Parallel Qualitative Simulation*

Marco Platzner, Bernhard Rinner, Reinhold Weiss
{marco, rinner, rweiss}@iti.tu-graz.ac.at

Institute for Technical Informatics
Graz University of Technology, AUSTRIA

The goal of qualitative simulation is to derive a characterization of the behavior of a dynamic system given only weak information about it. This fundamental strength of qualitative simulation is exploited more and more in applications, like design, monitoring, and fault diagnosis, nowadays. However, the poor performance of current qualitative simulators complicates or even prevents its application in technical environments.

In our research project [9] a special-purpose computer architecture for the widely-used qualitative simulator QSIM is developed. Two approaches are considered two improve the performance. Complex functions are parallelized and mapped onto a multiprocessor system. Less complex functions are directly implemented in hardware. These functions are executed on specialized coprocessors. This paper presents an overview of the design and implementation of this computer architecture.

keywords: special-purpose computer architecture,
qualitative simulator QSIM,
multi-DSP TMS320C40,
FPGA

1. Introduction

Qualitative simulation is a new and challenging simulation paradigm which belongs to the research area *qualitative reasoning (QR)*. In qualitative simulation physical systems are modeled on a higher level of abstraction than in other simulation paradigms — like continuous simulation. In continuous simulation the structural description of physical systems is modeled by a mathematical description in terms of differential equations. Qualitative simulations 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. Additionally, qualitative

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

simulation potentially predicts behaviors which are not physically possible. Hence, qualitative simulation is *complete* but *not sound*. Thus, the qualitative simulation paradigm is mainly used in applications where a detailed description is not required or even not known. Major applications areas are design, monitoring, and fault diagnosis.

The widely-used algorithm QSIM is the best-known representative of qualitative simulators. It has been developed by Kuipers [5]. In the past years QSIM has been widely studied, applied, and extended, both by the original developers and by researchers worldwide. None of these works study or analyze the computational complexity of QSIM or even present an empirical study of runtimes of qualitative simulators [1]. However, these complexity and performance issues are extremely important for applying qualitative simulation in embedded systems. High performance qualitative simulators are required in these environments. As a matter of fact current QSIM implementations lack in execution speed. Furthermore, investigations concerning guaranteed execution times are desired to allow the application of qualitative simulation in real-time environments.

In our research project [9] a special-purpose computer architecture for QSIM is developed with the primary goal to increase the performance. The design of this application-specific computer architecture is mainly based on an extensive analysis of QSIM implementations [11]. This analysis has been completed by runtime measurements of a QSIM implementation.

In the following sections of this paper we present the design and implementation of this special-purpose computer architecture — i.e. important results of QSIM analysis are presented in Section 2. An overview of the computer architecture design and first experimental results are given in Section 3. Some remarks for further work conclude this paper.

2. QSIM Analysis

The qualitative simulator QSIM is a very complex algorithm and has many optional features. Design considerations of this specialized computer architecture are restricted to kernel functions. Kernel functions are essential in calculating one simulation step, and they normally dominate the runtime of the complete simulator. Several model-based fault diagnosis and monitoring systems use qualitative simulation [2] [6]. These systems do not require the functionality of the whole simulator. However, QSIM kernel functions are required.

Figure 1 presents an overview of the kernel functions, which are hierarchically structured. The *constraint check functions (CCFs)* are primitive kernel functions but they dominate the overall kernel runtime. These functions 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 1 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 kernel functions require more than 50 % of the overall runtime. An important fact is that this

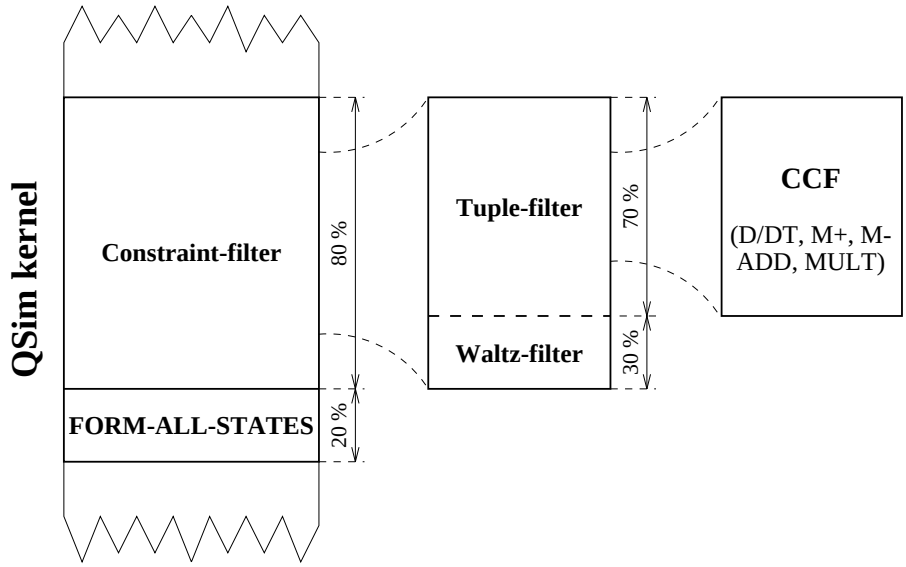

Figure 1. Runtime analysis of the QSIM kernel. Kernel functions are hierarchically structured and their runtimes are informally presented with regard to the runtime of the calling function.

