Parallel TLM simulation of MPSoC on SMP workstations: Influence of communication locality

Isaac Maïa Pessoa, Aline Vieira de Mello, Alain Greiner, François Pêcheux

Parallel TLM Simulation of MPSoC on SMP Workstations: Influence of Communication Locality

Isaac Maia Pessoa, Aline Mello, Alain Greiner, François Pêcheux
Lip6 - Laboratoire d'Informatique de Paris 6
Université Pierre et Marie Curie
4,place Jussieu - Paris - France
Email: {Isaac.Maia,Aline.Vieira-de-Mello,Alain.Greiner,Francois.Pecheux}@lip6.fr

***Abstract*— Simulation speed is a key issue in virtual prototyping of Multi-Processors System on Chip (MPSoCs). SystemC TLM2.0 (Transaction Level Modeling) is now commonly used to accelerate the simulation. However, the standard SystemC simulation engine uses a centralized scheduler that is clearly a bottleneck to parallelize the simulation of architectures containing hundreds of processor cores, and involving hundreds of *SC_THREADs* to be scheduled. In this paper, we describe a general modeling strategy for shared memory MPSoCs and associated tools for the parallel TLM simulation of these architectures. The proposed approach is based on the Parallel Discrete Event Simulation principles, and our parallel version of the SystemC kernel (named SystemC-SMP) that can run advantageously on multiprocessor workstations. As the speedup obtained by parallel simulation depends on the communication pattern between the parallel tasks, we study the influence of various locality characteristics for the software application running on the simulated MPSoC.**

I. Introduction

The emergence of multicore technologies in modern PCs and workstations has brought an incredible parallelism potential on our desks. On the other hand, Massively Parallel MPSoC (MP2SoC) will soon integrate in a multi-layered structure hundreds of processor cores, and the simulation of these complex architectures could benefit from the intrinsic parallelism of these multi-core workstations.

The Figure 1(a,b) presents a shared-memory, typical clusterized MP2SoC architecture. Each tile (or cluster) can contain a local interconnect, one or several processor cores, one or several memory bank, various peripherals, and a network interface controller to access a hierarchical network on chip.

An embedded software application running on such a massively parallel multi-processor architectures (MP2SoC) is generally designed as a set of parallel software tasks cooperating in a shared address space (see Figure 1a). The performance of the embedded application running on the MP2SoC relies on both the optimal mapping of the software tasks on the processors, and the optimal mapping of the software data (communication buffers, execution stacks, etc.) on the physical memory banks. As the cost of a remote access (to a remote memory in another cluster) is much higher than the cost of a local access (to a local memory in the same cluster), both the performance and the power consumption strongly depend on the quality of the mapping. For a given software application, a good indicator of the mapping quality is the ratio between the number of intra-cluster (local) transactions, and the number of inter-cluster (global) transactions.

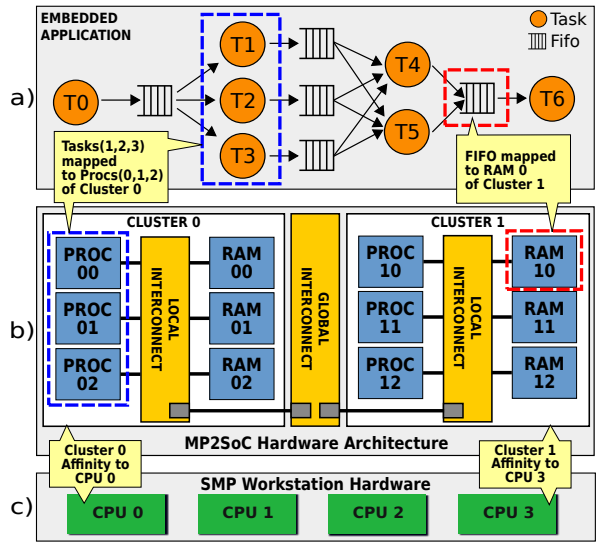

Fig. 1. TLM-DT Platform Example

Current modeling approaches for MP2SoCs are mostly based on SystemC, a library of C++ classes that allows description of MP2SoC hardware at various levels of abstraction, ranging from synthesizable RTL (most accurate, very slow) to Transactional Level Modeling or TLM (less accurate, very fast). However, when it comes to simulate a tiled architecture containing hundreds of processors, even the simulation speedup brought by TLM is not enough.

Unfortunately, the standard SystemC simulation kernel distributed by the OSCI consortium is sequential, and cannot exploit the parallelism of multi-cores workstations. In this paper we propose a new modeling style optimised for parallel execution, called TLM-DT, we present a new parallel simulation engine called SystemC-SMP, and we study the influence of the mapping of the embedded application tasks running on the MP2SoC processors, on the speedup obtained by parallel simulation.

