The Interplay of Run-Time Estimation and Granularity in HW/SW Partitioning

Jörg Henkel, Rolf Ernst

Institut für Datenverarbeitungsanlagen
Technische Universität Braunschweig
Hans-Sommer-Str. 66, D-38106 Braunschweig, Germany
{henkel,ernst}@ida.ing.tu-bs.de

Abstract
An important presupposition for HW/SW partitioning are sophisticated estimation algorithms at a high level of abstraction that obtain high quality results. Therefore the granularities of estimation and partitioning have to be adapted adequately. In this paper we discuss the effects that arise when the granularities of partitioning and estimation are not adapted in a necessary way. Furthermore we present our solution that allows to choose different levels of granularities adapted to the estimation and partitioning phase. The experiments show that this refinement in estimation at a high level of abstraction leads to an improvement (in terms of run-time and chip area) of the whole mixed HW/SW system.

1 Introduction
In recent years HW/SW cosynthesis has become a very interesting field of investigation. This is due to the promising advantages of this design method. In HW/SW cosynthesis a system is specified by its behavior — independent from later implementation in software or in hardware — instead of its structure. So the designer can focus on the really challenging problems as, for example, the correct specification of a system. In the case of HW/SW cosynthesis the term system is often a synonym for real-time system because of the large economical potential (areas of application are telecommunications, car electronics, office automation, consumer electronics, . . .). The correctness of a reactive system is not only determined by specifying an appropriate algorithm for a given problem (e.g. which compression algorithm to select given a limited bandwidth and a constrained quality) but also by the timing behavior: if a message of a multiple-process reactive system is not transmitted within a given time slot, other processes may have to wait and cannot serve the actuators in the right time. In this case the functionality would not be maintained. The designer’s task is to prevent such cases by analyzing the demands and specifying accordingly. This is the more urgent the complexer the system becomes. Otherwise the time-to-market factor would increase to such an extent that the resulting system will become too expensive. A typical time-to-market is about 18 months for complex HW/SW systems [15]. Presupposition for this increasing abstraction level in system design is the existence of an almost automatic synthesis process for both hardware and software parts. This is called HW/SW cosynthesis.

The well known design steps of a generic cosynthesis flow are:
I. High-level specification (C, C++, . . .)
II. Estimation (HW run-time, SW run-time, chip area, . . .)
III. HW/SW partitioning
IV. Synthesis (HW, SW, interface)

Steps II. and III. are strongly correlated, so only if estimation values are close enough to the real values (obtained when really implemented) partitioning makes sense.

The approaches used in the estimation phase — which is that relevant that it determines the quality (in terms of monetary costs) and the feasibility (given the structure of a target architecture) of a mixed HW/SW solution — cover a wide diversity of algorithms: it spans from simple profiling to sophisticated approaches that also take into account the architectural features (pipeline interlocks, register sets, . . .) of a given processor core that is provided to execute, for example, the SW part. The same holds for the hardware part: the question is, how well the HW runtime estimation is suited to that algorithm used when the HW part is really synthesized (see also [10]) and what will happen if there is a great deviation between estimation and real data. The same questions could similarly be applied to the communication mechanism between the hardware and software part and the implied costs of an application specific hardware (chip area).

In any case, the answer is that it is nearly impossible or at least economically of no interest to perform an automatic HW/SW partitioning unless the underlying analyzing/estimation algorithms are of an appropriate accuracy.

There are a lot of approaches to HW/SW cosynthesis that differ mainly in two points: the partitioning algorithm itself (automatic by means of a combinatorial optimization algorithm or manually driven) and the accuracy of the analyzing/estimation algorithms. Automatic partitioning is proposed by [7, 8, 13, 14, 19]. They differ in the way the partitioning process is performed: hardware-oriented/software-oriented approaches. Others [2, 3, 4, 5, 6, 12, 17, 18, 20] focus on input specification, interface synthesis, rapid prototyping . . .
An investigation of how the quality/accuracy of an analyzing/estimating algorithm influences the quality (i.e., costs) of a HW/SW system that has been automatically partitioned has not been performed so far. This is mainly due to the fact that the algorithms were imported from the high-level synthesis domain and have not been adapted in an adequate way to the cosynthesis problem. So it becomes necessary to perform a re-partitioning or limit the partitioning process to a coarse-grain granularity even though a limitation to a coarse-grain granularity only, might lead — at least potentially — to a sub-optimum result.

