Hardware/Software Partitioning using Integer Programming

Ralf Niemann, Peter Marwedel

Dept. of Computer Science XII
University of Dortmund
D-44221 Dortmund, Germany

Abstract

One of the key problems in hardware/software codesign is hardware/software partitioning. This paper describes a new approach to hardware/software partitioning using integer programming (IP). The advantage of using IP is that optimal results are calculated respective to the chosen objective function. The partitioning approach works fully automatic and supports multi-processor systems, interfacing and hardware sharing. In contrast to other approaches where special estimators are used, we use compilation and synthesis tools for cost estimation. The increased time for calculating the cost metrics is compensated by an improved quality of the estimations compared to the results of estimators. Therefore, fewer iteration steps of partitioning will be needed. The paper will show that using integer programming to solve the hardware/software partitioning problem is feasible and leads to promising results.

1 Introduction

Embedded systems typically consist of application specific hardware parts and programmable parts, i.e., processors like DSPs, core processors or ASIPs. In comparison to the hardware parts, the software parts can be developed and modified much easier. Thus, software is less expensive in terms of costs and development time. Hardware however, provides better performance. For this reason, a system designer's goal is a system which fulfills all performance constraints by using as few as possible hardware. Hardware/software codesign deals with the problem of designing embedded systems, where automatic partitioning is one key issue. This paper describes a new approach in hardware/software partitioning for multiprocessor systems working fully automatic. The approach is based on integer programming (IP) to solve the partitioning problem optimally. A formulation of the IP-model will be introduced in detail. The drawback of solving IP-models often is a high computation time. To reduce the computation time, a second approach has been developed which splits the partitioning approach in two phases. In a first phase, a mapping of nodes to hardware or software is calculated by estimating the schedule times for each node with heuristics. During the second phase a correct schedule is calculated for the resulting HW/SW-mapping of the first phase. It will be shown that this *heuristic scheduling* approach strongly reduces the computation time while the results are nearly optimal for the chosen objective function.

Another new feature of our approach is the cost estimation technique. The cost model is not calculated by estimators like other approaches, because the quality of estimations is often bad and estimators do not concern compiler effects. In our approach the tools (a compiler for the software parts and a high-level synthesis tool for the hardware parts) are used instead of special estimators. The disadvantage of an increased runtime for calculating the cost metrics is compensated by a better quality of the cost metrics compared to the results of estimators. Furthermore, better cost metrics lead to fewer partitioning iterations.

The outline of the paper is as follows: Section 2 gives an overview of related work in the field of hardware/software partitioning. In section 3 our own approach to partitioning is presented. A formulation of the hardware/software partitioning problem follows in section 4. Section 5 describes the problem by an IP-model. After experimental results of solving these IP-models have been presented in section 6, a conclusion is given in section 7 .

2 Related Work

There are only few approaches considering hardware/software partitioning. One of these is the

COSYMA system [EHB93], where hardware/software partitioning is based on simulated annealing using estimated costs. The partitioning algorithm is *software-oriented*, because it starts with a first non-feasible solution consisting only of software components. In an *inner loop partitioning (ILP)* software parts of the system are iteratively realized in hardware until all timing constraints are fulfilled. To handle discrepancies between estimated and real execution time, an *outer loop partitioning (OLP)* restarts the *ILP* with adapted costs [HE94]. The *OLP* is repeated until all performance constraints are fulfilled. Another hardware/software partitioning approach is realized in the *VULCAN* system [GCM92]. This approach is *hardware-oriented*. It starts with a complete hardware solution and iteratively moves parts of the system to the software as long as the performance constraints are fulfilled. In this approach performance satisfiability is not part of the cost function. For this reason, the algorithm will easily trap in a local minimum. The approach of Vahid [VGG94] uses a relaxed cost function to satisfy performance in an inner partitioning loop and to handle hardware minimization in an outer loop. The cost function consists of a very heavily weighted term for performance and a second term for minimizing hardware. The authors present a *binary-constraint search algorithm* which determines the smallest size constraint (by binary search) for which a performance satisfying solution can be found. The partitioning algorithm minimizes hardware, but not execution time. Kalavade and Lee [KL94] present an algorithm (GCLP) that determines for each node iteratively the mapping to hardware or software. The GCLP algorithm does not use a hardwired objective function, but it selects an appropriate objective according a global time-criticality measure and another measure for local optimum. The results are close to optimal and the runtime grows quadratically to the number of nodes. This approach has been extended to solve the extended partitioning problem [KL95] including the implementation selection problem.

