Co-Simulation of mixed HW/SW and Analog/RF systems at architectural level

Markus Damm
Institute of Computer Technology
Vienna University of Technology, Austria
damm@ict.tuwien.ac.at

Jan Haase
Institute of Computer Technology
Vienna University of Technology, Austria
haase@ict.tuwien.ac.at

Christoph Grimm
Institute of Computer Technology
Vienna University of Technology, Austria
grimm@ict.tuwien.ac.at

ABSTRACT

Analog systems are more and more functionally interwoven with digital hardware/software systems. SystemC offers the potential for a unified modeling approach for such systems: SystemC AMS extensions and SystemC TLM extensions are covering the domains of analog/signal processing systems, resp. hardware/software systems. Both extensions use different Models of Computation to gain simulation performance, most notably by abstracting timing information.

In this paper we present a method to couple SystemC AMS extensions with TLM 2.0 extensions while maintaining the simulation speedup for an overall system simulation.

1. INTRODUCTION

There is a growing trend for tighter interaction between embedded hardware/software (HW/SW) systems and their analog physical environment. This leads to systems in which digital HW/SW is functionally interwoven with analog and mixed-signal blocks such as RF interfaces, power electronics, sensors and actuators, as shown for example by the communication system in Figure 1. We call such systems ***E**mbedded **A**nalog/**M**ixed-**S**ignal* (E-AMS) systems. Examples of E-AMS systems are cognitive radios, sensor networks or systems for image sensing. A challenge for the development of E-AMS systems is to understand the interaction between HW/SW and the analog and mixed-signal subsystems at architecture level. This requires some means of modeling and simulating the interacting analog/mixed-signal systems and HW/SW systems at functional and architecture levels.

SystemC [2] supports the refinement of HW/SW systems down to RTL by providing a discrete-event (DE) simulation framework. A methodology for generalized modeling of communication and synchronization that builds on this framework is available: Transaction Level Modeling (TLM) [13] allows designers to perform abstract modeling, simulation and design of bus-oriented HW/SW system architectures. However, the SystemC simulation kernel has not been designed for the modeling and simulation of analog, continuous-time systems and lacks the support of a refinement methodology to describe analog behavior from a functional level down to implementation level.

System-level tools such as Simulink [1] and Ptolemy II [8] are often used for functional modeling and simulation. They may also capture continuous-time behavior, but do not target the design of E-AMS systems at an architecture-level. Hardware description languages (HDLs) such as VHDL-AMS [4] and Verilog-AMS [5] target the design of mixed-signal subsystems close to implementation level, but these languages have limited capabilities to provide efficient HW/SW co-design at high level of abstraction. Existing co-simulation solutions mixing SystemC and Verilog/VHDL-AMS do not provide high enough simulation performances and lack offering a seamless design refinement flow for modeling mixed discrete-event/continuous-time systems and HW/SW systems at architectural level.

In response to these needs from telecommunication, automotive, and semiconductor industries, SystemC AMS extensions where introduced for SystemC [9], providing a uniform and standardized methodology for modeling E-AMS systems. The SystemC AMS extensions are intended to extend the HW/SW oriented SystemC class library to provide a framework for functional modeling, architecture exploration, integration validation, and virtual prototyping of E-AMS systems [3, 15]. The core Model of Computation (MoC) of SystemC AMS is Timed Synchronous Dataflow (TDF), which offers high simulation performance since it is possible to compute a static schedule for the execution of processes during simulation. This feature can be exploited even more when using high data rates.

The Open SystemC Initiative (OSCI) recently released the TLM 2.0 standard [13]. TLM 2.0 has a dedicated focus on boosting simulation performance. It introduces different *coding styles* (which can roughly be regarded as MoCs) regarding the modelling of time, providing different trade-offs between simulation accuracy and speed. In the so called *loosely timed coding style*, processes are allowed to run ahead of the global SystemC simulation time (*temporal decoupling*). This coding style is the focus of our interest, since it offers the most simulation performance gain. In this paper,

