mentation. The partition function results with a very small number of good candidates for HW implementation. The bottleneck are the data transportation costs. This shows that these costs cannot be neglected and that the memory analysis phase build a very important part of our analysis method.

In this paper a detailed specification analysis method was presented. Cost functions for static and dynamic aspects were given as well as for interface costs. The described analysis procedure was implemented and examined by a set of benchmarks. It could be shown that specification analysis automatically generates a good HW/SW-partitioning and reduces the design space enormously. Further research will focus on the quantification of the overall speedup and interface design.

Literature

[Asha92] P. Ashar, S. Devadas, and A.R. Newton. *Sequential Logic Synthesis*. Kluwer Academic Publishers, Boston/Dordrecht/London, 1992.

[Benn93] T. Benner, U. Henkel, and R. Ernst. “Internal representation of embedded hardware-/software systems.” In *Handouts of the 2. IFIO Intern. Workshop on Hardware/Software Codesign* 1993.

[Camp87a] R. Camposano and R. Brayton. “Partitioning before logic synthesis.” In *Proc. of the ICCAD*, pages 324–326, Santa Clara, CA, 1987.

[Camp87b] R. Camposano and J.T.J. van Eijndhoven. “Partitioning a design in structural synthesis.” In *Proc. of the ICCD* pages 564–566, 1987.

[Cyp90] *Sparc/RISC User’s guide*, Cypress Semiconductor Ross Technology Subsidiary, 3901 North First Street, San Jose, CA 95134. 1990.

[DeMi90] G. De Micheli, D. Ku, F. Mailhot, and T. Truong. “The olympus system for digital design.” *IEEE Design & Test Magazine*, 1990.

[Edwa92] Martin Edwards. “A development system for hardware/software co-synthesis using FPGAs.” *Second IFIO International Workshop on Hardware/Software Codesign* 1992.

[Gene94a] R. Genevriere and R. Camposano. “Partitioning and Restructuring Designs on the Behavioral Level..” *submitted to DAC 95*, 1995.

[Gene94b] R. Genevriere and A. Hoffmann. “PMOSS - a modular synthesis and hw/sw-codesign system.” Technical Report SFB - 358 - B2 - 2/94, March 1994.

[Gupt90] R. Gupta and G. De Micheli. “Partitioning of functional models of synchronous digital systems.” In *Proc. of the ICCAD*, 1990.

[Gupt92] R.K. Gupta and G. DeMicheli. “System synthesis via hardware-software co-design.” In *CSL TR-92-548, Stanford*, 1992.

[Gupt93] R. Gupta and G. De Micheli. “Hardware-software cosynthesis for digital systems.” In *IEEE Design & Test*, 1993.

[Hard93] W. Hardt and R. Camposano. “Trade-offs in hw/sw codesign.” *Proc. of the ACM Workshop on Hardware/Software Codesign* 1993.

[Hard94] W. Hardt, A. Günther, and R. Camposano. “Pipelined interface for hw/sw- codesign.” Technical Report SFB - 358 - B2 - 3/94, University of Paderborn, Technical University of Dresden

[HePa90] D.A. Patterson J.L. Hennessy. Computer Architecture A Quantitative Approach. MORGAN KAUFMANN PUBLISHERS, INC., 1990.

[Holt93] U. Holtman and R. Ernst. “Speculative computation for coprocessor synthesis.” In *Proc. of the ICCD*, 1993.

[Kucu91] K. Kucukcakar and A. Parker. “Chop: A constraint-driven system level partitioner.” In *Proc. 28th DAC*, 1991.

[Lind62] B.W. Lindgren. *Statistical Theory*. MACMILLAN PUBLISHING CO., INC. 1962

[McFa90] M. McFarland and T. Kowalski. “Incorporation bottom-up design into hardware synthesis.” In *IEEE Transactions on Computer-Aided Design*, September 1990.

[Peng93] Z. Peng and K. Kuchicinski. “Speculative computation for coprocessor synthesis.” In *Proc. of the European Conference of Design Automation* 1993.

[Poll90] L.H. Pollard. *Computer Design and Architecture*. Prentice Hall, 1990.

[Stro90] B. Stroustrup. *C++ programming language*. Addison-Wesley Publishing Company, Inc., 1990.

[Vahi92] F. Vahid and D. Gajski. “Specification partitioning for system design.” In *Proc. 29th DAC*, 1992.

