Profiling High Level Abstraction Simulators of Multiprocessor Systems

Liana Duenha, Rodolfo Azevedo
Computer Systems Laboratory (LSC)
Institute of Computing, University of Campinas - UNICAMP
Campinas - SP, Brazil
lianaduenha@lsc.ic.unicamp.br, rodolfo@ic.unicamp.br

***Abstract*—Simulation has become one of the most time-consuming tasks in Electronic System Level design, required both on design and verification phases. As the complexity of modelled systems increases, so do the need for adequate use of available computational resources in multiprocessor computers or clusters. SystemC simulator models are designed to use only one core, even if the hardware is multi-core. In this paper, we analyse 20 platforms, designed in SystemC, varying from 1 to 16 cores with 4 different processor models (ISAs), and evaluated the SystemC kernel overhead for a set of 12 programs running over those platforms, totaling 240 configurations. We split the execution time into the simulation components and found out that the major contributor to the simulation is the SystemC kernel, consuming around 50% of the total simulator execution time. This finding opens space for new research focusing on improving SystemC Kernel performance.**

I. Introduction

The complexity of *Multiprocessor System-on-Chip* (MPSoC) designs and the required simulation performance to take advantage of design space exploration decisions in the reduced time-to-market window, forces designers into an even higher level design methodology, looking for faster tools to accomplish their tasks. One such design alternative is to replicate the full system in software to validate its functional behavior, to evaluate its performance, or to explore architecture design options [12].

A virtual platform is a fully functional model of a complete system, containing higher level models for each component, to mimic every piece of hardware and software of the target device. To construct virtual platforms, we focus on technologies available under open-source licenses, such as SystemC [1], OSCI TLM [4], and ArchC [3].

System Level Description Languages (SLDLs) provide a collection of libraries of data types, kernel simulation, and components for high level system modelling and simulation. *SystemC* is a SLDL with modelling environment based on C++ used for modelling and verification of systems at different abstraction levels. Recently, SystemC has become the leading choice of designers of SoCs (System-on-Chip) and embedded processors [7], [5].

A limiting factor in accelerating the simulation of systems modelled in SystemC is that the SystemC kernel is sequential and based on discrete event simulation. Consequently, the simulator executes a process at a time, even if the hardware supports execution of concurrent processes [7], [8]. For example, a virtual platform with 16 processors running in a real multicore architecture uses only one core to execute the simulation, scheduling the processors and other peripherals to execute one at a time.

Currently, research at reducing the simulation time are focused in distributed SystemC simulation, which consists of distributing the simulation in multiple cores, and the best performance improvement happens when the model design is partitioned manually [7], [10], [14], [6]. These techniques do not modify the SystemC scheduler.

The main goal of this paper is to profile CPU usage, during SystemC simulations, in order to guide future performance improvements efforts. We found out that, by splitting the execution time into parts, the SystemC kernel CPU usage approaches 50% of total platform simulation time. We evaluate 20 different platforms, varying from 1 to 16 processor cores of 4 distinct ISAs, and 12 different programs capable of running in all platforms. By showing how the CPU is used during simulations, we expect to contribute to future improvements for simulation speed research.

This paper is organized as follows. Section II discusses related work; Section III provides definitions and describes the tools used in the experiments and the developed components; Section IV briefly presents the ESLDiagram tool, and the experimental results are shown in Section V; finally, we show a brief analysis of the results in Section VI and conclude this paper in Section VII.

II. Related Work

In the past few years, the main focus on speeding up SystemC simulations were focused on distributing the simulations over the cores in a multicore computer, or over several computer clusters.

Maia, Greiner and Pecheux[13] proposed to accelerate SystemC simulators changing the SystemC kernel in order to improve the Transaction Level Models (TLM) performance. Based on this work, Mello et al. [14] defined a new engine of parallel and distributed simulation capable of exploiting the computational power of multiprocessor workstations. The new kernel, named *SystemC-SMP*, is dedicated to the DT-TLM model (TLM with distributed time). To use it, the platforms must be changed, which limit SystemC-SMP usage. Moreover,

considering the simulation of a system with multiple cores and some peripherals, the CPU time used by the processors is significantly longer than the CPU usage by TLM peripherals.