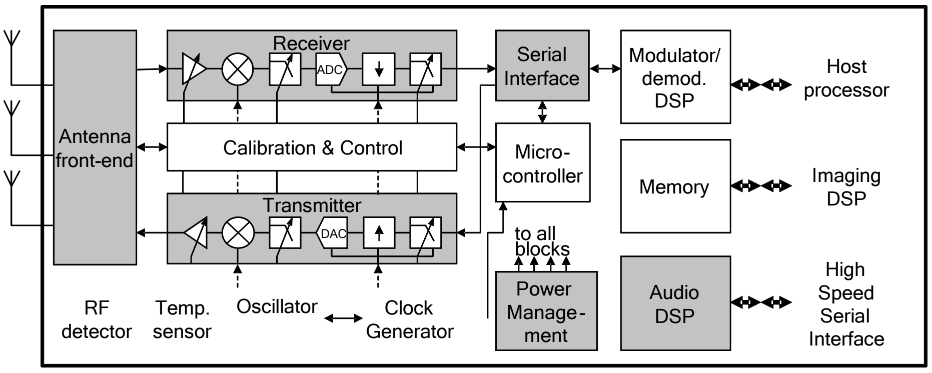

Figure 1: Example of an embedded analog/mixed-signal architecture: Communication System [9]

we present an approach for a MoC converter that can be used to connect loosely timed TLM models with TDF models. We do this in a way that the simulation performance enhancement capabilities of the two methodolgies are preserved effectively. This work falls roughly in the category of heterogeneous system modelling with different MoCs. There has been a lot of work in this area, especially with the focus of SystemC (e.g. [14],[10], and also by some of the authors in [7]), or with a more general approach (e.g. [11]).

The rest of this paper is organized as follows: We start with a brief introduction of the TLM 2.0 loosely timed coding style and the SystemC AMS TDF MoC. This is followed by a discussion on how these two models can be connected by exploiting certain similarities to maintain their simulation efficiency. We then present the general structure of converters from TLM to TDF and vice versa in Sections 5 and 6, and present an example where they are used in Section 7. We conclude in Section 8.

2. OSCI TLM 2.0 LOOSELY TIMED CODING STYLE

In the *loosely timed coding style* (LT-TLM), introduced in the draft 2 of the OSCI TLM 2.0 standard [13], a non-blocking method interface is used where initiator processes generate transactions (the *payload*) and send them with a method call to target processes. The speciality of LT-TLM is the possibility to annotate a transaction with a time delay d to mark it (in the case of $d > 0$) as a future transaction. That is, LT-TLM allows initiator processes to "warp ahead" of simulation time. The target processes must deal with these future transactions accordingly. They have to store them in a way such they can access and process delayed transactions at the right time, e.g. by using the *payload event queue* (PEQ) [6] of TLM 2.0.

The idea of this approach is that context switches on the simulation host system (generally triggered by `wait()` statements) are reduced and thus simulation performance is gained. Instead of letting initiator and target repeatedly produce and process a transaction respectively, an initiator can produce a chunk of transactions in advance, which is then processed by a target (ideally) at once. However, this may lead to *time-twisted* transactions, i.e. the order of arrival of two transactions at one target is different from their temporal order, causing potential causal errors.

LT-TLM basically allows processes to run according to their own, local simulation time. To organize this, TLM 2.0 provides the facility of the *quantum keeper*. Processes can use the quantum keeper to keep track of their local time warp, and yield to the SystemC simulation kernel after a certain time quantum is used. Typically, a smaller time quantum will reduce the chance of causal errors while a greater quantum increases the simulation performance.

3. SYSTEMC AMS TDF

The main MoC provided by SystemC AMS is the *Timed Synchronous Dataflow* (TDF) MoC. It is a timed version of the (originally untimed) *Synchronous Dataflow* (SDF) MoC [12], where processes communicate via bounded fifos. The number of data tokens produced and/or consumed by a process (the *data rate*) is constant for all process firings.

The advantage of SDF is that the schedule of process execution can be precomputed, such that the simulation kernel is only engaged with executing this static schedule, which makes the simulation of SDF models very fast. The speciality of the SystemC AMS TDF MoC is that a certain time span (the *sampling period*) is associated with token consumption and production. It is an attribute of the *TDF port* classes, which are analogous to the SystemC port classes, respectively. Via TDF ports, *TDF modules* are connected to each other by *TDF signals*. A TDF module encapsulates the actual process as a method with the standard name `sig_proc()`. In the current SystemC AMS prototype, the sampling period has to be set only at *one* TDF port of *one* TDF module of a connected set of TDF modules. The sampling periods of all other TDF ports within the cluster are then a result of this one given sampling period.

For example, if the sampling period of an input port p_1 of a TDF Module M_1 is set to 2 ms, with a data rate of 3, the consumption of one token takes 2 ms. such that the consumption of all 3 tokens (when M_1's `sig_proc()` fires) takes 6 ms. If M_1 contains also an output port p_2 with data rate 2, the sampling period of p_2 is 6ms divided by the data rate of 2, resulting in 3 ms. An input port p_3 of a TDF module M_2 which is connected to p_2 via a TDF signal now also has a sampling period of 3 ms.