[Ye93] W. Ye, R. Ernst, Th. Benner, and J. Henkel. “Fast timing analysis for hardware-software co-synthesis.” In *Proc. of the ICCD*, 1993.

umn contains the number of modules. .For reasons of simplicity a module was understood as a

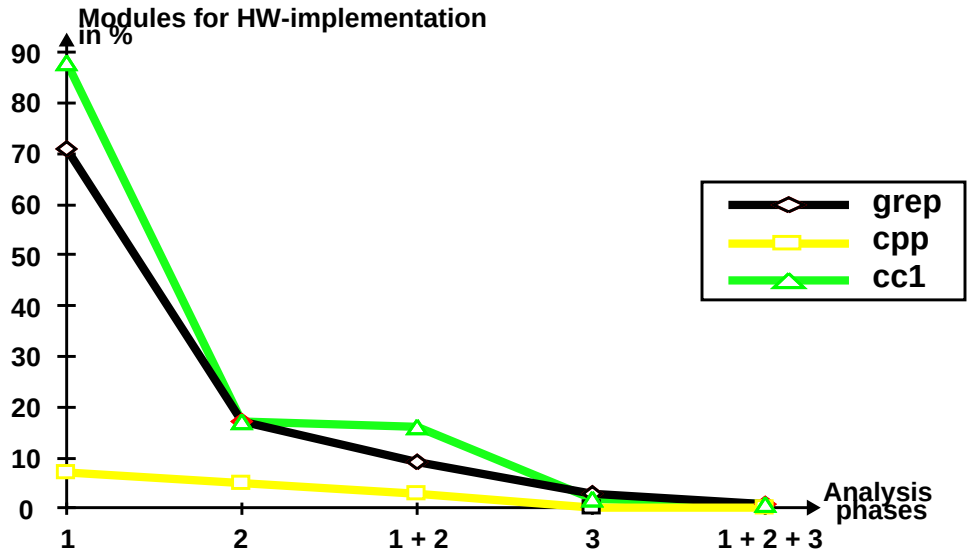

FIGURE 3. Result of specification anlysis for grep, cpp and

hierarchical element of the description language (function) Specification analysis was applied to this benchmarks and the modules qualified for HW-implementation are countedIn figure 3 and figure 4 the result of static (1), dynamic (2) and interface costs (3) analysis phase is given explicit as well as the combined results. It has been found, that each phase qualifies a part of

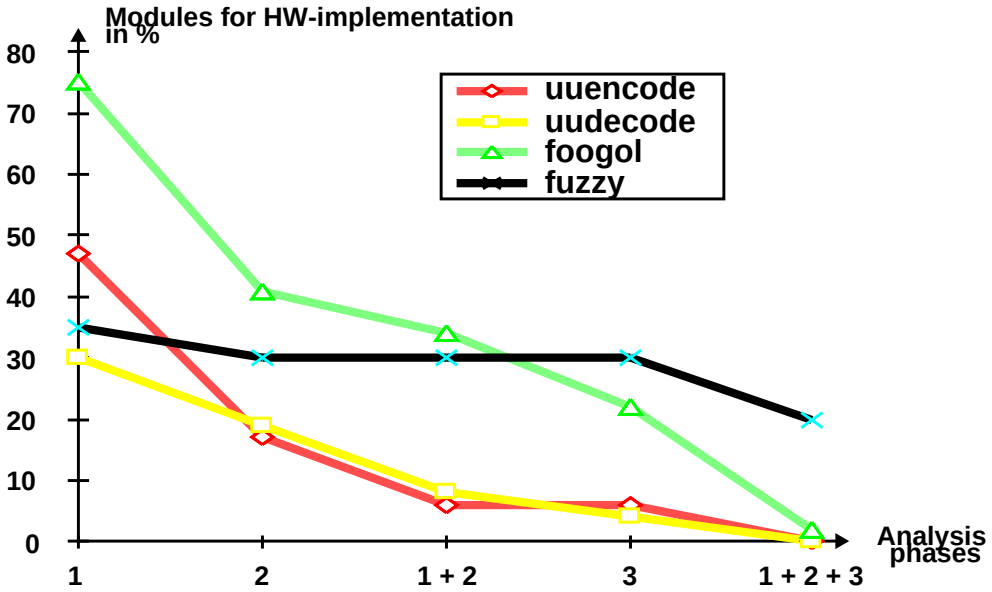

FIGURE 4. Result of specification anlysis for uuencode, uudecode, foogol, fuzzy

the design for HW-implementation. But only the combination of all analysis phases leads to good partitions and an enormous design space reduction.The considered design space could be reduced to 50% in average by static specification analysis. So a reasonable design space reduction could be reached. If only dynamic analysis is applied the design space is reduced to 20%. The correlation of both analysis phases leads to a further design space reduction of 5%. Considering only the interface costs only 10% (average) are found suitable for HW imple-

