CYCLE-BASED SIMULATION ON LOOSELY-COUPLED SYSTEMS

Denis Döhler, *Associate Member, IEEE*
IBM Germany, S/390 Development
P.O. Box 1380, 71003 Böblingen, Germany
doehler@de.ibm.com

Klaus Hering, *Member, IEEE* Wilhelm G. Spruth, *Senior Member, IEEE*
Department of Computer Science, University of Leipzig
Augustusplatz 10–11, 04109 Leipzig, Germany
{hering, spruth}@informatik.uni-leipzig.de

Abstract— ***Logic simulation is a crucial verification task in processor design. Aiming at significant acceleration of system simulation we have parallelized IBM's cycle-based simulator TEXSIM. Resulting parallelTEXSIM has already been employed successfully in simulating S/390 architectures on IBM SP systems. Here we present parallelTEXSIM together with its model partitioning environment.***

I. Introduction

Verification of processor designs is a real challenge due to rapidly growing design complexity. Ensuring correctness of logic designs containing millions of gates requires a large amount of simulation. For system simulation at register transfer level (RTL) and gate level (GL) it has proven to be a good practice to separate timing analysis from functional verification (logic simulation). The application of static timing verifiers and cycle-based simulators as tools targeted to the corresponding task is advantage [9]. Considering functional verification, cycle-based simulators are significantly faster than event-driven ones which offer rich functionality at the cost of performance and memory utilization [8]. Hardware accelerators as alternatives to software simulators show high performance but they are very expensive and difficult to adapt to changing simulation techniques.

Cycle-based simulation (CBS) can be abstractly represented by the ALTER-CLOCK-RETRIEVE pattern (Fig. 1), where CLOCK means simulating a number of cycles, ALTER stands for setting values of model components from outside and RETRIEVE is related to the check of such values. Formal description approaches for CBS can be found in [2] and [3].

With the availability of commercial programs such as Cyclone (*Synopsis*) or SpeedSim/3 (*SpeedSim*) in recent years, CBS has become more and more popular in industry. *IBM* has been doing CBS for processor verification at least since the early 80's. Work presented in this paper is related

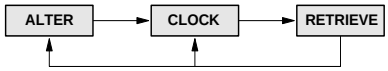

Fig. 1 – ALTER–CLOCK–RETRIEVE pattern

to the *IBM* internal cycle-based Texas Simulator (TEXSIM) developed by D.S.Zike. Since the production release of its first version in 1990 TEXSIM has become the most important logic simulator in the various *IBM* processor developments. It defines a client-server architecture offering a variety of interfaces for simulation control by different kinds of user programs. Simulation models are created by a simulator specific model compiler out of a structural circuit description which is generated by a HDL compiler in the context of the *IBM* Design Automation Database (DA_DB). Mixed level models simultaneously containing RTL and GL constructs are allowed. Originally, TEXSIM has been used for regression runs following a random simulation strategy based on a large number of mainly automatically generated small *test cases* (machine code or microcode sequences [1]) which

are (automatically again) distributed over a network of workstations. For regression runs related to complete processor models at system level [7] the evaluation of complex test cases (representing parts of boot processes, for instance) is very desirable. To cope with the tremendous amount of time required by such simulations we have parallelized TEXSIM based on the assumption that corresponding simulation processes should spend very little time in the ALTER and RETRIEVE steps relative to the CLOCK step.

Our parallelization approach makes use of model inherent parallelism (*replicated worker principle*). A parallelTEXSIM cycle simulation appears as a co-operation of several simulator instances evaluating parts of the original model, which are defined statically, on a loosely-coupled processor system. Choosing cone-based model partitioning gives the possibility of leaving the kernel of the optimized TEXSIM simulation engine unchanged. The performance of corresponding parallel simulations strongly depends on preceding model partitioning. Because of the relevance of one partitioning for several extremely time consuming simulations over one and the same model we allow complex partitioning algorithms regarding both component communication and workload aspects.

