The COSYMA environment for hardware/software cosynthesis of small embedded systems


Technische Universität Braunschweig, Hans-Sommer-Str. 66, D-38106 Braunschweig, Germany
The COSYMA environment for hardware/software cosynthesis of small embedded systems


Technische Universität Braunschweig, Hans-Sommer-Str. 66, D-38106 Braunschweig, Germany

Abstract

COSYMA is a platform for the investigation of hardware/software cosynthesis of small embedded systems. Target architecture is currently limited to processor-coprocessor configurations executing a single process or a system of communicating processes which are statically scheduled. Many aspects of cosynthesis such as automatic hardware/software partitioning, efficient hardware/software communication, timing and hardware overhead estimation and analysis, interdependence of different cosynthesis phases, data representation, etc., can successfully be investigated in this manageable domain. COSYMA covers the complete design flow from an input language similar to C down to netlist and object code. Current focus is on high performance data dominated systems, but first steps to incorporate control dominated subtasks can be presented. Using a specific high-level synthesis tool, the results show a considerable speedup of the resulting processor-coprocessor system even compared to modern RISC processors which is typically limited by memory bandwidth.

Keywords: Embedded systems; Hardware/software partitioning; Process scheduling; Run-time analysis

1. Introduction

Hardware/software co-design is a very diverse area with topics ranging from system specification all the way down to code generation and high-level synthesis. The target systems are of very different sizes, from large scale distributed control systems to telephone answering machines. In this context, it is crucial to define manageable research topics which are still relevant in the whole picture. One important, yet manageable, area is the design of small embedded systems consisting of few ICs with microprocessors. Such systems are ubiquitous in consumer electronics, multimedia, automotive control, office systems etc., and they show many of the co-design problems of the larger systems. Their rather simple architecture particularly allows the investigation of global optimization over hardware and software components in a cosynthesis process. This is the purpose of the COSYMA system (COSYnthesis of eMbedded Architectures).

Embedded systems have data dominated and control dominated tasks. In control dominated tasks, such as a car ignition control or a LAN controller, data operations are simple while response time requirements and signal waveforms can become very complex. In contrast, data dominated tasks, such as complex signal processing like image compression, show complex and computation intensive data and memory operations and a less complicated reactivity. Data dominated systems must handle data dependent execution times and complex compiler and high-level synthesis optimization, and memory contention problems, while asynchronous event handling, optimization with re-timing, and fast context switch and scheduling are particular problems of control dominated tasks. As a consequence, the algorithms are different, and most of the researchers have emphasized the one or the other category. CHINOOK [1], TOSCA [2] and the work of Vincentelli [3,4] give cosynthesis approaches for control dominated systems, while PTOLEMY [5] and [6–9] focus on data dominated systems. Some cosynthesis systems are not clearly biased towards one of the categories, such as VULCAN [10], and [11]. Cosynthesis systems which can handle combinations of both categories are useful since many control systems have both control and data dominated subtasks such as an engine management unit in a car. COSYMA was originally focussed on speeding up single-process systems with complex data dominated parts. With a
"scalable performance" process scheduling approach, which incorporates some ideas from control dominated tasks scheduling. COSYMA can now optimize multi-process systems with different timing requirements and control characteristics.

This paper gives an overview on the current status of the COSYMA system. Section 2 shows the COSYMA design flow. Section 3 introduces scalable performance scheduling as a preprocessing step to the hardware/software partitioning described in Section 4. Section 5 explains the high-level synthesis system, BSS, which is specifically developed for the cosynthesis process. An important COSYMA component is the hybrid timing analysis described in Section 6. In Section 7 some results are given. Finally, we draw some conclusions.

2. Design flow in COSYMA

The COSYMA target architecture (Fig. 1) consists of a standard RISC processor core (we use the SPARC architecture with 33 MHz clock and floating point coprocessor as implemented by LSI Logic), 2 fast RAMs for program and data with single clock cycle access time and an automatically generated application specific coprocessor. Processor and coprocessor communicate through shared memory using a CSP type protocol (communicating sequential processes). The COSYMA design flow is shown in Fig. 2.