Definition 18 : The **partitioning function** $\Phi: UUC \rightarrow \{SW, HW\}$ is defined as:

$$\Phi(mod) = \begin{cases} HW \Leftrightarrow \dfrac{Stat(mod) \times Dyn(mod) \times Intf_{limit}(mod)}{Intf_{total}(mod)} > 0 \\ SW \Leftrightarrow \dfrac{Stat(mod) \times Dyn(mod) \times Intf_{limit}(mod)}{Intf_{total}(mod)} = 0 \end{cases}$$

The suitability of a module for HW implementation found by static analysis is multiplied by the result of dynamic analysis and divided by the interface costs. If the overall interface costs are too high $Intf_{limit}(mod)$ results to zero. This partitioning function splits the UUC into a HW part P_{HW} and a SW part P_{SW} so that holds: $UUC = P_{HW} \cup P_{SW}$ and $P_{HW} \cap P_{SW} = \emptyset$. For optimization all modules of the HW partition may be regarded together. This may be subject of further research work.

3 Experimental Results and Conclusion

The HW/SW-partitioning process based on specification analysis as described above selects a small number of good candidates for HW-implementation. Comparisons of these results with manual candidate selection showed the partitioning quality. It has been found that the manually generated HW-partition is always included in the automatically generated HW-partition. For example, specification analysis applied to the UNIX tool „grep“ identifies the module executing the central pattern match algorithm for HW-implementation. According to our cost function this selection leads to the best possible overall speedup. Regarding the example FIP (Fuzzy Inference Process) four modules were automatically identified for HW-implementation. In figure 2 the module graph of the automatically generated HW-partition is given. It should be pointed out that the interface costs are minimized and strongly connected modules are not separated. Obviously such tree structured design parts are candidates which are chosen during manually partitioning. Our approach has been implemented on a UNIX platform. We

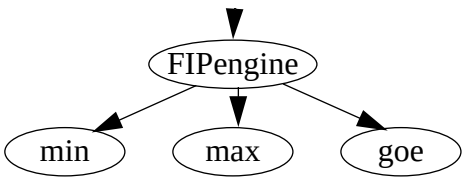

FIGURE 2. Module graph of FIP- HW part

applied our specification analysis method to a set of benchmarks of reasonable complexity..

UUC	uuencode	uudecode	foogol	grep	fuzzy	fuzzy	cc1
size	2.418	2.518	3.876	35.081	1462	33.926	564.380
modules	17	26	59	173	20	174l	2116

Table 2. Characteristics of examined benchmarks

Most of them are taken from public domain software. Their characteristics are summarized in table 2. The design size is given in lines of assembler code for a sparc machine. The next col-

for HW-implementations. The transport costs for SW-implementations are $Trans^{SW}_{direct}(mod)$, $Trans^{SW}_{memory}(mod)$ and $Trans^{SW}_{indirect}(mod)$.

Definition 14 : The function $Trans: M \rightarrow I\!R^{+}$ expresses HW-transportation costs in relation to the SW-transportation costs:

$$Trans(mod) = \frac{Trans^{HW}_{direct}(mod) + Trans^{HW}_{memory}(mod) + Trans^{HW}_{indirect}(mod)}{Trans^{SW}_{direct}(mod) + Trans^{SW}_{memory}(mod) + Trans^{SW}_{indirect}(mod)}$$

The evaluation of these transportations costs are subject of the memory analysis phase. First the (assembler) source code of the UUC is automatically modified. Thus a protocol of all memory accesses is generated. This protocol distinguishes the access direction (read, write).

The maximal acceptable interface costs are limited by a constant (INT_MAX_COST) (table 1).