3 Hardware/Software Partitioning Approach

Our hardware/software partitioning approach is depicted in figure 1. The designer has to specify the **target architecture** by defining the set of processors for the software parts and the component library to synthesize the hardware parts. The **system** has to be defined in VHDL as a set of interconnected instances of VHDL-entities. Moreover, the designer has to determine the **design constraints**, containing performance constraints (timing) and resource constraints (area, memory). Then, the VHDL specification is compiled into an internal syntax graph model. For each entity of this model, software source code (C or DFL) and hardware source code (VHDL) is generated. The software parts are compiled and the hardware parts are synthesized by a high-level synthesis tool (OSCAR [LMD94]). The results are software cost metrics (software execution time, memory usage) and hardware cost metrics (hardware execution time, area) for the entities. The disadvantage of an increased runtime for calculating the cost metrics is compensated by a better quality compared to the results of estimators. Moreover, better cost metrics lead to fewer partitioning iterations. After the compilation/synthesis phase a partitioning graph is generated. Nodes represent the instances of VHDL-entities of the system and edges represent the interconnections between them. The nodes are weighted with the hardware and software costs, the edges are weighted with interface costs which occur if an interface is used between the nodes of the edge. The interface costs are approximated by the number and type of data flowing between both nodes. The user-defined design constraints are also matched to the graph. Thus, the partitioning graph includes all information needed for partitioning. The partitioning graph is then trans-

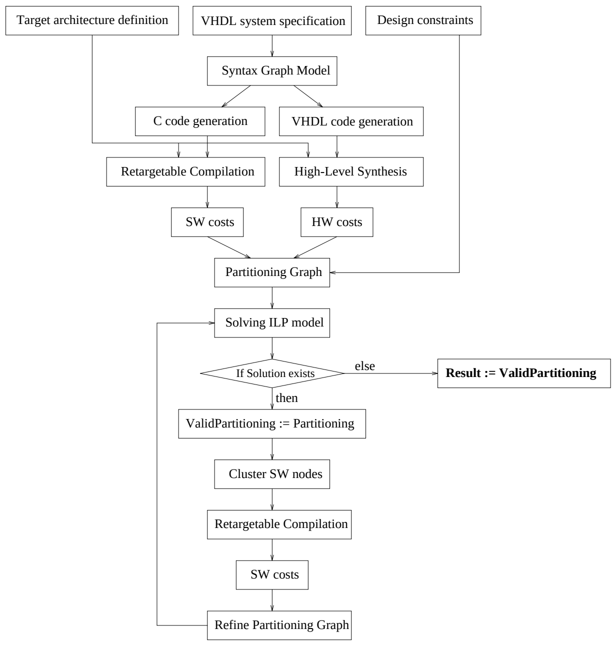

Figure 1: Hardware/Software Partitioning

formed into an IP-model, which is the key issue of this paper. Afterwards, the model is solved by an IP-solver. The calculated design is optimal for the generated cost model, but nevertheless it is possible to improve the design, because sharing between different instances of same entities is considered, but not sharing effects between different entities. This disadvantage can be removed by an iterative partitioning approach. We use a software oriented approach, because compilation is faster than synthesis and software oriented approaches seem to be superior to hardware oriented approaches (see [VGG94]). Sets of nodes which have been mapped on the same processor are clustered. For each cluster a new cost metric is calculated by compiling all nodes of the cluster together. Then, the partitioning graph is transformed by replacing each cluster by a new node attached with the new cost metric. Finally, the redefined graph is repartitioned. This iteration will be repeated until no solution is found. The last valid partitioning represents the resulting design. The clustering technique is illustrated in figure 2.

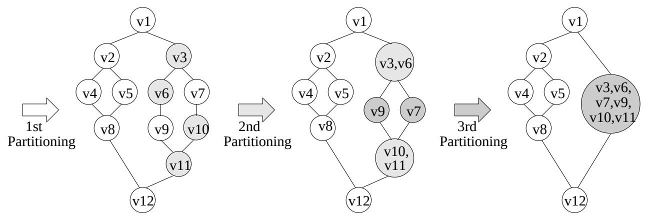

Figure 2: Partitioning refinement

4 Formulation of the HW/SW Partitioning Problem

This section introduces a formulation of the hardware/software partitioning problem. This formulation is necessary to simplify the description of the problem with the help of an IP-model. We have to define the target architecture used to realize the system and the system itself which has to be partitioned.

