MILP Based Task Mapping for Heterogeneous Multiprocessor Systems

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

CAD-systems supporting hardware/software codesign map different tasks of an algorithm onto processors. Some of the processors are programmable and others are application specific. We propose a new MILP (mixed integer linear program) model that allows to determine a mapping optimizing a trade off function between execution time, processor and communication cost. The mapping also guarantees that all specified execution deadlines are met. We demonstrate the efficiency with practical examples.

1. Introduction

The purpose of this paper is to present a new mapping approach to find optimal designs of real-time systems composed of hardware and software components for implementing a given algorithm. The result of such a codesign is an application specific real-time system, which is composed of standard processors (e.g. microprocessor, signal processor) and several application specific processors (ASICs). The hardware and software components are designed in such a way that no timing constraints are violated by performing the algorithm on the real-time system determined for this algorithm. The partitioning of the algorithm into tasks executable on components is performed in a preprocessing step.

A good mapping of tasks (allocation and scheduling) into hardware and software implementations depends on many factors like performance, timing constraints, hardware cost, etc. It is desirable to have a mapping approach that optimizes a function depending on such factors [1]. 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.

The hardware/software codesign process consists of several steps that are run through iteratively until the design goals are met. First of all, the algorithm has to be specified using a formal language [2, 3]. 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 the task using a high level synthesis system [4, 5]. For each other task one has to determine the execution time for at least one processor. The result is an annotated task-graph where each task node may have assigned several execution times for running the task on different processors. For each pair of tasks that communicate, an edge is introduced annotated with the communication time. We restrict our approach to those applications 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. The result of the mapping procedure is the decision which task runs on which processor(s) at which time.

We can classify the different approaches for hardware /software codesign as software and hardware oriented. In the software oriented approaches specifications are given in a programming language, like C, and the tasks are normally implemented on programmable processors. Only those tasks that do not meet real-time constraints are mapped onto hardware processors [6, 7]. For these applications one needs a high level synthesis system starting from C like algorithms to support the design process. Hardware oriented approaches normally take VHDL as a specification language and implement most tasks in hardware. Tasks that are not on the critical path may now be selected for a software implementation [8, 9, 10, 11].

As in all codesign proposals we only consider mappings onto a set containing one standard processor and some additional dedicated processors. The design decision is therefore restricted to hardware or software implementation. For cost reasons it seems to be better to allow also different types and several instances of standard processors. Universal microprocessors, signal processors and transputers are examples for the types we have in mind. We therefore consider a loosely coupled heterogeneous multiprocessor system as target architecture with global and local memories.

The focus of this paper is on modeling the task mapping problem as a MILP (mixed integer linear program) allowing the use of standard tools for solving it. 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 described as MILP. The MILP is solved with a standard software tool. In section 4 we show the practicability of our mapping approach by applying it on several typical signal and image processing algorithms. A more powerful model based on the model presented her cannot be presented in this paper because of the limited space. In this extension we can also optimize the latency between two successive runs of the algorithm for overlapped executions (pipelining) [12].

EURO-DAC '96 with EURO-VHDL '96
0-89791-848-7/96 $4.00

2. Mapping Approach - Overview

In this paper we consider a task-graph as a high level specification no matter what kind of implementation is intended for the different tasks. The mapping of tasks to processors under real-time conditions is a well known discipline in real time processing [13]. Our approach is restricted to those applications where the execution time of an algorithm is constant, that means it is also independent of specific input values. Many signal and image processing algorithms have this property. We can therefore statically allocate and schedule the tasks and there is no need to interrupt any task. Clearly, this makes things much simpler and allows the use of a mapping procedure resulting in an application specific real-time system with optimal allocation and schedule of the tasks.

The generic target architecture of this system is shown in Fig. 1. Normally it consists of one general purpose standard processor (e.g. microprocessor), different application specific components (ASICs) and application oriented standard processors (e.g. signal processors). In this paper we denote all these working units as processors. We assume that each processor has its own local memory and only one task can be executed at a time by one processor. Tasks on different processors can be executed in parallel. An execution of a task on a processor cannot be interrupted. Additionally we model a global memory as a processor with zero execution time. In this case we only have to consider 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. For communication each processor has its own communication processor.

The steps of a task mapping are shown in Fig. 2. 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 quality of the result of applying 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. The determination of execution times for each task is not easy since the implemented code or hardware structure has to be optimized by using optimizing compilers for software or hardware. 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.

