@fmi.uni-passau.de

Abstract

This paper presents an approach for mapping tasks optimal to hardware and software components in order to design a real-time system. The tasks are derived from an algorithm and are represented by a task-graph. The performance of the algorithm on the resulting real-time system will meet the specified timing constraints. Some of the hardware components are programmable and others are application specific hardware processors. We propose a powerful MILP (Mixed Integer Linear Program) model with and without functional pipelining. The efficiency of the method is demonstrated with practical examples.

1 Introduction

One important task of a hardware/software codesign is to map different tasks of an algorithm onto hardware or software components. Some components are programmable and others are application specific hardware processors. The aim of the codesign is to design a heterogeneous and loosely coupled multiprocessor system, which violates no timing constraints while performing the underlying algorithm. To reach this goal we split a given algorithm into tasks and map these tasks to several processors.

The known approaches can be divided with respect to the specification style into software or hardware oriented approaches. The COSYMA approach [5] is a software oriented approach, which means that the input specification is given in terms of a programming language. Only tasks that violate real-time constraints are implemented in hardware. They use an iterative heuristic approach. For the tasks to be implemented in hardware a partially automatic synthesis tool generates the hardware components. After generating the components they check the original timing constraints while regarding additional communication overhead. To check this they use a fast timing hardware/software co-synthesis analysis tool [9]. The hardware oriented VULCAN system [7] operates on the same system architecture (heterogeneous processors without local memories) as the COSYMA system. Starting from an implementation where all components are implemented in hardware some components are selected, which can be implemented in software. Alternatives for implementations of hardware components are not taken into consideration. Because of the restrictions of the input specification language (HardwareC) the designer is not able to describe dynamic data structures, pointers, etc. In the COBRA project [10] they have developed a prototyping system, which includes cost functions for hardware, software and communication costs. The system supports integration of standard processors as well as emulation of processors.

In this paper we formulate a Mixed Integer Linear Program (MILP) that allows to determine a mapping optimizing a trade off function between execution time, processor and communication costs. It is desirable to have a mapping approach that optimizes a function depending on such factors [14]. For some tasks it is obvious which task has to be implemented in hardware and which one in software. For example, a high speed packet manipulation should be implemented in hardware while recursive searching is always implemented in software. However, there are tasks, which may be implemented either in hardware or in software or should be further splitted into subtasks.

To solve the task mapping problem we have to determine on what hardware component a task is performed (allocation) and we have to compute a starting time for every task (schedule). In our approach this is done at the same time. Allocation decides for each task whether it should be implemented in hardware (e.g. ASICs) or in software (e.g. microprocessor). The partitioning of the algorithm into tasks which are executable on components is performed in a preprocessing step. The tasks are mapped on hardware and software components in such a way that no timing constraints are violated by performing the algorithm on the real-time system determined for this algorithm. In many signal and image processing applications it is necessary to deal with continuous data streams as input in the real-time system. To reach this requirement we extend our basic model [3] in this paper. We call this extension functional pipelining (minimize the time interval between two runs of the algorithm).

The paper is organized as follows. In section 2, we give a more detailed description of the problem addressed in this paper. In section 3 we introduce the formal model and a powerful extension for functional pipelining both described as MILPs. In section 4 we show the practicability of our mapping approach by applying it on several typical signal and image algorithms. We have a second powerful extension of the model, which we cannot present in this paper because of the limited space. In this extension we can deal with multiple computation of tasks (a task may be performed on several processors to avoid the need for (time expensive) communications and to exploit more parallelism).

ED&TC '96
0-89791-821/96 $5.00

2 Mapping Approach - Overview

The hardware/software codesign process consists of several steps. This design process is repeated iteratively until the design goals are met. First of all, the algorithm has to be specified using a formal language [2]. In a second step the algorithm is partitioned into tasks that are possibly processed on different processors. Now, the set of available processors of the generic multiprocessor target architecture has to be defined. The time for executing a task on an ASIC can be guessed by synthesizing this ASIC for that task using a high level synthesis system [1, 6]. For each other task one has to determine the execution time for a task on at least one processor. The result is an annotated task-graph where each task node has got information about execution times on different processors. For each pair of communicating tasks an edge is introduced. This edge is annotated with the communication time. We restrict our approach to those applications were the processing time is independent on the values that are processed. We also assume that the amount of data to be communicated between the different tasks has been determined. Although the restrictions seem to be hard there are many signal and image processing applications we can cope with. An application specific real-time system (including the necessary communication structure) with optimal allocation and scheduling of the tasks is the result of the mapping procedure.

General purpose standard processor (e.g. microprocessor), different application specific components (ASICs) and application oriented standard processors (e.g. signal processors, transputers) define our generic target architecture shown in Fig. 1. We call all these working components processors in this paper. We assume each processor has its own local memory and only one task can be executed at a time by one processor. To handle communication each processor has its own communication processor. Tasks on different processors are executed in parallel. A processor execution of a task cannot be interrupted. Additionally we model a global memory as a processor with zero execution time. In this case we have to consider only the communication time between the memory and the other processors. We assume a multibus system for the communication between processors where each bus has the same transmission rate.

