Scalably Distributed SystemC Simulation for Embedded Applications

Kai Huang, Iuliana Bacivarov, Fabian Hugelshofer, Lothar Thiele *

ABSTRACT

SystemC becomes popular as an efficient system-level modelling language and simulation platform. However, the sole-thread simulation kernel obstacles its performance progress from the potential of modern multi-core machines. This is further aggravated by modern embedded applications that are getting more complex. In this paper, we propose a technique which supports the geographical distribution of an arbitrary number of SystemC simulations, without modifying the SystemC simulation kernel. This technique is suited to distribute functional and approximated-timed TLM simulation. We integrate this technique into a complete MPSoC design space exploration framework and the improvement gained is promising.

1. INTRODUCTION

The complexity of modern embedded applications is steadily increasing. Due to this ever increasing complexity, the time spent to verify, analyze, and validate such applications is continuously growing that hampers the time-to-market, a fundamental factor in highly competitive markets.

Recently, SystemC becomes popular as an efficient system-level modelling language and simulation platform for embedded systems, allowing design and verification at different levels of abstraction. A commonly adopted implementation is provided by the Open SystemC Initiative (OSCI) [6]. Although SystemC is well accepted by the community, the sole-thread simulation kernel however obstacles its performance progress from the potential of modern multi-core machines and geographically distributed systems. This effect is further aggravated when the applications to be simulated are getting more complex.

Since SystemC has been designed for single host simulation only and does not provide by itself means to distribute, there are works in the literature focusing on distributing SystemC simulations. The approaches presented in [2, 5, 10] target geographically distributed IPCore verification, which is orthogonal to our scope. A synchronous data flow (SDF) extension of the simulation kernel is present in [7], where efficiency is gained by the concurrency of the SDF model. In [1], a functional parallel kernel has been developed by running multiple copies of the SystemC scheduler. The major drawback of the two last approaches is the modification of the SystemC kernel, which is not desired for generality and portability reasons. For instance, the work in [1] cannot support all SystemC features due to the modification. A concurrency re-assignment technique is present in [8], trying to act as a compiler front-end and not to modify the SystemC kernel. However, this transformation leads to the problem of semantic equivalence, which is undefined between the new model and the old one.

In this paper, we propose a new technique and implement a SystemC Distribution (SCD) library based on OSCI SystemC 2.2.0, supporting the geographical distribution of SystemC simulations. Distribution here means that an arbitrary number of Linux systems connected by a network can share the simulation workload. This is particularly important as modern computer systems contain multiple CPU cores or form clusters of distributed resources, which SystemC by itself is currently not able to utilize. Compared to the related works, our technique does not require any modification of the SystemC kernel, while it can utilize all the computation power of a multi-core simulation host or of a geographically distributed system. This technique is suited to distribute functional and approximated-timed Transaction Level Modeling (TLM) simulations. We integrate this technique into our MPSoC Design Space Exploration (DSE) framework [9] and the improvement gained is promising. The contributions of this paper can be summarized as follows:

- We propose and implement a new technique for geographically distributing SystemC simulation without modifying the existing SystemC kernel.
- We integrate the SCD library into a complete MPSoC DSE framework. This facilitated the automatic generation of our distributed simulation and the scaling to an arbitrary number of Linux systems. As well, we demonstrate the effectiveness and analyze the impact of our SCD library by a case study.

2. THE OSCI SYSTEMC KERNEL

To be able to distribute the SystemC-based simulation, it is important to understand in detail how its scheduling works. The SystemC simulation kernel is event-based. Execution of processes [1] depends on the occurrence of events that the processes are sensitive to. The kernel keeps track of the set of runnable processes and different event queues.

Figure 1 shows the flowchart of the scheduling behavior. Initially all processes are made `runnable`. As long as runnable processes exist, one of them is selected and executed until it returns or waits for an event to occur. If the current process has generated immediate events, all processes sensitive to these events are made `runnable` immediately. Checking for `runnable` processes and executing them constitutes the *evaluation phase*.