The input description may consist of several communicating processes with timing requirements. The communication mechanism is similar to [12]. The description consists of a function description in C', a superset of C with processes, and control file with user directives for communication channel mapping and timing constraints [13]. At this time, only shared memory communication is supported. There are also directives to predefine the binding of functions to specific user defined hardware components or to software constraining the partitioning process.

This input description is translated to a syntax graph (using both the SUIF compiler and an own compiler) which is extended by a data flow graph for each basic block and the global control flow (extended syntax graph: ESG). The processes are statically scheduled with the algorithm described in Section 3.

An automated partitioning process is one of the key problems in hardware/software cosynthesis as opposed to co-design. COSYMA can partition functions or basic blocks including basic blocks with function calls. The hardware/software-partitioning in COSYMA is solved with simulated annealing [14]. The approach is software-oriented, i.e. the simulated annealing starts with an all software solution and "extracts" hardware (this is called the inner partitioning loop). This is done iteratively [15] until all time constraints can be met (outer partitioning loop).

![Fig. 1. Target architecture of COSYMA.](image)

![Fig. 2. Design flow in COSYMA.](image)
In COSYMA the user has the choice between the High-Level Synthesis systems Olympus [16] BSS or VOTAN. Last step in the design flow is a hybrid timing analysis that combines behaviour simulation and formal methods in order to validate the timing constraints of the mixed hardware-software system.

Fig. 2 shows COSYMA design environment.

3. Scalable performance scheduling

Process scheduling is a well known problem in embedded system design. In many dynamic approaches priorities are assigned to the processes depending on their rates [17,18]. In static approaches [19,20,1] the order of the processes is determined at design time in order to guarantee timing constraints.

In all of the current process scheduling approaches, the hardware-software architecture is widely known at compile time. In hardware-software cosynthesis, the decision of the hardware or software implementation can be moved to a very late design stage (late binding) because changing system descriptions and generating and modifying hardware-software architectures has become much easier. This is very much like logic synthesis has simplified RT-level modifications. As a consequence, process scheduling could move in front of hardware architecture definition.

The major difference of the scheduling problem as compared to other process scheduling problems, is that the scheduler does not operate on a fixed architecture but on an architecture with scalable performance. Given a process system with time constraints, the scheduler cannot simply anticipate the execution time of the individual processes or process segments, because the partitioning tools will determine the parts to be accelerated by hardware.

The scalable performance [21] is defined relative to the performance of a given processor which shall be used as core processor in the processor-coprocessor system (the approach in general also holds for other architectures). To express the required performance, a performance scaling factor is defined,

\[ S_p = \frac{\text{required system performance}}{\text{given processor performance}} \]

such that \( S_p \) can be used as speedup factor for cosynthesis. To alleviate the scheduling problem, we assume that hard I/O-timing constraints are buffered by a peripheral device and that processes with cycle times of less than a few microseconds are moved to hardwired logic upfront.

The aim of the following is to find a valid schedule such that \( S_p \) is minimized. At the beginning of the scheduling process (Fig. 3), a process dependency graph (PDG) is derived from the C* description. The processes are split into basic blocks in order to get short time segments, which allow to schedule processes with a short iteration rate in the order of a few microseconds. Each basic block also ends at a blocking communication or at a label which corresponds to an intertask constraint. In a prescheduling step, the processes are first serialized such that the order of the processes does not violate the data and communication dependencies but without regarding the external time or rate constraints. Being executed with external stimuli data, the ordered segments show the correct function and timing when executed on a single target processor simulator (the prescheduling step saves an extra simulator). The sequential process provided by prescheduling is functionally correct, because all dependencies were considered, but the timing is incorrect. Therefore a different scheduling is required, now.

![Process scheduling diagram](image-url)
If the application cannot be implemented on a single core processor, the utilization $U$ of the processor becomes greater than 1. A scaling factor $S_c$ brings the processor utilization in the valid area by normalizing the execution time of each process by $S_c$. In addition to the processor utilization, $S_c$ depends on the task dependencies, the constraints, the communication and the overhead for context switching. We join all these components to a factor $a_{dep}$ to determine the scaling factor:

$$S_c = a_{dep} \cdot U = a_{dep} \cdot \sum_{i=1}^{n} \frac{T_i}{C_i}.$$  

Because $a_{dep}$ depends on the number of context switches and the order of the processes which are unknown before scheduling, it cannot easily be estimated, but is evaluated heuristically. The scheduling algorithm is iterated several times while adapting $a_{dep}$ and $S_c$ after each iteration. For the adaption we use simple binary search, starting with a heuristic value of $n$ (we typically used $n = 5$). If this is successful, the interval between 1 and $n$ is traversed, otherwise the one between $n$ and $2n$. This procedure minimizes $S_c$. By choosing the minimum interval the quality of the solution and the computation time can be controlled.

