Modeling Constructs and Kernel for Parallel Simulation of Accuracy Adaptive TLMs

Rauf Salimi Khaligh and Martin Radetzki
Embedded Systems Engineering Group (ESE) - ITI
Universität Stuttgart, Pfaffenwaldring 47, D-70569 Stuttgart, Germany
{salimi,radetzki}@informatik.uni-stuttgart.de

***Abstract*—We present a set of modeling constructs accompanied by a high performance simulation kernel for accuracy adaptive transaction level models. In contrast to traditional, fixed accuracy TLMs, accuracy of adaptive TLMs can be changed during simulation to the level which is most suitable for a given use case and scenario. Ad-hoc development of adaptive models can result in complex models, and the implementation detail of adaptivity mechanisms can obscure the actual logic of a model. To simplify and enable systematic development of adaptive models, we have identified several mechanisms which are applicable to a wide variety of models. The proposed constructs relieve the modeler from low level implementation details of those mechanisms. We have developed an efficient, light-weight simulation kernel optimized for the proposed constructs, which enables parallel simulation of large models on widely available, low-cost multi-core simulation hosts. The modeling constructs and the kernel have been evaluated using industrial benchmark applications.**

I. Introduction and Motivation

Transaction level modeling (e.g. [1]) is an increasingly popular simulation-centric modeling paradigm for development of system-level models of complex embedded systems and systems-on-chip. Previous research in system level simulation, even prior to the popularity of transaction level modeling (e.g. [2], [3]) has shown that in many use cases, the accuracy of a simulation model is only required in some intervals during simulation and in other intervals the chosen level of accuracy is not necessary. Hence, use of fixed accuracy models can result in unnecessary loss of simulation performance. One approach to address this problem is use of accuracy-adaptive models. Our experience [4], [5] has shown that development of ad-hoc adaptive models can result in complex models whose model-specific adaptivity mechanisms are not easily reusable in other contexts. In such models, implementation of adaptivity mechanisms can obscure the actual logic of a model. However, some adaptivity mechanisms are quite general and can be used in a wide variety of models. One example is a mechanism for modeling preemptable behaviors in buses and operating systems. Our aim in this work has been providing modelers with a reusable set of constructs which, while hiding the implementation details of adaptivity mechanisms, enable systematic development of adaptive TLMs.

Currently, languages such as SystemC [6] are the basis of most TLM frameworks. These languages have a general and rich set of features suitable for models ranging from functional models to synthesizable RTL models. These languages are accompanied by general, sequential discrete event simulation (SDES) kernels similar to those used in traditional RTL simulators. This generality results in unnecessary simulation performance loss in many use cases. For example, in [7] the authors address this issue for simulation of heterogenous models composed of different models of computation (MoCs). The authors propose specialized SystemC kernel extensions for efficient simulation of each MoC. For TLM, often an additional library is used (e.g. [8]) which is implemented as a layer on top of SystemC and its general SDES kernel. To the best of our knowledge, no TLM-specific simulation kernels have been developed so far. In this paper we show that a small set of TLM-specific modeling constructs together with a specialized, light-weight simulation kernel provide the necessary level of flexibility and performance for most TLM use cases.

A transaction level model is essentially a network of concurrent behaviors. Although TLMs inherently have a high degree of concurrency, not all will benefit from parallel simulation. This is mostly due to the overhead of inter-simulator communication and synchronization. Without an appropriate computation to communication and synchronization ratio, parallel simulation may even result in slow-down rather speed-up. However, many TLMs (e.g. hardware/software models of multiprocessor SoCs) have a coarse-grain concurrency which makes them suitable for parallel simulation. Currently, most often these models are simulated using sequential simulators. Considering the abundance of low cost parallel processing power on ordinary workstations, this is not justifiable. Our simulation kernel enables parallel simulation on multicore machines.

The remainder of this paper is organized as follows. In section II we give a brief overview of closely related work. In section III we present our modeling constructs and simulation kernel. Section IV summarizes the results of our evaluation and experiments. Section V concludes the paper with a discussion of the results and our directions for future research.

II. Related Work

Transaction level modeling is used for a wide variety of use cases such as system-level power and performance estimation, benchmarking, virtual platforms and validation. Although

This work has been funded by the German Research Foundation (Deutsche Forschungsgemeinschaft - DFG) under grant Ra 1889/1-1 .