In Fig. 2 we show the mapping approach. The constraint library contains the annotated task-graph, processor and bus costs and timing constraints. The granularity of the task-graph is crucial for the success of the mapping procedure. The finer the granularity is the more parallelism can be exploited. On the other hand, the complexity of the mapping problem grows with the number of tasks. The process of splitting an algorithm into tasks must still be left to a designer who will normally start with defining each small procedure or just basic block as a task. For the whole algorithm and sometimes for special task-regions we have to regard time constraints. Clearly, the process will be started by using only few hardware processors for those tasks that are assumed time critical. If the mapping procedure ends up by presenting no acceptable solution (i.e. some real-time constraints are not met) the number of tasks that are possibly assigned to hardware processors is enlarged in an iterative way. In each iteration step we normally implement more and more tasks in hardware. Therefore, we must introduce more resources in our constraint library in each step. Then we update the nodes of the annotated task-graph. This has to be done only for those tasks, which can be executed on the additional processors. At last, another MILP is generated with these new information. At the end of this mapping procedure all tasks from a given task-graph are mapped onto the processors so that the underlying algorithm does not violate any time conditions. Tasks, which are mapped on the microprocessor are "implemented in software". On the other hand all other tasks, which need special hardware (ASICs, signal processors, etc.) are "implemented in hardware". The designer decides whether he wants to generate automatically the MILP with or without functional pipelining corresponding to the requirements of the real-time system. For solving the MILP we use the standard software tool *CPLEX* [4]. To reduce the domain (range of possible values) of the variables in the MILP and therefore the complexity of the MILP we have developed a heuristic preprocessing [12] for large mapping problems.

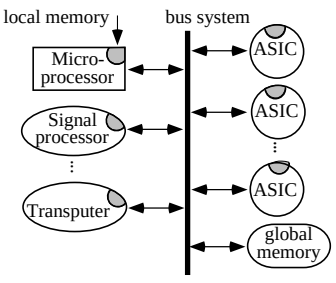

Fig. 1: Target architecture

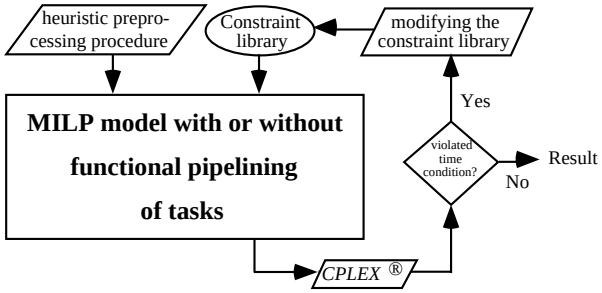

Fig. 2: Mapping approach

3 MILP model for functional pipelining

In section 3.2 we describe the MILP for the basic model and in section 3.3 we extend this model for functional pipelining. At first we introduce a notation needed to formulate the MILPs.

3.1 Notation

A short summary of the abbreviations used in a MILP-formulation is given first. N represents the number of given tasks, $M+1$ the number of available processors, and L the number of available buses. With processor P_0 we indicate the universal microprocessor.

T	set of tasks $T = \{T_1, \ldots, T_N\}$
B	set of buses $B = \{B_1, \ldots, B_L\}$
P	set of processors $P = \{P_0, \ldots, P_M\}$
$\rho(T)$	set of processors able to execute task $T \in T$
G	directed acyclic task-graph. $G = (V, E)$ where each vertex $i \in V$ represents one task T_i ; a directed arc $(i,j) \in E$ from task T_i to T_j represents a precedence constraint between these tasks. For all $(i,j) \in E$ it holds $i < j$.

$RE(i)$ set of all vertices reachable from vertex $i \in V$ in the given task-graph G

$\prec_2$ order relation with: $(i,j),(i',j') \in E$:
$(i,j) \prec_2 (i',j') \Leftrightarrow (i < i') \vee ((i = i') \wedge (j < j'))$

(i,j) represents a communication between task T_i and task T_j. $RE(i)$ represents the set of tasks data dependent from task T_i. Nodes i with $RE(i) = \{\ \}$ are called leaf nodes of the task-graph. The corresponding tasks are called leaf tasks. Each task $T \in \mathbb{T}$ can be executed on at least one processor, i. e. $\rho(T) \neq \{\ \}$. A MILP formulation needs the definition of variables and constraints.