4. CONNECTING LT-TLM AND TDF

At a first glance, bringing these two approaches together seems to be futile. On the one hand, there is the LT-TLM approach with its local time warps decoupled from the global simulation time. On the other hand, we have the strictly timed TDF approach which runs at an unstoppable pace. But by taking a closer look at how the TDF simulation works when using a static schedule (as it is the case with the current SystemC AMS prototype), we find surprising similarities.

The SystemC AMS simulation kernel is using its own simulation time, whose current value is returned by the TDF module method `sca_get_time()` (from now on denoted by t_{TDF}). The current SystemC simulation time (from now on denoted by t_{DE}) is returned by the DE module method `sc_time_stamp()`. By DE, we denote the *discrete event* MoC implemented by the SystemC simulation kernel, while a *DE module* denotes the `sc_module`-instances.

If a pure SystemC AMS TDF model is used in a simulation, the SystemC AMS simulation kernel is blocking the DE kernel all the time, so the DE simulation time doesn't proceed at all. However, there might be a need to connect and synchronize TDF modules to DE modules. SystemC AMS provides *converter ports* for this cause, namely `sca_scsdf_in` and `sca_scsdf_out`. They can be used within TDF modules to connect to instances of the SystemC DE `sc_signal`.

If there is an access to such a port within the `sig_proc()` method of a TDF module, the SystemC AMS simulation kernel interrupts the execution of the static schedule of TDF modules and yields to the SystemC DE simulation kernel, such that the DE part of the model can now execute, effectively proceeding t_{DE} until it is equal to t_{TDF}. Now, the DE modules reading from signals driven by TDF modules can read their new values at the right time, and TDF modules reading from signals driven by DE modules can read their correct current values.

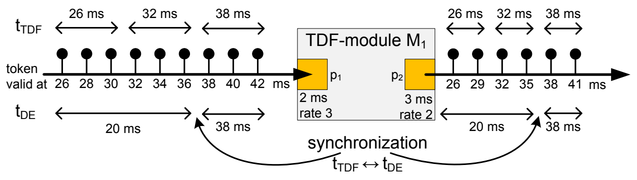

Figure 2: $t_{DE} \leftrightarrow t_{TDF}$ synchronization

Figure 2 shows an example using the TDF module M_1 from Section 3. The data tokens consumed are on the left axis, and those produced are on the right axis. The numbers beneath the tokens denote the time (in ms) at which the respective token is valid. The time spans above the tokens indicate the values of t_{TDF} when the respective token are consumed resp. produced. The time spans below indicate the according values for t_{DE}. At the beginning of the example, $t_{TDF} > t_{DE}$ already holds, until $t_{TDF} = 38ms$. Then the SystemC AMS simulation kernel initiates synchronization, for example because M_1 contains a converter port which it accesses at that time, or because another TDF module within the same TDF cluster accesses its converter port.

The important conclusion is that TDF modules also use a certain time warp. In general, TDF modules run ahead of SystemC simulation time, since $t_{TDF} \geq t_{DE}$ always holds. Further time warp effects result from using multi-rate data flow. When a TDF module has an input port with data rate > 1 it also receives "future values" with respect to t_{DE}, and even t_{TDF}. When a TDF module has an output port with data rate > 1, it also sends values "to the future" with respect to t_{DE} and t_{TDF}. The difference to TLM is that the effective local time warps are a consequence of the static schedule, with the respective local time offsets only varying because of interrupts of the static schedule execution due to synchronization needs.

In the following two Sections we describe how the streaming data of TDF signals can be converted to TLM 2.0 transactions and vice versa, effectively proposing general TLM2↔TDF converters. We do this in a way such that the temporal decoupling approach of LT-TLM is exploited to maintain a high simulation performance. The transaction class used will always be the TLM 2.0 generic payload.

5. CONVERTING FROM LT-TLM TO TDF

The principal idea of a TLM→TDF converter is to take write-transactions (i.e. with a command set to `TLM_WRITE_COMMAND`) and stream their data to a TDF signal. However, we are confronted with several difficulties.

First of all, we can't expect the data from the TLM side to arrive at certain rates, even if we take the time warp into account. We might get huge amounts of data within short time spans, and almost no data for long time spans. Nevertheless, we have to feed an unstoppable data token consumer, namely the TDF reading side.