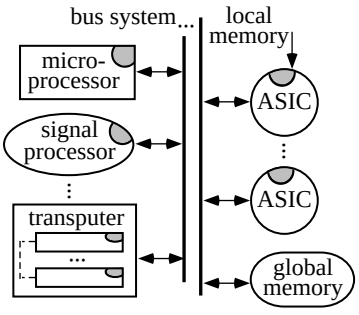

Fig. 1: Target architecture

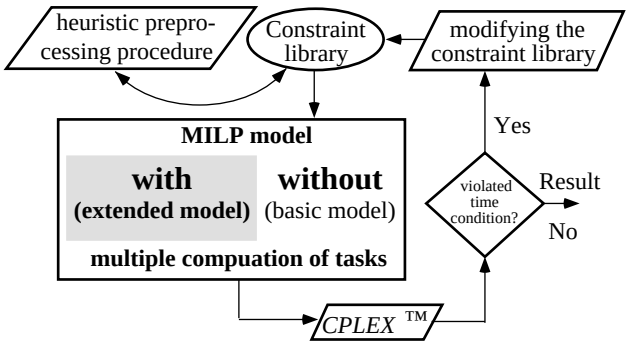

Fig. 2: Mapping approach

The usual goal is to meet the timing constraints of the underlying algorithm with respect to factors as mentioned above. To reach this goal we have introduced two MILP models (see Fig. 2). The first model [14] is our basic model for single computation of tasks (each task is performed once on one processor) and the second one is an extension dealing 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). In this paper we describe our second MILP model with multiple computation of tasks. In general, the designer can select one of these two MILPs depending on the task-graph complexity and the given real-time conditions. MILPs are automatically generated from the constraint library. For solving the MILP we use the standard software tool *CPLEX*™ [15]. For large mapping problems we have developed a heuristic for preprocessing [16]. As a result the domain (range of possible values) of the variables in the MILP is reduced and therefore the complexity of the MILP is decreased.

For practical examples some mapping iterations (see Fig. 2) may be necessary in order to meet the timing constraints. In each iteration step we have two choices. First, we can introduce more resources in our constraint library. Then we update the nodes of the annotated task-graph for those tasks, which can be executed on additional processors. Finally, a new MILP is generated with the updated information. Another choice is to switch to the second MILP model that supports multiple computations. If the value of the objective function for the solution of a MILP with single computation does not satisfy the demands of the user the solution can be used as a start solution of the MILP with multiple computations. This may result in a better value of the objective function. In any case, the start solution reduces the complexity of the more sophisticated MILP. If the previous problem was already based on the model with multiple computations we have only the first choice. Introducing new resources normally means to implement more and more tasks in hardware. The mapping procedure succeeds when 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 called "implemented in software". Tasks, which need special hardware (ASICs or signal processors) are called "implemented in hardware". In our mapping approach a real-time system is a composition of units executing tasks either implemented

in hardware or in software. The communication structure is based on multiple buses.

3. MILP model for multiple computation

We have described our basic model (each task is performed once on one processor) in [14, 17]. In section 3.3 we describe the MILP model that supports multiple computation of tasks. In [18] also a MILP is described but without multiple computation of tasks. First, we introduce notations used to formulate the MILP. Second, we explain how multiple computation can help to meet real-time constraints. We illustrate this by means of a small example.

3.1. Notation

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

$\mathcal{T}$	set of tasks $\mathcal{T} = \{T_1, ..., T_N\}$
$\mathcal{B}$	set of buses $\mathcal{B} = \{B_1, ..., B_L\}$
$\mathcal{P}$	set of processors $\mathcal{P} = \{P_0, ..., P_M\}$
G	directed acyclic task-graph. $G = (\mathcal{V}, \mathcal{E})$ where each vertex $i \in \mathcal{V}$ represents one task T_i; a directed arc $(i,j) \in \mathcal{E}$ from task T_i to T_j represents a precedence constraint between these tasks. For all $(i,j) \in \mathcal{E}$ it holds $i < j$.
$\rho(T)$	set of all processors able to execute task $T \in \mathcal{T}$
$RE(i)$	set of all vertices reachable from vertex $i \in \mathcal{V}$ in the given task-graph G
$\prec_3$	order relation with: $(i,j), (i',j') \in \mathcal{E}$ and $m, m' \in \mathcal{P}$: $(i,m,j) \prec_3 (i',m',j') \Leftrightarrow (m < m') \vee ((m = m') \wedge (i < i')) \vee ((m = m') \wedge (i = i') \wedge (j < j'))$

(i,m,j) represents a communication between task T_i and task T_j were data is sent to processor P_m, which is the processor T_j is allocated on. $RE(i)$ represents the set of tasks data dependent from task T_i. Nodes i with $RE(i) = \{\}$ are leaf nodes of the task-graph. The corresponding tasks are called leaf tasks. Each task $T \in \mathcal{T}$ 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:

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

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

$$h_{limj} = \begin{cases} 1 & \text{on bus } B_l \text{ data produced by task } T_i \text{ are} \\ & \text{sent to processor } P_m \text{ for executing task } T_j \\ 0 & \text{otherwise} \end{cases}$$

$$x_{ki} = \begin{cases} & \text{the processor } P_k \text{ that executes the task } T_i \\ 1 & \text{providing data for any task } T_j \text{ with } (i,j) \in \mathcal{E}; \text{ the} \\ & \text{two tasks are allocated on different processors} \\ 0 & \text{otherwise} \end{cases}$$

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

s_{mi}	start time of task T_i on processor P_m
b_{imj}	start time of communication between task T_i and T_j; task T_j is allocated on processor P_m

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

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 $(i,j) \in \mathcal{E}$

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 communication each processor has its own communication processor.

