Parallel Simulation of SystemC TLM 2.0 Compliant MPSoC on SMP Workstations

Aline Mello , **Isaac Maia** , **Alain Greiner** , and **Francois Pecheux**
{Aline.Vieira-de-Mello, Isaac.Maia, Alain.Greiner, Francois.Pecheux}@lip6.fr

lip6 - Laboratoire d'Informatique de Paris 6
Université Pierre et Marie Curie
4, Place Jussieu - Paris - France

ABSTRACT

The simulation speed is a key issue in virtual prototyping of Multi-Processors System on Chip (MPSoCs). The SystemC TLM2.0 (Transaction Level Modeling) approach accelerates the simulation by using Interface Method Calls (IMC) to implement the communications between hardware components. Another source of speedup can be exploited by parallel simulation. Multi-core workstations are becoming the mainstream, and SMP workstations will soon contain several tens of cores. The standard SystemC simulation engine uses a centralized scheduler, that is clearly the bottleneck for a parallel simulation. This paper has two main contributions. The first is a general modeling strategy for shared memory MPSoCs, called TLM-DT (Transaction Level Modeling with Distributed Time). The second is a truly parallel simulation engine, called SystemC-SMP. First experimental results on a 40 processor MPSoC virtual prototype running on a dual-core workstation demonstrate a 1.8 speedup, versus a sequential simulation.

Keywords

MPSoC, Parallel Simulation, SystemC, SMP workstations

1. INTRODUCTION

Despite their relative novelty, Multi-Processors System on Chip (MPSoCs) containing a few cores tend to be replaced by Massively Parallel MPSoCs (MP2SoCs), which integrate dozens or hundreds of processor cores interconnected through a possibly hierarchical network on chip. The increase of processing power and parallelism creates the need for faster yet accurate simulation tools for virtual prototyping, supporting both functional verification and performance evaluation.

Several industrial and academic frameworks appeared to help modeling, simulating and debugging these architectures. The SystemC hardware description language[6] is the effective backbone of all these frameworks. The SystemC library of C++ classes allows to describe the hardware at various levels of abstraction, ranging from synthezisable RTL to Transactional Level Modeling (TLM). However, when it comes to simulate an architecture containing hundreds of processors, even the simulation speed provided by the TLM is not enough.

Simultaneously, multi-core workstations are becoming the mainstream, and SMP (Symmetric Multi Processors) workstations will soon contains several tens of cores[11]. Unfortunately, the genuine SystemC simulation kernel is fully sequential and cannot exploit the processing power provided by these multi-cores machines.

The present work proposes a general modeling strategy for shared memory MPSoCs, called TLM-DT (Transaction Level Modeling with Distributed Time). Moreover, it describes a first implementation of a parallel simulation engine, called SystemC-SMP.

This paper is composed of five sections. After this introduction, section 2 introduces the principles of the PDES algorithm and describes the proposed TLM-DT approach. Section 3 delves into the parallel simulation kernel and the implementation of the critical synchronization primitives in a multicore environment. Section 4 presents a full-fledged experimental setup with a virtual architecture containing 40 processors, and gives the simulation speedup. Section 5 comments these first results and provides some perspectives on ongoing researches.

2. THE PROPOSED TLM-DT APPROACH

The Transaction Level Modeling (TLM) approach accelerates the simulation by using Interface Method Calls (IMC) to implement the communications between hardware components. Another source of speedup can be exploited by parallel simulation. Multi-core workstations are becoming the mainstream, and SMP workstations will soon contain several tens of cores[11]. Anticipating this trend, the Transaction Level Modeling with Distributed Time (TLM-DT) is proposed. It is an extension of the work presented in [12].

A SystemC transaction level model is generally a collection of *SC_THREADs* modeling the various hardware components of the simulated architecture. These *SC_THREADs* are good candidates to be executed in parallel on the multiple cores of a SMP workstation. The main difficulty for SystemC parallel simulation is that SystemC, as most hardware description language (including VHDL & Verilog), relies on the DES (Discrete Event Simulation) algorithm. In most event-driven simulation engines, the simulation is controlled by a central scheduler, that contains a list of time-ordered events and a global simulation time. This means that - in the standard SystemC simulation engine - all the *SC_THREADs* are controlled by the same central scheduler, that is clearly the bottleneck for a parallel simulation.

