A Computer Architecture to Support Qualitative Simulation in Industrial Applications*

M. Platzner IEEE†, B. Rinner IEEE‡, and R. Weiss ÖVE, IEEE§

Abstract

Monitoring and diagnosis of dynamic systems in industrial environments, like assembly lines and power plants, are challenging tasks. Faulty behaviors must be detected as soon as possible to avoid shutdown or damage. Recently, techniques from artificial intelligence (AI) have been applied to achieve these tasks. To overcome performance problems for the industrial application of the rather new AI technique *qualitative simulation*, a special-purpose computer architecture has been developed.

keywords: application-specific computer architecture, monitoring and diagnosis, qualitative simulator QSim, parallel processing, software/hardware migration

Die Überwachung und Fehlerdiagnose dynamischer Systeme in industriellen Umgebungen, wie z.B. Produktionsanlagen oder Kraftwerke, zur frühzeitigen Fehlererkennung und Vermeidung von Systemausfällen oder Systemschäden stellen eine große Herausforderung dar. In jüngster Zeit werden für diese Aufgaben Techniken aus dem Bereich der künstlichen Intelligenz (KI) eingesetzt. Herkömmliche Rechnersysteme weisen für den industriellen Einsatz der neuen KI-Technik *Qualitative Simulation* eine zu geringe Rechenleistung auf. Um dieses Problem zu überwinden, wurde eine Spezialrechnerarchitektur entwickelt.

Schlüsselwörter: anwendungsspezifische Rechnerarchitektur, Überwachung und Fehlerdiagnose, qualitativer Simulator QSim, Parallelverarbeitung, Software/Hardware Migration

*This work was partially supported by the Austrian National Science Foundation *Fonds zur Förderung der wissenschaftlichen Forschung* under grant number P10411-MAT.

†Dipl.–Ing Dr. Marco Platzner, Institute for Technical Informatics, Technical University Graz, Steyrergasse 17/4, A-8010 Graz (current address: GMD — German National Research Center for Information Technology, Schloss Birlinghoven, D-53757 St. Augustin, Germany. E-Mail: marco.platzner@gmd.de)

‡Dipl.–Ing. Dr. Bernhard Rinner, Institute for Technical Informatics, Technical University Graz, Steyrergasse 17/4, A-8010 Graz. E-Mail: rinner@iti.tu-graz.ac.at

§O. Univ.–Prof. Dipl.–Ing. Dr. Reinhold Weiss, Institute for Technical Informatics, Technical University Graz, Steyrergasse 17/4, A-8010 Graz. E-Mail: rweiss@iti.tu-graz.ac.at

1 Introduction

As many technical systems, like assembly lines and power plants, are becoming more and more complex, the need for automatic control systems is increasing enormously. The task of the control system is often to monitor the technical system and to detect faulty behaviors as soon as possible to avoid system shutdown or damage. It is nearly impossible to achieve this ambitious task without any computerized support. Recently, techniques from *artificial intelligence (AI)* have been applied in control systems for such technical systems.

A rather new AI technique is *qualitative simulation*. In qualitative simulation, physical systems are modeled on a higher level of abstraction than in other simulation paradigms, e.g., in continuous simulation. In continuous simulation, the structural description of a physical system is modeled by a mathematical description in form of differential equations. Qualitative simulation relies on a further abstraction of these differential equations — the so-called *qualitative differential equations (QDEs)*. Qualitative simulation requires neither a complete structural description of the physical system nor a fully specified initial state. The major strength of qualitative simulation is the prediction of all physically possible behaviors derivable from this incomplete knowledge.

1.1 Application Areas of Qualitative Simulation

The qualitative simulation paradigm is mainly used in applications where a detailed description of a physical system is not required or even not known. Qualitative simulation is furthermore a key inference technique of model-based reasoning — i.e., given the (qualitative) model of the system, the goal is to predict the possible behaviors consistent with that model. This inference technique is applied in areas, like monitoring, fault diagnosis and explanation:

- **monitoring**

  Based on a model of the system and a behavior predicted by that model, the task is to monitor the system's behavior and to check for malfunctions. This is achieved by identifying discrepancies between observations (measurements) and predictions (simulation

results).