This paper is composed of six sections. After this introduction, section II present the state of the art regarding SystemC parallel simulation. Section III introduces the principles of

the PDES algorithm and describes the proposed TLM-DT approach (Transaction Level Model with Distributed Time). Section IV details the parallel simulation kernel and the implementation of the critical synchronization primitives in a multicore environment. Section V presents the experimental results obtained for a 40 clusters architecture (640 processors). Section VI comments these first results and provides some perspectives on ongoing researches.

II. Related Work

Cox[**?**] developed a parallel implementation of the SystemC simulation kernel: RITSim. Cox proposes to integrate the parallel/distributed simulation engine into the SystemC kernel. Performance results are presented for a distributed simulator running on a cluster of machines connected over both a 100 Mb/s and 1000 Mb/s networks. The results show that a speedup is obtained if and only if a large amount of data is exchanged in a transaction. This is generally not the case in shared memory architectures.

Chopard et al.[**?**] proposed another implementation of SystemC. The approach requires that the user determines which modules are executed by which processors. For that purpose, the approach extends the SystemC library with a new class *sc_node_module*, which is only used to create a hierarchical module. The implementation has a centralized thread used by all the other threads for the synchronization tasks and does not implement any kind of lookahead. The centralized thread determines which threads are to be executed in parallel. There is a significant synchronization overhead that is discussed in [**?**].

In 2007 Bouzouzou[**?**] presented a parallel implementation of SystemC using the pthreads library. The parallel engine is executed on a SMP workstation, and not on a network of workstations. This implementation uses a centralized scheduler. An average 3x speedup is obtained on a highly parallel model running on a quad core workstation. However, the method is not convenient when the simulated architecture contains hardware components that are used by multiple initiator components, which is the case in shared memory architectures.

All these and others tentatives to parallelize the SystemC simulation engine are limited by the SystemC centralized scheduler, and the centralised representation of time associated to the event-driven simulation paradigm. In TLM simulation, this simulation paradigm causes numerous context switches between the *SC_THREADs* describing the various hardware components of the simulated MP2SoC. To reduce the penalties associated to the context switches, the Loosely-Timed TLM (TLMLT)[**?**] approach introduces the interesting notion of quantum keeper that allows a simulated processor to run ahead of the global simulation clock but even in this case, the simulator is able to run only one *SC_THREAD* representing one hardware activity at a time.

We believe that the parallelization of SystemC implies a change of perspective. To remove the central scheduler bottleneck, one must accept to shift to a new, distributed, simulation paradigm, where each *SC_THREAD* is handling its own private local clock, and the various *SC_THREADs* synchronize with each other, according to the PDES (Parallel Discrete Event Simulation) principles.

III. TLM-DT approach

A TLM[**?**] model is generally a collection of communicating *SC_THREADs* describing hardware components of the simulated architecture. These *SC_THREADs* are good candidates to be executed in parallel on multi-core workstations. The main difficulty for the parallel simulation is that the OSCI SystemC simulation kernel defines a central scheduler which contains a list of time-ordered events and a global variable representing the simulation time. This central scheduler is clearly the bottleneck for parallel simulation.

The TLM-DT approach implements the PDES principles[**?**], 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 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 be used for a shared memory MP2SoC, that is a strongly coupled parallel system. 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.

The proposed TLM-DT simulation models are fully compliant with the TLM2.0 standard[**?**]. The models use generic payload and phase, the initiator and target sockets, and the non blocking transport functions defined by the TLM2.0 standard. However shifting from global time to distributed time introduces some major 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[**?**] associate to the event-driven algorithm. In TLM-DT, the synchronization between timed processes is distributed by annotating all messages with timing information. Each *SC_THREAD* has an absolute local time and sends it as the third argument of the transport interface methods, as suggested by the TLM2.0. This absolute local time must be increased during simulation. It can still be interpreted as an offset relative to the SystemC global time because this global variable is not used, and never incremented during execution. A TLM-DT model uses only two basic synchronization primitives (*wait(event)* & *event.notify()*), and it can be simulated using the standard SystemC kernel and the standard TLM2.0 package, provided by the OSCI consortium.

A. Components Modeling

For a shared memory MPSoC as shown in Figure 1(b), the TLM-DT model contains three types of hardware components: initiator, target and interconnect. For each hardware component, the corresponding TLM-DT simulation model contains one *SC_THREAD* and a local *sc_time* member variable.