For both the prescheduling and the final scheduling we adapt the algorithm of Chou [1] based on the traversal search through a graph. As a result of scheduling, we get a serialized PDG and an $S_c$ which is close to the minimum for static scheduling. The cosynthesis system now tries to reach the required speedup by generating an application specific coprocessor.

A model train control serves as an example. The train is controlled by a personal computer, from which encoded control words are transferred over the rails. The model train has a rather powerful speed regulation which lets the train move similar to original large trains. The C* description consists of six processes (Fig. 5). Five of the processes are executed every 51 µs. The rate of the sixth process, a bit decoder, ranges from 3.12 KHz to 1.25 MHz ($R_{dec}$) depending on the chosen transfer rate between PC and train. In Fig. 4 the steps of the binary search for three different rates of the bit decoder are shown. Each of them converges to the minimum $S_c$ in a few steps. For this example the binary search terminates whenever the distance of the factors $a$ of the last valid and the next invalid schedule is smaller than 0.5.

![Fig. 4. Process scheduling for the train example.](image)

![Fig. 5. Block diagram of the train example.](image)
4. Hardware/software partitioning

The main interest in the project is the hardware/software partitioning. Approaches that also focus on an automated partitioning process are presented by [10] which start partitioning with an all-hardware solution (hardware oriented). All other approaches [11,6,5,22] (as well as in COSYMA) are software oriented.

Partitioning in COSYMA starts with an all software solution and tries to extract hardware components iteratively until all timing constraints are met. The partitioning goals are in order of decreasing importance:

- meet real-time constraints
- minimize hardware costs
- minimize the turnaround time and, thus, allow the user to investigate the influence of system modifications on systems costs.

While the first goal is a hard constraint (because it is a necessary condition), the other ones are (soft) optimization goals. Hardware/software partitioning can be considered as a scheduling problem which is known to be NP-hard. Given a user specified maximum amount of hardware components (i.e. ALU, Multiplier, ...), our task is to find a minimum set of code segments that lead to a sufficient speedup when implemented in hardware. We selected the Simulated Annealing [14], a stochastic optimization algorithm. The benefits are:

- flexibility in cost function
- the possibility of a quality/computation time trade off
- it is an algorithm with mathematically well investigated behavior
- it allows an incremental cost function computation.

The granularity in the partitioning process are basic blocks which seems to be be a manageable compromise between statement level (fine grain) on the one side and function level partitioning (coarse grain) on the other side. The total (estimated) costs of a single basic block \( b \) — assumed that it is moved from software to hardware — amounts to:

\[
\Delta E(b) = w \cdot (t_{HW}(b) - t_{SW}(b)) + t_{\text{com}}(Z) - t_{\text{com}}(Z \cup b) \cdot R(b) \tag{1}
\]

where \( \Delta E(b) \) is the estimated decrease in execution time, \( w \) is the weight factor to control simulated annealing [15]. \( t_{HW}(b) \) is the estimated coprocessor execution time of \( b \), \( t_{SW}(b) \) is the estimated execution time of \( b \) on the processor and \( t_{\text{com}}(Z) \) is the estimated processor-coprocessor communication time, given the current set \( Z \) of basic blocks on the coprocessor. It(b) is the number of iterations as determined by profiling/tracing on \( b \). All estimations are required previous to partitioning because simulated annealing needs a fast cost function computation.

\( t_{SW}(b) \) is estimated with a local source code timing estimation. Estimation inaccuracy results from data dependent instruction execution times (LSI-SPARC: e.g. mult, div), optimization and register allocation in the compiler. Alternatively, trace data from processor simulation could be used (not in this paper).

\( t_{HW}(b) \) is estimated with a path-based scheduler on \( b \) using the execution time (number of clock cycles) for each operator in high-level synthesis, given the user specified number of functional units used.