Several parallelization techniques have been used to get around this bottleneck. TLM-DT implements the PDES

(Parallel Discrete Event Simulation) principles[1], where the system is described as a set of logical processes that execute in parallel and communicate via point-to-point channels. In this approach, the global simulation time does not exist anymore, each logical process involved in the simulation has its own local time, and the processes synchronize themselves through timed messages. In the conservative PDES, a logical process is allowed to increase its local time if and only if it has the guarantee that it cannot receive on any of its input channels a message with a timestamp smaller than its local time. This constraint can be violated in the optimistic PDES, but a rollback mechanism is needed to restore a process into a previous state in case of violation. This rollback mechanism is very expensive and cannot reasonably be used with MPSoC. To solve this issue, the conservative PDES algorithm uses null messages, that contain no data, but only timing information. The null messages must be sent by each process at regular and bounded time intervals in order to prevent deadlocks.

2.1 TLM-DT compliance with TLM2.0

The TLM-DT simulation models use the generic payload and phase, the initiator and target sockets, and the non blocking transport functions defined by the TLM2.0 standard. Although TLM-DT models are compliant with TLM2.0, shifting from global time to distributed time induces some differences.

In TLM2.0, the synchronization between the hardware components is accomplished by yielding control to the SystemC central scheduler, that executes sequentially each process, respecting the general evaluate-update paradigm[2] that is the basis of the DES algorithm. Regarding the time representation, TLM2.0 suggests two coding styles: approximately-timed and loosely-timed[6]. The approximately-timed coding style (TLM-AT) is the strict implementation of the DES algorithm. Timed processes (implemented as *SC_THREADs*) are annotated by specific delays, and synchronize with the central scheduler using the *wait()* primitive, which means that processes are unscheduled at each synchronization point. This allows an accurate description of the timing behaviour of shared memory MPSoC, but the simulation speed can be very low, as the number of context switches is very large. The loosely-timed coding style (TLM-LT) supports temporal decoupling, where processes can run ahead of the global simulation time (without unscheduling) for a bounded quantum of time. In this approach, the transport interface methods are annotated with delays, that are interpreted as local time offsets relative to the global simulation time. The loosely-timed coding style permits a significant simulation speed improvement by reducing the number of synchronization events, but it does not ensure correct system synchronization.

In TLM-DT, the synchronization between timed processes is not anymore centralized in the scheduler, but distributed by annotating all messages with timing information. For this reason, TLM-DT proposes its own coding style. In TLM-DT coding style, each *SC_THREAD* has an absolute local time and sends it as the third argument of the transport interface methods, as suggested by the TLM-LT. This absolute local time must be set to zero at the beginning of simulation and is increased during simulation. It can still be interpreted as an offset relative to the SystemC global simulation time because this time is never incremented in TLM-DT. Only three synchronization primitives are allowed: *wait(sc_event)*, *wait(SC_ZERO_TIME)*, and *sc_event.notify(SC_ZERO_TIME)*. Any TLM-DT description can be simulated using the standard SystemC and the standard TLM2.0 package, provided by the OSCI consortium. As presented in the experimental section, and thanks to the intrinsic properties of the PDES algorithm, the dispersion between the various local time is limited, and the timing error of the TLM-DT simulation (versus a cycle accurate simulation) is low.

2.2 Components Modeling

In most cases, a complex MPSoC is structured in several sub-systems (or clusters). Each cluster can contain several initiators, several targets, and a local interconnect. A global interconnect manages the communications between clusters. The figure 1 depicts a typical shared memory, clusterized architecture.