3.2. Motivation for multiple computation

In our basic model [14] multiple computation of tasks cannot be modeled since the constraints only allow to allocate a task on one processor at one time. Multiple computation means that a task may be performed on several processors in order to save communication overhead and to exploit more parallelism. In this manner we sometimes can meet real-time requirements, which cannot be met by just allowing single computation without introducing additional processors in our constraint library.

Example:

In Fig. 3b) we present a valid mapping of the tasks $T_i, T_j, T_{j'}$ on processors P_k, P_m without multiple computation and in c) with multiple computation for the underlying task-graph (see Fig. 3a).

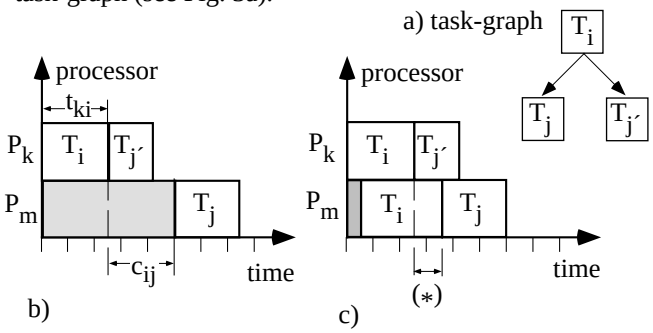

Fig. 3: Advantage of multiple computation

In Fig. 3b) let $P_k, P_m \in \mathcal{P}$ with $P_k \neq P_m$ and the tasks $T_i, T_{j'}$ are mapped on processor P_k and task T_j on processor P_m. Although $t_{mi} > t_{ki}$ may hold, if task T_i is additionally mapped on processor P_m (Fig. 3c) and is fulfilled

$$\underbrace{s_{mj} - (s_{ki} + t_{ki})}_{(*)} < c_{ij}$$

then the multiple computation of task T_i is better because the resulting execution time of the real-time system is reduced. ◇

If T_i is allocated on processor P_k a start time s_{ki} is determined. T_i can only be allocated once on processor P_k. Since we are dealing with multiple computation, T_i can be allocated on several processors $P_k \in \rho(T_i)$. We therefore may have n instances of T_i executed on n processors. For data dependent tasks $(T_i \rightarrow T_j)$ we have to observe that for each instance of T_j at least one instance of T_i must have finished before the instance of T_i can start in order to provide data for T_j. Furthermore, we have to observe communication time if no T_i is allocated on the processor T_j is allocated on.

With respect to the quality of a solution of a task mapping problem with time constraints an approach supporting multiple computation is superior to the basic model. On the other hand, the number of variables and therefore the complexity for the MILP for the extended model is much higher than for the basic model. Therefore, it is useful to solve at first the MILP without multiple computation (see Fig. 2). If there are some time restrictions violated in the solution we have two ways to proceed. If we can change our constraint library by including faster processors then we can update the annotated task-graph as described above and then we generate with these new information another MILP for the basic model. Alternatively, we formulate a modified MILP supporting multiple computation based on information of the MILP solution already computed. One modification e.g. is a reduction of the domain of variables. Another advantage is that the generated optimal solution for the basic model can be used as an initial solution for the extended model. In this way the computation time of the modified MILP with multiple computation is reduced.

3.3. Model with multiple computation

Now, we formulate the MILP for the model supporting multiple computation. 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). We want to minimize an objective function depending on the total execution time (*TET*), the total processor costs (*TPC*), and the total communication costs (*TCC*). The weights k_1, k_2 and k_3 of the costs *TET*, *TPC* and *TCC* have to be tuned by the designer.

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

$$minimize\left(k_1 \cdot TET + k_2 \cdot TPC + k_3 \cdot TCC\right) \qquad \text{(MOF)}$$

subject to objective function constraints, data dependency constraints, and resource conflict constraints:

objective function constraints

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

$\forall\, 0 \le m \le M$ processor P_m
$\forall\, 1 \le i \le N:$ task T_i

$$RE(i) = \{\ \} \wedge d_{mi} = 1 \quad \Rightarrow \quad s_{mi} + t_{mi} \le TET$$

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

$$s_{mi} + t_{mi} \ \le\ TET + \begin{cases} \infty & \text{if } d_{mi} = 0 \\ 0 & \text{if } d_{mi} = 1 \end{cases}$$

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

$$s_{mi} + t_{mi} \ \le\ TET + (1 - d_{mi}) \cdot\ C \qquad \text{(MOFC1)}$$

For internal reasons of the MILP solver C should be as small as possible. The sum $s_{mi} + t_{mi}$ in (MOFC1) represents the end time of the execution time of task T_i on processor P_m. For each computation of a leaf task we need an inequation.