Variables used in the MILP formulation:

$d_{mi} = \begin{cases} 1 & \text{task } T_i \text{ is executed on processor } P_m \\ 0 & \text{otherwise} \end{cases}$

$y_{ij} = \begin{cases} 1 & \text{task } T_i \text{ starts execution before task } T_j; \\ & \text{both tasks are allocated on the same processor} \\ 0 & \text{otherwise} \end{cases}$

$h_{lij} = \begin{cases} 1 & \text{communication of task } T_i \text{ and } T_j \text{ is performed} \\ & \text{on bus } B_l \\ 0 & \text{otherwise} \end{cases}$

$\gamma_{mij} = \begin{cases} 1 & \text{both tasks } T_i \textbf{ and } T_j \text{ are executed on} \\ & \text{processor } P_m. \text{ It holds: } \gamma_{mij} = d_{mi} \wedge d_{mj} \\ 0 & \text{otherwise} \end{cases}$

$w_{(ij)(i'j')} = \begin{cases} 1 & \text{the communication } (i,j) \text{ starts before} \\ & \text{communication } (i',j') \\ 0 & \text{otherwise} \end{cases}$

s_i start time of task T_i

b_{ij} start time of communication between task T_i and T_j

Variables d are only introduced for these tasks $T \in \mathbb{T}$ which can be performed on the processor $P \in \mathbb{P}$. We introduce no variable for $P_m \notin \rho(T_i)$.

Constants used in the MILP formulation (given in the constraint library)**:**

t_{mi} execution time of task T_i on processor P_m

c_{ij} communication time for sending data computed by T_i to T_j; it holds $(i,j) \in E$

A communication time is only considered when the two tasks are allocated on different processors. c_{ij} is independent of the actual bus allocation because each bus has the same transmission rate. Therefore, the communication time only depends on the tasks. In this manner, we can observe the amount of data to be transferred but we cannot model the kind of the transfer, which may be processor dependent (e.g. protocol type). For the communication each processor has its own communication processor.

3.2 MILP Formulation for the basic model

Now, we formulate the MILP for the basic model. The solution of a MILP determines on which hardware unit (allocation) and at which time the task is started (schedule). Allocation involves the decision for each task whether it should be implemented in hardware (e.g. on ASICs) or in software (e.g. on microprocessor). With the following objective function we want to minimize the overall execution time (OET), the overall processor costs (OPC), and the overall communication costs (OCC). The weights k_1, k_2 and k_3 of the costs OET, OPC and OCC have to be tuned by the designer.

Objective function (with: $k_1, k_2, k_3 \in \mathbb{R}_0^+$) **:**

$$minimize\ (k_1 \cdot OET + k_2 \cdot OPC + k_3 \cdot OCC) \qquad \text{(OF)}$$

subject to:

The finishing time of each leaf task is less or equal OET. Constraints for bounding the OET have only to be introduced for each leaf tasks ($RE(i) = \{\ \}$).

objective function constraints

$\forall 0 \le m \le M$ processor P_m

$\forall 1 \le i \le N$: task T_i

$$RE(i) = \{\ \} \wedge d_{mi} = 1 \Rightarrow s_i + t_{mi} \le OET$$

$d_{mi} = 1 \Rightarrow s_i + t_{mi} \le OET$ has to be represented by a linear inequation. Therefore, this condition is translated into the resulting constraint:

$$\forall 1 \le i \le N \wedge RE(i) = \{\ \}: \quad s_i + \sum_{m=0}^{M} (t_{mi} \cdot d_{mi}) \le OET \qquad \text{(OFC1)}$$

In inequation (6) (will be given later) we force that each task is allocated on exactly one processor. Therefore, the sum in (OFC1) represents the execution time of the leaf task T_i on the assigned processor P_m. Normally a task-graph has more than one leaf task (see Fig. 4).

The costs for processors and buses are taken from the constraint library. (OFC2) models the overall processor costs and (OFC3) models the overall communication costs.

$$OPC = \sum_{\substack{m=0 \\ \exists 1 \le i \le N : d_{mi} = 1}}^{M} \text{costs of processor } m \qquad \text{(OFC2)}$$

$$OCC = \sum_{\substack{l=1 \\ \exists (i,j) \in E : h_{lij} = 1}}^{L} \text{costs of bus } l \qquad \text{(OFC3)}$$

Conditions with existential quantification can be modeled in the MILP by introducing 0/1 variables. Because of the limited space we do not present the resulting constraints in this paper. They are described in [12]. We now introduce the constraints ensuring that an assignment to the variables determine a valid allocation and schedule of the tasks. We have to observe data dependencies and have to avoid resource conflicts. First, we model data dependency constraints.

Data dependencies

If two tasks T_i and T_j in the task-graph are connected by an arc $(i,j) \in E$ then the execution of task T_i on processor $P_m \in \rho(T_i)$ with execution time t_{mi} has to be finished before the execution of task T_j can start (modeled with 1a).

task execution times

$\forall 0 \le m \le M$ processor P_m

$\forall (i,j) \in E$ consider all tasks with precedence constraint

$\wedge\ P_m \in \rho(T_i) \cap \rho(T_j)$: processor P_m can execute the two tasks T_i and T_j

$$d_{mi} = d_{mj} = 1 \quad \Rightarrow \quad s_i + t_{mi} \le s_j \qquad \text{(1a)}$$

This condition has to be represented by a linear inequation. Therefore, this condition is translated into:

$$s_i + t_{mi} \le s_j + \begin{cases} 0 & \text{if } d_{mi} = d_{mj} = 1 \\ \infty & \text{otherwise} \end{cases}$$

Instead of ∞ we choose a sufficiently big number C. The resulting constraint is

$$\Rightarrow \quad \underbrace{s_i + t_{mi}}_{(*)} \le s_j + \underbrace{\left(2 - d_{mi} - d_{mj}\right) \cdot C}_{(**)} \qquad (1a')$$

(∗) This part of the inequation need only to be observed if $d_{mi} + d_{mj} = 2$.