At the end of the evaluation phase, the scheduler calls the `update()` function, which fulfills the update requests

*ETH Zürich, Switzerland, {huang, bacivarov, fabianhu, thiele}@tik.ee.ethz.ch

[1] Besides channels, processes are one of the main components of a SystemC model. They describe functionality of an application and allow parallel activities.

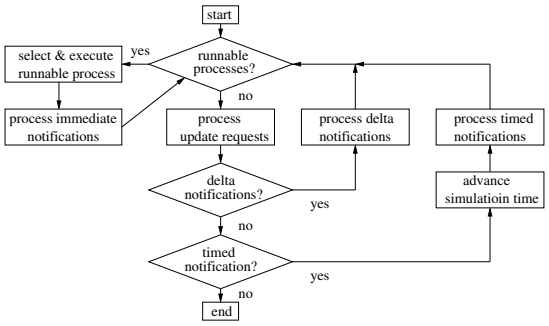


Figure 1: SystemC scheduler.

of respective processes. This *evaluation-update paradigm* enables processes not to consider new values immediately, but before the next *delta cycle*. Only *delta events* and *timed events* can be generated by the `update` function.

After all `update` requests have been processed, the scheduler makes all processes which are sensitive to pending delta notifications `runnable` and the simulation enters the *evaluation phase* again. Simulation is then said to be advanced by one *delta cycle*, which leaves the simulation time unchanged.

When the evaluation phase is finished and no delta events exist, the scheduler checks for *timed events*. Then, the simulation time is advanced to the time of the earliest timed event. All processes sensitive to timed events at the current simulation time are made `runnable` and again, the evaluation phase is entered. If no pending timed events exist, the simulation terminates.

The SystemC simulation can be started by using the method `sc_start`:

- `sc_start()` – Runs the simulation until no more events exist, as described above.
- `sc_start(timeval)` – Advances the simulation with `timeval`. After returning, the simulation time is increased with `timeval` and processes sensitive to timed events at this simulation time are made `runnable`.
- `sc_start(SC_ZERO_TIME)` – Runs the simulation for one *delta cycle* only. Processes sensitive to delta events of the next delta cycle are made `runnable`.

The OSCI SystemC executes only one process at any time, even if the hardware supports execution of concurrent processes. The drawback is that not all computational resources of modern computer systems can be utilized.

3. DISTRIBUTION OF SYSTEMC SIMULATION

The idea of how to distribute is to run the simulation delta cycle wise by calling `sc_start(SC_ZERO_TIME)` repeatedly. After every delta cycle, the simulation returns to the main function, i.e. `sc_main`, where communication and synchronization take place. Communication means transmitting data among simulators. Synchronization refers to globally controlling the simulation activity, for advancing the simulation time or terminating the simulation. By this means, a fixed point of synchronization for the purpose of communication is not enforced; the simulation can be continued immediately if no data has arrived or a remote simulator is not able to accept data. The advantage over previous works is that communication and synchronization are completely decoupled from the SystemC simulation. The simulation time is only advanced if `sc_start(timeval)` is called, independently of the existence of events, currently or for later simulation times. To control the progress of the simulation, we use two methods, i.e. `sc_pending_activity_at_current_time()`[2] and `sc_get_curr_simcontext()->next_time()` .

3.1 SCD Control State Machine

The key features of our SystemC Distribution (SCD) library are (a) the synchronization of the communication and (b) the control among different simulators. A single simulator might be paused or reactivated based on the data availability from a remote simulator. For this, three simulation states are defined:

- `busy` – The simulator has events at the current simulation time and is simulating.
- `idle` – The simulator has currently no events, but at a later simulation time, and it is waiting for the simulation time to be advanced.
- `done` – The simulator has no events at all and is waiting for the simulation to be terminated.

To manage the global simulation state, a master-slave approach is taken. A master simulator is connected with every other involved simulator. The slave simulators continuously inform the master about their local simulation state. The master collects this information and advances the simulation time or terminates the simulation if the global state is `idle` or `done`, respectively. As the communication of the local state takes some time, the master does not have an accurate view of the slaves' local state at every time. To enable this centralized simulation, three finite state machines have been implemented, i.e. for the master, slave and slave wrapper that represents the master's current view of a slave's state, as shown in Fig. 2.