In the proposal by Chopard et al. [6], a copy of the scheduler will execute on each processing node and simulate a subset of application modules. The consistency of the modified data on a node is guaranteed through events between processes. Later, Huang et al. [10] proposed a technique to increase the performance of the SystemC simulations consisting of the geographical simulation distribution based on the same concepts described in [6]. To use these techniques, the designer needs to decide how to distribute the peripherals over the available processing units.

Ezudheen et al. [7] proposed a new scheduler to support the parallel execution of SystemC process. The amount of processes is linearly reduced with number of available cores. The best performance gain happened when using the manual partitioning technique. Through experiments, they concluded that the simulation with the new scheduler obtained advantages when compared to serial simulation and can contribute in complex projects; however, it is a considerable limitation to rely on manual partitioning to get the the biggest gain.

Schumacher et al. [16] implemented parallel simulation through changes in the SystemC simulation kernel, in order to accelerate the simulation of multiprocessor systems. It is known that the synchronous approach introduces a high overhead and requires centralized communication. The authors reduced the communication latency and synchronization by using an adaptive scheduling technique. To correctly handle atomicity of shared resources data, all related process code should be adapted.

III. Infrastructure

In this paper, we use virtual platforms modelled in SystemC and using Transaction Level Models (TLM) [12]. TLM allows the benefit of reducing model development time, and leads to a large improvement on simulation speed and modelling productivity, enabling new design methodologies. Although TLM is language independent, SystemC fits perfectly its representation style by allowing adequate abstraction levels and by providing elements for isolating computation and communication. *ArchC* is a *Architecture Description Language* (ADL) following a SystemC syntax style, which provides enough information in order to allow users to explore and verify a (new or legacy) processor's architecture by automatically generating not only software tools for code generation and inspection (like assemblers, linkers, and debuggers), but also executable processor models for platform representation [15], [3]. In our experiments, we use ArchC processor models and other peripherals developed with TLM methodology.

Many processors include hardware performance monitoring, which take the form of one or more counters, incremented each time an event occurs. The so called hardware performance counters allows very precise profiling tools like OProfile [2]. OProfile is a system wide profile for Linux, capable of profiling all running code at low overhead using the hardware performance counters. It is composed by a kernel driver, a daemon for collecting sample data, and several post-profiling tools for manipulating the results into information. We used Oprofile tool in our experiments to generate the statistics about the CPU usage, distinguishing between the CPU usage of the SystemC kernel and of our components.

A. Platforms

This paper used a set of platforms ranging from 1 to 16 cores in a shared memory environment. Each platform contains: a set of processors (1 to 16), a shared memory, a hardware lock device to enable the construction of atomic operations, and an interconnection structure used to connect all platform components, called router. Figure 5 shows an example of a platform containing 16 processors.

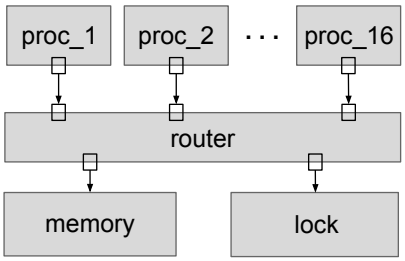
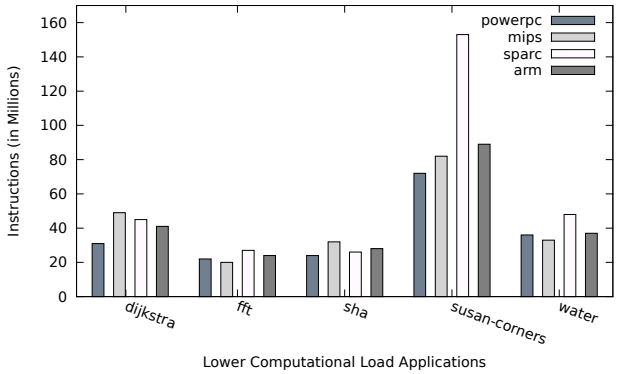


Fig. 1. Schematic of the 16-core platform example containing a router, one shared memory and the lock peripheral.