- **fault diagnosis**

  Based on a model of the system, the task is to detect faults and to identify faulty components. The predicted behavior is compared with the actual observations, producing discrepancies. Discrepancies in turn give rise to a possible diagnosis, i.e., identification of the faulty device(s). Compared to rule-based systems where only already known faults can be identified, a model-based diagnosis covers a broader range of faults by viewing misbehavior as anything other than what the model predicts.

- **explanation**

  Based on a sequence of observations, the task is to find one or more models consistent with the observation. This is done by extracting the relevant features of those models, e.g., cause-effect relationships, and communicating them to the user.

1.2 Application Examples of Qualitative Simulation in Dynamic Systems

For industrial applications diagnosis and monitoring are probably the most interesting domains. An increasing amount of research has been done in the field of qualitative simulation for these domains. Most of these papers demonstrate the successful application of qualitative simulation in monitoring and diagnosis based on small to medium complex dynamic systems. Dvorak and Kuipers [1] describe a monitoring system using qualitative simulation in the example of water heater. In Lackinger and Nejdl [5], a diagnosis and monitoring system for a central heating system is presented. Subramanian and Mooney [8] demonstrate a multiple-fault diagnosis system for the reaction control system of the space shuttle.

Condition Monitoring for Gas Turbines

Probably the most advanced industrial application of qualitative simulation in the area of monitoring and diagnosis is the recently completed ESPRIT III project Tiger [9]. In the Tiger project, a condition monitoring system for gas turbines has been developed. Qualitative simulation is

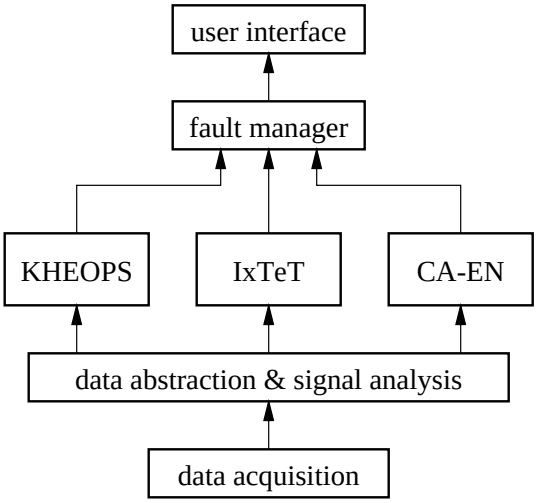

Figure 1: Diagnosis architecture of Tiger. Reasoning is performed at three different levels using the tools KHEOPS, IxTeT and CA–EN.

used to predict the behaviors of the turbine at start up and in response to load changes. Application sites include a large industrial turbine at Exxon Chemical in UK and a small aircraft auxiliary power unit turbine at Dassault Aviation in France.

The diagnosis architecture of Tiger [10] (see Figure 1) consists of 3 independent tools. They can work in parallel, each reasoning with knowledge at different levels. At the lower level, KHEOPS reacts in real-time to the current state of the process. It is a high speed rule-based system and is used primarily for limit checking. However, it does not have a global view on the past history and evolution of the process. Such a view is given by the second level in the hierarchy. Here in IxTeT, the user is able to specify a sequence of events corresponding to normal or faulty situations. IxTeT then monitors the turbine to ensure that the sequence is properly followed. The more complex causal reasoning mechanisms are devoted to the third level. Reasoning at this level is based on CA–EN, a model-based diagnosis system using qualitative simulation. Because diagnosis at this level may require non-deterministic searches, it does not meet real-time constraints. The results of all tools are coordinated by the fault manager, and the conclusions are communicated to the user (technician) via the user interface.

1.3 A Specialized Computer Architecture for QSim

For industrial applications of qualitative simulation it is of utmost importance that diagnosis and monitoring are done on-line. High-performance qualitative simulators are required in these environments. However, the main drawback of current qualitative simulators is poor runtime performance. Models of only low to medium complexity can be simulated within reasonable execution time.