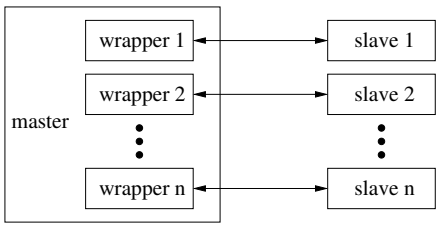
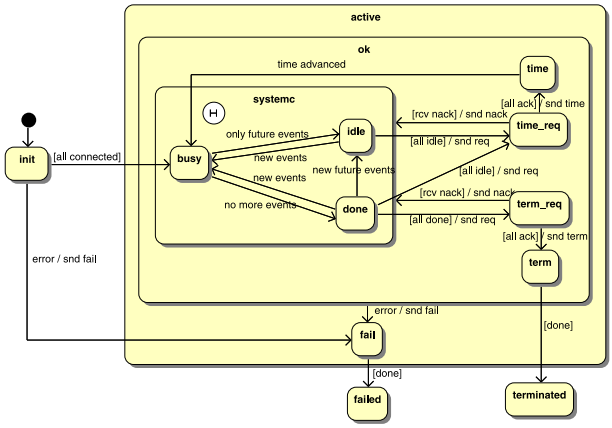


Figure 2: SCD framework.

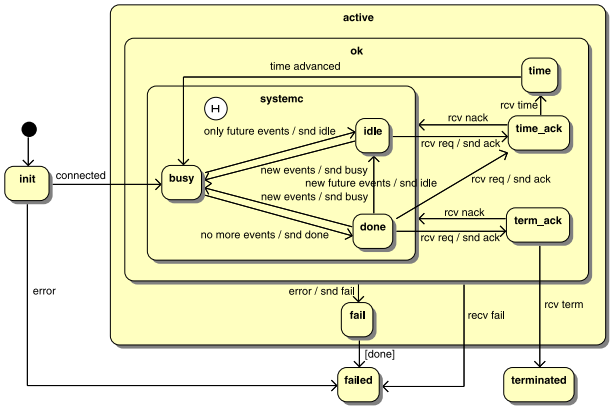
Fig. 3 shows the state charts of the master and slave control state machines, respectively. The slave wrapper state machine is omitted, since it reflects the master's view on a slave's state. Master and slave controllers behave the same, except that the slave controller will always inform the master about state transitions. Some of the main states are explained in the following:

`init` and `busy` states. All controllers start in the `init` state. As soon as the connection between a slave and the master controller has been established, the slave moves to

[2]This method is introduced only in the release notes of the newest SystemC version 2.2.0.



(a) Master



(b) Slave

Figure 3: Control state machines.

the `busy` state. The master controller does so, if all slaves are connected.

`fail` and `failed` states. If an error occurs during the simulation, a slave informs the master by sending a *fail* message and moves from `fail` to `failed` state as soon as this message has been transmitted. In case that the master observes an error, it sends an *fail* message to all slaves. This will bring all slaves to the `failed` state immediately. The master moves to `failed` as soon as all *fail* messages have been sent.

`systemc` state. The `systemc` super state [3] reflects the state of the local simulation, i.e. `busy`, `idle` or `done`. Transitions between these states are triggered by events at the current or later simulation times. As described before, moving from `busy` to `idle` or `done` is only allowed, if there is no inter-simulator communication activity.

`done` and `terminated` states. A simulator will terminate, if its control state machine leaves the `active` super state. This can be because the simulation either failed or terminated. The termination procedure is the following: As soon as all slaves and the master seem to be `done`, the master will send a *termination request* to all slaves. If one of the slaves responds with *non-acknowledgement*, all slaves will be informed about the *abortion of the termination attempt*. Otherwise, the master sends the *termination message* which leads all slaves to `terminated`. The master terminates as soon as all termination messages have been sent.