Definition 4.1 *The* **target architecture** *consists of an ASIC h, a set of processors $\mathcal{P} = \{p_1, \ldots, p_{n_P}\}$, external memory and busses between them. The set of target architecture components is defined as: $\mathcal{TA} = \{h\} \cup \mathcal{P}$.*

To simplify the notations in the following chapters, let the ASIC be the first element of $\mathcal{TA}$ with index 0, followed by the processors: $ta_0 := h; ta_k := p_k, \forall k \in \{1, \ldots, n_P\}$.

Definition 4.2 *A* **system** *is defined as a tuple $\mathcal{S} = (\mathcal{E}, V, E, I)$ with the following definitions:*
$\mathcal{E} = \{en_1, \ldots, en_{n_\mathcal{E}}\}$ defines the set of entities. The set of nodes $V = \{v_1, \ldots, v_{n_V}\}$ consists of instances of entities, defined by the function $I : V \rightarrow \mathcal{E}$. The set of edges $E \subset V \times V$ represents the interconnections between nodes.

The following **cost metrics** are defined for each entity en_l: $c^a(en_l)$ represents the hardware area, $c^{th}(en_l)$ the hardware execution time, $c^{dm}(en_l)$ the used software data memory, $c^{pm}(en_l)$ the used software program memory and $c^{ts}(en_l)$ the software execution time. The costs $c^a(v_j)$, $c^{th}(v_j)$, $c^{dm}(v_j)$, $c^{pm}(v_j)$ and $c^{ts}(v_j)$ for the instances v_j of an entity en_l are equal to the costs of en_l:

$$I(v_j) = en_l \qquad \Rightarrow \qquad c^x(en_l) = c^x(v_j) \tag{1}$$

The following interface costs for an edge $e = (v_1, v_2)$ are considered: $ci^a(e)$ defines the additional hardware area and $ci^t(e)$ defines the communication time for e.

A **design** represents the realization of a system $\mathcal{S}$ on a target architecture $\mathcal{TA}$. The **design quality** can be expressed by the following **design metrics**: $C^a(S)$ represents the hardware area, $C^{pm}(S)$ the used software program memory, $C^{dm}(S)$ the used software data memory and $C^t(S)$ the total execution time of $\mathcal{S}$. The set of **design constraints** $\mathcal{C}$ consists of $MAX^a(S)$, $MAX^{pm}(S)$, $MAX^{dm}(S)$ and $MAX^t(S)$ according to the design metrics of $\mathcal{S}$.

Definition 4.3 *The* **hardware/software partitioning problem** *is the problem of finding a mapping $map : V \rightarrow \mathcal{TA}$ in such a way that all performance and resource constraints are fulfilled and the design costs are minimized.*

The definitions will be used in the following example:
Example 1:

In figure 3 a system is specified consisting of 2 entities en_1 (circle), en_2 (box) and 7 instances $v_1, \ldots, v_7$ of these entities. This system will be partitioned for a target architecture containing one ASIC, one DSP, memory and a bus connecting these components.

5 The IP-Model

Linear optimization problems can be solved optimally by using integer programming (IP). This paper

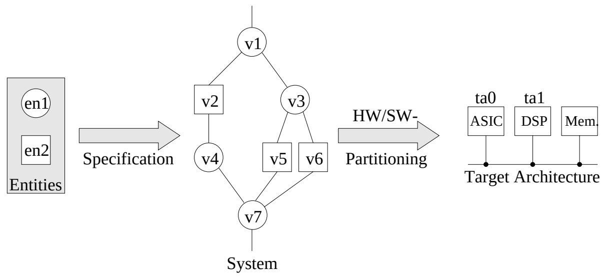

Figure 3: Unpartitioned system

will show that our IP-model is able to solve the hardware/software partitioning problem with the following characteristics: multiprocessor systems are supported, timing constraints are guaranteed, interface costs are included, sharing effects between different instances of the same entity are considered, and user constraints can be adapted easily. To describe the IP-model the following notations are necessary:

Definition 5.1 *Let $J = \{1, \ldots, n_V\}$ represent the indices of $v_j \in V$, $K = \{0, \ldots, n_P\}$ the indices of elements $ta_k \in \mathcal{TA}$ and $L = \{1, \ldots, n_{\mathcal{E}}\}$ the indices of elements $en_l \in \mathcal{E}$.*
Let $c^x_{l,k}$ be the cost metric $c^x(en_l)$ for entity en_l and $c^x_{j,k}$ the cost metric $c^x(v_j)$ for node v_j on target architecture component ta_k.
Let C^x_k be the system cost $C^x(S)$ on ta_k of system $\mathcal{S}$ and MAX^x_k the according maximum of C^x_k.
Let T^S_j be the execution starting time of node v_j.
Let T^D_j be the execution time of node v_j.
Let T^E_j be the execution ending time of node v_j.