In Section II, our static model partitioning strategy for parallel cycle-based simulation is outlined. Then, parallelTEXSIM is introduced in Section III focussing on its general structure and performance relevant issues. Experimental results related to parallel simulations of a real processor model belonging to the *IBM* S/390 architecture are presented in Section IV.

II. Model Partitioning

A. Overview

For parallelTEXSIM simulations a set of TEXSIM models representing parts of an original design has to be provided together with a cross-reference list and one signal-cut list per model. The cross-reference list contains all elemental design components which are accessible from outside during simulation and information about their distribution to models. Signal-cut lists comprise model-related input and output points indicating communication relations to other models. In Fig. 2 the process of model-related input generation for parallelTEXSIM is depicted with *protos* embodying DA_DB objects for the structural representation of RTL / GL designs.

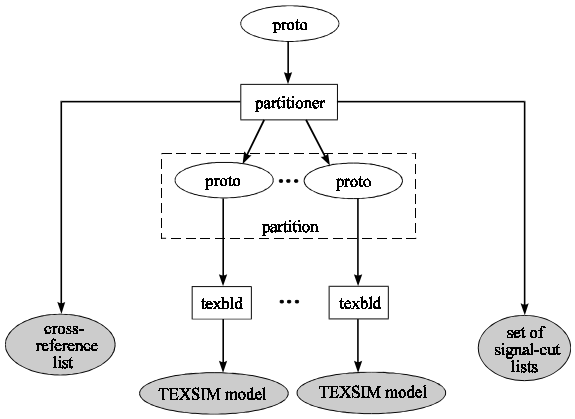

Fig. 2 – Model-related parallelTEXSIM input

Partitioning starts from a proto generated by a HDL compilation. The number of resulting protos can be both given in advance or fixed by the partitioning process itself. For every proto delivered a model building process realized by texbld (the same as for sequential TEXSIM simulation) follows. The parallel simulator automatically adapts to the number of models generated.

B. Partitioning Strategy

As basis for development and analysis of proto partitioning algorithms we have introduced a formal *Structural Hardware Model (SHM)* [5] embodying a directed graph with vertices to be interpreted as wires, latches, elements of combinatorial logic or input/output elements, respectively. For each proto describing a synchronous design a corresponding *SHM* can be constructed easily. Then, the problem of proto partitioning can be considered as a problem of *SHM* partitioning. For a given *SHM* M, the *fan-in* cones with heads representing latches or outputs form the set $Co(M)$ of basic building blocks for partitions of M. The latter ones appear as partitions of $Co(M)$ in a mathematical sense. In corresponding parallel simulations, inter-processor communication between simulator instances is restricted to cycle boundaries and can be realized by collective communication operations. Obviously, cones may overlap. Assigning overlapping cones to different partition components results in replication of simulation work.

The *SHM* partitioning problem can be stated as NP-hard combinational optimization problem. Using a formal model of *Parallel Cycle Simulation (PCS)* [3] we have developed a parameterized cost function [4] estimating upper run-time bounds for simulations corresponding to a given *SHM* partition. This partition valuation function takes into consideration both workload aspects and interprocessor communication overhead.

Due to the complexity of models representing complete processor structures we have introduced a hierarchical partitioning strategy [5] allowing combination and competition of partitioning algorithms and a special merging method of their results called *superposition*. Within a two-level partitioning scheme after fast pre-partitioning (reducing problem complexity) more expensive algorithms working on the basis of hypergraph structures are applied. Specifically, the employment of *Evolutionary Algorithms* at the second level has proven successful.

C. Partitioning Tools

A realization of the two-level partitioning scheme mentioned above is given by PAMB (Partitioning And Model Build) which has been developed by R.Haupt. It comprises a C-library (containing partitioning algorithms, functions for proto handling, model build and the creation of hypergraph structures) together with a script-based application framework in the context of the DA_DB. PAMB will be integrated in our partitioning environment parallelMAP (Model Analysis and Partitioning). The latter one embodies a client-server architecture which allows the implementation of parallel partitioning algorithms on message-passing basis.