The obvious solution for this problem is to use an internal fifo within the converter to buffer the incoming data. If a transaction causes a buffer overflow (when the internal buffer is chosen to be of a fixed size), it is returned with an error response. If, on the other hand, the buffer is empty, default values are written to the TDF side to fill the gaps (e.g. zeros). We could also choose to throw a simulation error in that case. Another advantage of using an internal buffer is that the size of the data section of the write-transactions can be set independent from the data rate of the TDF output of the converter. Especially, the transaction data size can vary over the course of the simulation.

However, as already mentioned in 2, transactions may arrive with twisted time warps, e.g. since they came from different initiators. Therefore, we can't write the transaction data to the buffer right away. Instead, we write the transaction to a PEQ, which stores them in the order of their procession time, such that the twists are resolved. When the local time of the converter proceeded far enough for a transaction in the PEQ to be processed, its data gets written to the buffer. To this end, the PEQ is checked for transactions with a time stamp smaller or equal to the current local time of the converter at every `sig_proc()` execution.

Another issue is synchronization. Since the converter is a TDF module, it might run way ahead of t_{DE}, and the initiators feeding the converter might not have had the chance

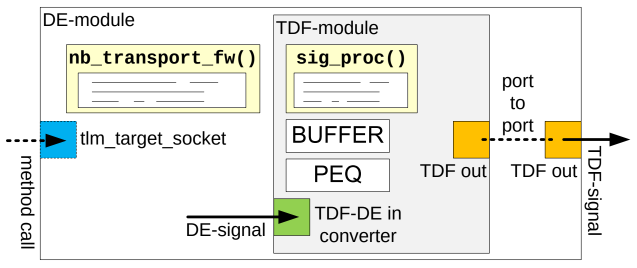

Figure 3: The TLM→TDF converter

to produce transactions sufficiently. Therefore, if there are no transactions available in the PEQ when the `sig_proc()` is processed, and the buffer also holds not enough data for the output, the SystemC simulation kernel must get the chance to catch up. This is done by connecting the converter to an `sc_signal` using a SystemC AMS converter port. If a reading access is now performed on this converter port, the SystemC AMS simulation kernel interrupts the procession of the static schedule, and the SystemC simulation kernel regains control and can proceed the SystemC simulation (including the TLM initiator modules) until $t_{DE} = t_{TDF}$. Figure 3 shows an overview of the architecture of the proposed converter. The core is a TDF-module, which contains the PEQ, the buffer, and the port to the TDF side. It is encapsulated within a DE-module, which implements the TLM 2.0 transport interface. For synchronization, the TDF module is connected to a DE-signal via a SystemC AMS converter port. Note that the DE-signal needs no driver; simply accessing it from within the TDF module is sufficient to trigger synchronization.

6. CONVERTING FROM TDF TO LT-TLM

When converting from TDF to TLM, we want to bundle the streaming TDF data into the data section of a transaction and send it to the TLM side. This would be an easy and straightforward task if we would consider the converter (i.e. the TDF side) to act as an TLM initiator. In this case, the transaction's command would be set to `TLM_WRITE_COMMAND`, and the delay of the transaction could be set to the difference of t_{DE} and the valid time stamp of the last token sent via the transaction (i.e. t_{TDF} + token number·sampling period).

However, the TDF models we focus on here just provide streaming input to the TLM side, and the idea of such models acting as a TLM initiator is as realistic as an A/D converter acting as a bus master. It can make sense to view certain TDF models as TLM initiators, e.g. models of signal processing algorithms which are supposed to run on a digital signal processor in the implementation. But these TDF models might access resources via a bus in various ways, such that there is no obvious conversion semantics here. For example, it is not clear how the TDF side should provide the transactions with an address.

Therefore, we focus on an approach where the initiators are on the TLM side, sending read-transactions to the converter, which copies the data tokens it receives from the TDF side into the data section of the transaction and returns it. The advantage of this approach is that it is pretty similar to the TLM→TDF conversion direction. For example, the converter needs an internal buffer to store the incoming TDF data tokens, for similar reasons as discussed in Section 5. Here, the TLM side might *request* data of varying length at varying times, while the TDF side provides data at an unstoppable pace.