5.1 The Decision Variables

Our IP-model uses the following 0/1-variables:

Definition 5.2 *Let the following 0/1-variables be defined as:*

$$x_{j,0} = \begin{cases} 1 & : \ v_j \text{ is not shared on } ta_0, \\ 0 & : \ \text{otherwise.} \end{cases}$$

$$y_{j,k} = \begin{cases} 1 & : \ v_j \text{ is shared on hardware } ta_0, \\ 1 & : \ \text{processor } ta_k (k \geq 1) \text{ executes } v_j, \\ 0 & : \ \text{otherwise.} \end{cases}$$

$$sh_{l,k} = \begin{cases} 1 & : \ en_l \text{ is shared on hardware } ta_0, \\ 1 & : \ \text{processor } ta_k (k \geq 1) \text{ executes } en_l, \\ 0 & : \ \text{otherwise.} \end{cases}$$

$$i_{j_1,j_2} = \begin{cases} 1 & : \ v_{j_1} \text{ and } v_{j_2} \text{ need an interface,} \\ 0 & : \ \text{otherwise.} \end{cases}$$

$$b_{j_1,j_2,k} = \begin{cases} 1 & : \ v_{j_1}, v_{j_2} \text{ are executed on} \\ & : \ \text{different components,} \\ 1 & : \ v_{j_1} \text{ ends before } v_{j_2} \text{ starts on } ta_k, \\ 0 & : \ \text{otherwise.} \end{cases}$$

Example 2:

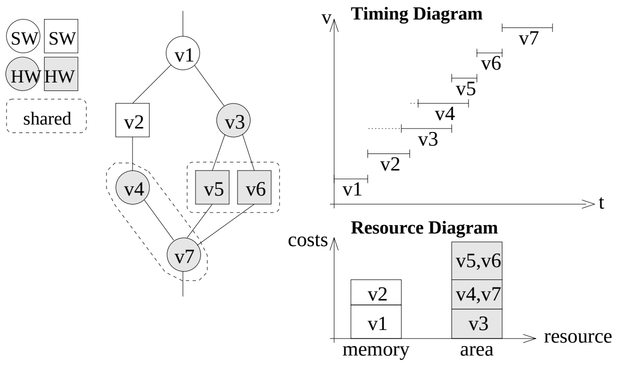

Figure 4: Partitioned system

The result of hardware/software partitioning of the system depicted in figure 3 is shown in figure 4. Gray shaded nodes are realized in hardware. Shared nodes are enclosed by a dashed line. The following table shows the 0/1-variables for executing nodes shared or not shared on hardware or software.

HW/SW	$v_j \rightarrow$ var	v_1	v_2	v_3	v_4	v_5	v_6	v_7
unshared HW	$x_{j,0}$	0	0	1	0	0	0	0
shared HW	$y_{j,0}$	0	0	0	1	1	1	1
SW	$y_{j,1}$	1	1	0	0	0	0	0

The nodes v_1 and v_2 are executed on the processor ta_1 ($y_{1,1} = y_{2,1} = 1$). Therefore, $sh_{\underline{1},1} = sh_{\underline{2},1} = 1$, because v_1 is of entity type $en_{\underline{1}}$ and v_2 is of entity type $en_{\underline{2}}$. v_3 is executed unshared on the hardware ($x_{3,0} = 1$). The other four nodes $v_4, \ldots, v_7$ are executed shared on the hardware. In total, 2 interfaces are needed: $i_{2,4} = i_{1,3} = 1$. The timing diagram shows that unshared nodes in hardware (v_3) can be executed in parallel to instances of the same entity (v_4). Shared nodes (v_5, v_6) have to be sequentialized, but result in less hardware area as shown in the resource diagram.