978-3-9810801-6-2/DATE10 © 2010 EDAA

originally used for modeling and simulation of traditional bus-based systems, TLM is being used in a number of emerging application areas such as Networks-on-Chip (NoCs) (e.g. [9]). The popularity of TLM is in part due to the very high simulation speed and in part due to the ability to model hardware and software using a single language such as SystemC [6].

In addition to commercial TLM products and TLM standards and libraries (e.g. [8]), there exist a growing number of research projects in this field. The closest of these projects to our work are the multi-level models [10], adaptive models [4], [5] and result-oriented models (ROM) [11]. These projects present ideas which have been applied to specific models, but they do not provide a set of reusable modeling constructs. Runtime switching between different abstraction levels has been mentioned and encouraged in the recent OSCI TLM standard [8] but no details have been presented. In this work we present a small but sufficient and reusable set of modeling constructs for accuracy adaptive TLMs.

There exist a growing number of parallel simulation frameworks used for transaction level simulation. However, these are not optimized and specialized for TLM. Most of these frameworks are based on parallel and distributed discrete event simulation (PDES/DDES) principles (e.g. [12], [13]) and are *conservative* as they strictly avoid causality violations. In [14] authors present a framework which uses a modified SystemC kernel. Their framework is for general SystemC models and deals with low-level communication constructs, which means the simulators synchronize and exchange information at the end of every update phase. Another framework which is based on the same principles but does not require modification of the SystemC kernel is [15]. Another recent work is [16] which exploits the properties of temporally decoupled [8] TLMs to reduce inter-simulator synchronization overhead. Our simulation kernel enables parallel simulation. However, it is specialized and optimized for our modeling constructs and does not deal with low-level communication channels (e.g. signals) which are not used in TLM.

A much more efficient alternative to instruction-set simulators (ISS) especially when simulating large MPSoCs is the source-level annotated model of software. In these models, the software executes on the simulation host but is annotated at the source level with delays which represent the duration of execution of code fragments on given targets (e.g. [17], [18]). In contrast to existing work, we use hierarchical source-level annotation. This enables modelers to incorporate multiple levels of accuracy in a single model and dynamically change the accuracy at simulation time.

III. Modeling Constructs and Simulation Kernel

A. Definitions and Assumptions

Despite recent standardization efforts (e.g. [8]), there does not exist a unified set of definitions in the TLM community. To clarify the scope of our work, here we briefly give some basic definitions. A *model* is a set of communicating *modules* which in object oriented programming terminology are objects of a certain class, and represent logical or physical entities. Behavior of the modules is specified in a sequential, imperative manner. *Active* modules have a process of their own, which executes their specified behavior. *Passive* modules on the other hand do not have a process and their behavior is executed by the processes of active modules. Modules can only communicate via direct, or interface-based calls of their methods (i.e. no *sc_signal*-like constructs or shared memory accesses). Behavior of a module is composed of *actions* (i.e. computations), *transactions* (communication of data between modules) or *synchronizations*.

B. Modeling Constructs

The following constructs enable development of accuracy-adaptive models by providing the means for: incorporating multiple levels of timing accuracy in a single model, changing the level of accuracy at runtime, and efficient modeling of preemptable behaviors.

1) Multiple levels of accuracy: We incorporate timing information in the behavior of modules using hierarchical (i.e. nested) source-level annotations. The annotations represent timing of regions of code which are referred to as single-entry, single-exit (SESE) regions in the compiler design community. SESE regions can be individual statements within basic blocks, basic blocks or larger compound blocks.

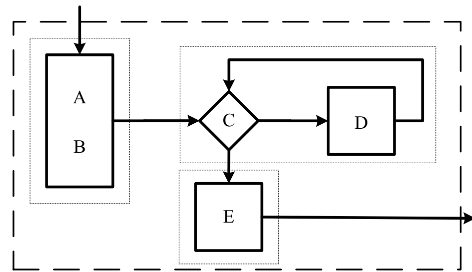

Fig. 1. A hierarchy of SESE regions.

Figure 1 shows the control flow graph of a region of code, consisting of straight line segments and a loop, which is annotated at two levels. The two levels of timing annotation correspond to two levels of abstraction and two levels of timing accuracy. At the higher abstraction level, regions A,B,C,D and E are annotated as a whole and at the lower abstraction level three regions AB, CD and E are annotated.