QSim [4] is the best-known and widely-used algorithm for qualitative simulation. In the past years, QSim has been widely studied, applied and extended both by the original developers and by researchers worldwide. The lack in runtime performance of current QSim implementations is basically caused for two reasons. First, QSim tends to generate a huge number of system behaviors during simulation. This can be at least partially avoided by improved filter algorithms. Research in this area is done by the Qualitative Reasoning group at UT Austin. Second, QSim is implemented in LISP and executed on general-purpose computers. In a research project at the Institute for Technical Informatics, Technical University Graz, a special-purpose computer architecture for QSim has been developed. The primary goal of this application-specific computer architecture is to increase the performance. Improved performance is achieved by methods of computer architecture — i.e., parallelizing and mapping of QSim functions onto a multiprocessor system and migrating functions from software to hardware. Furthermore, plans have been already made to integrate qualitative simulation into the *multi-agent system for model-based real-time fault diagnosis* at our institute (compare article in this issue).

The remaining part of this paper deals with the design and implementation of this specialized computer architecture. Section 2 introduces briefly the qualitative simulator QSim. In Section 3, the design of a multiprocessor and specialized coprocessors for QSim as well as experimental results are presented. Section 4 concludes this paper.

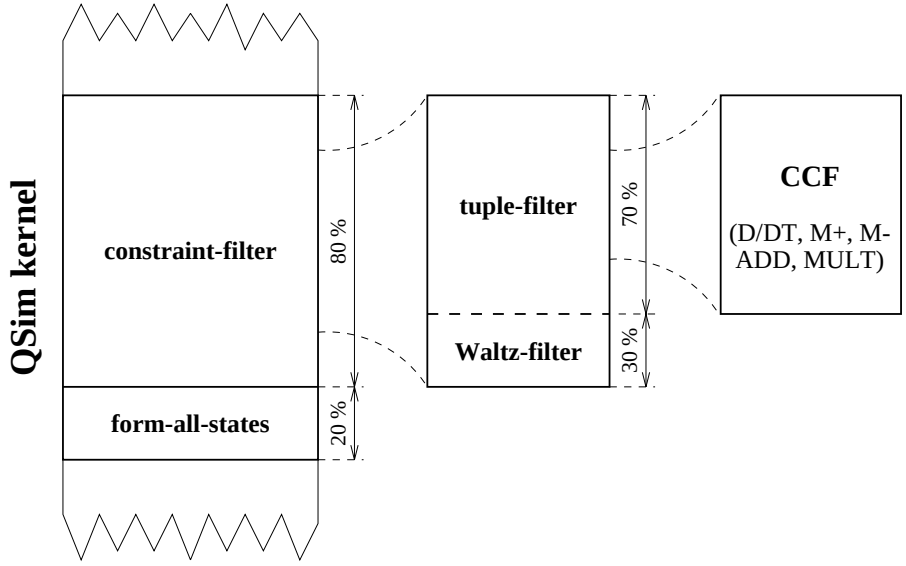

Figure 2: Hierarchical structure and runtime analysis of the QSim kernel. The runtimes are informally presented with regard to the runtime of the calling function.

2 Qualitative Simulator QSim

In QSim, models are described as qualitative differential equations or equivalently as *constraint–networks*, which consist of *variables* and *constraints*. Variables represent system parameters, e.g., speed or temperature. The values of qualitative variables are expressed by two parts, a *qualitative magnitude* (qmag) and a *qualitative direction* (qdir). Constraints describe relations between system parameters. QSim uses several types of constraints which represent arithmetic relations (e.g., ADD–, MULT–, D/DT–constraints) and functional dependencies (e.g., M^+–, M^-–constraints) between variables.

The qualitative simulator QSim is a very complex algorithm and has many optional features. The design considerations for our specialized computer architecture are restricted to the QSim kernel functions. The kernel functions are essential in calculating one simulation step, and they normally dominate the overall runtime of QSim. Furthermore, several model-based fault diagnosis and monitoring systems do not require the functionality of the whole simulator [1] [5]. These systems are based on the QSim kernel functions.

Figure 2 presents an overview of the hierarchical structure of the QSim kernel functions.

The *constraint check functions (CCFs)* are primitive kernel functions. For each constraint type, an individual CCF exists. The CCFs are called by the *tuple-filter*. For each constraint of the input model one tuple-filter is required. The *constraint-filter* is generated by all tuple-filters and the *Waltz-filter* which is used for efficiency reasons. The final kernel function is called *form-all-states*. The presented runtime ratios in Figure 2 are extracted from various runtime measurements of a QSIM system implemented on a TI Explorer LISP workstation. Many input models were simulated and the runtimes of the individual functions were measured. The runtime ratios represent an average of all simulated models. For most models, the kernel functions require more than 50 % of the overall QSIM runtime. An important fact is that this percentage is positively correlated to the complexity of the model. Qualitative models for 'real-world' technical systems usually have many constraints and variables [3]. For these models, kernel runtime ratios of up to 90 % were observed. The empirical runtime analysis reveals that the tuple-filter and subsequently the CCFs dominate the kernel runtime.