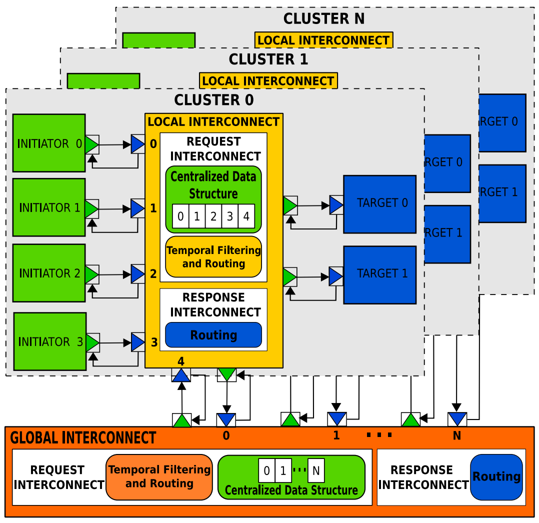

Figure 1: TLM-DT system example

A **TLM-DT VCI-OCP initiator** contains at least one initiator socket and one *SC_THREAD*. An initiator *SC_THREAD* runs until it reaches an explicit synchronization point or when it has consumed a predefined time quantum. In case of explicit synchronization, such as a memory access, a transaction object is initialized and sent by means of the *nb_transport_fw()* method. Afterwards, the corresponding *SC_THREAD* is unscheduled by an explicit *wait(rsp_event)*, and waits for the response. When the response is received by means of the *nb_transport_bw()* method, the initiator local time is updated, and the *SC_THREAD* is resumed by a *rsp_event.notify()* primitive. The local time of an initiator is updated in two cases: (1) it is directly increased by the initiator itself (processing time) and (2) it is updated with the return time of the *nb_transport_bw* method (communication time). Whenever the local time is updated, the TLM-DT VCI-OCP initiator checks if its time quantum has been reached. If this is the case, it sends a null message with its current local time and is unscheduled.

A **TLM-DT VCI-OCP target** has at least one target socket and one *SC_THREAD*. A target is reactive, i.e. the *SC_THREAD* remains sleeping until it receives a request transaction by means of the *nb_transport_fw()* method, that executes a *req_event.notify()*. When this is the case, the transaction is processed, the target local time is set to the transaction time value, and the response transaction is returned to the initiator by calling the *nb_transport_bw()* method. The local time of a target is updated in two cases: (1) it is updated to the request transaction time if and only if this time is greater than the local time (this condition is satisfied when the interval between two consecutive request transactions is greater than the target processing time), and (2) it is directly increased by the target itself (response processing time).

A **TLM-DT VCI-OCP interconnect** represents actually two fully independent networks for requests and responses respectively, in order to avoid deadlocks. The request network is associated to the forward path and the response network is associated to the backward path.

The request network has two functionalities: (1) it performs the conservative PDES algorithm (transactions must be processed in a strictly time-ordered manner), and (2) it implements the routing function (the transaction address field is analyzed, and the transaction is routed to the proper target). The request network contains a centralized data structure (called PDES buffer) to store the transactions and one *SC_THREAD*. The PDES buffer has a reserved slot for each input channel (there is of course one target socket per input channel). When a target socket receives a transaction by means of the *nb_transport_fw()* method, it is stored in the corresponding slot. The interconnect *SC_THREAD* is not activated until all slots contain at least one transaction. When this condition is reached, a temporal filtering is performed to select the request with the smallest timestamp. If the selected transaction is a null message, it is deleted and not sent. In another case, the transaction time is increased by the interconnect latency and the transaction is send on the proper correct initiator socket.

In principle, the response network can implement a similar approach to route the *nb_transport_bw()* calls. In order to increase the simulation speed, the contention is not modeled in the response network. There is neither *SC_THREAD* nor temporal filtering, and the responses to different initiators can be sent out of order. In the response network, there is only a routing function that analyzes the transaction source_identifier field to route the response to the proper initiator.

This modeling strategy supports hierarchical interconnects. In a two-level interconnect, the local interconnect connects the initiators to the targets that belong to the same cluster, and the global interconnect connects the different clusters. Both levels have the same structure and behavior.

When it uses standard SystemC simulation engine, the TLM-DT coding style combines the advantages of the loosely-timed coding style (simulation speed), and the approximately-timed coding style (high accuracy). But the main advantage of the TLM-DT approach is that it does not use anymore the SystemC global simulation time, and it becomes possible to use a truly parallel simulation engine.

3. PARALLEL SIMULATION

The main idea of the **SystemC-SMP** simulation engine is to take advantage of the TLM-DT distributed approach to perform parallel simulation on SMP workstations. SystemC-SMP is dedicated to the TLM-DT coding style and does not require any modification in the simulation models running with the standard SystemC[6] simulation kernel.