The costs for processors and buses are taken from the constraint library. (MOFC2) models the total processor costs and (MOFC3) models the total communication costs.

$$TPC = \sum_{\substack{m=0 \\ \exists 1 \le i \le N: d_{mi} = 1}}^{M} \text{costs of processor } m \qquad \text{(MOFC2)}$$

$$TCC = \sum_{m=0}^{M} \sum_{\substack{l=1 \\ \exists (i,j) \in \mathcal{E}: h_{limj} = 1}}^{L} \text{costs of bus } l \qquad \text{(MOFC3)}$$

Conditions with existential quantification can be modeled in the MILP by introducing 0/1 variables. Because of the limited space the resulting constraints are presented elsewhere [16].

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 dependency constraints

If two tasks T_i and T_j in the task-graph are connected by an arc $(i, j) \in \mathcal{E}$ 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 M1a for execution of T_j on the same processor P_m). When the tasks are allocated on different processors $P_k \ne P_m$ then we additionally have to observe communication time c_{ij}, i. e. T_j can start c_{ij} time units after T_i has finished (modeled with M1b).

task execution times

$\forall 0 \le m \le M$ processor P_m
$\forall (i, j) \in \mathcal{E}$ 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

$$\Rightarrow \quad \underbrace{s_{mi} + t_{mi}}_{(*)} \le s_{mj} + \underbrace{(2 - d_{mi} - d_{mj}) \cdot C}_{(**)} \qquad \text{(M1a)}$$

(*) 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.

$$\forall 0 \le m,k \le M\ \forall (i,j) \in \mathcal{E} \wedge k \ne m \wedge P_k \in \rho(T_i) \wedge P_m \in \rho(T_j):$$

$$\Rightarrow \quad s_{ki} + t_{ki} + c_{ij} \le \quad s_{mj} \quad + \underbrace{\left(2 - d_{mj} + d_{mi} - x_{ki}\right) \cdot C}_{(***)} \tag{M1b}$$

The term $(***)$ is zero iff task T_i is allocated on processor P_k, T_i is not allocated on processor P_m, and processor P_m receives the computed data for task T_j from processor P_k.

Because we are supporting multiple computation a task $T \in \mathcal{T}$ can be allocated on several processors. We have to consider **all** possible multiple allocations of a task T on all processors $P \in \rho(T)$. Therefore, we have to introduce processor specific start times. Additionally, we have to know which processor provides the data for task T_j. This is done by variable x_{ki}. $x_{ki} = 1$ iff processor P_k provides data computed by task T_i for each task T_j with $T_i \rightarrow T_j$. This is one reason why the complexity in our extended model is much higher than in the basic model (much more variables and constraints).

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. For the allocation of tasks with direct precedence constraint allocated on different processors we have to consider the communication time. We assume that 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 (M2). The duration of a communication is observed in (M3).

beginning of communication

$$\forall 0 \le m,k \le M\ \forall (i,j) \in \mathcal{E} \wedge m \ne k \wedge P_m \in \rho(T_j) \wedge P_k \in \rho(T_i):$$

$$\Rightarrow \quad s_{ki} + t_{ki} \le b_{imj} + \underbrace{\left(1 - x_{ki}\right) \cdot C}_{\text{observation condition}} \tag{M2}$$

In the extended model, there may be several processors $PP_i = \{P_m \in \mathcal{P} \mid d_{mi} = 1\}$ executing T_i and several processors PP_j executing task T_j. The processor $P_k \in PP_i$ that provides data for each $P_m \in PP_j$ is identified by $x_{ki} = 1$, where $x_{mi} = 0$ for $m \ne k$ holds. Therefore, we need for each processors P_m, P_k and for each $(i,j) \in \mathcal{E}$ an inequation. In the basic model there is only one processor P_k executing task T_i. Therefore, we need only one inequation of this type for every $(i,j) \in \mathcal{E}$ in our basic model. b_{imj} denotes the start of the communication from processor P_k already executed task T_i to processor P_m going to execute task T_j. In this manner, the inequation (M2) is only observed if task T_i is allocated on processor P_k and task T_i provides data for each task T_j with $(i,j) \in \mathcal{E}$ and T_j is allocated on processor P_m. In this case $x_{ki} = 1$ holds. Any other processor executing an instance of T_i does not have to communicate despite data dependencies $T_i \rightarrow T_j$. The constraints for consistent x_{ki} are modeled in (M8a) and (M8b) (will be given later).