(∗∗) Therefore, this term is zero if to be observed and at least C if not to be observed.

For internal reasons of the MILP solver C should be as small as possible. When the tasks are allocated on different processors $P_k \neq P_m$ then we additionally have to observe communication time c_{ij} and T_j can start c_{ij} time units after T_i has finished (modeled with 1b).

$$\forall 0 \le m,k \le M \; \forall (i,j) \in E$$
$$\wedge \; k \neq m \wedge P_k \in \rho(T_i) \wedge P_m \in \rho(T_j):$$
$$\Rightarrow \quad s_i + t_{ki} + c_{ij} \le s_j + \left(2 - d_{ki} - d_{mj}\right) \cdot C \qquad (1b)$$

In hardware/software codesign the communication costs have a great influence on the quality of a modular implementation. Therefore, communications on buses are also considered. After the allocation of tasks on processors we have to consider the communication time for two tasks with direct precedence constraint which are allocated on different processors. Each processor that has to communicate has its own communication processor, i.e. communication involving P_k can take place in parallel with executing a task on P_k. Inequations related with the beginning of a communication on bus $B \in \mathcal{B}$ are modeled in (2). The duration of a communication is observed in (3).

beginning of communication

$$\forall (i,j) \in E: \quad s_i + \sum_{\substack{m=0 \\ P_m \in \rho(T_i)}}^{M} \left(t_{mi} \cdot d_{mi}\right) \le b_{ij} \qquad (2)$$

duration of communication

$$\forall (i,j) \in E:$$
$$\underbrace{\overbrace{b_{ij}}^{(*)} + \overbrace{c_{ij}}^{(**)}}_{(****)} \le \overbrace{s_j}^{(***)} + \overbrace{\left(1 - \sum_{l=1}^{L} h_{lij}\right) \cdot C}^{(*****)} \qquad (3)$$

(∗) start of communication between T_i and T_j
(∗∗) duration of communication between T_i and T_j
(∗∗∗) start of task T_j
(∗∗∗∗) observation condition
(∗∗∗∗∗) end of communication between T_i and T_j

For the introduction of such inequations we can use the property that $h_{lij} = 1$ (communication between task T_i and T_j is done on bus l) holds for at most one l. This property we force in inequation (7) (will be given later). If the sum in (∗∗∗∗) is zero, T_i and T_j are executed on the same processor. Therefore, the observation condition equals C. This means that the constraint is automatically fulfilled and denotes no restriction concerning the whole model. If the observation condition equals 0 the data depending has to be taken into account. The constraints to observe data dependencies are now complete. In the next step we introduce the constraints to avoid resource conflicts.

Resource conflicts

Two tasks must not be executed on the same processor at the same time. For data dependent tasks, this is ensured by (1a´). For all other pairs of tasks, we have to introduce nonoverlapping constraints (4a) and (4b). With the first inequation type (4a) we describe the possibility that task T_i will be executed after task T_j on the same processor (i. e. $y_{ij} = 1$). In (4b) we formulate the other possibility (i. e. $y_{ji} = 1$).

nonoverlapping constraints for executions on processors

$$\forall 0 \le m \le M \; \forall 0 \le i,j \le N \wedge P_m \in \rho(T_i) \cap \rho(T_j)$$
$$\wedge \; \underbrace{i \notin RE(j)}_{(*)} \wedge j \notin RE(i):$$
$$\Rightarrow \quad s_i + t_{mi} \le s_j + \left(3 - d_{mi} - d_{mj} - y_{ij}\right) \cdot C \qquad (4a)$$
$$\Rightarrow \quad s_j + t_{mj} \le s_i + \left(3 - d_{mi} - d_{mj} - y_{ji}\right) \cdot C \qquad (4b)$$

(∗) T_j has not been executed after completion of T_i because of data dependencies; This avoids inequations which would be implied by (1a´) and (1b).

y_{ji} and y_{ij} are only relevant for $d_{mi} = 1 \wedge d_{mj} = 1$. In this case we can assume $y_{ji} = 0 \Leftrightarrow y_{ij} = 1$ and $y_{ji} = 1 \Leftrightarrow y_{ij} = 0$. This allows to save some y variables. Therefore, we add $i < j$ to the premise and can omit $i \notin RE(j)$ because of $i < j \Rightarrow i \notin RE(j)$. The additional term in (4b) indicating the relevance of the inequation (see notes for 1a´) has to be replaced by $\left(2 - d_{mi} - d_{mj} + y_{ij}\right) \cdot C$.