From the simulation kernel viewpoint, a TLM-DT platform can be seem as a set of communicating *SC_THREADs* that use *sc_event* objects to synchronize themselves. Regarding the *SC_THREADs* scheduling, neither timed nor immediate notifications are used. This means that the evaluate-update scheduling algorithm[2] is not adapted to the SystemC-SMP parallel simulation kernel.

3.1 Simulator Software Architecture

The SystemC-SMP uses a gang-scheduling [10] approach by grouping related neighboring *SC_THREADs* on the same physical CPU of the SMP workstation. Communicating *SC_THREADs* must be executed on the same physical CPU in order to benefit the cache hierarchy [7]. Moreover, the gang-scheduling can minimize the blocking bottleneck [3]. This approach can be used because the communication graph is fully determined by the MPSoC hardware architecture, and can be statically analysed by the system designer. The *SC_THREAD* mapping can be explicitly controlled by the system designer through configuration directives that are described in the section 3.3.

The SystemC-SMP software architecture is represented in the Figure 2. The **TLM-DT Virtual Platform** is the user code after elaboration phase[2]. It is presented in section 2 and it is not part of the simulation engine. The **SystemC-SMP kernel** is responsible for the creation and termination of simulation objects. It implements shared objects visible to all local schedulers. A *local scheduler* is responsible for scheduling all *SC_THREADs* that are executed on the same physical CPU. Each *SC_THREAD* is implemented as a QuickThread[4], as in the standard SystemC. The **O.S Kernel** is the host Operating System kernel that provides a POSIX-thread[10] API. Each *local scheduler* of SystemC-SMP is executed in a **POSIX-thread** (pthread) [10] of the host O.S. With this software architecture, it is possible to use the CPU-affinity[5] functions provided by the host O.S to associate each pthread to a physical CPU. The **SMP Hardware** is the Symmetric Multiprocessing[7] hardware implementing a shared memory and several physical CPUs (with both private L1 caches, and shared L2 caches).

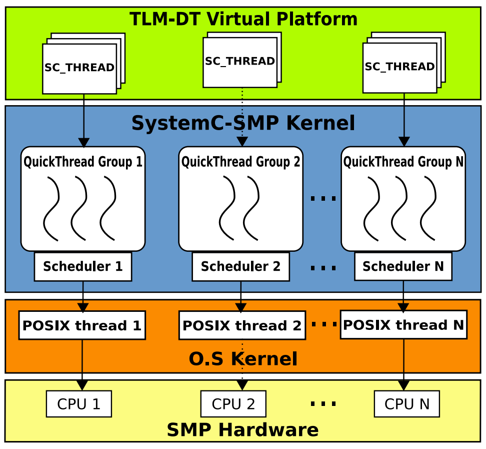

Figure 2: Simulator Software Architecture

3.2 Synchronization

The *SC_THREADs* running in parallel on the SMP workstation use *sc_event* objects to synchronize. As mentioned before, TLM-DT models use three primitives: *wait(sc_event)*, *sc_event.notify(SC_ZERO_TIME)* and *wait(SC_ZERO_TIME)*. Considering a certain *sc_event* instance named ***e***, two scenarios can occur in a SMP environment. The first scenario is when ***e.notify()*** is executed after ***wait(e)***. This is the normal situation where a *SC_THREAD* blocks and yields on behalf to another *SC_THREAD* by calling the *wait(e)* primitive. It resumes when the *e.notify()* is called. The second scenario is when ***e.notify()*** is executed before ***wait(e)***. In this case, the *SC_THREAD* executing the *wait(e)* must continue its execution without block.

An *sc_event* can be in one of three states: *IDLE*, *WAITING* and *NOTIFIED*. The state of a *sc_event* can be changed by different *SC_THREADs* in the same time, therefore *wait(e)* and *e.notify()* have been implemented using atomic test-and-set[10] instructions.

3.3 Configuration Directives

SystemC-SMP provides a way to bind groups of *SC_THREAD* to a physical CPU using the macro *MAP_CPU(cpu_n, module_instance)*. In this macro *cpu_n* is a CPU index and *module_instance* is an hardware component instantiated in the simulated architecture (i.e. a *sc_module* instance name). Using this macro, all *SC_THREADs* within a certain module will be mapped to a specific scheduler and it will be mapped within CPU with the same index. These configuration macros can be included in the SystemC top-cell, using conditional compilation directives to keep fully compatibility with standard SystemC.