This paper focuses on one part of this problem, namely the problem of partitioning. The dominating question to consider is: in which way is the additional amount of run-time and/or application specific hardware (i.e., chip area) to be avoided in order to prevent a cost-intensive mixed HW/SW system. Also, a solution is presented. Experiments in section 4 underline the relevance of applying a sophisticated estimation technique. The quality metric is given by the hardware area and hardware run-time. Finally, section 5 gives a conclusion.

2 Problem definition

2.1 Meeting real-time constraints

An essential goal in HW/SW cosynthesis is the high-level specification that the system is to be implemented. The designer should care about the question whether to implement dedicated parts in hardware or in software.

Hardware/Software partitioning is the most challenging step in HW/SW cosynthesis since the best hardware/software tradeoff promises the cheapest design while meeting the specified constraints. Independent of the fact whether HW/SW partitioning is performed automatically or manually, the quality of the estimation techniques that control the partitioning process will inherently determine the costs of a mixed HW/SW system. This is one of the presuppositions to guarantee the correct functionality of a real-time system.

Let $T_{HW/SW SYS, Ci}^H$ be a time-constraint assigned to a code segment $Ci$ in the behavioral description of the system. The goal is to find an implementation for $Ci$ such that

$$T_{HW/SW SYS, Ci}^H \leq T_{HW/SW SYS, Ci}^{CS}$$

where $T_{HW/SW SYS, Ci}^{CS}$ denotes the execution time of $Ci$ after implementation (i.e., assigned to the software part or to the application specific hardware part of the mixed HW/SW system). This is one of the presuppositions to guarantee the correct functionality of a real-time system. In this context the task of estimating is to find a

---

1 We will show later that the choice of coarse-grain granularity hides the inherent problems since the relative deviations are smaller.

---

Figure 1: Defining a granularity from the behavioral description a) for the purpose of HW/SW partitioning b) $T_{HW/SW SYS, Ci}^E$ that represents a lower bound for the execution time of each $ci$ and that is as close as possible to the according constraint $T_{HW/SW SYS, Ci}^H$ in order to obtain a cheap design.

2.2 Estimating and granularity

Estimating in HW/SW cosynthesis does not only mean estimating from behavioral specification but also estimating at a specific level of granularity. Thereby granularity is defined as follows:

**Definition 1** Given the behavioral description of an algorithm the granularity determines how to cut this algorithm into a set $B$ of $n$ pieces $b_i$ for the purpose of distributing these pieces among the hardware part and the software part of a mixed hardware/software system (see fig. 1.)

$B := \{b_0, b_1, \ldots , b_{n-1}\}$

Examples for obvious granularities, directly derived from the structure of the behavioral descriptions, are: operator-level, statement-level, base-block-level, control-block-level and task/process-level.

Note that the choice of the granularity does not necessarily mean that the smallest code-segment to be implemented either in hardware or in software is limited to the extension of the according granularity. But it does mean that the smallest code-segment can not be smaller than the smallest element $b_i \in B$. So the granularity determines a lower bound of a code-segment.

Once the granularity has been determined, for each piece $b_i$ the execution time $T_{HW, b_i}^E$ has to be estimated in order to guide the HW/SW partitioning. Thereby $T_{HW, b_i}^E$ is the estimated execution time of piece $b_i$ for the assumption that $b_i$ would be implemented in hardware. Principally there are two different ways how to obtain $T_{HW, b_i}^E$:

---

2 We use the term control-block for a set of adjacent base blocks containing also control statements (IF, FOR, . . .) and clustered by a closeness metric.