III. The Parallel Simulator

A. General Structure

We have implemented parallelTEXSIM based on the partitioning framework outlined above. A production release of the simulator is employed for regression runs in the *IBM* S/390 processor development allowing significantly larger test cases. From the user's point of view, the parallel program offers the same options and interfaces as the sequential one. Designed to run on loosely-coupled systems, it represents a master-slave structure (Fig. 3) with component communication via message-passing. A master simulator instance (mTEXSIM) derived from sequential TEXSIM provides Application Programming Interfaces (APIs) to the environment and controls a set of slave simulator instances (sTEXSIM). These instances contain the original TEXSIM simulation engine encapsulated within a communication shell. parallelTEXSIM permits parallelization of user programs for simulation control by assigning corresponding program instances to simulator instances. The software platform currently supported is *IBM* AIX/6000 with the *IBM* Parallel Environment [6]. This allows networks of *IBM* RS/6000 workstations and *IBM* RS/6000 SP machines as target hardware.

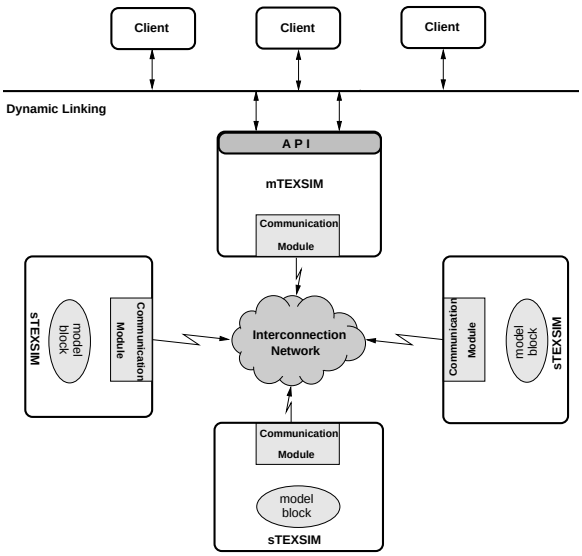

Fig. 3 – Parallel simulator structure

B. Facility Management

For referencing model elements (signals or arrays) from outside during simulation, TEXSIM provides the concept of *facilities* which are represented by bit matrices. Model partitioning for parallel simulation leads to distribution of accessible elements of the original model to different models (described by a cross-reference list mentioned in Section II) possibly related with replication and splitting. Therefore the facility notion has been extended to a facility hierarchy to ensure efficient element referencing within parallelTEXSIM. We have introduced *global*, *parallel* and *local facilities*. A local facility is a usual facility handled

by a slave simulator instance. The global facilities serve to support the usual facilities on a master simulator instance as if they were referenced on a non-parallel simulator. In fact they are vectors of parallel facilities and those are vectors containing references to local facilities.

Moreover, we have introduced *communication facilities* to achieve fast access to data structures which are involved in collective communication operations during parallel cycle simulation. These data structures are related to *cut signals* which arise from model partitioning. Cut signals are made known to parallelTEXSIM via model-related signal-cut lists (cf. Section II).

C. Component Co-operation

Within parallelTEXSIM three types of collective message-passing are distinguished based on the components involved: "master and one slave", "master and all slaves" and "all slaves". Processes using communication operations of the corresponding type are for instance facility referencing, creating an initial status protocol and simulating cycles in parallel.

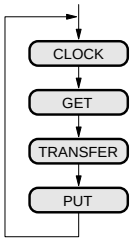

Fig. 4 – Parallel clock-cycle implementation

The main function of parallelTEXSIM is the realization of the *parallel clock-cycle* as shown in Fig. 4. During CLOCK one cycle is simulated on all slave instances over the corresponding models without interaction. In the GET step, these instances independently provide model-related global output values within communication vectors. TRANSFER embodies a personalized all-to-all communication between the slave instances. Thereby, communication vector components are transferred between instances according to their vector position. This results in new model-related communication vectors. In the final PUT step, all slave instances (again independent of one another) update facilities based on the corresponding communication vectors.