There is a need for communication when two tasks are allocated on different processors and when there is a precedence constraint between these tasks. With the objective function (OFC3) introduced above we also consider the number of buses. Similar to the scheduling of tasks we need a constraint that prevents an overlapping of two communications on the same bus at the same time. Therefore, we introduce the following inequations as *nonoverlapping constraints on buses* :

$$\forall (i,j),(i',j') \in E$$
$$\wedge \; (i,j) \prec_2 (i',j') \wedge i' \notin RE(j) \; \forall 1 \le l \le L:$$

$$\begin{aligned} b_{ij} \quad + c_{ij} \quad &\le b_{i'j'} + \left(3 - h_{li'j'} - h_{lij} - w_{(ij)(i'j')}\right) \cdot C \\ b_{i'j'} + c_{i'j'} \quad &\le b_{ij} \quad + \left(2 - h_{li'j'} - h_{lij} + w_{(ij)(i'j')}\right) \cdot C \end{aligned} \qquad (5)$$

The inequations in (5) are defined according to (4a) and (4b). In addition to the observation of data dependencies and to the avoidance of resource conflicts we need the constraints (6) to (9b). With inequation (6) we ensure that each task has to be allocated on exactly one processor. The equation (7) formulates this in a similar way for buses. That means, we need exactly one bus for a communication between two data dependent tasks allocated on different processors.

$$\forall 1 \le i \le N: \quad \sum_{\substack{m=0 \\ P_m \in \rho(T_i)}}^{M} d_{mi} = 1 \qquad (6)$$

$$\forall (i,j) \in E: \quad \sum_{l=1}^{L} h_{lij} = 1 - \sum_{\substack{m=0 \\ P_m \in \rho(T_i) \cap \rho(T_j)}}^{M} \gamma_{mij} \qquad (7)$$

For the equation (7) we need the condition $\gamma_{mij} = d_{mi} \wedge d_{mj}$. A possible formulation for this condition in a MILP is given in (8). To get only positive start times we need inequation (9a) and (9b).

$$\forall (i,j) \in E \;\; \forall 0 \le m \le M \wedge P_m \in \rho(T_i) \cap \rho(T_j): \\ (d_{mi} + d_{mj}) - 1 \le 2 \cdot \gamma_{mij} \le (d_{mi} + d_{mj}) \qquad (8)$$

$$\forall 1 \le i \le N: \quad s_i \ge 0 \qquad (9a)$$

$$\forall (i,j) \in E: \quad b_{ij} \ge 0 \qquad (9b)$$

Now the formalization of the MILP model without functional pipelining is complete. The inequations (2) and (3) imply the inequation (1b). Therefore, we do not have to regard this restriction in our MILP and we replace (1a´) and (1b) by (1), which does not contain a case decision. This reduces the complexity of the overall MILP.

$$\forall (i,j) \in E: \quad s_i + \sum_{\substack{m=0 \\ P_m \in \rho(T_i)}}^{M} (t_{mi} \cdot d_{mi}) \le s_j \qquad (1)$$

3.3 Extension: functional pipelining

In many real-time applications it is necessary to deal with continuous data streams as inputs [8]. With our MILP model previously described we can handle this only in a restricted way. After the execution of all tasks we can start with a new iteration. This means that the first computation of a task in the (i+1)-th iteration is only possible if all leaf tasks are finished in the i-th iteration. Therefore, we have $n \cdot OET$ as the overall computation time (OCT) after n iterations. Normally it holds: $n \cdot OET >> OCT$. We illustrate this for two iterations with a small example in Fig. 3.

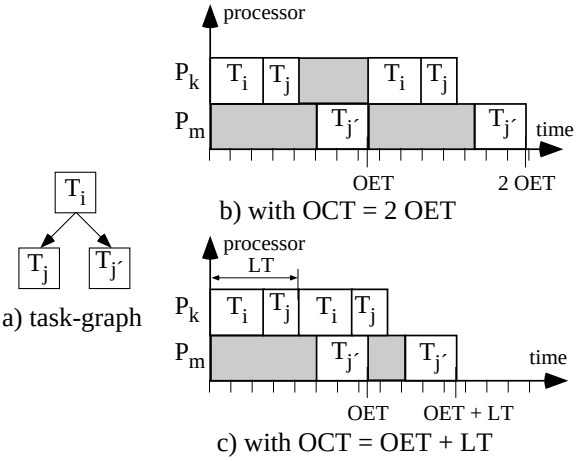

Fig. 3: Advantage of functional pipelining

To minimize the OCT it is necessary to start with the computation of the first task in iteration (i+1) as soon as possible. The time interval between two successive runs of the algorithm is called latency (LT). To be able to minimize the latency we have to extend the objective function given above.