\( t_{\text{com}}(Z \cup b) \) is estimated by data flow analysis of adjacent basic blocks in the control flow. A global data flow analysis for each \( Z \cup b \) is too computation intensive (exponential). For shared memory, communication costs are proportional to the number of variables to be communicated: Let \( C_{a,b} \) the number of variables flowing from \( a \) to \( b \) for the case that \( a \) and \( b \) are adjacent blocks, and \( 0 \) otherwise. At fixed transfer costs \( t_{\text{trans}} \) the communication costs \( t_{\text{com}}(Z \cup b) \) are:

\[
t_{\text{com}}(Z \cup b) = t_{\text{com}}(Z) - \sum_{a \in Z \setminus b} C_{a,b} - \sum_{d \cup b \setminus Z} C_{d,b} \cdot t_{\text{trans}} \tag{2}
\]

Here \( Z \) is the set of blocks which are not on the coprocessor.

- Since we have had to handle hard constraints and soft optimization goals a special treatment of the applied Simulated Annealing is necessary. So the weight \( w \) is chosen such that it drives the estimated system execution time \( T_S \) towards the required execution time \( T_r \) with a minimum number of basic blocks in the coprocessor while slowing down the automatic cooling process [14] as long as the timing requirements are met.

\[
w = \text{sign}(T_C - T_S) e^{\frac{T_C - T_S}{T_r - T_C}} \tag{3}
\]

With the increments \( \Delta C \) and \( \Delta t \), the total costs \( C \) and the time \( T_S \) of the codes are updated as follows (assuming \( t \) refers to the last move of the annealing algorithm):

\[
C = C_{-1} \pm \Delta C \tag{4}
\]

\[
T_S = T_{S-1} \pm \Delta t \tag{5}
\]

where the upper operators refer to a SW \( \rightarrow \) HW move and the lower operators refer to a HW \( \rightarrow \) SW move.

Table 1 shows the behaviour of the Simulated
Annealing for a benchmark of three different examples: smooth represents a filter that smooths the edges of a digital image; key is part of an HDTV studio equipment that calculates the parameters for a chromakey mixer and trick implements a part of an algorithm for a professional online trick animation. The columns show the following: loc are the lines of C-code, BBs gives the number of basic blocks moved to hardware using the constraint constr. and moves is the corresponding number of moves needed by the Simulated Annealing.

Intuitively it should be expected that the number of moves increases with loc (increasing complexity). In fact there is no simple relationship between increasing complexity and number of moves since the cost function tries to hit the condition \( T_S = T_C \) as close as possible. If simulated annealing cannot closely match the given constraint (e.g. the level of granularity is too coarse), the temperature of the annealing decreases slowly and thus the number of moves increases. This is the tribute we have to pay when using the very helpful cost function 3.

The computations in every move of the annealing schedule are inexpensive (column time). However, they would require a lot of computation time when timing and cost would be computed exactly at annealing time. In order to generate a codesign that is of high quality with acceptable computation time the estimations of all the cost function components (for hardware runtime estimation see [23]) are precomputed. For a more detailed description of the Simulated Annealing behaviour see [24]. Partitioning effects are described in [25].

5. Hardware synthesis

COSYMA can use 3 high-level synthesis systems: OLYMPUS [16], VOTAN, and BSS. For VOTAN, a synthesis tool from Siemens, and SYNOPSIS Design Compiler, RT-level VHDL-code is generated using the path-based scheduler of the COSYMA hardware timing estimator. BSS (Braunschweig Synthesis System) is a high-level synthesis system developed specifically for fast coprocessor design. In this paper, we will focus on BSS.

During partitioning, those basic blocks and functions which shall be implemented in hardware are marked in the ESG. In the first step, they are translated to the internal format of BSS which is an hierarchical CDFG. The marked basic blocks typically form sequences which are grouped as segments. When the coprocessor is activated by the processor, the processor sends a segment identifier to a reserved address, which activates the coprocessor and lets it execute the corresponding segment. If we regard a segment as a complex instruction, we could consider the coprocessor to be extremely vertically microcoded. The data are passed in shared memory. Array variable accesses are split to index computation and memory read or write access. Pointed variables are treated similarly. Passing array pointers instead of scalar variables drastically speeds up communication. The current version of BSS cannot handle functions, so they are inlined at translation time. So, the generated CDFG contains the segment identifier recognition, reading and writing operations to the shared memory for variable passing and the translated ESG segment. This segment can contain variables which are not in shared memory. Finally, the profiling and tracing information is included for synthesis and timing optimization.