3 In fact in many cases it is advantageous to put several adjacent $b_i$'s to hardware or software.
Case 1
The hardware execution time is estimated taking the whole code without dividing it into a set of pieces $B$. So the whole behavioral description would be assumed to get executed by the hardware part. Let $T^{HW}_{HW,bi}$ be the according execution time. If we intend to design a mixed HW/SW system we now have to choose a granularity i.e. we have to find a set $B$. The HW run-time $T^{HW}_{HW,bi}$ for each $bi$ can be derived from $T^{HW}_{HW,sys}$ by simply dividing the whole description according to the granularity and assigning all of the run-time attributes that are found within each piece to $bi$.

This method is advantageous for the case that there is already a run-time estimator available (e.g. a scheduler from the domain of High-Level Synthesis). Nevertheless, this method lacks accuracy: if the granularity is fine-grain (a large $n$ i.e. a lot of $bi$’s in $B$) with only a few operators within a $bi$, we should be aware of the fact that a schedule strongly depends on what has been scheduled some control steps before. Say, the schedule of $bi$ depends on data/control dependencies outside of $bi$. Hence, the schedule of $bi$ is only 100% correct if all adjacent $bi$’s are also implemented on the hardware side (see above).

Case 2
This is the reverse case. $T^{HW}_{HW,bi}$ is obtained by an estimation technique that is limited to the boundaries of a $bi$ (e.g. a simple List-Scheduler). It works great if we do not have any code segment to put to hardware that is larger than a single code segment $bi$. But remember that the choice of a granularity is only a basis for partitioning (see beginning of this subsection). So, a larger code segment (consisting of more single-estimated $bi$’s) has also an inherent inaccuracy coming from the inaccuracy implied by each $bi$. This is because none of the $bi$’s does take advantage of synergetic effects.

Definition 2
A synergetic effect arises if in the case of an interplay of two or more $bi$’s an effect is obtained that is different from the effect obtained by the sum of effects of the single $bi$’s.

In the context of scheduling we could observe that synergetic effects will lead to a better result. So, without taking these effects into account we will get sub-optimum results.

We can summarize the two cases by formulating an inequality expressing the all-over relative error $(Err)^{(case1,2)}_H$ assuming that some of the $bi$’s are intended to get implemented in hardware:

$$Err^{case1}_H = \sum_{bi \in H} \frac{T^{case1}_{HW,bi} - T^{Opt}_{HW,bi}}{T^{Opt}_{HW,bi}} \leq 1 \leq \sum_{bi \in H} \frac{T^{case2}_{HW,bi} - T^{Opt}_{HW,bi}}{T^{Opt}_{HW,bi}} = Err^{case2}_H$$

Consequently case 1 is a lower bound and case 2 is an upper bound according to the optimum solution.

Obviously the estimation error can be reduced if we simply try to reduce the number $n$ of blocks $bi$. This is equivalent to choosing a coarse-grain granularity. Task/process-level granularity belongs to this group. During the partitioning the question will arise whether to move the whole task/process to hardware or to software — independent from the extension of the task (i.e. lines of code). But note that the area of application are small embedded systems which — in most cases — are very cost sensitive: even a few more gate equivalents for the part of the application specific hardware lead to an increase in price that makes this product uninteresting in the market of e.g. consumer products (audio, video, ..) where competition becomes harder and harder. So, if the designer describes an embedded system only by aspects like modularity, portability etc. — i.e. programming style2 — this might not be a good choice as a basis for the partitioning process. Therefore it is very probable that a coarse-grain granularity hides inherent optimization potential.

Or, on the other side, a fine-grain granularity will keep all the potential open.

In the next section we will start to discuss effects implying the inaccuracies in more detail and present our approach.

3 Effects in estimating and how to solve
3.1 Effects
As described in the last section HW/SW partitioning swaps code-segments $bi$ of a previous chosen granularity between hardware and software in order to find the best hardware/software tradeoff. This procedure is driven by estimation of relevant aspects. As for this paper we investigate the HW run-time estimation although the principles can also be applied to SW run-time estimation for example.