Extended objective function (with: $k_1, ..., k_4 \in \mathbb{R}_0^+$):

$$minimize \; (k_1 \cdot OET + k_2 \cdot OPC + k_3 \cdot OCC + k_4 \cdot LT) \quad \text{(EOF)}$$

LT is a continuous variable in the MILP. The model is restricted in the following way. To get not too much additional variables in the MILP we allocate each task (communication) in each iteration on the same processor (bus). In addition, we require that all tasks of the i-th iteration are finished on a processor before a task of the (i+1)-th iteration can start on this processor. We need the following inequation to formulate that task T_i can start its execution in iteration (i+1) after task T_j is finished in the i-th iteration for all tasks T_i, T_j which are allocated on the same processor P_m.

$$\forall 0 \le m \le M \;\; \forall (i,j): 1 \le i,j \le N \wedge i \notin RE(j) \\ \wedge \;\; P_m \in \rho(T_i) \cap \rho(T_j): \\ \overbrace{s_j + t_{mj}}^{(*)} \le \overbrace{s_i + LT}^{(**)} + \overbrace{(2 - d_{mj} - d_{mi}) \cdot C}^{(***)} \qquad \text{(E1)}$$

(*) end of task T_j in iteration i
(**) start time of task T_i in iteration (i + 1)
(***) The inequation is only to be observed if both tasks are allocated on the same processor

In each solution there is one inequation for each processor P_m containing the first task T_i executed on processor P_m and the last task T_j executed on P_m that influence the latency time. We have to introduce inequations for all pairs and all processors able to execute both tasks in order to cover this case in each solution. According to this *multiple period nonoverlapping constraints on processors* we need a *multiple period nonoverlapping constraints on buses* as an extension of the single period constraints in the basic model. Therefore, we introduce the inequation (E2), which ensures that the last communication in the i-th iteration has to be finished before the first communication in the (i+1)-th iteration on bus l is possible.

$$\forall (i,j),(i',j') \in E \wedge i' \notin RE(j) \;\; \forall 1 \le l \le L: \\ b_{ij} + c_{ij} \le b_{i'j'} + LT + (2 - h_{lij} - h_{li'j'}) \cdot C \qquad \text{(E2)}$$

When we extend our previous MILP by the inequations (E1) and (E2) and change the objective function as we did in (EOF) we get a new MILP, which describes a real-time system for continuous data streams as input.

4 Experimental Results

This section is organized as follows. First, we consider an image processing application in detail and we illustrate for this practical example the advantage of functional pipelining. Then we describe the basic structure of three different task-graphs. For each example the underlying annotated task-graph is considered shortly. Then we show for these task-graphs the results of our mapping approach. To solve the following MILPs we used CPLEX [4] on a SPARCstation 20 (192 MB main memory, two 60 MHz SuperSPARC CPUs).

4.1 Results of a typical application

As an example we use a typical complex image processing application based on the CCITT recommendation H.261 [13]. This application consists of several tasks with execution times independent of the input values. Each of these tasks have different demands on the hardware components to be used.

4.1.1 The underlying algorithm

Fig. 4 shows the corresponding task-graph derived from the video coding algorithm H.261. This graph is automatically generated. Vertices are marked with the possible allocation of tasks to processors and the individual execution time. Arcs are marked with the communication time in clock cycles between two tasks allocated on different processors.

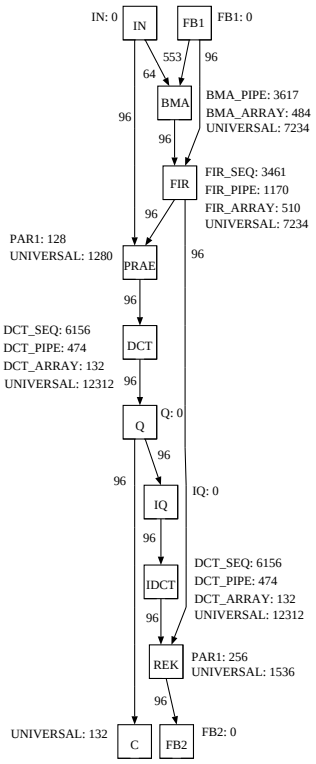

Fig. 4: Task-graph

The constraint library contains one microprocessor (UNIVERSAL) and 9 different ASIC´s (BMA_PIPE, ...) on which we can allocate the tasks. The vertex description is a mnemonic abbreviation for the tasks. For example DCT denotes the Discrete Cosine Transformation. We have extended the original description by execution times for each task on the microprocessor. Tasks IN and FB model external memories. To avoid cycles in the task-graph we model the FB memory with FB1 for read and FB2 for write. These tasks as well as the tasks Q and IQ are used to model the environment. Therefore, the execution times are set to zero.

4.1.2 Mapping - Results without functional pipelining

We assume for the task C the timing constraint $s_C + t_C \leq 2500$. With s_C we identify the start time of task C on the allocated processor and with t_C the execution time of C on this processor. Therefore, this inequation states that the execution of task C must be finished after 2500 clocks. This is a typical timing constraint for the design of real-time systems. Such constraints are contained in our constraint library.