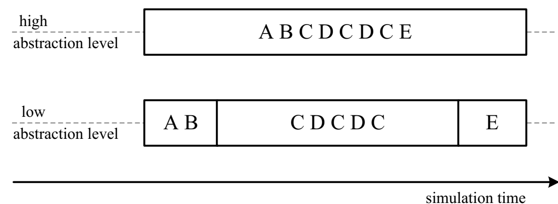

Fig. 2. Available timing information at different abstraction levels.

By annotating a region with timing, we indicate that we are interested in the total duration of the enclosed activities.

This also indicates that the individual timing of those activities is not of relevance at the given abstraction level. Let r be a region annotated to have a duration of d and assume that during simulation this region is entered at simulation time t. The simulation kernel guarantees that the code in the region is executed between t and $t+d$ but no other assumption can be made.

Timing accuracy of simulation of the control flow graph of figure 1 at two levels of abstraction is shown in figure 2. At the higher level of abstraction, individual execution times of actions A, B, C, D and E are not identifiable. More timing information is available at the lower abstraction level, which comes at the price of reduced simulation speed. The abstraction level of modules can be selected statically at the start of simulation or dynamically at simulation-time.

Listing 1 shows a code fragment annotated at two levels using timing annotation macros HTA_BEGIN and HTA_END.

```
HTA_BEGIN(T1);
for(int i=0;i<n;i++)
{
    HTA_BEGIN(T2);
    C;
    D;
    HTA_END();
};
HTA_END();
```

Listing 1. Timing annotation macros.

The simulation semantics of these macros is best explained by what occurs upon entering an annotated region and upon exit from that region, summarized in listing 2. In the following, $G = (R_M, E)$ is the hierarchy tree of annotated SESE regions R_M of module M, with $depth_r$ being the depth of a region r in that tree. Function $d_M : R_M \times S_M \rightarrow T \cup \{\Omega\}$ assigns to each region its duration $t \in T$ depending on state $s \in S_M$ of the module M. T is a totally ordered set of simulation time values. $\Omega \notin T$ is a special value representing *absence* of timing information. $level_M$ is the current chosen abstraction level of module M and $local_time_M \in T$ is the local time of module M. Operation *synchronize* synchronizes the local simulation time of a module with the global simulation time by halting the execution of the behavior of the module until the global simulation time reaches $local_time_M$. Two cases are differentiated: The first case is when an estimate for the duration of an SESE region r exists prior to entry to that region (e.g. the loop in listing 1). This will postpone the execution of the actions enclosed in r for the annotated amount of time when the module is at the corresponding abstraction level (listing 2 lines 4-5). The second case is when such an estimate does not exist and depends on the execution of the enclosed actions (e.g. a loop with a complex termination condition). In this case, the actions enclosed by r are executed and the local time proceeds according to timing information of sub-annotations (line 9). The synchronization with the global simulation time in this case occurs upon exit from region r (line 14). It should be noted that inconsistencies between the annotated duration of a region and total duration of its subregions are allowed (e.g. when no accurate estimates exist). Whether the resulting inaccuracy is tolerable depends on the model and the use case.

```
1  on entry to region r:
2  if (level_M = depth_r){
3      if (d_M(S_M, r) ≠ Ω){
4          local_time_M ← local_time_M + d_M(S_M, r)
5          synchronize;
6      }
7  }
8  else if (level_M = depth_parent(r))
9          local_time_M ← local_time_M + d_M(S_M, r)
10
11 on exit from region r:
12 if (level_M = depth_r){
13     if (d_M(S_M, r) = Ω){
14         synchronize;
15     }
16 }
```

Listing 2. Simulation semantics of annotation macros.

2) Modeling preemptable behaviors: Preemption is a recurring pattern in system level models of hardware and software (e.g. operating systems [19] and busses [5]). Assume a behavior consisting of N atomic actions which can be preempted upon occurrence of a certain condition. The simplest approach for modeling this behavior, which is often used in cycle-based models would be checking for the occurrence of that condition after each atomic step and terminating the behavior accordingly. For TLM, more efficient but equally accurate approaches have been proposed (e.g. [5], [11], [19]). We provide constructs for modeling preemptable behaviors based on these approaches without obscuring the actual logic of the model. Listing 3 shows an example of a preemptable region delineated by HTA_BEGIN_PREEMPT and HTA_END_PREEMPT macros. This behavior consists of several sub-regions each annotated by HTA_BEGIN_P_ACTION and HTA_END_P_ACTION macros.

```
HTA_BEGIN_PREEMPT(D, condition)
    HTA_BEGIN_P_ACTION(D1);
    C;
    HTA_END_P_ACTION();
    /* more actions */
    HTA_BEGIN_P_ACTION(D2);
    D;
    HTA_END_P_ACTION();
HTA_END_PREEMPT();
```

Listing 3. Timing annotation macros - preemptable behaviors.