Definition 15 : The function $Intf_{limit}: UUC \rightarrow I\!R^{+}$ is defined as:

$$Intf_{limit}(mod) = \begin{cases} \text{INT\_MAX\_COST} - Trans(mod) & if \quad \geq 0 \\ 0 & else \end{cases}$$

The overall transportation costs depend on the number of module accesses $Acc^{SW}_{dyn}(mod)$ which can be determined by dynamic analysis.

Definition 16 : $E_{Acc^{SW}_{dyn}}(mod)$ is defined as expected value of all found profiling values

$Acc^{SW}_{dyn}(mod)$ of module $mod \in UUC$.

.

The overall interface costs of a given module are defined by function $Intf_{total}(mod)$.

Definition 17 : The function $Intf_{total}: UUC \rightarrow I\!R^{+}$ summarizes the interface costs for a given module $mod \in UUC$:

$$Intf_{total}(mod) = E_{ACC^{SW}_{dyn}}(mod) \times Trans(mod)$$

2.5 Partitioning Cost Function

The results of static, dynamic and memory specification analysis are correlated. The partitioning $\Phi(mod)$ decides for a given module $mod \in UUC$, if a HW implementation should be considered.

Note, it is not intended to generate statistical independent test patterns, but to avoid design decisions caused by invalid or hardly relevant data. It is assumed that the input data was chosen with respect to its statistical distribution.

Definition 10 : $E_{RT_{abs}^{SW}}$ is the expected value of all profiling values $R_{abs}^{SW}(mod)$ found during dynamic analysis.

The terms $E_{average}^{SW}(mod)$ and $E_{rel}^{SW}(mod)$ are defined in the same way.

Definition 11 : $\sigma_{RT_{abs}^{SW}}(mod)$ is defined as standard derivation of all found profiling values $RT_{abs}^{SW}(mod)$ and if $E_{RT_{abs}^{SW}}(mod) \neq 0$ the **confidence** $Conf_{RT_{abs}^{SW}}(mod)$ is defined as:

$$Conf_{RT^{SW}_{abs}}(mod) = \text{DYN\_CONFIDENCE} - \frac{\sigma_{RT^{SW}_{abs}}(mod)}{E_{RT^{SW}_{abs}}(mod)}$$

Now, the cost function for each runtime aspect can be defined. The expected value is correlated with the confidence value.

Definition 12 : $Dyn_{abs}(mod) = \begin{cases} E_{RT_{abs}^{SW}}(mod) \times Conf_{RT_{abs}^{SW}}(mod) & if \geq 0 \\ 0 & else \end{cases}$

$Dyn_{average}(mod)$ and $Dyn_{rel}(mod)$ are defined analogously.

This correlations measure the importance of the given module for the overall runtime behavior. The next definition introduces one additional normalization factor for each runtime aspect in order to direct the analysis method.

Definition 13 : The function $Dyn: UUC \rightarrow \mathbb{R}^{+}$ summarizes the suitability of module $mod \in UUC$ for HW implementation found by dynamic analysis:

$$Dyn(mod) = \frac{Dyn_{abs}(mod)}{\text{SW\_ABS\_RT}} + \frac{Dyn_{average}(mod)}{\text{SW\_AVERAGE\_RT}} + \frac{Dyn_{rel}(mod)}{\text{SW\_REL\_RT}}$$

2.4 HW/SW- Interface Costs

As mentioned before specification analysis must take the interface costs also into account. For quantification several reasons causing interface costs are to be distinguished. Global data and parameter passed to

and returned from a module $Trans_{direct}^{HW}(mod)$, accesses to main memory $Trans_{memory}^{HW}(mod)$ and

calls of further submodules $Trans_{indirect}^{HW}(mod)$ are time consuming actions. The costs of direct and

indirect data transportation are determined by static analysis. But such costs are not only known from HW-designs. Data transports are expensive for SW-implementations as well as

Definition 8 : The average of all execution times of module mod found during design execution is called **average SW runtime**: $RT^{SW}_{average}(mod)$.

Definition 9 : The percentage of a modules absolute runtime to the design execution time is named **relative SW runtime** of module mod: $RT^{SW}_{rel}(mod)$.