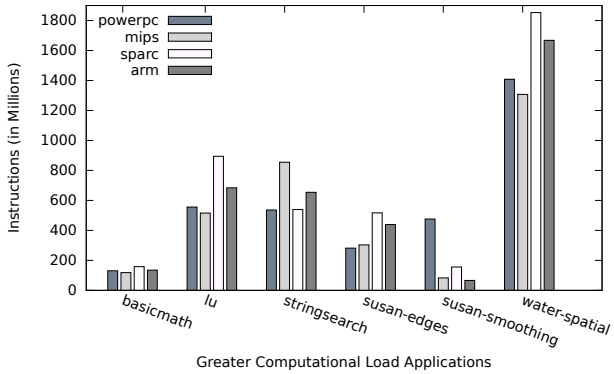
All processors instantiated in each platform share an external 512 MB memory that is connected by a TLM channel. We also include a hardware lock device to allow the execution of parallel applications since our processors and platforms do not support atomic operations required to create software locks. The platforms use a communication device called router to interconnect all components. The router is a very simple module, which essentially consists of TLM ports to connect it to the memory and lock devices, followed by communication interfaces with each instantiated processor.

Our platforms may have one of the following processors: PowerPC, SPARC, MIPS, and ARM. All processors are modelled on ArchC and the communication between processor and external devices uses TLM ports. Al processors have 32-bit wordsize. **PowerPC** model implements the PowerPC 32 bits instruction set, including ABI emulation; **MIPS** is a MIPS-I ISA description, including delay slots, and ABI emulation; **SPARC** model implements the V8 version of the SPARC architecture, including delay slots, window registers and ABI emulation; the **ARM** model implements the ARMv5e instruction set, including ABI simulation. Table I list a few more details of each processor.

For each processor, five different base platforms were designed, varying the number of cores in each one (1, 2, 4, 8, and 16 cores). As we have four different available processor models, we have 20 different platform configurations. Considering that a platform is not directly related to the application, and we have 12 different multicore applications, our experiments use a



(a) Applications with lower computational load



(b) Applications with greater computational load

Fig. 2. Number of instructions executed for each application. To make it easy to visualize the results, we divided the applications into two categories, one with lower computational load (lower total instruction count) and the other with higher computational load (higher total instruction count).

TABLE I
PROCESSOR ARCHITECTURAL CHARACTERISTICS

Processor	Register Bank	Orhers Registers	ISA	Endian
PowerPC	32	13	181	big
MIPS	32	3	59	big
SPARC	32	256	118	big
ARM	31	20	100	little

universe of 240 different configurations of parallel software on multicore platforms. Still, it is possible to perform 16 single-core application of Mibench benchmark suite simultaneously and independently in a 16-core environment, adding extra 8 configurations.

B. Applications

All applications were compiled for each of the four already mentioned processor models using an appropriate cross-compiler infrastructure. The application source code were adapted from their original benchmark suites:

- **ParMiBench [11]:** Is a benchmark suite composed by parallel versions of the traditional MiBench benchmark. We used the following applications: sha, dikjstra, susan-corners, stringsearch, susan-smoothing, susan-edges, basicmath;
- **Splash-2 [17]:** Is a benchmark suite composed by highly configurable parallel applications. We used the following applications: fft, lu, water, water-spatial;
- **MiBench [9]**: Is a benchmark suite composed of several sequential applications divided into categories. We used the following applications: bitcount, susan-corners, susan-edges, susan-smoothing, basicmath, dijkstra, qsort, stringsearch, sha, rijndael-enc, rijndael-dec, fft, blowfish-enc, blowfish-dec, adpcm-enc, adpcm-dec. We used the single-core versions and adapted them run simultaneously on a 16-core platform.

We chose some of the possible configurations to characterize the applications according to their computational load.