Thereby an error is invoked that is mainly due to effects at the borders of a code-segment at which control is transmitted from software to hardware, from hardware to software or — in case of a parallel execution of hardware and software parts — a fork or a join is performed.

Let $G = \{V, E\}$ be a directed acyclic graph (DAG) in which each node $vi \in V$ represents a basic block that contains at least one single operation and where a directed edge $e_{i,j} \in E$ specifies the direction of the control flow, meaning that there is an edge from $vi$ to $vj$. A node $vi \in pred(vj)$ is a node in $G$ that is called a predecessor of $vj$ since there is an edge $e_{i,j}$. Accordingly a node $vj \in succ(vj)$ is called a successor of $vj$. A granularity determines code-segments $bi \in B$, where

$$bi = \{vj \in G : \bigvee_{v_{j1}, v_{j2}, v_{j3} \in b_i} v_{j1} \in pred(v_{j2}) \lor v_{j2} \in pred(v_{j3}) \lor v_{j3} \in succ(v_{j2})\}$$

So, this recursive definition says that a $bi$ is a connected part of the origin code.

Different number of resources
Fig. 2 shows the example of a graph $G$ with $bi$ consisting of 3 elementary nodes $v$. $bi$ is now assumed to get implemented in hardware, the rest of $G$ should remain in software. The number of hardware resources (ALUs, multipliers, shifters, ..) available on the software side $HW Res^{SW}$ and on the hardware side $HW Res^{HW}$ will be different. In order

4Detailed effects will be the topic of the next section.
to obtain a speedup by using an application specific hardware, it is $HW Res_{HW} > HW Res_{SW}$. That implies that scheduling across point $P_{SW \rightarrow HW}$ where control is transmitted from SW to HW (or vice versa) causes an inaccuracy unless $P_{SW \rightarrow HW}$ is identical to the point where a new $b_i$ starts.

Let us assume that there is $HW Res_{SW} = 1^6$ with no parallel invoking of the resources and $HW Res_{HW} = 2$ with possible parallel execution of the resources in the example. For the adjacent area (statement before and behind) of $P_{SW \rightarrow HW}$ we obtain a scheduling result of 2 control steps for the case that scheduling has no information about the existence of $P_{SW \rightarrow HW}$. In the other case a more precise estimation obtains 4 control steps. The difference of 2 control steps has to be put in relation to the schedule of the following code-segment that will be put to hardware (see inequality 1). The same is for the point $P_{HW \rightarrow SW}$ where control is transmitted back from HW to SW.

The effect is the larger the smaller a $b_i$ is and run-time estimation does not use the same granularity as partitioning does.

Data transfer

At a control transfer point there is also transmitted data for the general case that hardware side and software side are data dependent. So, HW run-time consists not only of the scheduling result of the pure code-segment that is intended to be put to hardware, but it also consists of the data transfer time $T_{\text{transfer}}$. The choice of the granularity i.e. the extension of the $b_i \in B$ should be adapted to the amount of $T_{\text{transfer}}$. Otherwise a performance increase by using an application specific hardware with more resources than the software has, will be decreased drastically.

The extend of influence depends on the specific application (data- or control-dominated) and therefore it is indispensable to dynamically determine the granularity at which estimation should take place. *Dynamically* means that it is adapted to the application and not fixed to structural peculiarity of the behavioral description like the composition of a program in its tasks/processes as one extreme and its statements as the other extreme. Fig. 3 shows the data transfer overhead of different granularities relative to the hardware execution time. For different granularities (the horizontal axis points in direction of decreasing granularities)

$$
\sum_{b_i \in B} \frac{T_{\text{transfer}}}{T_{\text{exec}}_{HW,b_i}}
$$

is shown (vertical axis), where $T_{\text{exec}}_{HW,b_i}$ is the pure hardware execution time. This application comes from the domain of digital signal processing and therefore is strongly data dominated. Concerning a selection of an appropriate granularity it can be seen that an index of 78 really makes sense since the data transfer overhead is beyond 50%. The HW/SW partitioning will prefer moving code-segments to hardware that are expected to obtain a large speedup. So the granularity for estimation should be chosen accordingly (see above).