Often the designer will guide the partitioning process exploring the design space by predefined restrictions. Typical examples are maximal or minimal module size. Some approaches pass parameters to the design process indicating the optimization priority, e.g. time versus size [DeMi90] or many partitions versus few. On the other hand relative directives are not always sufficient especially if there are restrictions for all partitions. This can be handled, if module characteristics are normalized. The normalization base defined by the designer as constant is given to the partitioning process. The values listed in table 1 lead to results presented in figure 3 and figure 4. But the designer may change this constants also for further design space exploration.

constant	value	note
SW_APPROX_RT	1000	This constant defines the intended size of the approximated software runtime in cpu cycles.
SW_ABS_RT	10	The intended absolute runtime of a given module is set to 10 ms.
SW_AVERAGE_RT	5	Normalization factor for the average module runtime in ms.
SW_REL_RT	5	Definition of the intended relative module runtime in percent of the design runtime.
DYN_CONFIDENCE	3	Maximal value of sigma / mu indicating the confidence of dynamic analysis.
INT_MAX_COST	3	Interface costs of a HW implementations may not be higher than 3 times the costs of a SW implementation.

Table 1. constant definitions

Dynamic analysis generates profile data for all modules according to each input value. For example, if 1000 different inputs are considered dynamic analysis terminates with 1000 data values per module for each runtime aspect. For one module these values may differ slightly or dramatically. The probability for a correct profiling result depends on the found differences. So, we regard the dynamic analysis for one input value as statistical experiment and the found profiling value as the result of this experiment. Now, the terms probability P, the expected value E and standard deviation σ from statistical theory e.g. [Lind62] are used to classify the generated input data. If σ grows too much the probability of a bad profiling result is rather high. This has been found as good classification base. To apply these statistical aspects to profile data the UUCs behavior have to be examined with a set of input data which is large enough. The correctness of an experiment is heuristically measured by the quotient of σ and E. The worst acceptable value (DYN_CONFIDENCE) depends on the size of the input data set and may be defined by the designer (table 1). If DYN_CONFIDENCE is set to 1 and $\sigma < E$ holds the experiment will be accepted.

and a set of edges $E_{MG} = \{access_0, access_1, ..., access_{m-1}\}$ is called **module-graph**.

Nodes as well as edges are annotated with results from static, dynamic and memory specification analysis.

2.2 Static Specification Analysis

Static specification analysis examines the design specification already compiled to assembler code. This implies a fixed target architecture. A standard sparc architecture [Cyp90] was chosen for it is rather powerful and widely spread. But the same concepts can be applied to other architectures as well if an appropriate compiler environment is available. Static specification analysis approximates a lower bound of the runtime of the SW implementation $\left(RT^{SW}_{approx}(mod) \right)$ as defined in [Hard93] and the module's control dominance.

Definition 5 : If $Inst(mod)$ is the number of instruction per module mod and $Jmp(mod)$ the sum of conditional and unconditional jump-instructions of module mod, the **control dominance** is defined as:

$$Ctrldom(mod) = \frac{Jmp(mod) \times 100\%}{Inst(mod)}$$

Experiments have shown that the reached speed-up of HW-partitions is high if the control dominance is high. E.g. consider a comparison of two single bits. This can be done in HW very fast by only one gate. A SW implementation e.g. for a pipelined architecture needs one cycle at least.

Furthermore it has been found that reasonable speed-ups can be reached only if the approximated runtime is not to small. The appropriate size is defined as constant (SW_APPROC_RT) (table 1) by the designer. These effects lead to a result function of static analysis measuring the **suitability** of a given module for HW implementation.

Definition 6 : The function $Stat: UUC \rightarrow I\!R$ summarizes the suitability of module $mod \in UUC$ for HW implementation found by static analysis with respect to the normalization factor:

$$Stat(mod) = \frac{RT^{SW}_{approx}(mod) \times Ctrldom(mod)}{\text{SW\_APPROX\_RT}}$$

2.3 Dynamic Specification Analysis

Dynamic specification analysis examines the runtime behavior of the given UUC. Profiling data generated by execution of the SW implementation of the UUC is analyzed. Considering a single module three aspects, the absolute, average and relative runtime of a module are taken into account.

Definition 7 : The time consumed by a module mod during the whole design execution is called **absolute SW runtime**: $RT^{SW}_{abs}(mod)$.

centrated on designs described in “C” or “C++” [Stro90], because a variety of suitable tools is available in a UNIX environment. Also synthesis systems handle this description language GeHo94, [DeMi90] and [Benn93]. So, parts of a C/C++ design specification can be synthesized nearly automatically. The analysis scenario is presented in figure 1. Static analysis and dynamic analysis as well as memory analysis use the same internal data structure, which is based on a graph model. The design is represented in terms of modules and accesses in between. This is an abstraction of the control-dataflow graph commonly used for system synthesis. Nevertheless there may be references to detailed design information provided by our synthesis system PMOSS [GeHo94].