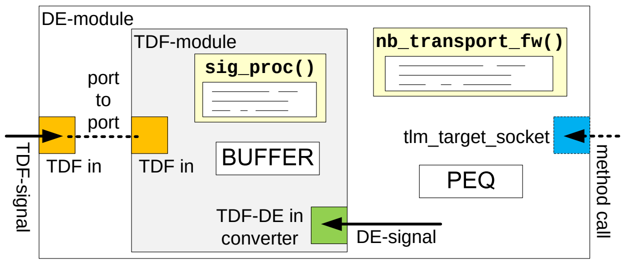

Figure 4: The TDF→TLM converter

We also use a PEQ to store the incoming transactions, such that time delay twists are resolved. Here, we use the PEQ in a way such that an event is produced when a transaction within the queue is ready. In that case, the transaction is taken from the queue, and it is checked whether the internal buffer provides sufficiently many data tokens to fill the transaction's data section. Here, we also have to make sure that we don't return "future" tokens from the transaction's point of view. Note that the presence of such tokens in the internal buffer is perfectly possible when using multirate data flow. If enough valid data tokens are present, the transaction is returned with them. If not, it is returned with an error response.

When the internal fifo is chosen to be of finite size, buffer overflows can occur. Therefore, at every `sig_proc()` execution, it is checked whether the internal buffer contains enough space to take the next chunk of data tokens provided by the TDF side. If not, the converter yields to the SystemC simulation kernel with the same converter port access technique described in Section 5. This gives the TLM side the chance to produce more reading transactions, and might proceed t_{DE} far enough for transactions in the PEQ to become ready. If there is still not enough space in the internal buffer, the surplus data tokens are simply discarded and a warning is raised. We could also choose to throw an simulation error here.

As it can be seen in Figure 4, the architecture of the TDF→TLM converter is pretty similar to the architecture of the TLM→TDF converter. Since the TDF part now does not need to access the PEQ, it is contained in the toplevel DE-module.

7. EXAMPLE SYSTEM

To test our conversion approach, we implemented an example system containing two TDF sources, two TDF drains, three TLM digital signal processing (DSP) modules and a TLM bus (see Figure 5). The idea of this system is that the data coming from $source_i$ is processed by any of the DSPs, and the results are then passed to the respective $drain_i$ ($i = 1, 2, 3$). Here, the exact nature of the computations performed by the DSPs were not the focus of our interest. However, a possible example would be a software defined radio, where the TDF sources would provide data to be modulated (or demodulated). The modulation (or demodulation) schemes to be applied to the source data could be different for every source, but every DSP provides the capabilities to perform them. That is, every DSP checks the sources for new data, reads them, processes them accordingly, and writes the results to the appropriate drain.

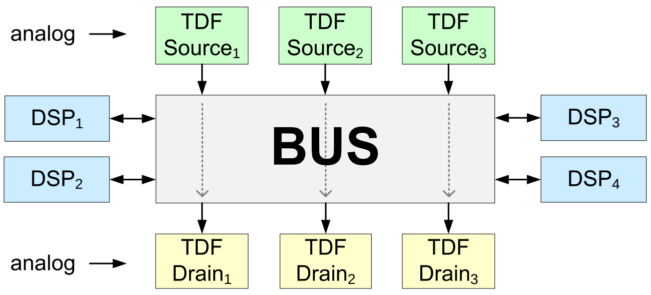

Figure 5: Example system

Our interest here is to demonstrate the functional correctness of our converters and to observe accuracy / simulation speed tradeoffs, typical for loosely timed TLM models. The loss of accuracy in this case manifests itself in data packages arriving at the signal drains in the wrong order. This is possible in principle, since the DSPs run independently from each other. Nevertheless, when the DSPs run in lock step with the simulation kernel (i.e. their time warp is set to 0), the procession delay will make sure that the data package order is preserved. However, when we allow the DSPs to warp ahead in time locally, such data package twists can occur.

Regarding the simulation speed, we measure the number of context switches in the TLM initiators, since a high number of context switches usually increases simulation time. That is, every point in simulation time the simulator switches to the process of one specific DSP accounts for one context switch. We simulated about 16 minutes of time, in which about 14,000 data packages with 128 values each were received by the drains. The sampling period of the sources was 1 ms.

Figure 6 shows the results when using different time warp values. The straight line indicates the simulation speedup factur ranging from 1 to 120. The other line shows the corresponding error rate. With a time warp of 100ms, we didn't experience any errors, while getting about 20 times less context switches. A time warp of 250ms still results in an error rate well below 0.1%, with a speedup factor of 50. If errors are acceptable at all, this is an excellent accurcy / simulation speed tradeoff. On the other hand, a 600ms time warp results in a 10% error rate, while the speedup factor of about 120 is not much of a further gain.