The following subsection describes the method we use to find a granularity adapted to estimation.

3.2 Clustering for estimation

In order to prevent low quality results for run-time estimation due to the fact that the granularity it takes place on is either too small or too large (section 2.2) we introduce in short terms our approach. Note that this clustering does not fix the extension of code-segments to get implemented in hardware but it does find a granularity that is suited for run-time estimation and that is acting as a *basis* for the following HW/SW partitioning process. In our cosynthesis design flow clustering is located as follows:

I. High-level specification
II. Clustering for estimation (HW run-time, SW run-time, chip area,...)

---

*6*meaning 1 ALU, 1 multiplier, ...

*9*This is the performance increase of a mixed HW/SW system compared to a simple SW system. It is illustrated in more detail in the section 4.
III. Estimation (HW run-time, ...)

IV. HW/SW partitioning

V. ...

For clustering we define a graph $G$ with the following characteristics: $G = \{V, E^H, E^V\}$ is a directed acyclic graph where each node either represents a basic block\(^{10}\) or a control statement like if, for, ... . There are two kinds of directed edges, both pointing in direction of the control flow. The edges $e^H \in E^H$ do not change the hierarchy level whereas the edges $e^V \in E^V$ do. This implies the definition of a hierarchy level: the hierarchy level is an attribute of a node $v_i$ that is incremented by one compared to $\text{pred}(v_i)$ if $\text{pred}(v_i)$ represents a control node and $v_i$ is nested in the hierarchy of $\text{pred}(v_i)$. As an example assume there is an if-statement that encloses a base block executed when the condition is true. Then this base block is called to be nested in the if-statement and consequently has hierarchy incremented by 1. A statement that follows the whole if-construct would have the same hierarchy level as the if-statement.

The edges $E^H$ are drawn from left to right and the edges $E^V$ are drawn from top to bottom. Fig.4 gives an example of a code and its representation as graph $G$. All nodes at a specific line own the same hierarchy level. Counting starts with 0 that is reserved for the head of the function/task. Furthermore the graph has the characteristic that the feed back edges are implicit: If we arrive at a leaf node we have to skip to node $\text{succ}(v_i)$ where $v_i$ is the node we previously descended to the hierarchy we are currently in.

Clustering always starts at the leaf nodes of the graph. The cost function for controlling contains mainly two components:

**Number** $n$ of pieces $b_i$

The number of pieces determines the computation effort for both estimating and following HW/SW partitioning. As for the run-time estimation it means that we have to execute $n$ schedules\(^{11}\). But the complexity for a based path-based scheduling depends on the number of possible paths within each $b_i$. The number of paths grows exponentially with the number of operations within a $b_i$ in the worst case. Since the complexity of the combinatorial optimization problem for partitioning grows exponentially with $n (O(2^n))$ the requirements concerning $n$ seem to be contradictory. We have overcome this problem by using a path-based estimation technique\(^{11}\).

So the user can determine the number $n$ of pieces $b_i$ as one constraint for clustering.

**The "quality" of pieces $b_i$**

For each possible piece $b_i$ a quality metric is defined as follows:

$$\text{quality}(b_i) = \frac{T_{SW,b_i}^{E}}{T_{HW,b_i}^{\text{pre-exec}} + T_{b_i}^{\text{transfer}} + T_{b_i}^{\text{misc}}}$$

The term $T_{SW,b_i}^{E}$ expresses the run-time estimation for the case $b_i$ would be executed at the software part and $T_{HW,b_i}^{\text{pre-exec}}$ expresses the run-time estimation for the assumption of executing $b_i$. Since we only want to find a possible granularity where a sophisticated run-time estimation can take place afterwards, $T_{HW,b_i}^{\text{pre-exec}}$ is determined by a simple list schedule. The transfer time for data from software to hardware and vice versa is obtained by a data flow analysis that determines the $\text{IN}\_\text{Set}(b_i)$ and $\text{OUT}\_\text{Set}(b_i)$ similar to the method described in [1]. A possible overlap of $b_i$ can get expressed by $T_{b_i}^{\text{misc}}$ (then there is $T_{b_i}^{\text{misc}} < 0$).