`time` state. The synchronization for advancing the simulation time behaves similar to the termination. It is triggered if none of the slaves is `busy`. If all slaves acknowledge their *time request*, the master globally determines the time of the next event. It then sends the simulation advancing time step to the slaves in a *time message*. All simulators are moved to the `time` state. As soon as the simulation time has been advanced, the slaves become `busy` again.

3.2 Automated Simulation Generation

As a proof-of-concept, we include our SCD library to distribute the functional SystemC simulation automatically generated by our MPSoC DSE framework [9]. Our MPSoC DSE framework takes as input three separate parts: (1) the application, specified in a process network fashion [4], i.e. as concurrent autonomous processes communicating via point-to-point FIFO channels; (2) the architecture, abstractly specified and containing just structural, performance, and parametric data necessary for design decisions; (3) the mapping, referring to binding of application processes to computation resources and channels to communication resources, and the scheduling policies accordingly. The structure of the application, the abstract architecture and the mapping are specified in customized XML format.

For the generation of the distributed simulation, the architecture XML schema is extended in order to support the definition of the number of concurrent simulators and their allocations to simulation hosts. The source code of each process remains unmodified, a wrapper of which is generated to glue it into a `sc_thread`. In this way, a process can be bundled to a specific simulator in conformity to the mapping specification. The channels connecting processes within the same simulator are replaced by the SystemC `sc_channel`, while those connecting processes located on different simulators adopt the *internet protocol*. The communication primitives in the source code, i.e. read and write primitives of the FIFO channel, will be redefined as SystemC `sc_port` and *socket-based interface*, depending on if they operate on the same simulator or inter-simulator communication, respectively.

Taking advantage of the existing framework, the distributed simulation can be automatically generated using the `visitor` design pattern [3], based on the application, architecture, and mapping specifications. The simulation can thereby scale to an arbitrary large number of simulation hosts, just by redefining the amount of simulators and simulation hosts in the architecture specification. For the users convenience, a script is generated to control the distribution, execution, and result collection of the simulation.

4. EXPERIMENT

This section analyzes the impact of our SCD library, by applying it to the distribution of the functional simulation of an MPEG-2 video decoder application [9].

4.1 Experiment Setting

The MPEG-2 video decoder application is specified as a process network [9], with 20 parallel processes. It decodes 15 s of a compressed video sequence, with a resolution of 704×576 pixels and a frame rate of 25 fps.

The simulation hosts have identical configurations: each

[3] `systemc` is a super state because it includes other states and reflects their dependencies.

machine has two AMD Opteron 2218 dual core processors at 2.6 GHz, i.e. four cores in total. The operation system is Debian Linux with kernel version 2.6.23. The simulation hosts are connected by Gigabit Ethernet where round trip times measured with `ping` were below 0.1 ms.

We measure two time criteria: (1) simulation time (`Sim.`), corresponding to the time from the start of the simulation until it completes, as a measurement with a normal stopwatch, which gives information about how fast an execution is, and (2) accumulated computing time (`Comp.`), indicating how much computation power the application has actually expended. All times are rounded to seconds.

4.2 Measurements

Tab. 1 shows the results of 6 different cases, using different libraries and different numbers of simulation hosts (`#Hosts`) and simulators per host (`#Sims`).

Case	Library	#Hosts	#Sims	Sim.	Comp.
1	Pthread	1	1	31 min 46 s	80 min 47 s
2	SystemC	1	1	12 min 13 s	12 min 13 s
3	SCD	1	1	12 min 14 s	12 min 13 s
4	SCD	1	4	5 min 21 s	12 min 34 s
5	SCD	4	1	5 min 23 s	12 min 42 s
6	SCD	5	4	2 min 41 s	13 min 24 s

Table 1: Runtimes of the MPEG-2 example.

`Case 1` shows the performance of a Posix Threads (Pthreads) implementation. Pthreads are, in contrast to the SystemC threads, preemptive, i.e. all available CPU cores can be in use in principle. As shown in the table, the computation time of `Case 1` is higher than the simulation time. The Pthread implementation is however inefficient, running almost 30 min and computing 80 min. The inefficiency is mainly caused by the massive system calls made by Pthreads, implying a large context switch overhead.