percentage is positively correlated to the complexity of the model. Qualitative models for serious technical processes usually have many constraints and variables [4]. For these models kernel functions consume definitely more than 50 % of the overall runtime.

Constraint-filter The constraint-filter is generated by mutually independent functions (tuple-filter) and the Waltz-filter. The number of tuple-filters depends on the input simulation model. Waltz-filtering can be considered as a preprocessing step for the successive function of the QSIM kernel (FORM-ALL-STATES). It is used for efficiency reasons to reduce the search space for FORM-ALL-STATES as soon as possible.

FORM-ALL-STATES FORM-ALL-STATES is actually a backtracking algorithm to solve a constraint satisfaction problem (CSP) [8]. Although solving CSPs is NP-complete we did not experimentally observe an exponential behavior of this function [11]. The runtime of FORM-ALL-STATES remains nearly constant — even with complex models the runtime of FORM-ALL-STATES does not exceed 20 % of the kernel runtime.

Constraint Check Functions (CCF) Current QSIM implementations include many types of CCFs. Although these CCFs vary in their complexity, they only consist of primitive operations. Examples of these operations are the evaluation of boolean functions, comparisons, and table-lookups. Due to their frequent execution the CCFs dominate the overall kernel runtime.

3. QSIM Computer Architecture

According to the complexity of the kernel functions different approaches to increase the performance are considered. Complex kernel functions (like constraint-filter and FORM-ALL-STATES) are parallelized and mapped onto a multiprocessor system. Less complex functions (CCFs) are HW-implemented using FPGAs. These runtime intensive functions are executed on specialized coprocessors.

3.1. QSIM Multiprocessor

Parallelization of the constraint-filter is trivial. Tuple-filters are executed on individual processors. After all tuple-filter results have been received Waltz-filtering is executed. Since the Waltz-filter requires global access to all tuple-filter results, parallelization of the Waltz-filter is not considered.

FORM-ALL-STATES is parallelized by a *parallel agent based (PAB)* strategy [7]. The overall search space is partitioned into smaller independent subproblems which can be solved with any sequential CSP-algorithm. The overall result is formed by merging the results of the subspaces. Two aspects are of special interest for an efficient parallelization. First, the overall search space has to be partitioned into *equally* complex subproblems, and second, for a given subproblem the *fastest* sequential algorithm has to be chosen.

Both kernel functions (constraint-filter, FORM-ALL-STATES) can be parallelized in the same way. Important design issues of the multiprocessor are:

Topology A *wide* tree topology is used as a compromise between the optimal structure (star) and the scalability of the multiprocessor system.

Scheduling/Mapping The number of independent functions for parallelization is not known before runtime. Hence, dynamic mapping and dynamic scheduling of these functions is required.

We have introduced and evaluated several partitioning heuristics for QSIM-CSPs by a worst- and best-case speedup estimation [10]. Using certain heuristics a linear speedup is expected for a parallel implementation. Several sequential CSP-algorithms have also been compared using CSPs traced from QSIM. Especially for complex CSPs, some algorithms are more than 7 times faster than the sequential algorithm used in QSIM.

3.2. CCF Coprocessor

The CCF coprocessor is designed at the gate- and register-level. This is necessary to obtain maximum execution speed. Main features of the design are:

- data structures are optimized for the application QSIM
- operations use maximum parallelism
- customized memory architectures allow parallel access

Communication between the coprocessor and the hostprocessor is established via two separate communication channels. These unidirectional connections ease the I/O-controller design of the coprocessor and allow parallel input and output operations.