The simulation semantics of the macros is shown in listing 4 which shows entry to preemptable regions and atomic sub-regions. The preemptable region is annotated with an estimate for its duration, assuming that it is executed atomically. Satisfaction of a condition c denotes violation of that assumption at D_c simulation time units after entry to the region. Operation *wait(D,c)* resumes the behavior of the module after

$min\{D, D_c\}$ units of simulation time and returns the amount of time elapsed since its initiation.

```
1   on entry to preemptable region r:
2   if (level_M = depth_r) {
3       time_budget ← wait(d_M(S_M, r), c)
4       local_time_M ← local_time_M + time_budget
5   }
6
7   on entry to a sub-region r_s:
8   if (level_M = depth_parent(r)) {
9       if (time_budget − d_M(S_M, r_s) ≥ 0)
10          time_budget ← time_budget − d_M(S_M, r_s)
11      else
12          skip;
13  }
```

Listing 4. Simulation semantics - preemptable region annotation.

Operation *skip* causes the enclosed region to be left without executing its enclosed actions. In summary, upon entry to a preemptable region, its execution is postponed for its estimated non-preempted duration (listing 4 line 3). If the non-preemption assumption is not violated, all enclosed subregions are executed. If the non-preemption assumption is violated after D_c simulation time units, only those subregions whose total duration is less than D_c are executed and others will be *skipped* (lines 9 to 12). If necessary, the constructs can be used in a loop, to model resumption of the preempted behavior.

C. Simulation Kernel

We have implemented the aforementioned modeling constructs in a C++ modeling library, which borrows some of its features from the open source SystemC library [6] (exports, interface method calls and module hierarchy) but is otherwise developed from scratch. A specialized, discrete event simulation kernel has been developed which is directly based on and is optimized for the modeling constructs. Special care has been taken to keep this simulation kernel light and efficient. For example, instead of a full-fledged, user-level threading package we have opted for a more efficient coroutine library [20] which has been used by other researchers in large projects (e.g. [21]). As another example, for performance reasons shown in section IV we have decided to use a raw 64 bit integer time base instead of a more flexible but less efficient *sc_time*-like object.

A single instance of the simulation kernel can be used to simulate all modules in a model sequentially on a single processor core. For parallel simulation of a model on n cores of a multicore host, n instances of the kernel $S_1 \ldots S_n$ are used. The set of modules $\mathcal{M}$ in the model is partitioned into disjoint sets $M_1 \ldots M_n$ and each set M_i is assigned to one instance of the kernel. At global simulation time T, each kernel instance S_i simulates its behaviors up to T and reports the time of the next behavior execution T_i to a simulator instance S_c which is statically chosen to act as the synchronizer. S_c calculates the next global simulation time $T' = min\{T_1, \ldots, T_n\}$ and reports this value to all simulators as the next global simulation time, to which the simulators proceed. This continues until a global simulation termination condition is reached. In PDES/DDES terminology, this scheme is sometimes referred to as *synchronous* parallel discrete event simulation. The simulation kernel automatically detects whether it is being used in parallel simulation and enables the global simulation time synchronization algorithm accordingly. Communication between modules assigned to different simulators is handled automatically by the library using MPI [22] and requires no user code. The partitioning and load balancing between simulators is currently static and is done using a module constructor parameter.

IV. Experimental Results

All following experiments have been performed on a 2.5 GHz, Intel Core 2 Quad based machine running 32-bit Linux.

A. Simulation Kernel Performance

Figure 3 compares our kernel (when used for sequential simulation) with the standard OSCI SystemC kernel. In this experiment models with increasing number of active modules were simulated. The modules performed repetitive synchronizations with each other and the simulation time, with no computation between synchronizations. The results show the efficiency of our lightweight coroutine-based concurrency management and the effect of choosing a raw 64-bit integer time base. It can be seen that our kernel is almost twice as fast compared to the standard SystemC kernel.