3 QSIM Computer Architecture

The specialized computer architecture presented in this paper improves the runtime of the QSIM kernel by using two strategies. First, the parallelism in the complex kernel functions constraint-filter and form-all-states is exploited. These functions are parallelized and mapped onto a multiprocessor system. Second, the less complex CCFs are accelerated by software to hardware migration, i.e., the CCFs are directly implemented in hardware. In the following sections, design considerations for this specialized computer architecture, a prototype implementation and experimental results are described.

3.1 Design of the QSIM Kernel Multiprocessor

3.1.1 Constraint-Filter

The constraint-filter consists of a number of calls to the function tuple-filter and a final call to the Waltz-filter function. A data depencency analysis reveals a high parallelism within the

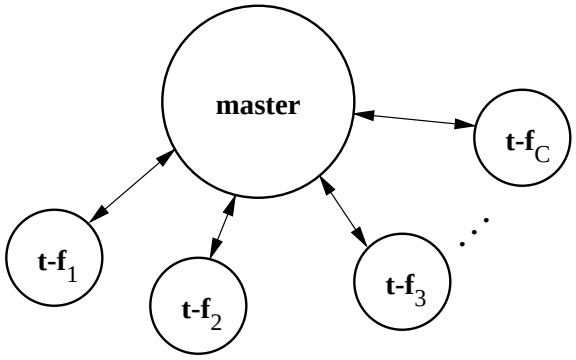

Figure 3: Logical structure of the constraint-filter.

constraint-filter. All tuple-filter functions are independent of each other and can be executed in parallel.

Figure 3 presents the logical structure of the constraint-filter. It consists of a set of tasks and communication links. The tasks are partitioned into two groups — a *master* task and a set of *slave* (tuple-filter) tasks. The master task is responsible for the transmission of the input data to all tuple-filter tasks, the reception of the tuple-filters' results and the execution of the Waltz-filter. Since all tuple-filters are independent of each other, no communication among the slaves is required. The maximum degree of parallelism for the constraint-filter is the number of constraints C. Therefore, the logical structure consists of C slaves.

The logical structure forms a *star* with the master as the central element. However, in a star structure the master becomes a bottleneck as the number of slaves increases. This limits the scalability of the computer architecture. To achieve a scalable architecture, our multiprocessor system is connected in a *wide tree* topology which is a compromise between logical structure and scalability. In a wide tree, each processing element has a constant node degree and, hence, a fixed number of required communication links. The root node of the tree corresponds to the master task; all other nodes correspond to the slaves of the logical structure.

3.1.2 Form-All-States

The kernel function form-all-states solves a *constraint satisfaction problem (CSP)* by a back-tracking algorithm. A big search space has to be processed by a depth-first search to find all

solutions of the CSP. Contrary to the constraint-filter, there is no obvious parallelization given by the function hierarchy of form-all-states. For a parallel implementation of form-all-states, the CSP must be partitioned artificially. A *parallel-agent-based (PAB)* strategy [6] is used for the parallelization of the CSP in our QSim architecture. The basic idea of PAB is to partition the overall search-space into smaller independent subspaces which can be solved by any sequential CSP algorithm in parallel. The partitioning step is essential for the performance of the parallel implementation. A *variable-based partitioning (VBP)* heuristic [7] is used to partition the search space of QSim CSPs.

The logical structure of the parallel form-all-states algorithm is similar to the logical structure of the constraint-filter. Due to the PAB strategy, a master/slave structure is also derived. The master task is responsible for the generation and transmission of subproblems to the slave tasks and for merging the partial results to the overall result. The maximum degree of parallelism is determined by the number of generated subproblems. The same architectural considerations for the constraint-filter are also valid for form-all-states.

3.2 Design of the CCF Coprocessor

The CCFs are executed on specialized coprocessors. This section presents the coprocessor design for the MULT-CCF. The MULT-CCF is one of the most complex CCFs; CCFs for other constraint types are less complex but very similar in structure.