5.2 The Constraints

The following constraints have to be fulfilled:

1. **General Constraints**: Each node v_j is executed exactly on one target architecture component ta_k.

$$\boxed{\forall j \in J: \qquad x_{j,0} + \sum_{k \in K} y_{j,k} = 1 \qquad (2)}$$

2. **Resource Constraints**: The values for used data memory C_k^{dm} (eq. 3) and program memory C_k^{pm} (eq. 4) on each processor ta_k may not exceed a given maximum. The used hardware area C_0^a (eq. 5) is the sum of hardware area of unshared instances, shared entities, and the total interface area CI_0^a (eq. 14). C_0^a may not exceed a given maximum.

$$\forall k \in K \backslash \{0\}: \quad C_k^{dm} = \sum_{l \in L} sh_{l,k} * c_{l,k}^{dm} \leq MAX_k^{dm} \tag{3}$$

$$\forall k \in K \backslash \{0\}: \quad C_k^{pm} = \sum_{l \in L} sh_{l,k} * c_{l,k}^{pm} \leq MAX_k^{pm} \tag{4}$$

$$C_0^a = \sum_{j \in J} x_{j,0} * c_{j,0}^a + \sum_{l \in L} sh_{l,0} * c_{l,0}^a + CI_0^a \leq MAX_0^a \tag{5}$$

3. **Timing Constraints**:
The timing costs cannot be calculated by accumulating the execution time of the nodes, because nodes, that are not shared on the ASIC can be executed in parallel. To determine the starting time and ending time for each node, scheduling has to be performed. The execution time T_j^D (eq. 6) of v_j is either the hardware or the software execution time. The ending time T_j^E (eq. 7) is the sum of starting time T_j^S and execution time T_j^D. The starting times T_j^S (eq. 9) of nodes have to be in their ASAP/ALAP-range which can be calculated in a preprocessing step. Data dependencies (eq. 10) have to be considered for all edges $e = (v_{j_1}, v_{j_2})$ including interface communication time T_{j_1,j_2}^I of equation 12. The system execution time C^t (eq. 11) is the maximum of all ending times and may not violate the constraint.

$$\forall j \in J : \forall e = (v_{j_1}, v_{j_2}) \in E :$$

$$\begin{aligned}
T_j^D &= x_{j,0} * c_{j,0}^{th} + y_{j,0} * c_{j,0}^{th} + \sum_{k \in K \backslash \{0\}} y_{j,k} * c_{j,k}^{ts} && (6)\\
T_j^E &= T_j^S + T_j^D && (7)\\
& && (8)\\
ASAP(v_j) &\leq T_j^S \leq ALAP(v_j) && (9)\\
T_{j_2}^S &\geq T_{j_1}^E + T_{j_1,j_2}^I && (10)\\
T_j^E &\leq C^t \leq MAX^t && (11)
\end{aligned}$$

5.3 Interfacing

An interface has to be realized for an edge $e = (v_{j_1}, v_{j_2})$, if v_{j_1} and v_{j_2} are realized on different target architecture components. This fact is formulated with help of additional constraints for the interface 0/1-variable i_{j_1,j_2} (see [NM95]). Then, the following interface costs can be calculated: interface execution time T_{j_1,j_2}^I (eq. 12), interface hardware area A_{j_1,j_2}^I (eq. 13), and the area of all interfaces CI_0^a (eq. 14).

$$\forall e = (v_{j_1}, v_{j_2}) \in E :$$

$$T_{j_1,j_2}^I = i_{j_1,j_2} * ci_{j_1,j_2}^t \tag{12}$$

$$A_{j_1,j_2}^I = i_{j_1,j_2} * ci_{j_1,j_2}^a \tag{13}$$

$$CI_0^a = \sum_{e=(v_{j_1},v_{j_2}) \in E} A_{j_1,j_2}^I \tag{14}$$

5.4 Sharing

An entity en_l is shared on hardware ta_0 (eq. 15), if at least two nodes v_{j_1}, v_{j_2} which are instances of entity en_l are executed shared on ta_0. An entity en_l is shared on processor ta_k (eq. 16), if at least one instance of entity en_l is executed on ta_k.