The observation condition in (M3) is zero if there is a communication between task T_i and T_j on some bus B_l and T_j is allocated on processor P_m. In inequation (M7´) (will be given later) we force that $h_{limj} = 1$ holds for at most one l. If the sum in $(****)$ is zero, T_i and T_j are executed on the same processor P_m.

duration of communication

$$\forall 0 \le m \le M\ \forall (i,j) \in \mathcal{E} \wedge P_m \in \rho(T_j):$$

$$\Rightarrow \quad \underbrace{\overbrace{b_{imj}}^{(*)} + \overbrace{c_{ij}}^{(**)}}_{(*****)} \le \overbrace{s_{mj}}^{(***)} + \overbrace{\left(1 - \sum_{l=1}^{L} h_{limj}\right) \cdot C}^{(****)} \tag{M3}$$

$(*)$ start of communication between T_i and T_j; T_j allocated on processor P_m
$(**)$ duration of communication between T_i and T_j
$(***)$ start of task T_j on processor P_m
$(****)$ observation condition
$(*****)$ end of communication between T_i and T_j

We now have introduced the constraints that observe data dependencies. In the next section we introduce the constraints that avoid resource conflicts.

Resource conflict constraints

Two tasks must not be executed on the same processor at the same time. For data dependent tasks, this is ensured by (M1a). For all other pairs of tasks, we have to introduce nonoverlapping constraints (M4). With the first inequation type (M4a) we describe the possibility that task T_i will be executed after task T_j on processor P_m (i. e. $y_{mij} = 1$). In (M4b) we formulate the other possibility (i. e. $y_{mji} = 1$).

nonoverlapping constraints for executions on processors

$$\forall 0 \le m \le M\ \forall 0 \le i,j \le N \wedge P_m \in \rho(T_i) \cap \rho(T_j) \wedge \underbrace{i \notin RE(j)}_{(*)} \wedge j \notin RE(i):$$

$$\Rightarrow \quad s_{mi} + t_{mi} \le s_{mj} + \left(3 - d_{mi} - d_{mj} - y_{mij}\right) \cdot C \tag{M4a}$$

$$s_{mj} + t_{mj} \le s_{mi} + \left(3 - d_{mi} - d_{mj} - y_{mji}\right) \cdot C \tag{M4b}$$

$(*)$ T_j has not been executed after completion of T_i because of data dependencies; This avoids inequations which would be implied by (M1a) and (M1b).

y_{mij} and y_{mji} are only relevant for $d_{mi} = 1 \wedge d_{mj} = 1$. In this case we can assume $y_{mji} = 0 \Leftrightarrow y_{mij} = 1$ and $y_{mji} = 1 \Leftrightarrow y_{mij} = 0$. 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 observation condition in (M4b) indicating the relevance of the inequation (see notes for M1a) has to be replaced by $\left(2 - d_{mi} - d_{mj} + y_{mij}\right) \cdot C$. In this manner, the value of y_{mij} decides which of (M4a) or (M4b) is relevant for $d_{mi} = 1 \wedge d_{mj} = 1$. For $d_{mi} = 1 \wedge\ d_{mj} = 1$ the value of y_{mij} is irrelevant, because (M4a) and (M4b) are relevant in any case.

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 (MOFC3) 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*:

$$\forall 0 \le m,m' \le M\ \forall (i,j),(i',j') \in \mathcal{E} \wedge (i,m,j) \prec_3 (i',m',j') \wedge P_m \in \rho(T_j) \wedge P_{m'} \in \rho(T_{j'})\ \forall 1 \le l \le L: \tag{M5}$$

$$\Rightarrow b_{imj} + c_{ij} \le b_{i'm'j'} + \left(3 - h_{li'm'j'} - h_{limj} - w_{(imj)(i'm'j')}\right) \cdot C$$
$$b_{i'm'j'} + c_{i'j'} \le b_{imj} + \left(2 - h_{li'm'j'} - h_{limj} + w_{(imj)(i'm'j')}\right) \cdot C$$

The inequations in (M5) are defined according to (M4a) and (M4b). In addition, to the observation of data dependencies and to the avoidance of resource conflicts we need the constraints (M6) to (M9b). We have to ensure that each task will be computed:

$$\forall 1 \le i \le N: \quad \sum_{\substack{m=0 \\ P_m \in \rho(T_i)}}^{M} d_{mi} \ge 1 \qquad \text{(M6)}$$

The following condition ensures that each communication will be allocated only on one bus:

$$\forall 0 \le m \le M \; \forall (i,j) \in \mathcal{E}: \sum_{l=1}^{L} h_{limj} = \left(d_{mj} \wedge \neg d_{mi}\right) \qquad \text{(M7)}$$