An analysis of the MULT-CCF reveals that this CCF can be partitioned into several subfunctions. The partitioned MULT-CCF is shown in Figure 4. Subfunction SF1, *value check*, tests the signs and directions of change of the input values. Subfunction SF2, *infvalue check*, tests relations between infinite and zero input values. Subfunction SF3, *cval check*, tests the input values against all tuples from the list of corresponding value tuples. Corresponding value tuples are stored in an internal memory of the coprocessor. SF1 to SF3 form the functionality of the MULT-CCF. SF4 performs a logical AND operation on the partial results of SF1 to SF3.

The specialized coprocessor implementing the functionality of the MULT-CCF is designed at the gate- and register-level to obtain maximum performance. The main features of the design

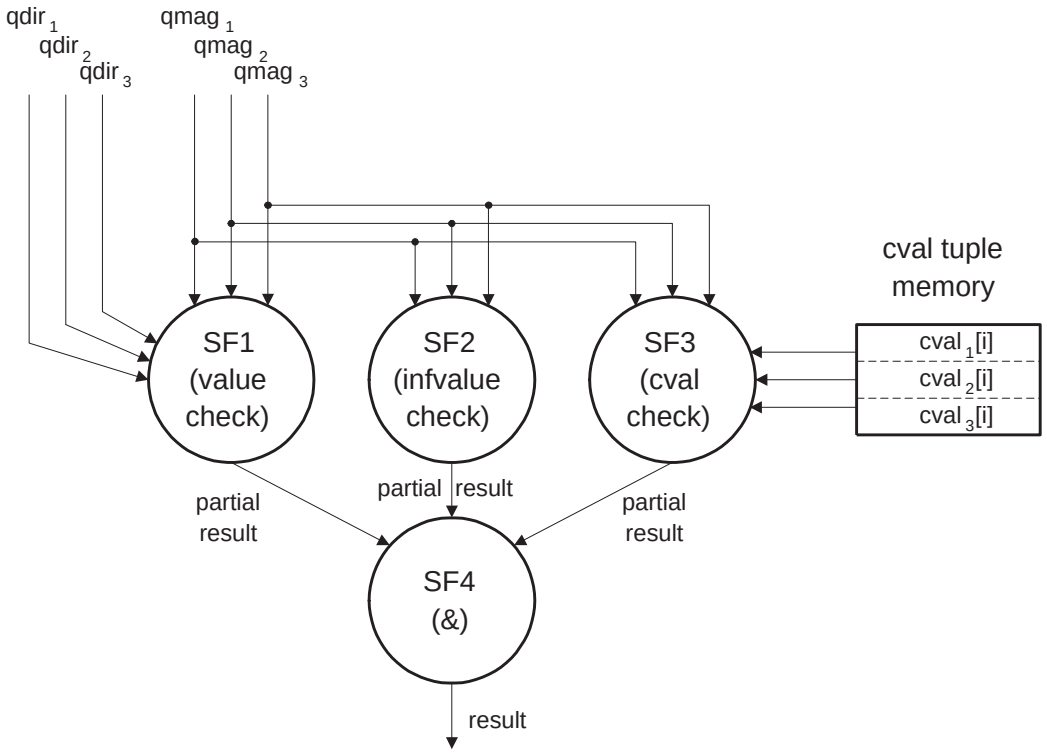

Figure 4: Partitioned MULT-CCF. The MULT-CCF is partitioned into three subfunctions SF1 to SF3.

are (i) *exploitation of parallelism*, i.e., the parallel execution of SF1, SF2 and SF3, (ii) *use of optimized data types*, i.e., the number of bits and the coding scheme of the input values and the corresponding value tuples, and (iii) *use of customized memory architectures*, i.e., the internal organization and the access mode of the cval tuple memory.

The coprocessor design contains functional blocks for SF1 to SF4, the internal memory, an I/O controller and a function controller [2]. The I/O controller establishes communication to a host processor via two separate communication channels which enable simultaneous input and output operations. The function controller decodes the instructions and controls the operation of all other functional blocks of the coprocessor. Three instructions are defined for the MULT-CCF coprocessor. Two instructions update the internal memory; the third instruction actually executes the MULT-CCF.