The unmarked, software part of the ESG is translated to C including segment identifier and parameter passing to the coprocessor. Suitable type definition and additional statements make sure that shared variables are not register allocated at the time of coprocessor activity. The GNU C compiler with maximum optimization then generates object code for the processor.

BSS uses loop pipelining [26] and speculative computation with multiple branch prediction (MBP-SC) [27], [28]. MBP-SC is an adaptive (dynamic) scheduling approach, which allows to efficiently apply deep loop pipelining even if there are several if-statements within a loop. This has shown to be very powerful and the speedup almost always ends with a bandwidth limitation of the fast memory.

6. Timing analysis

Timing analysis delivers the timing data to ESG for the cost estimation of the hardware/software partitioning, and checks the timing constraints given in C'. Checking of the timing constraints must be repeated in the iterative hardware/software partitioning. Therefore, a fast timing analysis approach is required.

There are many software timing estimation and analysis techniques in use today: program profiling [29,30], simulation [31], formal approaches [32–34]. Formal approach take much less simulation time than the simulation approach [35]. We found that, to be accurate enough, complex processor cores with load-store architecture, pipelining, and concurrent units (or floating point coprocessor) require an analysis that models the data and control flow in the processor, which known approaches to runtime analysis cannot provide. Experimental with larger examples, however, lead to RT level simulation times of more than 0.5 hours on SPARC 10/41 for each input pattern, even in the unpartitioned case. Iteration would be impractical under these circumstances. So, we developed a hybrid approach combining RT-level simulation and formal analysis (HTA), giving results that in our experiments with software timing analysis, deviate less than 10% from RT-level simulation with a few seconds of computation time per input pattern [36]. Originally, it was limited to single process systems.

We have now extended HTA to analyze system timing

7. Results

Table 2: Hardware cosynthesis

| Table 2: Hardware cosynthesis
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthesis</td>
</tr>
</tbody>
</table>

with multi-process performance analysis (execution time) 1.

1 Synthesis is done for the complete system.

2 The speedup is given for the complete system.

8. Conclusion

COSYMA cosynthesis can translate high-level system descriptions into high-performance hardware/software partitions with speedup limits on the architectural level. The partitioning tool COSYMA is implemented in C++ and is available as shareware.

The experimental results indicate that the presented approach is a viable solution for the hardware/software partitioning problem. The detailed analysis of the resulting hardware/software partitioning provides valuable insights into the design process of complex system. The presented approach can be extended to handle more complex system descriptions, such as those containing multiple processors and communication networks.

Table 2: Hardware cosynthesis

| Table 2: Hardware cosynthesis
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthesis</td>
</tr>
</tbody>
</table>

The presented approach is evaluated on several benchmark examples, including a high-level synthesis system for the translation of C-code into hardware and software partitioning. The experimental results show that the approach is able to generate high-performance hardware/software partitions with speedup limits on the architectural level.

The presented approach can be extended to handle more complex system descriptions, such as those containing multiple processors and communication networks.
with multiple concurrent processes. The extended timing analysis has a protocol table to determine worst case execution times (worst case may change during partitioning). Timing constraints checking and process scheduling of hardware/software partitioning will be performed using the timing data from this table. To assert the execution time of tasks, we currently enhance HTA for COSYMA with symbolic simulation [37].

7. Results

Table 2 shows the benchmarks presented in Section 4 Hardware/software partitioning after partitioning and synthesis1. Column geq shows that the total hardware effort (in terms of gate equivalents) is quite small though the gained speedup2 is relatively high. The benchmark trick for example achieves a speedup of 9.59 with additional hardware costs of only 24,070 gate equivalents (a small part of the processor costs). Hereby the controller effort is not included.

Experiences with other examples have shown that an additional overhead of about 10–15% has to be added for the controller. Synthesis used an ES2 1.0 μm standard cell library and a frequency of 33 MHz (same as used for the standard processor kernel, a LSI-SPARC).