8. CONCLUSION AND FUTURE WORK

In this paper, an approach on how to connect SystemC AMS models with loosely timed TLM 2.0 models using temporal decoupling was presented, with the focus on the SystemC AMS side acting as a streaming data producer and/or consumer. It was shown that the loosely timed coding style of TLM 2.0 can be exploited efficiently to fit with SystemC AMS's TDF, preserving the high simulation performance of both Models of Computation. We described generic converter elements implementing our approach. A small example model was implemented which indicated the converters functional correctness, while the general simulation performance / accuracy tradeoff typically found in loosely timed TLM models could still be observed.

Future work in this area will focus on two aspects: To formalize the conversion problem at hand more rigidly, possibly including a more formal description of the TLM 2.0 loosely timed coding style. And to explore and implement a more structured and sophisticated TLM↔TDF conversion approach which might be able to cover also the case of TDF models acting as TLM initiators.

9. REFERENCES

[1] *The Mathworks Simulink.* http://www.mathworks.com/products/simulink.

[2] *SystemC*$^{\mathrm{TM}}$, 2005. IEEE Std. 1666.

[3] *SystemC AMS extensions Requirements Specification*, 2007. OSCI AMS Working Group.

[4] *VHDL-AMS*, 2007. IEEE Std. 1076.

[5] Accellera. *Verilog-AMS Language Reference Manual Version 2.2*, 2004. http://www.verilog.org/verilog-ams/.

[6] J. Aynsley. OSCI TLM2 User Manual. Technical report, Open SystemC Initiative, 2007.

[7] M. Damm, F. Herrera, J. Haase, E. Villar, and C. Grimm. Using Converter Channels within a Top-Down Design Flow in SystemC. In *Proceedings of the Austrochip 2007*, 2007.

[8] J. Eker, J. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. R. Sachs, and Y. Xiong. Taming heterogeneity — The Ptolemy approach. *Proceedings of the IEEE, Special Issue on Modeling and Design of Embedded Software*, 91(1):127–144, January 2003.

[9] C. Grimm, M. Barnasconi, A. Vachoux, and K. Einwich. An Introduction to Modeling Embedded Analog/Mixed-Signal Systems using SystemC AMS Extensions, June 2008.

[10] F. Herrera and E. Villar. A Framework for Embedded System Specification under different Models of Computation in SystemC. In *Proceedings of the Design Automation Conference*, 2006.

[11] A. Jantsch. *Modelling Embedded Systems and SoCs.* Morgan Kaufmann, June 2003.

[12] E. A. Lee and D. G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on Computers*, C-36(1):24 – 35, 1987.

[13] Open SystemC Initiative. *OSCI TLM2.0.* http://www.systemc.org.

[14] H. Patel and S. Shukla. *SystemC Kernel Extensions for Heterogeneous System Modeling: A Framework for Multi-MoC Modeling.* Springer, July 2004.

[15] A. Vachoux, C. Grimm, and K. Einwich. SystemC Extensions for Heterogeneous and Mixed Discrete/Continuous Systems. In *International Symposium on Circuits and Systems, Kobe, Japan*, May 2005.

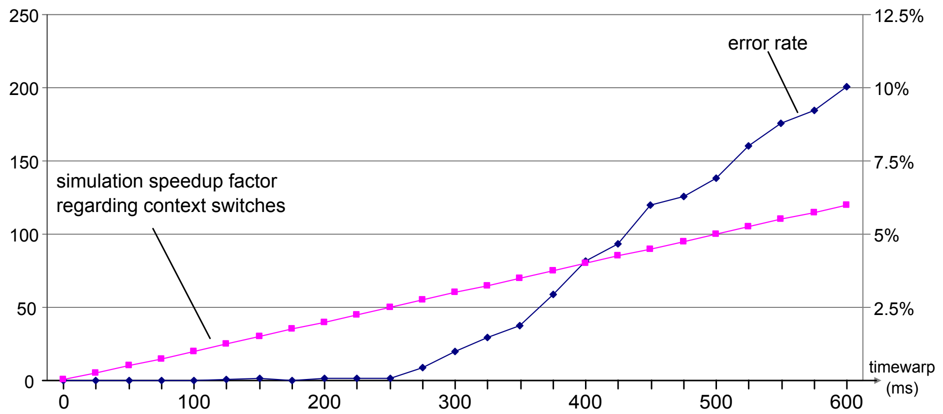

Figure 6: Speed vs. accuracy tradeoff