A **TLM-DT initiator** runs freely until it reaches a synchronization point (corresponding to a read or write transaction) or when it has consumed a predefined time quantum. In case of an explicit data transaction, the corresponding *SC_THREAD* is blocked (descheduled), waiting for response. When the response is received, the local time of the initiator is updated, and the *SC_THREAD* is resumed. Whenever the local time is updated, the TLM-DT 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 descheduled.

A **TLM-DT target** is reactive, i.e. the *SC_THREAD* remains sleeping until it receives a read or write transaction. 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.

A **TLM-DT interconnect** is modeled as two fully independent crossbars for requests and responses respectively. The request crossbar 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 decoded, and the transaction is routed to the proper target). The request crossbar implements a centralized data structure (called PDES buffer) that contains a slot for each input channel. When all slots contain at least one request, a time filtering is performed to select the request with the smallest timestamp, and the selected transaction is transmitted to the proper target. In order to avoid dead-locks, the contention is not handled in the response crossbar, and there is neither *SC_THREAD* nor time filtering, but only a routing function to route the response to the proper initiator. In hierarchical interconnects, the local and the global interconnect have the same structure and behavior.

When simulated with the standard SystemC simulation engine, a TLM-DT model combines the advantages of both the loosely-timed style (simulation speed), and the approximately-timed (high accuracy) proposed by TLM2.0. 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.

IV. SystemC-SMP : Parallel Simulation of TLM-DT

SystemC-SMP is an implementation of a new simulation kernel optimized to take advantage of the TLM-DT modeling approach for parallel simulation on SMP (multiprocessor) workstations. SystemC-SMP is dedicated to the TLM-DT coding style and does not require any modification in the simulation models that can still be simulated with the standard (sequential) SystemC[**?**], [**?**] simulation kernel and the standard TLM2.0 package.

From the simulation kernel viewpoint, a TLM-DT platform can be seem as a set of communicating *SC_THREADs* (simulation tasks) that use *sc_event* objects to synchronize themselves. A TLM-DT platform uses only three synchronization primitives that must be implemented by the SystemC-SMP kernel:

- *wait(sc_event e)* : The calling sc_thread is blocked, and enters a waiting state on event *e*
- *e.notify(SC_ZERO_TIME)* : notifies the event *e*
- *wait(SC_ZERO_TIME)* : The calling sc_thread is blocked but its state is not modified

A. Architecture and Implementation

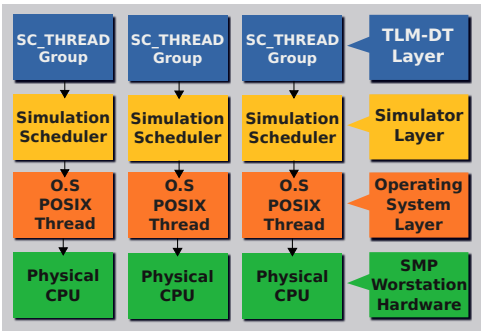

Fig. 2. SystemC-SMP Architecture

SystemC-SMP uses a gang-scheduling [**?**][**?**] approach by scheduling related neighboring *SC_THREADs* on the same physical CPU of the multi-core workstation.

To control the placement of the *SC_THREADs* on the CPUs, SystemC-SMP kernel creates one Operating System POSIX thread (PT)[**?**] per CPU and attach each PT to a given CPU using affinity functions from PT user interface. A statically defined set of *SC_THREADs* is controlled by each POSIX thread. There is actually one local scheduler for each set (group) of *SC_THREADs* (i.e. one scheduler per CPU of the SMP workstation). The Figure 2 shows this scenario.

Statically defined placement of the *SC_THREADs* is used in order to exploit the cache locality [**?**]. The communication graph is fully determined by the MP2SoC hardware architecture and can be statically analysed by the system designer (see dashed rectangles in Figure 1(a,b)). For example, each *SC_THREAD* associated with a hardware component within a cluster can be scheduled in turn on the same CPU, as shown in Figure 1(c). The *SC_THREAD* mapping is explicitly controlled by the system designer through a configuration file.

For synchronization SystemC-SMP re-implements the standard SystemC event class (*sc_event*). In this new implementation an event has only one state variable with two possible values (*WAITING* and *NOTIFIED*). This state variable is used by the local scheduler to evaluate the next task to execute. With this simple synchronization approach the simulator kernel does not use any blocking mechanism such as spinlocks, mutex or barriers.