This condition models that we need exactly one bus if $d_{mj} = 1$ (task T_j is allocated on processor P_m) and $d_{mi} = 0$ holds. In this case it holds $h_{limj} = 1$. A possible formulation of this condition in a MILP is given in (M7´). Because multiple computation of tasks is possible we have to determine for the formulation of the beginning of a communication (see M2) what task instance can provide the data. (M8a) models that x_{ki} is 1 for exactly one k. We have to ensure that $x_{ki} = 1 \Rightarrow d_{ki} = 1$ holds. This ensures that we only can send the computed data from processor P_k if task T_i is allocated on this processor. This is formulated in inequation (M8b). To keep all start times positive we need inequations (M9a) and (M9b).

$$\forall 0 \le m \le M \; \forall (i,j) \in \mathcal{E}:$$
$$\left(d_{mj} - d_{mi}\right) \le 2 \cdot \sum_{l=1}^{L} h_{limj} \le \left(d_{mj} - d_{mi}\right) + 1 \qquad \text{(M7´)}$$

$$\forall 1 \le i \le N: \quad \sum_{k=0}^{M} x_{ki} = 1 \qquad \text{(M8a)}$$

$$\forall 0 \le k \le M \; \forall 1 \le i \le N \wedge P_k \in \rho(T_i): \quad x_{ki} \le d_{ki} \qquad \text{(M8b)}$$

$$\forall 0 \le m \le M \; \forall 1 \le i \le N \wedge P_m \in \rho(T_i): \quad s_{mi} \ge 0 \qquad \text{(M9a)}$$

$$\forall 0 \le m \le M \; \forall (i,j) \in \mathcal{E} \wedge P_m \in \rho(T_j): \quad b_{imj} \ge 0 \qquad \text{(M9b)}$$

Now the formalization of the MILP model with multiple computation is complete. The inequations (M2), (M3) and (M7´) imply the inequation (M1b). Therefore, we do not have to regard this restriction in our MILP. In this manner, the complexity of our overall MILP can be handled.

4. Experimental Results

In this section we first consider an image processing application, which is a practicable example of reasonable size. We illustrate the advantage of models supporting multiple computation. Then we present statistics for four other task-graphs.

4.1. Results of a typical application

Our example is a typical complex image processing application based on the CCITT recommendation H.261 [19]. This application consists of several tasks with execution times independent of the input values. Each of the 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. Vertices are labeled with the set of processors able to execute the corresponding task and the individual execution time. Arcs are labeled with the communication time between two tasks if they are allocated on different processors. Times are given in terms of clock cycles.

The constraint library contains one microprocessor (UNIVERSAL), one signal processor (SIG_PROC) and 9 different ASIC´s (BMA_PIPE, ...) on which we can allocate the tasks. The vertex label uses a mnemonic abbreviation for the tasks. For example DCT denotes the Discrete Cosine Transformation. We have supplemented the original description by adding for each task the execution time if the task would be executed on the microprocessor. Furthermore, we assume that the tasks DCT, Q, IQ and C can be executed on the signal processor. Tasks IN, FB1 and FB2 model access to external memories. We model the FB access to memory with FB1 for read and FB2 for write in order to avoid cycles in the task-graph. These tasks as well as the tasks Q and IQ are used only to model the interaction with the environment. Therefore, the execution times are set to zero.

4.1.2. Mapping - Results

A timing requirement $s_C + t_C \le 4800$, where s_C denotes the start time of task C on the allocated processor and t_C denotes the execution time of C on this processor, represents the constraint that the execution of task C has to be finished before 4800 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 [20] for computing the ASAP- (as soon as possible) and ALAP-times (as late as possible) for each task and communication. We computed 7100 clocks as an upper bound for the total execution time.