So the quality metric gives a hint whether $b_i$ could be a good candidate for the HW/SW partitioning with the difference that in HW/SW partitioning itself a more sophisticated cost function is used (9, 71).

The task is now to find the $n$ best clusters within $G$. Since a detailed description of the used algorithm would fill a separate paper, and we mainly concentrate on the interplay of run-time estimation and granularity in this paper, a shortened textual description is given here:

**Phase 1**

I. Get all leaf nodes in $G$.

II. Define each leaf node as a primary cluster.

III. Extend each of these clusters by merging a new $v_i$. This $v_i$ is obtained by walking against the direction of edges $e^H$.

IV. If there is no more edge $e^H$ found then climb up one hierarchy by walking against the direction of an edge $e^V$.

V. If this proceeding causes that two or more clusters will touch then merge them to a new cluster.

VI. For each merge operation note the number of all clusters the quality.

**Phase 2**

VII. Now determine for all $n$ clusters the quality.

VIII. By a random function two clusters a picked out. One of the clusters is enlarged by applying the steps III-V while the other one is reduced applying the according reverse operations. Thereby the total number has to be kept constant (constraints).

IX. For each cluster gained in the previous step the quality is calculated. The acceptance of a cluster depends on more aspects than the pure qual-

---

\(^{10}\)The term basic block is used in the same manner as in [1].

\(^{11}\) $n$ is the number of all $b_i$'s in $B$. 
4 Experiments and Results

Aim of the experiments is to show that a refinement in run-time estimation and an adaptation of granularity for estimation and partitioning leads to better results. In this context better could possibly mean a lot of things like simply the improvement of a run-time estimation technique compared to the real schedule. We will start to discuss this point. Much more interesting is the question in which way the quality of the whole mixed HW/SW system will be improved by only refining one (namely the HW run-time estimation) estimation technique. Of course we cannot expect that the improvements to the whole system will be as large as the comparison of two single algorithms would imply. The main question is whether it is worth to refine an estimation at a high abstraction level. Does it have any influence on the implementation of the whole system?

We first start to compare the scheduling results of using our path-based estimation technique [11], that operates on the graph described in the last section, and a simple list scheduler that has been used before in our system. Table 1 shows the results in terms of clock cycles that could be obtained after scheduling and profiling. The last column gives the fraction of time the according test program uses for execution of the HW/SW system. The hardware execution time, HW/SW transfer time for data, the time for transmitting control from software to hardware and vice versa and synchronization time. For the different test-programs in table 2 we changed two parameters: the first one is the constraint-speedup (constr) of the system and the the second is the variation of the run-time estimation algorithm (LS for the small granularity of base blocks only and PBET for estimating path-based at various levels of granularity) in order to measure its influence according to the performance of the mixed HW/SW system. The HW/SW partitioning has been performed automatically as described in [7, 9]. The column spu shows the estimated speedup as described in equation above. If we compare according lines, for example lines 2 and 6 of test program trs (both have the same constraint speedups) we can note that in all cases estimation using PBET obtains better results (in most cases about 5%) than LS meaning that PBET is superior due to the effects described in section 3.2.

Next part of the experiments discusses the question whether the increased quality can also influence the total monetary costs of the mixed HW/SW system. A quality metric is the chip area in terms of gate equivalents. Therefore the test programs have been automatically partitioned as above and in the following steps they have been synthesized using the High-Level synthesis system BSS and SYNOPSYS for implementing the application specific hardware on Xilinx FPGA's. Afterwards we transformed the specific CLB-usage into gate equivalents (g eq). The results are shown in column area of table 2. For two of the benchmarks (signed by asterisks) we could detect that the number of gate equivalents is different at the same constraints (constr) due to the two different run-time estimation techniques. In all other cases the area is smaller.