V. Experimental Results

This section analyses the influence of the communications locality on the parallel simulation speedup. The experiment can be described by the triplet: Embedded Software Application (**ESA**), SoC Hardware Architecture (**SHA**), and SMP Workstation (**SMP**). The **ESA** is a synthetic application that we can adjust the ratio between local (intra-cluster) and global (inter-cluster) communications. The **SHA** is a shared-memory NUMA (Non Uniform Memory Architecture) clustered architecture with a 2D mesh topology. It contains 40 clusters, and each cluster contains 16 initiators, 16 memory banks, and a local interconnect. This SHA totalizes 1321 *SC_THREADs*. The **SMP** machine on which the simulation takes place is an Intel Xeon Quad Core Processor 3GHz with 512K L1 cache and 2MB L2 cache performing on Linux 2.6.18 Operating System. The simulations use up to four CPUs of the workstation.

A. Experimental setup

Each initiator corresponds to a synthetic traffic generator, which can create local and global transactions. In this experiment the percentage of local transactions varies from 0% up to 100%. A traffic with a high percentage of local transactions represents a good mapping of ESA on the SHA, and the inverse represents a bad one. Regarding the mapping of SHA onto SMP, all hardware components that belong to the same cluster are mapped in the same CPU of the SMP. The simulations were done using 1 up to 4 CPUs. The 40 clusters were equally spread on the CPUs to balance the load.

B. Results

Figure 3 shows the influence of traffic locality on the speedup. When the local transactions represent more than 80%, the speedup is almost optimal. However, the speedup decreases when the percentage of local transactions is below 50%, because the global interconnect becomes a contention point for the synchronization of the simulation. Additionally, the initiator and the target of a local transaction being mapped in the same CPU, we benefit from SMP workstation cache locality.

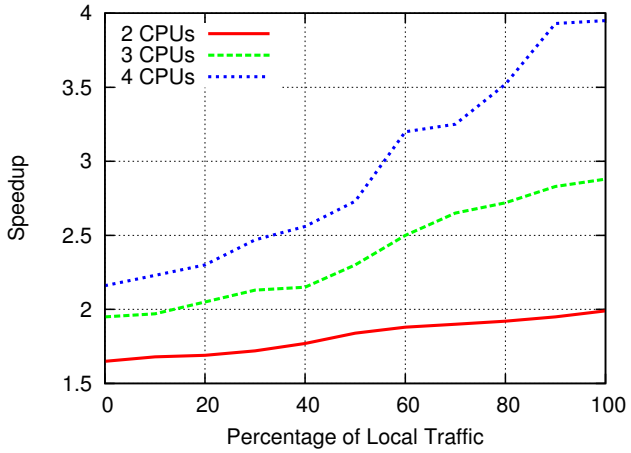

Fig. 3. Local Traffic x Speedup Chart

VI. Conclusion

The results show that the performance of a parallel simulation of a TLM-DT platform is given by the local/external traffic amount between components of different clusters.

In this paper, we presented a modeling approach for timed TLM virtual prototyping of shared memory MP2SoCs. These TLM-DT models implement the Parallel Discrete Event Simulation principles, that is well suited for parallel simulation on multi-cores workstations because it does not use the central SystemC scheduler and the centralised time representation. We developed a parallel simulation kernel called SystemC-SMP optimized for these TLM-DT models, and we shown that a quasi-linear speedup can be obtained on a quad-core workstation, for a shared memory clustered MP2SoC architecture (containing 640 processors) as long as the communication pattern of the embedded software application has a locality larger than 80%. All TLM-DT models and the SystemC-SMP engine will be available as open-source software in [**?**].

Experiments using real applications are being developed to evaluate the performance of both TLM-DT and SystemC-SMP.

References

[1] Y. Bouzouzou. *Semantics-Preserving Parallelization of the SystemC Scheduler for Reduced Simulation Times*. PhD thesis, Universite Joseph Fourier de Grenoble, 2007.

[2] 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.

[3] P. C. Chopard and J. Zory. A conservative approach to systemc parallelization. *Springer Berlin / Heidelberg*, 3994:653–660, 2006.

[4] P. Combes, E. Caron, F. Desprez, B. Chopard, and J. Zory. Relaxing synchronization in a parallel systemc kernel. In *ISPA Symposium Proceedings*, pages 180–187, 2008.

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

[6] D. R. Cox. Ritsim: Distributed systemc simulation. Master's thesis, Rochester Institute of Technology, 2005.

[7] D. G. Feitelson and L. Rudolph. Gang scheduling performance benefits for fine-grain synchronization. *Journal of Parallel and Distributed Computing*, 16:306–318, 1992.

[8] IEEE. *1003.1c-1995: Information Technology — Portable Operating System Interface (POSIX) - System Application Program Interface (API) Amendment 2: Threads Extension (C Language)*. IEEE Computer Society Press, 1995.

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

[10] OSCI. TLM-2.0 User Manual. http://www.systemc.org.

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

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

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