The rest of this paper is structured as follows. In section 2 basic definitions are given. Also cost functions for all analysis phases are presented. Subsection 2.4 focuses on measuring costs of the HW/SW-interface and in subsection 2.5 the overall partitioning cost function is defined. Results are presented in section 3.

2 Specification Analysis

During specification analysis a design is thought of as a set of interacting modules. Typically design descriptions are structured by hierarchy which may be expressed by different constructs with respect to the specification language. Well-known examples are functions (C), methods (C++) and procedures (VHDL) etc. Also more abstract terms can be used to identify hierarchy [Gene94a]. However the specification hierarchy can be understood as an initial system modularization and our specification analysis is based on this modularization. The suitability of each modul for HW-implementation is examined. Thus, each module can be moved either into the HW-partition or SW-partition.

2.1 Basic Definitions

Some preliminary definitions are necessary.

Definition 1 : A sequence of constructs of the description language is defined as **module.**

A module may initially be seen as a function of the description language. Such a module definition is heuristic but is has been found a useful experimentation base. However, the specification analysis method is not limited to restrictions like this.

Definition 2 : The activation of module mod_j by module mod_i is called **access**:

$$access = (mod_i, mod_j) .$$

Definition 3 : A **unit under codesign UUC** is defined by the set of all modules mod_i:

$$UUC = \{mod_0, mod_1, ..., mod_{n-1}\} \qquad mod_i \cap mod_j = \varnothing \Leftrightarrow (i \neq j),\ i \in \{0, ...,n-1\}$$

For analysis under HW/SW-partitioning aspects a design is represented by a module-graph. This granularity was chosen because of the correspondence to the objects handled during the partitioning process.

Definition 4 : A graph $MG = (V_{MG}, E_{MG})$ with a set of nodes $V_{MG} = \{mod_0, mod_1, ..., mod_{n-1}\}$

[Hard93]. We call this process **static specification analysis**. But static analysis is not sufficient, because data dependencies influence the control flow enormously. So we introduce an additional **dynamic specification analysis** phase. This can be done by design execution or simulation [HePa90], [Ye93], [Peng93]. Generating profiling data by simulation means to execute a design specification on a simulated processor. But simulation run time will prevent examination of large input data sets. Other approaches compile the design and generate profiling data during execution e.g. Ed93. This is of great use if more than one set of input data is considered. Our dynamic analysis also uses this method because large sets of input data are examined. We found this necessary because varying input data cause differences of the design behavior with respect to the control flow. In general the control flow is data dependent. Now, our aim is to reach a high overall speedup. Thus the partitioning should speed up the design for most sets of input data. So the design behavior must be examined for a large number of inputs. This leads to detailed runtime information. Combining these results with results of static analysis more realistic partitioning criteria is defined. However, **interface costs** caused by HW/SW-partitioning must be taken into account also. In literature different interface concepts are used [Gupt92], [Poll90] and [Hard94]. Always, the data-transfer between the different system parts is found as important bottleneck. This aspect also should influence the partitioning process. Within two steps an interface cost analysis was integrated into our specification analysis. First, the number and size of all input- and output parameters are determined by static analysis. And second, all access to main memory are regarded. This is very important because reference parameters are used often. The parameter is interpreted as pointer to the data which is needed. Obviously, the data size is independent of the pointer size and often much more data has to be transported between the regarded system parts. Furthermore, temporary data structure may be build and stored into the main memory. This causes also a lot of data traffic. Therefore a protocol of all memory access is generated and evaluated during an additional **memory analysis** phase. Our cost function defined in subsection 2.4 takes static and dynamic as well as the mentioned interface aspects into account.. Our activities are con-