3.3 Prototype Implementation and Experimental Results

A prototype of the overall heterogeneous multiprocessor architecture is shown in Figure 5. The digital signal processor TMS320C40 was chosen as the processing element because of its high I/O performance and its 6 independent communication channels. Software is developed in 'C' under the distributed real-time operating system Virtuoso [11], which supports a portable and flexible software design. The specialized CCF coprocessors are implemented on field programmable gate arrays (FPGAs).

3.3.1 QSim Kernel Multiprocessor

The experimental evaluation of the QSim kernel multiprocessor is based on a comparison of the execution times of the sequential implementation, t_{seq}, and the parallel implementation using n processing elements, $t_{par}(n)$. With these execution times, the speedup $S(n) = \frac{t_{seq}}{t_{par}(n)}$ can be determined. The execution times are measured on a prototype of the QSim kernel multiprocessor by using a 32 bit timer of the TMS320C40 with a resolution of 80 ns. The sequential implementation of the QSim kernel is executed on the root node of the QSim kernel multiprocessor. In the next two sections, the QSim kernel multiprocessor is evaluated independently for both

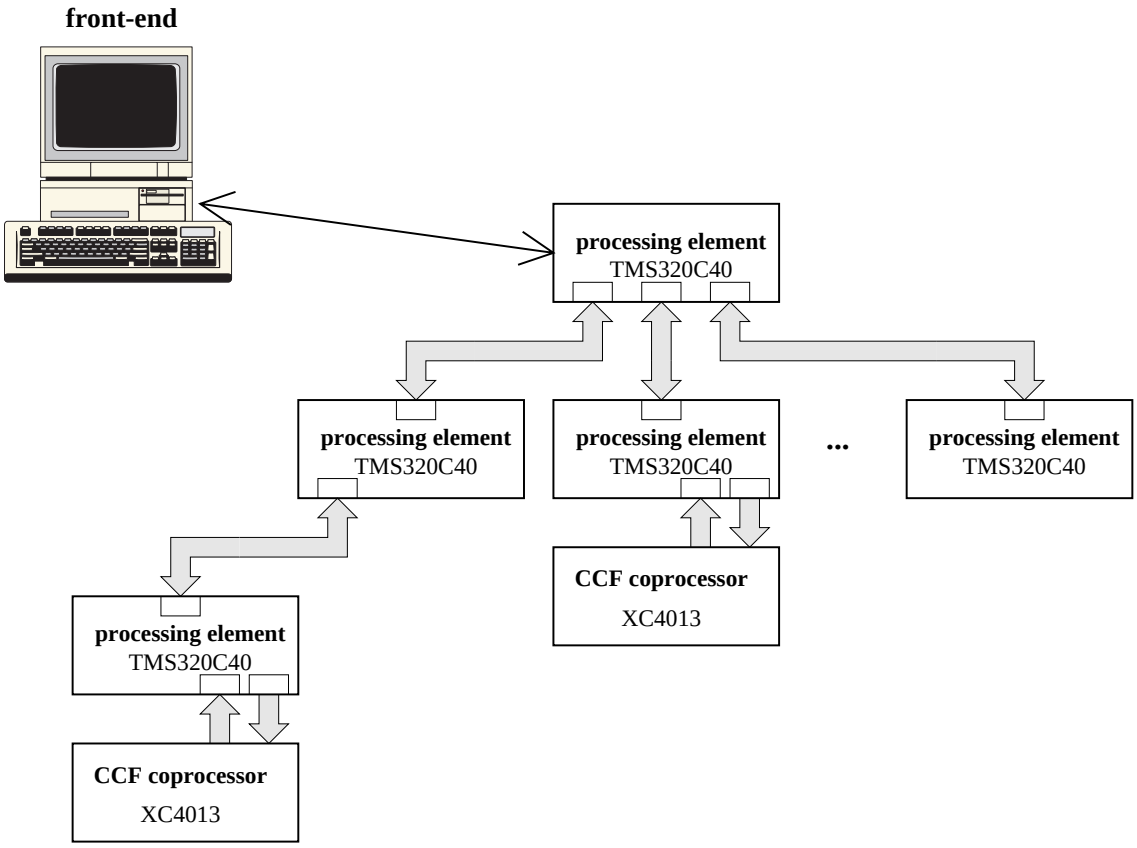

Figure 5: Example of the overall architecture for the QSim kernel multiprocessor. The processing elements are connected in a wide tree structure. Some processing elements are equipped with CCF-coprocessors.