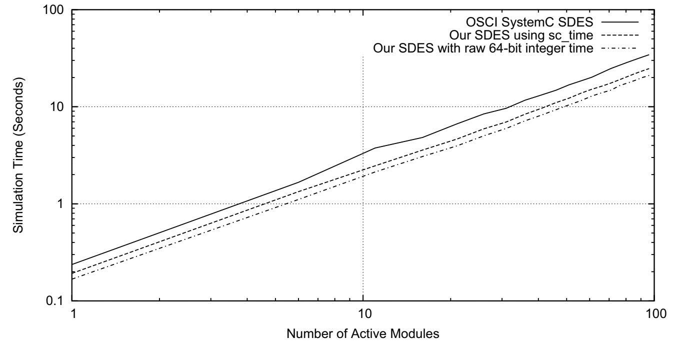

Fig. 3. SDES kernel performance.

To measure the efficiency of parallel simulation and the overhead of global simulation time synchronization, we simulated models with theoretical maximum achievable speedups of 2, 3 and 4. The actual speed-up was measured for different computation to communication/synchronization ratios. Results are summarized in figure 4, where each unit of computation corresponds to computation requiring roughly 5 μs of CPU time on our simulation host. The difference in the break-even points can be attributed to the fact that on the simulation host cores share the L2 caches pair-wise.

B. Efficiency of Adaptivity Constructs

To measure the efficiency of hierarchical timing annotations and dynamic accuracy control, we modelled a vector cross product operator which could be used at two different abstraction levels. At the higher level, only the total duration

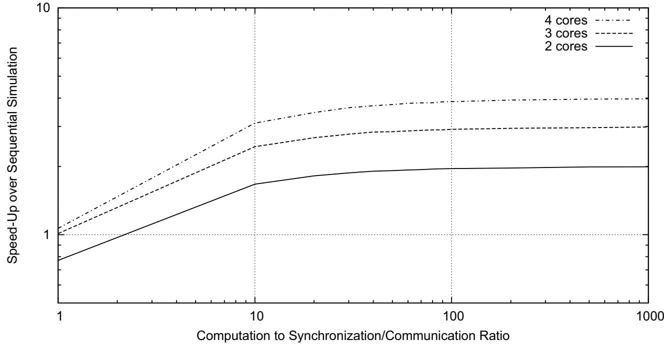

Fig. 4. Parallel simulation performance.

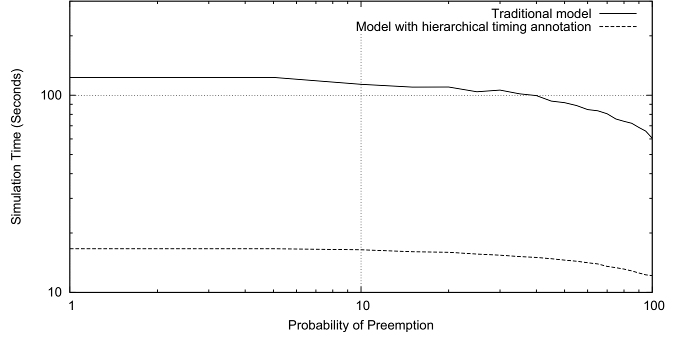

Fig. 6. Preemptable behaviors.

of a complete cross product was known to the outside. At the lowest level the duration of computation of each element of the output vector was visible. To do this we annotated the behavior of the module modeling this operator at two levels. The model was used in a scenario requiring 10^7 cross-product operations. Simulation at the lower level was approximately 30 times slower than simulation on the higher level. Now assuming the lower level of detail was only required for some operations, we varied the percentage of time in which the model was at the lower abstraction level. The results are shown in Figure 5, which clearly shows the benefit of using dynamic accuracy control even for a small example.

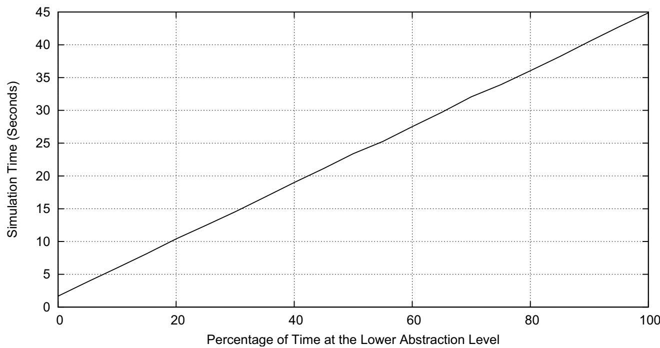

Fig. 5. Dynamic control of abstraction level.