$$\forall l \in L : \forall j_1, j_2 \in J : I(v_{j_1}) = I(v_{j_2}) = en_l :$$

$$sh_{l,0} \geq y_{j_1,0} + y_{j_2,0} - 1 \tag{15}$$

$$\forall k \in K \backslash \{0\} : \forall l \in L : \forall j \in J : I(v_j) = en_l :$$

$$sh_{l,k} \geq y_{j,k} \tag{16}$$

5.5 Scheduling

If two nodes v_{j_1}, v_{j_2} should be sequentialized, then the scheduling variables $b_{j_1,j_2,k}$ and $b_{j_2,j_1,k}$ have to be different, otherwise both have to be 1. The additional constraints for $b_{j_1,j_2,k}$ and $b_{j_2,j_1,k}$ are defined in [NM95]. With $b_{j_1,j_2,k}$,$b_{j_2,j_1,k}$ nodes can be sequentialized (eq. 17,18) by:

$$\begin{aligned}
\forall k \in K : \qquad T_{j_1}^S &\geq T_{j_2}^E - \infty * b_{j_1,j_2,k} && (17)\\
T_{j_2}^S &\geq T_{j_1}^E - \infty * b_{j_2,j_1,k} && (18)
\end{aligned}$$

5.6 Heuristic Scheduling

Optimal scheduling of the nodes is a complex problem, because the number of the 0/1-variables $b_{j_1,j_2,k}$ can grow quadratically in the number of nodes. An idea to solve this problem is to execute partitioning while iterating the following steps:

1. Solve an IP-model for the hardware/software mapping with help of approximated time values.
2. Solve an IP-model for calculating an exact schedule with nodes mapped to hardware or software.
3. If the resulting total time violates the timing constraint, repeat the first two steps with a timing constraint that is tighter than the approximated total time of step 1. (see figure 5).

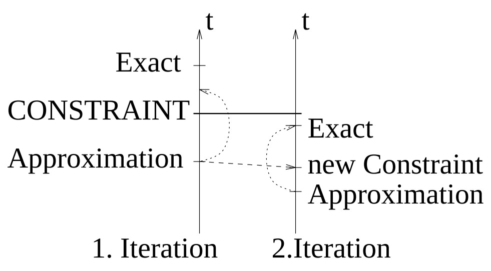

Figure 5: Heuristic scheduling

The following constraints are used additionally to the equations 6-11 to approximate time values:

- The starting time T_j^S of a node v_j is equal or greater than the accumulated software execution times of all predecessor nodes v_j.
- T_j^S is equal or greater than the accumulated hardware execution times of all shared predecessor nodes of v_j.
- T_j^S is equal or greater than the sum of the ending time of each dominator node v_i of v_j and the software execution times on processor ta_k of all nodes on the paths between v_i and v_j.
- T_j^S is equal or greater than the sum of the ending time of each dominator node v_i of v_j and the hardware execution times of all shared nodes on the paths between v_i and v_j.

The correct constraints can be found in [NM95].

6 Results

The interesting parameter for partitioning is the number of nodes n which have to be partitioned. For this reason, we have developed some examples containing a lot of instances of small VHDL-entities. The target architecture for all examples consists of a processor, an ASIC, memory and a bus connecting all components. All calculated partitionings concern interface costs and sharing effects between nodes. The computation times of the examples represent CPU seconds on a Sun SPARCstation20.

The heuristic partitioning approach can be evaluated by examining

- the quality and
- the computation time

compared to the optimal result.

The quality of the heuristic approach can be evaluated by determining the deviation between the exact and the approximated solution. If the heuristic partitioning approach does not consider interfacing, then the results are always exact, and therefore optimal. If interfacing is considered however, then the approximated system execution time may differ from the exact value. Therefore, we have partitioned 6 different systems (see figures 6,8) with the optimal and the heuristic approach. For each system, solutions have been calculated for a set of constraints. In figure 6 it is

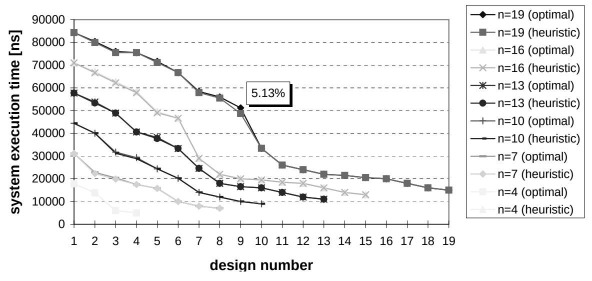

Figure 6: System execution time (exact/heuristic approach)

shown that the approximated execution time is equal or very close to the exact value. The maximal deviation between the exact and the approximated execution time is 5.13%, the average deviation is smaller than 1% for all examined systems.