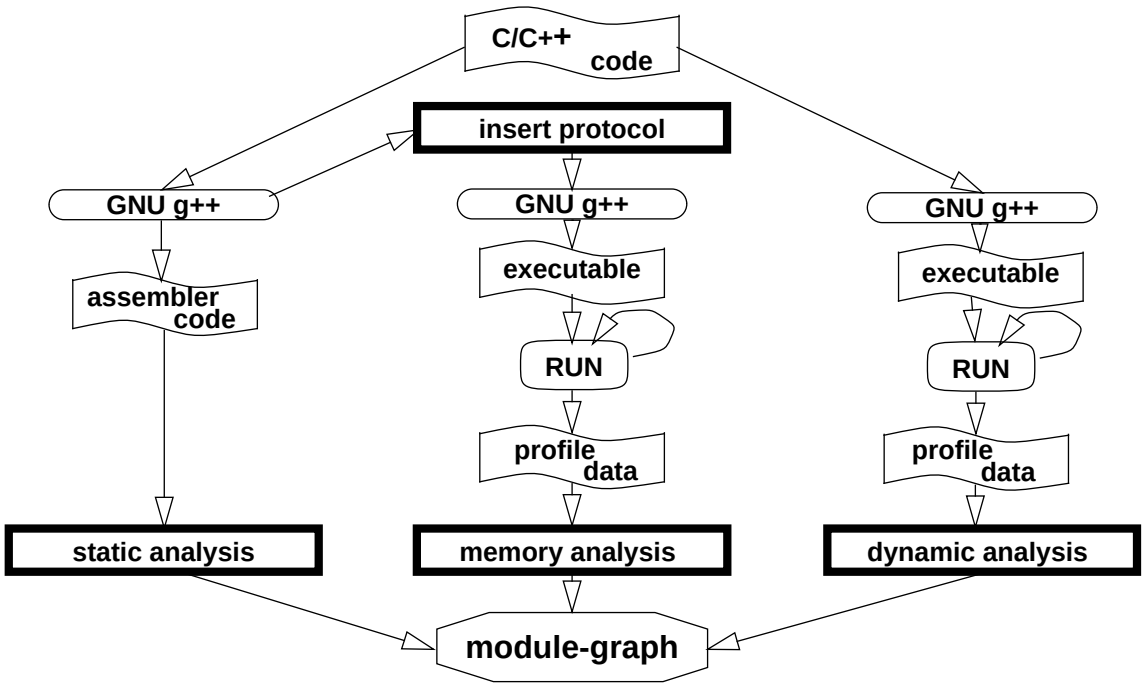

Figure 1. Specification analysis scenario

Specification Analysis for HW/SW-Partitioning

Wolfram Hardt[†], Raul Camposano[††]
email: hardt@uni-paderborn.de

[†]University of Paderborn
Warburger Str. 100
33 095 Paderborn
Germany

[††]Synopsys, Inc.
700 E. Middlefield Road
Mountain View, California 94 043
USA

Abstract: Embedded system design faces the HW/SW-partitioning problem. This is often solved by explicit designer decisions. In this paper we describe a partitioning method based on design specification analysis. Static and dynamic aspects influence the overall time behavior and are examined during two analysis phases. Each phase evaluates a cost function and classifies the suitability of the regarded partition for HW-implementation. Furthermore costs of the HW/SW-interface are taken into account because this overhead may not be neglected. The analysis method is demonstrated by several benchmarks and reasonable results are pointed out.

1 Introduction

Top-down design of embedded systems starts from behavioral design specification. HW/SW-partitioning is an essential problem during the design flow. Different goals may guide this process, e.g. minimization of power consumption and chip size or performance maximization. Also the intended target architecture, e.g. full custom or semi custom, and the available technology may influence the partitioning process.

A variety of partitioning approaches have been developed and have been applied to design specifications on different levels of abstraction. Some focus on logic partitioning including behavioral operations CaBr87 and [Camp87b]. Partitioning on control-dataflow design representations can be found in [McFa90],[Kucu91],[Gupt90] and [Vahi92]. All these partitioning approaches are driven by the costs of the generated HW in terms of area, latency, power etc. But the number of transistors integrated on chips and the chip clock rate increase continuously. So the hardware limits are widened and the importance of area costs decrease. Design optimization focuses much more on performance. Partitioning HW/SW our main objective is optimization of overall performance. Case studies point out that introducing additional HW-components into the system architecture speed-ups for the whole system can be reached, e.g. [Gupt93] and [Holt93]. But obviously it is difficult to find portions of a design specification which cause noticeable speed-ups when moved to HW. At this point specification analysis becomes important.

Static design characteristics allow a best case speed-up approximation. Such criteria can be generated very fast by specification examination on an appropriate level of abstraction