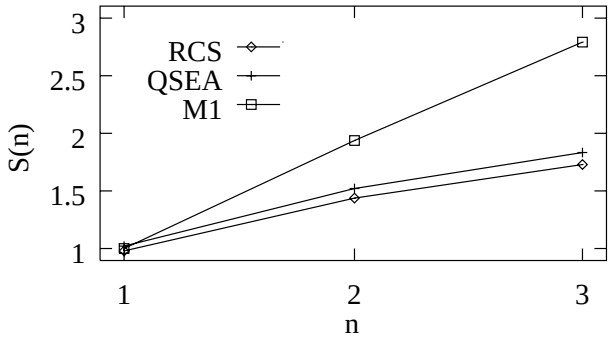

Figure 6: Speedup $S(n)$ of the parallel implementation of the constraint-filter for the models RCS, QSEA, and M1 using $n = 1 \ldots 3$ slave processors.

kernel functions, constraint-filter and form-all-states.

Constraint-Filter. The parallel implementation of the constraint-filter is evaluated using three different sets of input data. Two sets are derived from the QSim models RCS (48 constraints) [3] and QSEA (21 constraints) and one set is constructed artificially (M1). In the data sets RCS and QSEA, the numbers and types of the tuple-filter tasks as well as the numbers of tuples which must be checked vary. The data set M1 consists of 30 tuple-filters of type MULT; each tuple-filter must check 64 tuples. From all data sets, M1 has the longest execution times of the individual tuple-filter tasks.

Figure 6 presents the speedups of the parallel implementation of the constraint-filter using 1, 2, and 3 slave processing elements. The best speedup is achieved with input data set M1.

Form-All-States. The parallel implementation of form-all-states is evaluated using CSPs derived from the simulation of the QSim models RCS and QSEA. Figure 7 presents the speedups of the parallel implementation of form-all-states by using up to 7 slave processors. Parallel execution of the RCS model reveals a superlinear speedup using one and two slave processor(s). This occurs because the partitioning algorithm discards many inconsistent subproblems, and the total execution time of the remaining consistent subproblems is smaller than the execution time of the unpartitioned problem.

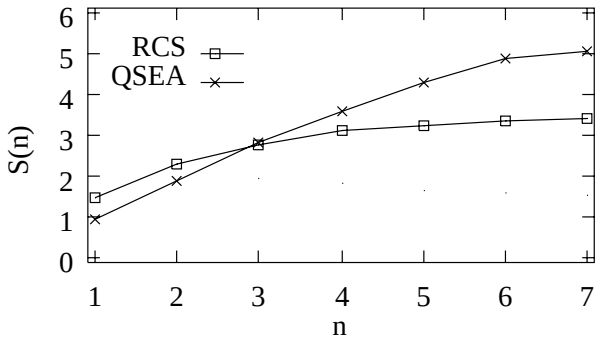

Figure 7: Speedup $S(n)$ of the parallel implementation of form-all-states for the models RCS and QSEA using $n = 1 \ldots 7$ slave processors.

3.3.2 CCF Coprocessor

The experimental evaluation of the CCF coprocessors is based on a comparison of the execution times of the software CCF, t_{sw}, with the execution times of the pair host and coprocessor, t_{hw}, for the CCF. From these execution times, the overall speedup of the coprocessor, $S_{tot} = \frac{t_{sw}}{t_{hw}}$, is calculated. This overall speedup also respects the required communication between the host and the coprocessor. The coprocessor execution times are measured on a MULT-CCF coprocessor prototype with the sequential execution of SF3 iterations. This coprocessor is implemented on an FPGA of type Xilinx XC4013 and is operated at a clock frequency of 15 MHz [2]. The measured execution times and the calculated speedups are subdivided into 6 cases according to the subfunction which causes termination of the MULT-CCF. For the short-circuit-evaluation of the software CCF, the execution order SF1, SF2, SF3 is assumed. Six cases are differentiated where case 1 denotes only execution of SF1, and case 6 denotes execution of SF1, SF2 and four iterations of SF3. These six cases reflect the most likely situations. In Figure 8, the overall speedup of the coprocessor, S_{tot}, is presented based on the six execution cases.