For the next experiment, we developed two models representing the general pattern of preemptable behaviors. One was a simple, "traditional" model, which checked for the preemption condition after each step, and the second one used the annotation macros presented in section III. We varied the probability of preemption per one execution of the preemptable behavior and measured the simulation time of both models. Figure 6 shows the result where the behavior consists of 64 steps (e.g. a 64-beat burst). The gain in simulation performance without having to leave out preemption from the model can clearly be seen here. It should be kept in mind that this performance gain is application dependent (e.g. depends on the number of atomic steps of the preemptable behavior).

C. Realistic Benchmarks

For evaluation under more realistic conditions, we chose two benchmarks from the commercial EEMBC Multibench Suite [23]. The first benchmark was RGB to CMYK color space conversion, running on an eight-core MPSoC. We annotated the software with timing at three levels of accuracy: processing of large sub-picture blocks, 64-by-64 pixel blocks and individual pixels (abstraction levels 1, 2 and 3). The model was simulated using one to four cores. The results are summarized in Table I. The benchmark can be configured to use different images and repetitions resulting in different simulation times. For example, simulating conversion of a sequence of 20 800x600 image at abstraction level 1 required approximately one second. For easy comparison, in the following tables all simulation times are relative to the sequential simulation time at the highest abstraction level. In this experiment the abstraction level of the models was fixed during simulation. It can be seen that the sequential simulation of the model at level 3 is almost two order of magnitude slower than simulation at level 1. Assuming that we are only interested in exact timing for some of the pixels (e.g. image boundaries), we can switch the model to a higher level of abstraction while other pixels are being processed and hence increase the simulation performance. When simulated at levels 1 and 2, parallel simulation results in noticeable speed up. When simulated at level 3, parallel simulation results in a slow-down. This is due to the very fine-granular timing annotations which result in frequent synchronization between simulator instances.

Abstraction Level	Sequential	2 SDES	3 SDES	4 SDES
1	1	0.53	0.4	0.26
2	≈ 1	0.55	0.4	0.33
3	93	115	108	136

TABLE I
SIMULATION TIMES FOR PARALLEL RGB-CMYK BENCHMARKS

The second benchmark was the MD5 digest algorithm running on a quad core MPSoC. The central part of this particular implementation is a function to calculate the digest for a 64 Byte block of data. We annotated the software at two levels: processing of large blocks of data, and smaller 64

Abstraction Level	Sequential	2 SDES	3 SDES	4 SDES
1	1	0.51	0.51	0.26
2	3.4	4.7	5.4	5.7
Dynamic	1.1	0.74	0.82	0.42

TABLE II
SIMULATION TIMES FOR PARALLEL MD5 DIGEST BENCHMARKS

byte blocks (abstraction levels 1 and 2). Results can be seen in Table II. We simulated the model with fixed abstraction levels, and a additionally with dynamic abstraction level to collect statistical information for 1% of the processed 64 Byte blocks (approximately $3 * 10^5$ blocks). While delivering the required level of accuracy for this use case, the simulation speed of the dynamic accuracy model was only slightly lower than simulation at level 1. While simulation with fixed, high-level of detail does not benefit from parallelization, simulation with dynamic accuracy can be made faster by parallel simulation. The minimal difference between speedups with two and three simulators is due to static partitioning and load balancing.

V. CONCLUSIONS AND FUTURE WORK

We have presented a small set of modeling constructs for systematic development of adaptive transaction level models, without having to deal with low-level implementation details of adaptivity mechanisms. These constructs are applicable in a large number of modeling use cases. Our experiments have shown the efficiency of the adaptive models developed with these constructs over their fixed-accuracy counterparts. The constructs are directly supported by the underlying specialized, light-weight simulation kernel. The simulation kernel can take advantage of the increasing availability of low cost parallel processing power in development workstations. Multiple instances of the kernel can be used to easily create a synchronous PDES simulator. The only additional user code required in this case would be assignment of modules to kernel instances (partitioning) by setting a single module attribute upon construction. Depending on the properties of the model and its partitioning, parallel simulation can result in significant speedup on multicore simulation hosts. Extending the parallel simulation capabilities for clusters and dynamic load balancing are our planned future work.

REFERENCES

[1] F. Ghenassia, *Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems*. Springer-Verlag Inc., 2006.