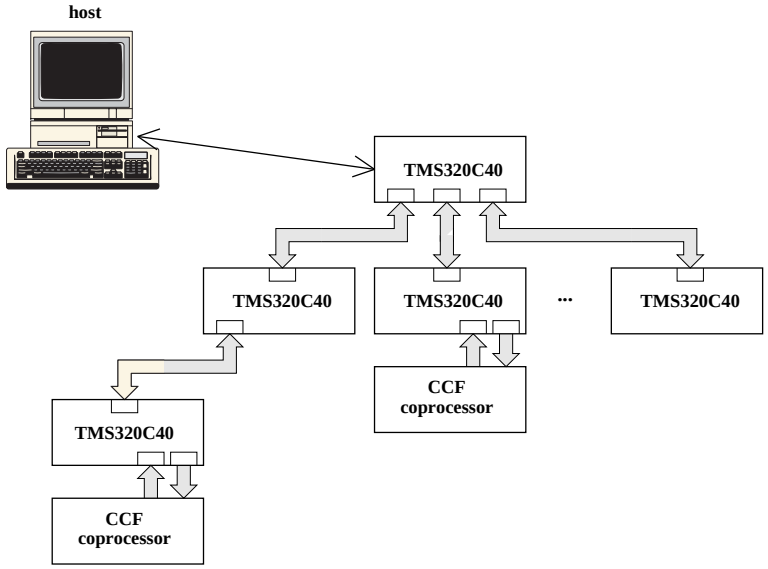

Figure 2. Example of the overall architecture. The processing elements are connected in a wide tree structure. Some processing elements are equipped with CCF-coprocessors.

First experimental results [3] show that the implementation of a CCF in an FPGA of type Xilinx XC4013 leads to a usage of 50 % totally occupied CLBs and a maximum clock frequency of 15 MHz. In order to compare the coprocessor to a SW reference system we consider a worst- and best-case execution path of the CCF. In best-case the runtime improvement is given by a factor of 6, in worst-case the gain is 20.7. It should be mentioned that these experimental results are first results. Further improvements are expected due to routing optimizations and overlapping of computation and communication.

4. Overall Architecture, Future Work

A prototype of the overall multiprocessor architecture, consisting of digital signal processors TMS320C40, is shown in Figure 2. The digital signal processor TMS320C40 was chosen because of its high I/O performance and its 6 independent communication channels [12]. Some processing elements of the multiprocessor architecture are equipped with a CCF coprocessor. The distributed real-time operating system Virtuoso [13] supports a portable and flexible SW implementation.

Further work on this project will focus on

- implementation of the parallelized kernel functions on the multi-DSP system
- design and routing optimization of the CCF coprocessor
- integration of the coprocessors into the multi-DSP system

REFERENCES

1. Ernest Davis. An engaging exploration of QSIM and its extensions. *IEEE Expert*, 9(6):70–71, December 1994. book review.
2. Daniel Dvorak and Benjamin Kuipers. Process Monitoring and Diagnosis: A Model-Based Approach. *IEEE Expert*, pages 67–74, June 1991.
3. Gerald Friedl. Entwurf und FPGA-Implementierung eines Coprozessors für qualitative Simulation. Master's thesis, Institute for Technical Informatics, Graz University of Technology, 1995.
4. Herbert Kay. A qualitative model of the space shuttle reaction control system. Technical Report AI92-188, Artificial Intelligence Laboratory, University of Texas, September 1992.
5. Benjamin Kuipers. *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*. Artificial Intelligence. MIT Press, 1994.
6. Franz Lackinger and Wolfgang Nejdl. Diamon: A Model-Based Troubleshooter Based on Qualitative Reasoning. *IEEE Expert*, pages 33–40, February 1993.
7. Q.P. Luo, P.G. Hendry, and J.T. Buchanan. Strategies for Distributed Constraint Satisfaction Problems. In *Proceedings 13th International DAI Workshop*, Seattle, WA, 1994. DAI.
8. Alan K. Mackworth. Constraint Satisfaction. In Stuart C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, volume 1, pages 285–293. John Wiley & Sons, Inc., 1992.
9. Marco Platzner, Bernhard Rinner, and Reinhold Weiss. A Distributed Computer Architecture for Qualitative Simulation Based on a Multi-DSP and FPGAs. In *3rd Euromicro Workshop on Parallel and Distributed Processing*, pages 311–318, San Remo, January 1995. IEEE Computer Society Press.
10. Johannes Riedl. Parallele Algorithmen und Laufzeitmessungen für Constraint Satisfaction im qualitativen Simulator QSim. Master's thesis, Institute for Technical Informatics, Graz University of Technology, 1995.
11. Bernhard Rinner. Konzepte zur Parallelisierung des qualitativen Simulators QSIM. Master's thesis, Institute for Technical Informatics, Graz University of Technology, October 1993.
12. Christian Steger, Marco Platzner, and Reinhold Weiss. Performance Measurements on a Multi–DSP Architecture with TMS320C40. In *International Conference on Signal Processing Applications & Technology*, Santa Clara, California, USA, September 1993.
13. Eric Verhulst. Virtuoso: A virtual single processor programming system for distributed real-time applications. *Microprocessing and Microprogramming*, 40:103–115, 1994.