The interplay of the partitioning step and synthesis step can be seen by the good correspondence of the given constraint constr. and the actual speedup received by BSS. For more details of adaptation of partitioning and synthesis see [38].

8. Conclusion

COSYMA has shown that hardware/software co-synthesis of embedded processor-coprocessors configurations can provide considerable speedup with little overhead even compared to 32-bit-RISC implementations with fast memory and compiler optimization. Speedup limitations are mainly due to the memory architecture. Scalable performance scheduling allows the global speedup of control tasks consisting of many processes with different requirements.

References


Table 2

<table>
<thead>
<tr>
<th>Benchm.</th>
<th>lOC</th>
<th>BSBs</th>
<th>geq</th>
<th>constr.</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>smooth</td>
<td>95</td>
<td>5</td>
<td>18 320</td>
<td>2.0</td>
<td>4.39</td>
</tr>
<tr>
<td>trick</td>
<td>240</td>
<td>5</td>
<td>20 260</td>
<td>5.0</td>
<td>6.50</td>
</tr>
<tr>
<td>key</td>
<td>1421</td>
<td>11</td>
<td>24 070</td>
<td>10.0</td>
<td>9.59</td>
</tr>
</tbody>
</table>

1 Synthesis is done with the Braunschweig Synthesis System BSS.
2 The speedup values are results of the high-level synthesis.


[34] P. Puchner and Ch. Koza, Calculating the Maximum Execution Time of Real-Time Programs, Real-Time Syst. (September 89) 159–176.


Rolf Ernst is a professor of electrical engineering at the Technical University of Braunschweig, Germany. His research interests are VLSI CAD and digital circuit design. Previously, he was a member of the technical staff in the CAD and Test Laboratory of AT&T Bell Laboratories and a research assistant at the University Erlangen, Germany. He holds a diploma in computer science and a PhD in electrical engineering from the University of Erlangen. He is a member of the IEEE, the IEEE Computer Society, and the German GI (Society for Computer Science).

Jörg Henkel is pursuing a PhD in electrical engineering from the Technical University of Braunschweig, Germany. Since 1991 he is involved in the development of the experimental system COSYMA for hardware/software codesign. His interests include hardware/software codesign, high-level synthesis and computer architecture. He holds a diploma in electrical engineering from the Technical University of Braunschweig. Email: henkel@ida.ing.tu-bs.de

Thomas Benner is research assistant at the Technical University of Braunschweig, working on the Cosyma project. His interests are hardware-software codesign, process scheduling and computer architectures. He holds a diploma in computer science from the Technical University of Braunschweig, where he is pursuing a PhD in electrical engineering.

Wei Ye is pursuing a PhD in computer science from the Technical University of Braunschweig, Germany. His interests include timing analysis of real time systems, hardware/software codesign and high-level synthesis. He holds a diploma in computer science from the Huazhong University of Science and Technology, Wuhan, China.

Ubirch Holtmann received the M.S. degree in computer science from the Technical University Braunschweig, Germany, in 1989. He is currently working towards the PhD degree in computer science at the Department Entwurf integrierter Schaltungen, Technical University Braunschweig. His research interests are high level synthesis and full-custom design.

Dirk Herrmann, born in 1968, holds a diploma in computer science from Technical University Braunschweig, Germany. Since August 1994 he is research assistant at the Institut für Datenverarbeitungsanlagen, TU Braunschweig. Currently he is working on a further improvement of the high level synthesis system BSS (Braunschweig Synthese System). His research interests are high level synthesis, compiler optimization techniques and memory architectures.

Dipl.-Ing. Ulrich Holtmann received the M.S. degree in computer science from Technical University Braunschweig, Germany, in 1995. At the Institut für Datenverarbeitungsanlagen he is working as research assistant since 1994. Mr. Holtmann is involved in the development of COSYMA. His interests include hardware/software codesign, especially compiling techniques.

Rolf Ernst is a professor of electrical engineering at the Technical University of Braunschweig, Germany. His research interests are VLSI CAD and digital circuit design. Previously, he was a member of the technical staff in the CAD and Test Laboratory of AT&T Bell Laboratories and a research assistant at the University Erlangen, Germany. He holds a diploma in computer science and a PhD in electrical engineering from the University of Erlangen. He is a member of the IEEE, the IEEE Computer Society, and the German GI (Society for Computer Science).