4 Conclusion

In this paper, we presented the design and the prototype implementation of a specialized computer architecture necessary for the qualitative simulator QSim to overcome the performance

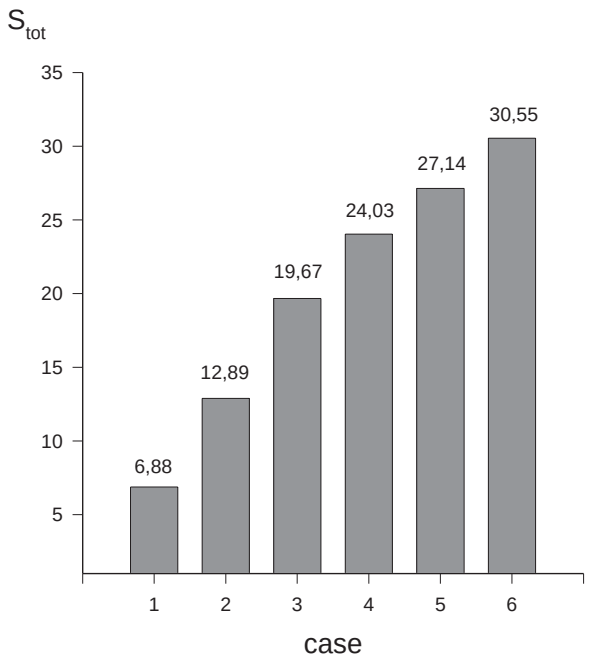

Figure 8: Speedup S_{tot} of the MULT-CCF coprocessor dependent on 6 different execution cases.

problems of current QSim implementations. The experimental results proved that a significant speedup can be achieved with this computer architecture.

In general, our computer architecture exploits the parallelism in the QSim kernel functions. The degree of parallelism depends strongly on the input simulation model. More complex models, i.e., models with many constraints and variables, lead to higher degrees of parallelism. Due to the high complexities of models for 'real-world' problems, the efficiency of existing qualitative simulators poses a significant barrier to their application. The presented special-purpose computer architecture will help to remove this barrier.

References

[1] Dvorak, D., Kuipers, B.: Process Monitoring and Diagnosis: A Model-Based Approach. IEEE Expert, 5(3) (1991), p. 67–74.

[2] Friedl, G., Platzner, M., Rinner, B.: A Special–Purpose Coprocessor for Qualitative Simulation. Proceedings of the First International EURO-PAR Conference, (1995) p. 695–698, Stockholm.

[3] Kay, H.: A qualitative model of the space shuttle reaction control system. Technical Report AI92-188 (1992), University of Texas.

[4] Kuipers, B.: Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge. MIT Press, 1994.

[5] Lackinger, F., Nejdl, W.: Diamon: A Model-Based Troubleshooter Based on Qualitative Reasoning. IEEE Expert, 8(1) (1993), p. 33–40.

[6] Luo, Q., Hendry, P., Buchanan, J.: Strategies for Distributed Constraint Satisfaction Problems. Proceedings 13th International DAI Workshop, (1994) Seattle, WA.

[7] Platzner, M., Rinner, B., Weiss, R.: Exploiting Parallelism in Constraint Satisfaction for Qualitative Simulation. J.UCS The Journal of Universal Computer Science, 1(12) (1995), p. 811–820.

[8] S. Subramanian, S, Mooney, R. J.: Multiple-Fault Diagnosis Using General Qualitative Models with Behavioral Modes. *International Joint Conference on Artificial Intelligence*, (1995), Montreal, CA.

[9] Esprit Project 6862: Real-Time Situation Assessment of Dynamic, Hard to Measure Systems, D510.36 Final Application Report (1995).

[10] Travé-Massuyès, L., Milne, R.: Diagnosis of Dynamic Systems Based on Explicit and Implicit Behavioural Models: An Application to Gas Turbines in Esprit Project TIGER. Applied Artificial Intelligence Journal, 10(3) (1996), p. 257–277.

[11] Verhulst, E.: Virtuoso: A virtual single processor programming system for distributed real-time applications. Microprocessing and Microprogramming, 40 (1994), p. 103–115.