Because a complex MILP can be expected we first apply a heuristic preprocessing, which is based on the metropolis algorithm [11]. This results in 3400 clocks as an upper bound for the overall execution time. With this information we determine ASAP- (as soon as possible) and ALAP-times (as late as possible) for the start times for each task and communication. We have used these bounds in our MILP to reduce the computation time for solving the MILP. With respect to the given constraint library we can find a vaild design of a real-time system with all tasks allocated and scheduled. The corresponding constants k_1 and k_3 in the objective function (OF) were given the value 0.4 and the constant k_2 was given the value 0.2.

The MILP generated without functional pipelining for this application consists of 196 (275) inequations with 111 (152) variables. The values in brackets denote the MILP without ASAP- and ALAP-times from the preprocessing. The CPU time was 6.44 (18.46) seconds.

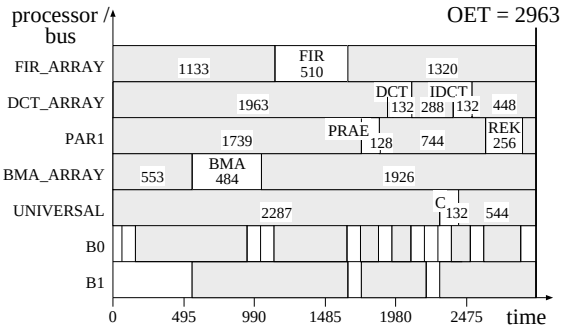

Fig. 5: Gantt diagram I

The results are shown in Fig. 5 with an automatically generated Gantt diagram, which is a standard illustration for tasks executed in parallel on several processors. The new designed real-time system consists of two buses (B0, B1) and five different processors. For example the tasks PRAE and REK are allocated on processor PAR1. The task PRAE starts after 1739 clocks. The execution time of this task is 128 clocks (see Fig. 4) on the processor PAR1. At the beginning we have a data transfer from the FB memory to the BMA_ARRAY, which needs 553 clocks for the communication on bus B1. The corresponding architecture for the designed real-time system is shown in Fig. 6. The allocation and the scheduling for this system is optimal for the given task-graph (Fig. 4) with respect to OET, OPC, OCC and the given constraint library. The timing constraint for task C is also met.

In Fig. 5 and 6 it can also be seen that we have to realize six tasks on four ASIC´s and only one task (C) is computed as software on the microprocessor (UNIVERSAL). Therefore, we have six hardware implemented tasks and only one software implemented task. Tasks with execution time zero are omitted. The overall execution time of the video coding algorithm H.261 on this real-time system is 2963 clocks, which is also shown in Fig. 5.

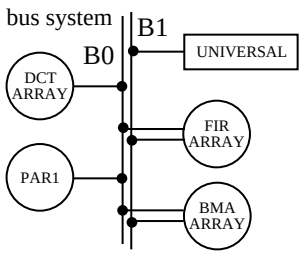

Fig. 6: Designed system

4.1.3 Mapping - Results with functional pipelining

In section 3.3 we have introduced two additional inequations, which we need to minimize the time interval between two successive runs of the algorithm. For this example we used no heuristic preprocessing procedure. With the extended objective (EOF) we have choose (k_1, k_2, k_3, k_4) = (0.02, 0.04, 0.04, 0.9) because our main goal was to get a minimized latency time. The necessary MILP for the above application consists of 544 inequations with 137 variables. The CPU time was 27.06 seconds.

The results are shown in Fig. 7 with an automatically generated Gantt diagram. Again, we have six hardware implemented and one software implemented task. The latency time of the coding algorithm on the resulted real-time system is 1320. This latency time is defined by bus B0. Also, we show the overall execution time in Fig. 7, which is greater as in the previous section. It can be seen,

that the execution of tasks FIR and BMA can start before the last execution of the tasks in the previous iteration are finished. One reason for this is that the communication to task BMA can start very early. Without functional pipelining 29,630 clocks are needed for ten iterations. With the minimization of the latency time we need only 14,907 clocks for ten iterations.

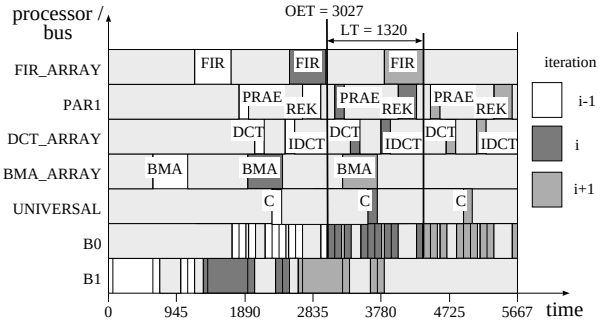

Fig. 7: Gantt diagram II

4.2 Further results

In this section we present the results of three task-graphs with a typical structure by applying our mapping approach. We start our mapping procedure from the given annotated task-graphs. In Table 1 we describe each annotated task-graph shortly. Each task can be performed on the microprocessor. Normally we have at least one additional processor for each task. In example 1 and 2 the communication times are lower as the execution times of the tasks on the processors. In example 3 the communication times are much higher. For each example two buses are available. In Table 2 we show the principle structure of the automatically generated MILPs, the solving time, the overall execution time and the latency time. We distinguish between our basic model (run 1) and the extension (run 2) concerning functional pipelining.