In contrast to Pthreads, SystemC simulations are much faster due to the low switching and synchronization overhead of the non-preemptive threads. But the sole-thread SystemC simulation kernel can use only one CPU core. As depicted in the table, `Case 2` has equal simulation and computation times.

To find out the overhead induced by our SCD library, `Case 3` runs the SCD simulation on only one simulator. The difference with respect to `Case 2`, is that now the simulation runs delta cycle wise, while `Case 2` is started with `sc_start()`. The overhead of delta cycle execution is considerably small, i.e. 100 ms, not even explicit in Tab. 1 where times are rounded to seconds.

To demonstrate the speedup of the new SCD simulation, the application is mapped onto four simulators. `Case 4` runs all four simulators on the same simulation host, using all four CPU cores, while `Case 5` runs the same four simulators on distinct simulation hosts, which enables network communications. In both `Cases 4` and `5`, a factor of more than 2x speedup is achieved. Compared to `Case 3`, the slight increase in computation time is due to additional communication and synchronization cycles. The overhead is only 2 % of the overall computation time, which is impressive since about 3 GB of data has been transferred during the simulation. Note that the simulation time is not reduced by a factor of 4, although 4x more resources are used. The reason is that the computation is not equally distributed to the four simulators: actually, one computes for more than 5 min. while the others are computing only about half of this time. This non-optimal mapping is due to the simulation distribution currently based on designers' experience and empirical estimation of the process complexity.

Finally, the maximum speedup of 4.5x is depicted in `Case 6` where the 20 simulators, one for each process, are distributed among 5 simulation hosts. For further speedup, one needs to redefine the granularity of the process network of the application.

5. CONCLUSIONS

We propose a new technique and implement a library, namely SCD, which allows the distribution of the SystemC simulation workload among several simulation hosts, consequently reducing the overall simulation time. This is particular important as modern computer systems have multiple CPU cores which SystemC by itself is not able to use. We integrate this library into our MPSoC DSE framework. A distributed functional simulator can thereby be automatically generated and scaled to arbitrary simulation hosts. In the future, we plan to integrate this library to our TLM simulation and automatize the generation of this distributed TLM simulation.

Acknowledgements

This research has been funded by European Integrated Project SHAPES under IST Future Emerging Technologies - Advanced Computing Architecture (ACA). Project number: 26825.

6. REFERENCES

[1] B. Chopard, P. Combes, and J. Zory. A conservative approach to systemc parallelization. In *International Conference on Computational Science (ICCS) Part IV*, pages 653–660, 2006.

[2] A. Fin, F. Fummi, and D. Signoretto. The use of SystemC for design verification and integration test of IP-cores. In *Proceedings of the 14th Annual IEEE International ASIC/SoC Conference*, pages 76–80, September 2001.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[4] G. Kahn. The semantics of a simple language for parallel programming. In *Information Processing '74: Proceedings of the IFIP Congress*, pages 471–475. 1974.

[5] S. Meftali, A. Dziri, L. Charest, P. Marquet, and J.-L. Dekeyser. SOAP based distributed simulation environment for System-on-Chip (SoC) design. In *Forum on Specification and Design Languages, FDL'05*, December 2005.

[6] O. S. I. (OSCI). Website. `http://www.systemc.org`.

[7] H. Patel and S. Shukla. Towards a heterogeneous simulation kernel for system-level models: a systemc kernel for synchronous data flow models. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(8):1261–1271, Aug. 2005.

[8] N. Savoiu, S. Shukla, and R. Gupta. Automated concurrency re-assignment in high level system models for efficient system-level simulation. *Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings*, pages 875–881, 2002.

[9] L. Thiele, I. Bacivarov, W. Haid, and K. Huang. Mapping applications to tiled multiprocessor embedded systems. In *ACSD '07: Proceedings of the Seventh International Conference on Application of Concurrency to System Design*, pages 29–40, 2007.

[10] M. Trams. Conservative distributed discrete event simulation with SystemC using explicit lookahead. `http://digital-force.net/publications/`, 2004.