Table 1: Comparison of schedules obtained by simple List-Schedule and path-based estimation technique (PBET) using an appropriate granularity

<table>
<thead>
<tr>
<th>testp</th>
<th>loc</th>
<th>List-Schedule [cys]</th>
<th>PBET [cys]</th>
<th>frac rtime [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>contour</td>
<td>127</td>
<td>360,524</td>
<td>261,441</td>
<td>72.5</td>
</tr>
<tr>
<td>distance</td>
<td>695</td>
<td>256,680</td>
<td>184,255</td>
<td>71.8</td>
</tr>
<tr>
<td>fuzzy</td>
<td>100</td>
<td>80,353</td>
<td>72,321</td>
<td>90.0</td>
</tr>
<tr>
<td>median</td>
<td>302</td>
<td>26,680,323</td>
<td>21,980,162</td>
<td>82.4</td>
</tr>
<tr>
<td>table</td>
<td>664</td>
<td>17,623,269</td>
<td>10,108,341</td>
<td>57.4</td>
</tr>
</tbody>
</table>

In this equation $T_{SWsys}$ is execution time in clock cycles for the case the whole system is implemented in software i.e. running on a standard processor core. The term $T_{HW/SWsys}$ denotes the execution time in cycles after partitioning in hardware and software parts. So $spu_{HW/SWsys} = 2$ for example means that a mixed HW/SW implementation is twice as fast as a pure software solution of the same behavior. The time $T_{HW/SWsys}$ contains all relevant times like the part the software needs for execution of the HW/SW system, the hardware execution time, HW/SW transfer time for data, the time for transmitting control from software to hardware and vice versa and synchronization time. For the different test-programs in table 2 we changed two parameters: the first one is the constraint-speedup (constr) of the system and the second is the variation of the run-time estimation algorithm (LS for the small granularity of base blocks only and PBET for estimating path-based at various levels of granularity) in order to measure its influence according to the performance of the mixed HW/SW system. The HW/SW partitioning has been performed automatically as described in [7, 9]. The column spu shows the estimated speedup as described in equation above. If we compare according lines, for example lines 2 and 6 of test program trs (both have the same constraint speedups) we can note that in all cases estimation using PBET obtains better results (in most cases about 5%) than LS meaning that PBET is superior due to the effects described in section 3.2.

Next part of the experiments discusses the question whether the increased quality can also influence the total monetary costs of the mixed HW/SW system. A quality metric is the chip area in terms of gate equivalents. Therefore the test programs have been automatically partitioned as above and in the following steps they have been synthesized using the High-Level synthesis system BSS and SYNOPSYS for implementing the application specific hardware on Xilinx FPGA's. Afterwards we transformed the specific CLB-usage into gate equivalents (g eq). The results are shown in column area of table 2. For two of the benchmarks (signed by asterisks) we could detect that the number of gate equivalents is different at the same constraints (constr) due to the two different run-time estimation techniques. In all other cases the area is smaller.

A quality metric describing the whole HW/SW system is...
cases the total hardware area is the same.

The complexity of a part of the test programs has been so large that they consumed a CPU time of 1000 (more than 90% due to the SYNOPISS tool). This also confirms the need of estimation tools at a high level of abstraction that can execute much faster at acceptable accuracy.

As a general result we can note that the accuracy in HW run-time estimating from behavioral description does influence the quality of the mixed HW/SW system in spite of the fact that a lot of other aspects (estimation of SW run-time, data transfer time, ...) are superposed.

5 Conclusion and future work

We have shown that estimating HW run-time at a high-level of abstraction is a challenging problem. By a sophisticated technique where the granularity of estimation is adapted dynamically and adapted to the following HW/SW partitioning we could demonstrate that the all over quality of the hardware/software system can be increased in terms of speedup and area.

The results seem not to be specific to the problem of HW run-time estimation only. So we are motivated to refine other high-level estimation techniques (e.g. area estimation) also. Our future work will concentrate on this point.

References