	# arcs	# tasks	max. of parallelism	# available processors
example 1	12	8	3	5
example 2	10	8	4	3
example 3	7	8	2	4

Table 1: Structure of the annotated task-graphs

	# const.	# var.	sec.	*OET*	LT
example 1 / run 1	190	87	0.82	341	-
example 1 / run 2	620	136	17.7	647	240
example 2 / run 1	199	93	0.36	198	-
example 2 / run 2	363	99	0.51	202	140
example 3 / run 1	166	84	6.16	240	-
example 3 / run 2	288	85	1.93	270	200

Table 2: Further results

For the underlying annotated task-graphs we have got optimal real-time systems with respect to the objective functions (EOF) and to the constraint libraries. In all examples we could reduce the overall computation time

6 Conclusion

We have presented an approach for automatically designing application specific real-time systems, which defines a bus oriented, heterogeneous, loosely coupled real-time system composed of resources available from a constraint library. Corresponding to the requirements the designer decides whether he wants to generate a MILP with or without considering pipelining. Also the cost function can be tuned by the designer in order to determine the weights of hardware costs and execution times. The determined allocation and schedule is optimal with respect to the costs and the time requirements for performing the underlying algorithm (e.g. image or signal processing algorithm) on the designed real-time system. For solving the optimization problem bounds have been introduced, so that the time for solving the problem is acceptable for problems of reasonable size. Several experimental results show the practicability of our mapping approach in hardware/software codesign for industrial applications. To illustrate the results where tasks are executed in parallel on several processors we use automatically generated Gantt diagrams, which allows the designer to evaluate the results.

References

[1] H. Achatz: *SUSAN: System for Universal Scheduling and Allocation*, SASIMI, Nara, Japan, 1993, pp. 138 - 144.

[2] E. Barrows, W. Rosenstiel, X. Xiong: *Hardware/Software Partitioning with UNITY*, Handouts of the 2nd Int. Workshop on Hardware/Software Codesign, Cambridge, 1993.

[3] A. Bender: *Optimal Task Mapping in a Hardware/Software Codesign Environment*, Proc. of the Workshop on Design Methodologies for Microelectronics, Slovakia, 1995, pp. 177-186.

[4] *Using the CPLEX Callable Library*, User Guide, CPLEX Optimization Inc., Incline Village, U.S.A., 1994

[5] R. Ernst, J. Henkel, T. Benner: *Hardware-Software Cosynthesis for Microcontrollers*, IEEE Design&Test of Computers, Vol. 10, No. 4, 1993, pp. 64-75.

[6] W. Grass, M. Mutz, W.Tiedemann: *High Level Synthesis based on Formal Methods*, Proc. of the 20th EUROMICRO conference, Liverpool, 1994, pp. 83-91.

[7] R. Gupta, C. Coelho, G. De Micheli: *Program Implementation Schemes for Hardware-Software Systems*, Computer, Vol. 27, No. 1, 1994, pp. 48-55.

[8] W. A. Halang, A. D. Stoyenko: *Real Time Computing*, Springer, 1994.

[9] D. Herrmann, J. Henkel, R. Ernst: *An Approach to the Adaption of Estimated Cost Parameters in the COSYMA System*, Proc. of the 3rd Int. Workshop on Hardware/ Software Codesign, Grenoble, 1994, pp. 100-107.

[10] G. Koch, U. Kebschull, W. Rosenstiel: *A Prototyping Environment for Hardware/Software Codesign in the COBRA Project*, Proc. of the 3rd Int. Workshop on Hardware/Software Codesign, Grenoble, 1994, pp. 10-16.

[11] N. Metropolis, et. al.: *Equation of State Calculations for Fast Computing Machines*, Journal of Chemical Physics, Vo. 21, 1953.

[12] P. Scholz: *Static mapping of program tasks onto multiprocessor systems for real-time applications* (In German), diploma thesis, University of Passau, 1994.

[13] M. Schwiegershausen, et. al.: Mapping Complex Image Processing Algorithms onto Heterogeneous Multiprocessors Regarding Architecture Dependent Performance Parameters, Proc. of the 3rd Int. Workshop on Algorithms and Parallel VLSI architectures, Leuven, Belgium, 1994.

[14] N. Woo, et. al.: *Compilation of a single specification into hardware and software*, Handouts of the 1st Int. Works. on Hardware/Software Codesign, Estes Park, Colorado, 1992.