4. EXPERIMENTAL RESULTS

The conducted experiments can be described by the triplet: Embedded Application (**EA**), Hardware Architecture (**HA**), and Simulation Engine (**SE**). The retained **EA** is an integer implementation of the Smith-Waterman (SW) algorithm [8] that performs sequence alignment in biocomputing. We used a parallel, multi-threaded implementation of this algorithm, written in C language. The **HA** is a shared-memory NUMA structured in clusters, as presented in Figure 1. This architecture contains 10 clusters interconnected by a global Network on Chip (global interconnect) supporting read/write communication primitives in the shared address space. Each cluster contains four 32-bits processors (with data and instruction caches), a local memory and a local interconnect. The **SE** corresponds to the physical multi-core workstation on which the simulation takes place. The experiments have been performed under Linux 2.6.18 on AMD Athlon(tm) Dual Core Processor 2.3GHz with 128KB L1-cache, 512KB L2-cache and 1GB RAM.

The experimental results show that a speedup factor equal to **1.9** is obtained when the performance of the dual-core SystemC-SMP (**263** seconds) is compared to the single-core Standard SystemC (**506** seconds), which is very close to the threoretical upper bound given by Amdahl's law.

In order to evaluate the timing accuracy of the parallel TLM-DT simulation using the SystemC-SMP simulation engine, we performed a cycle accurate simulation for the the same EA. The HA was described using the CABA (Cycle-Accurate, Bit-Accurate) simulation models available in [9]. The cycle accurate simulation completed in 68264502 cycles, while the TLM-DT simulation completed in 64400433 cycles, corresponding to a **6%** timing error. The execution time of the cycle accurate simulation (single-core) is **45x** (11980 seconds) slower than the dual-core SystemC-SMP.

5. CONCLUSION

In this paper, a new modeling approach for timed TLM virtual prototyping of shared memory MPSoCs was presented. The proposed TLM-DT coding style is fully compliant with the TLM2.0 standard. It enforces the Parallel Discrete Event Simulation principles and uses a distributed representation of time instead of the SystemC global simulation time. A TLM-DT model can be simulated with the standard OSCI simulation engine and the standard TLM2.0 package. In order to take advantage of TLM-DT parallel behavior, a new parallel simulation kernel was presented (SystemC-SMP). The SystemC-SMP running on a dual-core workstation obtained a 1.9 speedup in relation to the standard SystemC. This is is very close to the threoretical upper bound given by Amdahl's law.

TLM-DT coding style and SystemC-SMP simulation engine reduce locking by using small critical sections. For this reason, these approaches can be truly scalable and used on SMP workstations containing more of processing cores. This still has to be demonstrated in future work.

6. REFERENCES

[1] K. M. Chandy and J. Misra. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Trans. on Softw. Eng*, 5(5):440–452, 1979.

[2] D. A. S. Committee. IEEE Std 1666 - 2005 IEEE standard SystemC language reference manual, 2006.

[3] M. A. Jette. Expanding symmetric multiprocessor capability through gang scheduling. In *IPPS Workshop Proceedings*, page pages. Springer, 1998.

[4] D. Keppel. Tools and techniques for building fast portable threads packages. Technical Report UWCSE 93-05-06, University of Washington, 1993.

[5] B. Lewis and D. J. Berg. *Threads primer: a guide to multithreaded programming.* Prentice Hall Press, Upper Saddle River, NJ, USA, 1995.

[6] OSCI. SystemC. http://www.systemc.org.

[7] C. Schimmel. *UNIX systems for modern architectures: symmetric multiprocessing and caching for kernel programmers.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1994.

[8] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[9] SoCLib. Soclib project mainpage. http://www.soclib.fr/.

[10] W. Stallings. *Operating Systems (6th ed.): Internals and Design Principles.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2008.

[11] D. Strom and W. Gruener. Pat gelsinger: A shot at running intel. Tom's Hardware Guide, May, 2005.

[12] E. Viaud, F. Pêcheux, and A. Greiner. An efficient tlm/t modeling and simulation environment based on conservative parallel discrete event principles. In *DATE Conference Proceedings*, pages 94–99, 2006.