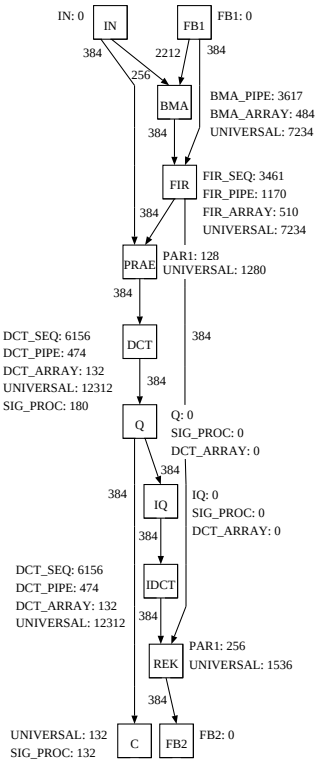

Fig. 4: Task-graph

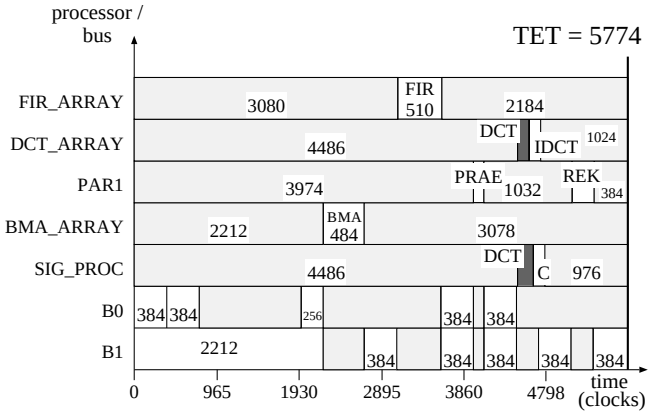

Fig. 5: Supporting multiple computation

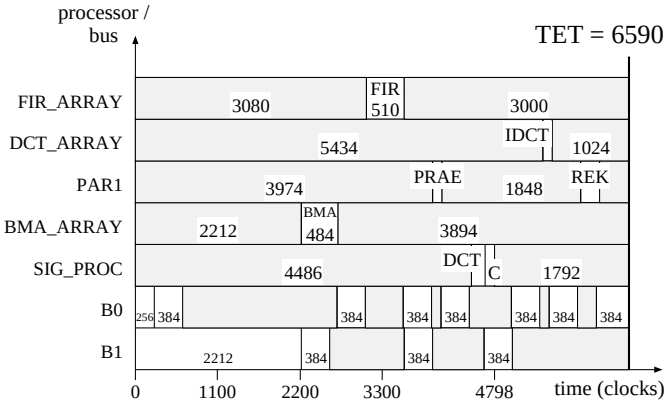

Fig. 7: Without multiple computation

These bounds are used to reduce the computation time of the MILP solver. A feasible solution could be computed, which determines a real-time system for which all tasks could be allocated and scheduled without violating timing constraints. We have chosen a cost function for minimizing the total execution time (*TET*) and the total communication costs (*TCC*).

The MILP generated for the model supporting multiple computation consists of 718 (147) inequations with 249 (86) variables. The values in parentheses refer to the MILP without multiple computation (i. e. basic model). Both MILPs were solved using CPLEX™ [15]. The CPU time on a SPARCstation 20 was 84 (0.36) seconds. The weights k_1 and k_3 in the objective function were 0.5 ($k_2 = 0$).

The results from scheduling and allocation are shown in Fig. 5 and 7 using automatically generated Gantt diagrams, which are standard illustrations for tasks executed in parallel on several processors. The new designed real-time system consists in both cases 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 3974 clocks and its execution time is 128 clocks (see Fig. 4) on the processor PAR1. At the beginning we have a data transfer FB1 from the memory to the BMA_ARRAY, which needs 2212 clocks for the communication on bus B1. The architecture for the designed real-time system corresponding to Fig. 5 is shown in Fig. 6. For the other model corresponding to Fig. 7 we need the same components but a different bus structure. For both real-time systems, the allocation and the scheduling is optimal for the given task-graph (Fig. 4) with respect to *TET*, *TCC* and the given constraint library.

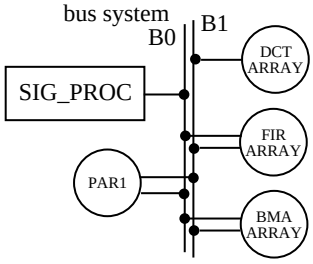

Fig. 6: Designed system

The total execution time of the video coding algorithm H.261 on this real-time system is 5774 clocks when we use the MILP model supporting multiple computation. Without multiple computation the total execution time is 6590 clocks. With the multiple computation of task DCT (see Fig. 5) we can avoid the communications between the tasks DCT-IQ-IDCT. This is the reason why the total execution time is reduced by about 816 clocks without introducing additional hardware components. In both cases the timing constraint for task C is met, because the execution of C is finished after 4798 clocks.

In Fig. 6 it can also be seen that we have to realize six tasks on four different ASIC´s and only one task (C) is software implemented on the signal processor (SIG_PROC). Therefore, we have six hardware implemented tasks and only one software implemented task. Tasks with execution time zero are omitted. Task DCT is computed twice on two processors (DCT_ARRAY, SIG_PROC, see Fig. 5).

4.2. Further results

In this section we present the results of applying our mapping approach on four typical task-graphs. In Table 1 we present statistics for each annotated task-graph. Each task can be performed on the microprocessor. Normally, we have at least one additional processor (ASIC) for each task. In example 1 and 3 the communication times are less than the execution times of the tasks on the processors. In example 2 and 4 the communication times are much greater. For each example we provide two buses. In Table 2 we show statistics for the automatically generated MILPs, the time for solving the MILPs with *CPLEX*™ and the resulting total execution time. We present results for our basic model [14] (run 1) and the model supporting multiple computation (run 2).