In contrast to the partitioning quality, the computation times are very different. Figure 7 depicts the computation times of both approaches for a system, which consists of 7 nodes. This system has been partitioned for 8 different system execution time constraints. The maximal computation time is 246 seconds for the optimal partitioning approach and 2 seconds for the heuristic one, i.e., computation time is drastically decreased. In figure 8 the computation time of the heuristic approach is depicted for 6 different systems. For all of these systems several different

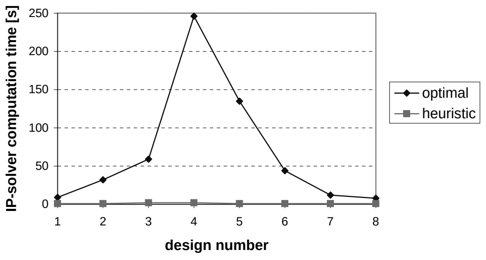

Figure 7: Computation time (exact/heuristic approach)

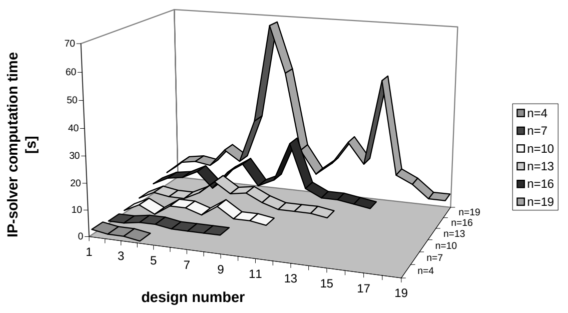

Figure 8: Computation times of the heuristic approach

designs have been calculated with help of a set of system execution time constraints for each system.

It becomes clear, that the heuristic approach is superior to the optimal approach, because the results are always nearly optimal and the computation times have been drastically reduced.

7 Conclusion

This paper presents a new approach of full-automated hardware/software partitioning supporting multi-processor systems, interfacing and hardware sharing. The partitioning approach itself is based on integer programming leading to optimal results. In contrast to other approaches, where hardware and software costs are estimated, our approach follows the idea of 'using the tools' for cost estimation. The disadvantage of an increased calculation time is compensated by better metrics and therefore fewer iteration steps. The presented results are very promising, because nearly optimal results are calculated in short time. Future work will deal with design studies of real system level examples.

References

[EHB93] Rolf Ernst, Jörg Henkel, and Thomas Benner. Hardware-software cosynthesis for microcontrollers. *IEEE Design & Test*, Vol.12, pages 64–75, 1993.

[GCM92] Rajesh K. Gupta, Claudionor Nunes Coelho Jr., and Giovanni De Micheli. Synthesis and simulation of digital systems containing interacting hardware and software components. *29th ACM, IEEE Design Automation Conference*, pages 225–230, 1992.

[HE94] D. Henkel J. Herrmann and R. Ernst. An approach to the adaption of estimated cost parameters in the cosyma system. *Third International Workshop on Hardware/Software Codesign, Grenoble*, pages 100–107, 1994.

[KL94] Asawaree Kalavade and Edward A. Lee. A global critically/local phase driven algorithm for the constrained hardware/software partitioning problem. *Third International Workshop on Hardware/Software Codesign, Grenoble*, pages 42–48, 1994.

[KL95] Asawaree Kalavade and Edward A. Lee. The extended partitioning problem: Hardware/software mapping and implementation-bin selection. *Proceedings of the 6th International Workshop on Rapid Systems Prototyping*, 1995.

[LMD94] B. Landwehr, P. Marwedel, and R. Dömer. OSCAR: Optimum Simultaneous Scheduling, Allocation and Resource Binding Based on Integer Programming. *Proceedings of the EURO-DAC*, pages 90–95, 1994.

[NM95] R. Niemann, and P. Marwedel. Hardware/Software Partitioning using Integer Programming. *Technical Report 586, Dept. of Computer Science XII, University of Dortmund*, 1995.

[PK93] Zebo Peng and Krzysztof Kuchcinski. An algorithm for partitioning of application specific systems. *Proceedings of the European Conference on Design Automation (EDAC)*, pages 316–321, 1993.

[VGG94] Frank Vahid, Jie Gong, and Daniel Gajski. A binary-constraint search algorithm for minimizing hardware during hardware/software partitioning. *European Design Automation Conference (EURO-DAC)*, pages 214–219, 1994.