[2] M. Rosenblum, S. Herrod, E. Witchel, and A. Gupta, "Complete Computer System Simulation: The SimOS Approach," *IEEE Parallel and Distributed Technology: Systems and Applications*, no. 4, December 1995.

[3] K. Hines and G. Borriello, "Dynamic Communication Models in Embedded System Co-Simulation," in *Proceedings of the Design Automation Conference (DAC '97)*, June 1997.

[4] M. Radetzki and R. Salimi Khaligh, "Accuracy-Adaptive Simulation of Transaction Level Models," in *Proceedings of Design, Automation and Test in Europe 2008 (DATE 08)*, March 2008.

[5] R. Salimi Khaligh and M. Radetzki, "Adaptive Interconnect Models for Transaction-Level Simulation," *Languages for Embedded Systems and their Applications (LNEE 36)*, 2009.

[6] *Standard SystemC Language Reference Manual, Standard 1666-2005*, IEEE Computer Society, March 2006.

[7] H. D. Patel and S. K. Shukla, *SystemC Kernel Extensions For Heterogenous System Modeling: A Framework for Multi-MoC Modeling & Simulation*. Kluwer Academic Publishers, 2004.

[8] *Transaction Level Modeling Standard 2 (OSCI TLM 2)*, Open SystemC Initiative (OSCI) TLM Working Group (www.systemc.org), June 2008.

[9] A. Kohler and M. Radetzki, "A SystemC TLM2 Model of Communication in Wormhole Switched Networks-on-Chip," in *Proceedings of the Forum on Specification and Design Languages (FDL '09)*, September 2009.

[10] G. Beltrame, D. Sciuto, and C. Silvano, "Multi-Accuracy Power and Performance Transactions-Level Modeling," *IEEE Transactions on Computer-Aided Design Of Integrated Circuits and Systems (TCAD)*, October 2007.

[11] G. Schirner and R. Dömer, "Fast and Accurate Transaction Level Models Using Result Oriented Modeling," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, November 2006.

[12] R. M. Fujimoto, "Parallel and Distributed Simulation," in *Proceedings of the 31st Winter simulation conference (WSC '99)*, 1999.

[13] J. Misra, "Distributed Discrete-Event Simulation," *Computing Surveys*, March 1986.

[14] B. Chopard, P. Combes, and J. Zory, "A Conservative Approach to SystemC Parallelization," in *Proceedings of the International Conference on Computational Science (ICCS 2006)*, May 2006.

[15] K. Huang, I. Bacivarov, F. Hugelshofer, and L. Thiele, "Scalably distributed SystemC simulation for embedded applications," in *Proceedings of the International Symposium on Industrial Embedded Systems (SIES'2008)*, June 2008.

[16] R. Salimi Khaligh and M. Radetzki, "Efficient Parallel Transaction Level Simulation by Exploiting Temporal Decoupling," in *Proceedings of the International Embedded Systems Symposium (IESS '09)*, September 2009.

[17] T. Meyerowitz, M. Sauermann, D. Langen, and A. Sangiovanni-Vincentelli, "Source-Level Timing Annotation and Simulation for a Heterogeneous Multiprocessor," in *Design Automation and Test Europe*, March 2008.

[18] P. A. Hartmann, H. Kleen, P. Reinkemeier, and W. Nebel, "Efficient Modelling and Simulation of Embedded Software Multi-Tasking using SystemC and OSSS," in *Proceedings of the Forum on Specification and Design Languages (FDL '08)*, 2008, pp. 19–24.

[19] G. Schirner and R. Dömer, "Introducing Preemptive Scheduling in Abstract RTOS Models using Result Oriented Modeling," in *Proceedings of the conference on Design automation and test in Europe (DATE)*, 2008.

[20] D. Libenzi, "Portable Coroutine Libraray (PCL)," http://www.xmailserver.org/libpcl.html.

[21] C. Liao, O. Hern, B. Chapman, W. Chen, and W. Zheng, "OpenUH: an Optimizing, Portable OpenMP Compiler," in *In: 12th Workshop on Compilers for Parallel Computers*, 2006, p. 2006.

[22] MPI Forum, "MPI: A Message-Passing Interface Standard," http://www.mpi-forum.org/.

[23] The Embedded Microprocessor Benchmark Consortium (EEMBC), "Multicore Benchmark Software (Multibench) 1.0 ," http://www.eembc.org.