	# arcs	# tasks	max. of parallelism	# available processors
example 1	12	8	3	5
example 2	17	12	4	4
example 3	10	8	4	3
example 4	7	8	2	4

Table 1: Structure of the annotated task-graphs

For these annotated task-graphs we obtained a real-time systems, allocation, and scheduling optimal and feasible with respect to the corresponding constraint library. In

example 1 we could not reduce the total execution time by allowing multiple computation. In example 2, 3 and 4 we could reduce the total execution time without introducing additional hardware components. To achieve this reduction it was sufficient to switch to the model with multiple computation. For solving the MILPs we have used ASAP- and ALAP-times obtained in the preprocessing step mentioned above. The results show that an automatic and optimal hardware/software codesign for real-time systems is supported in a reasonable way.

	# const.	# var.	CPU sec.	*TET*
example1 / run 1	190	87	1.10	341.2
example1 / run 2	200	90	0.2	341.2
example2 / run 1	665	238	115.56	320
example2 / run 2	719	250	9374.28	260
example3 / run 1	209	98	0.4	198
example3 / run 2	342	137	0.7	174
example4 / run 1	162	82	7.69	240
example4 / run 2	254	143	27.03	210

Table 2: Further results

6. Conclusion

We have presented an approach for automatically designing an application specific real-time system that can perform an image or signal processing algorithm observing real-time constraints. In our mapping approach, tasks obtained by splitting the algorithm are mapped to hardware units (e.g. ASICs) or software units (e.g. microprocessors) available from a constraint library. We have allowed that a task may be performed on several processors. A Mixed Integer Linear Programming (MILP) model has been developed that defines correct allocations, i.e. correct assignments of tasks to units that have to perform the task, and correct schedules, i.e. correct assignments of start times to tasks. The model also observes communication overhead that arises with data exchange between tasks executed on different processors. Solving the MILPs leads to designs optimal with respect to given resources. The cost functions can be tuned by the designer in order to determine the weights of hardware costs and execution time. The practicability of our approach has been turned out by solving MILPs for several image and signal processing algorithms. For these reasons, this paper has presented a powerful model that supports an automatic and optimal hardware/software codesign for real-time systems for image and signal processing applications. A main topic for future research is to implement a variant of a MILP solver, which is especially designed for solving the described type of MILPs.

Acknowledgments

I am very grateful to Matthias Mutz for his inspiring and indeed very valuable comments and ideas on this work. I also acknowledge Peter Scholz and Erwin Harbeck for their help by formulating the model and by computing experimental results, respectively.

References

[1] N. Woo, W. Wolf, A. Dunlop: *Compilation of a single specification into hardware and software*, Handouts of the 1st International Workshop on Hardware/Software Codesign, Estes Park, Colorado, 1992.

[2] S. Som, R. Mielke, W. Stoughton: *Prediction of Performance and Processor Requirements in Real-Time Data Flow Architectures*, IEEE Trans. on Parallel and Distributed Systems, Vol. 4, No. 11, 1993, pp.1205-1216.

[3] E. Barrows, et. al.: *Hardware/Software Partitioning with UNITY*, Handouts of the 2nd International Workshop on Hardware/Software Codesign, Cambridge, 1993.

[4] H. Achatz: *Extended 0/1 LP Formulation for the Scheduling Problem in High-Level Synthesis*, EURO-DAC, Hamburg, 1993, pp. 226 - 231.

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

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

[7] W. Hardt, et. al.: *Specification Analysis for HW/SW-Partitioning*, 3th GI/ITG Workshop Anwendung formaler Methoden beim Entwurf von Hardwaresystemen (eds. W. Grass, M. Mutz; Passau), Shaker , Aachen, 1994, pp.1-10.

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

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

[10] J. Buck, et. al.: Ptolemy: *A Framework for Simulating and Prototyping Heterogeneous Systems*, Int. Journal in Computer Simulation, Vol.4, No. 2, 1994, pp.155-182.

[11] A. Kalavade, A. Lee: *A Hardware-Software Codesign Methodology for DSP Applications*; IEEE Design & Test of Computers, Vol. 10, No. 3, 1993, pp. 16-28.

[12] A. Bender: *Design of an Optimal Loosely Coupled Heterogeneous Multiprocessor System*, European Design & Test Conference, Paris, 1996, pp. 275-281.

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

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

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

[16] E. Harbeck: *Extension and improvement of several mapping procedures for the design of multiprocessor systems with hard real-time constraints* (In German.), diploma thesis, University of Passau, 1995.

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

[18] R. Niemann, P. Marwedel: *Hardware/Software Partitioning using Integer Programming*, European Design & Test Conference, Paris. 1996, pp. 473-479.

[19] M. Schwiegershausen, M. Schönfeld, P. Pirsch: *Mapping Complex Image Processing Algorithms onto Heterogeneous Multiprocessors Regarding Architecture Dependent Performance Parameters*, Proceedings of the 3rd Int. Workshop on Algorithms and Parallel VLSI architectures, Leuven, Belgium, 1994.

[20] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller: *Equation of State Calculations for Fast Computing Machines*, Journal of Chemical Physics, Vol. 21, 1953.