Tab. 1. System simulation for an IBM S/390 processor model with TEXSIM / parallelTEXSIM

Test Cases	Simulation Performance (cps)			
	1-way	4-way	8-way	12-way
pa013	7.60	22.46	30.43	34.67
PG060	7.75	22.42	31.76	35.78
vb010	7.70	23.71	33.47	38.93

IV. Experimental Results

In the following, we present performance values for parallelTEXSIM running on an *IBM* SP2 system (160 MHz Thin Nodes, 1 GByte RAM per node, 97 MByte/s High Performance Switch) and simulating a processor model of the *IBM* S/390 architecture containing 1 Processing Unit, 4 L2-Caches, 8 Bus Switching Networks and 8 Storage Controllers. The model consists of about 2.7 million elements at RTL / GL. We have chosen a representative set of test cases which are microcode sequences of different size (pa013 : 294 cycles, PG060 : 7464 cycles, vb010 : 31466 cycles). Parallel simulations based on model partitions with 4, 8 and 12 components are considered in comparison with corresponding sequential (1-way) TEXSIM simulations (Tab. 1). Performance is measured in cycles per second (cps).

V. Concluding Remarks

We have presented parallelTEXSIM, a parallel logic simulator running on loosely-coupled systems. It essentially accelerates time consuming system simulation processes in the *IBM* S/390 processor development. Simulation performance strongly depends on preceding model partitioning. Our partitioning environment allows consideration of workload and inter-processor communication aspects via parameterized partition valuation functions. In future work we will use our experiences with parallelTEXSIM for the parallelization of the *IBM* Multivalue Simulator MVLSIM. We will concentrate on problems related to multiply referenced sub-designs, dynamic load balancing in multiuser environments and parallelization of user programs controlling simulation.

Acknowledgment

Work presented was supported by Deutsche Forschungsgemeinschaft (DFG) under grant

Sp 487/1-1. We are grateful to K.Lamb et al. (*IBM* Laboratories Böblingen) and W.Roesner et al. (*IBM* Laboratories Austin (TX)) for valuable assistance. Special thanks to the research workers and students involved in the project at the University of Leipzig.

References

[1] A. Aharon, A. Bar-David, B. Dorfman, E. Gofman, M. Leibowitz, and V. Schwartzburd. Verification of the IBM RISC System/6000 by a dynamic biased pseudo-random test program generator. *IBM Systems Journal, 30(4)*, pages 1–81, 1991.

[2] D. Döhler. *Entwurf und Implementierung eines parallelen Logiksimulators auf Basis von TEXSIM.* Diploma Thesis, Univ. of Leipzig, Dept. of Mathematics, 1996.

[3] K. Hering. Parallel Cycle Simulation. Technical Report 13(96), Dept. of Computer Science, Univ. of Leipzig, 1996.

[4] K. Hering, R. Haupt, and U. Petri. Parameterized Partition Valuation for Parallel Logic Simulation. In *Proc. of the Conference on Parallel and Distributed Computing and Networks (PDCN'97)*, pages 144 – 150, 1997.

[5] K. Hering, R. Haupt, and T. Villmann. Hierarchical Strategy of Model Partitioning for VLSI–Design Using an Improved Mixture of Experts Approach. In *Proc. of the Conference on Parallel and Distributed Simulation (PADS'96)*, pages 106–113, 1996.

[6] M. Snir, P. Hochschild, and D. D. Frye. The Communication Software and Parallel Environment of the IBM SP2 (PE). *IBM Systems Journal, 34(2)*, pages 185–204, 1995.

[7] W. G. Spruth. *The Design of a Microprocessor.* Springer Verlag, Berlin, 1989.

[8] C. C. Tung and C. Ussery. Face off : Cycle–Based vs. Event Driven Simulation. *Computer Design's ASIC DESIGN*, pages A14–A17, 1994.

[9] K. Westgate and D. McInnis. Reducing Logic Verification Time with Cycle Simulation. *SpeedSim, Published in the Internet, http://www.speedsim.com/tech/cyc_sim.htm*, 1996.