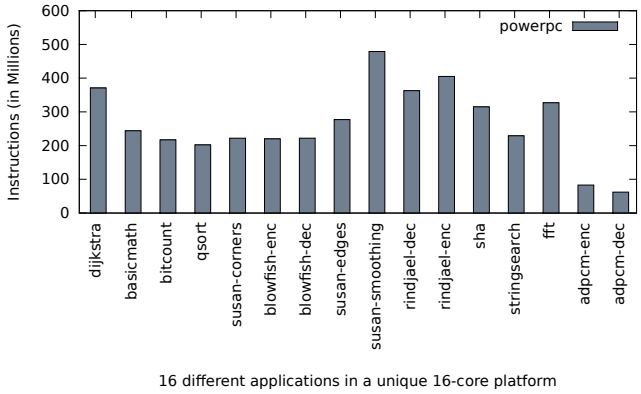


Fig. 3. Number of instructions executed in the 16-core platform running different and independent single-core applications.

Figures 2(a) and 2(b) show the amount of instructions executed on a 16-core platform running parallel applications with lower computational load (low total number of instructions) and higher computational load (higher total number of instructions), respectively. Figure 3 shows the amount of instructions executed by 16 different applications running on a 16-core PowerPC platform.

IV. PLATFORM DIAGRAM

Any attempt to exploit SystemC simulators requires knowledge about the data structures created for model description. To facilitate this task and to make easy to visualize platform diagram, we designed a runtime tool to be called during the elaboration phase of a platform to automatically create a diagram of the platform. The ESLDiagram has access to the SystemC simulation kernel data structures and collects the data needed to generate the diagram, such as the name of instantiated modules, classes they belong to and the connections between them. To use this new tool, it is only necessary to update the `sc_main()` to include a call to ESLDiagram, as outlined in Figure 4, immediately before calling `sc_start()`. It is not

necessary to modify any SystemC module. After executing the platform, ESLDiagram generates a PDF file containing the representative diagram of all created modules and their connections. An example of this output is shown in Figure 5, where a dual-core PowerPC processor is connected to a router which is connected to peripherals memory and a lock.

```
1  sc_simcontext* my_sim = sc_get_curr_simcontext();
2  ESLDiagram dotDiagram (my_sim);
3  dotDiagram.startCapture();
```

Fig. 4. Code that must be inserted after elaboration phase for generating diagram with ESLDiagram tool.

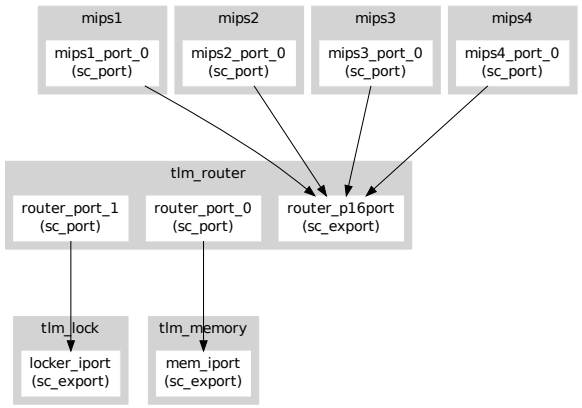


Fig. 5. A example of a 4-cores PowerPC platform, including all SystemC components.

V. Experimental Results

We started performing simulations on our set of platforms using the PowerPC processor, and used the OProfile tool to characterize the CPU usage by the various components in the platform and by the simulation kernel. Figure 6 compares the CPU usage by the simulation kernel in simulators with 1 and 16 processors, running 11 different applications. As we can see, the kernel overhead is near 50% with a small variability when increasing the number of components in most of the platforms.

Any conclusion on the CPU usage by the simulation kernel should consider that the platform with only one processor contains, in total, four modules (processor, memory, router, and lock); in contrast, a simulation of a platform with 16 processors contains 19 modules (16 processors, memory, router, and lock). For example, 53% of the simulation corresponds to the simulation kernel on a platform of 16 processors running `basicmath`; this means that the other 19 modules used the 47% of the CPU. In a single core version of this platform the kernel CPU usage was 62%, while the CPU usage of the others four modules was 38%. To demonstrate that this

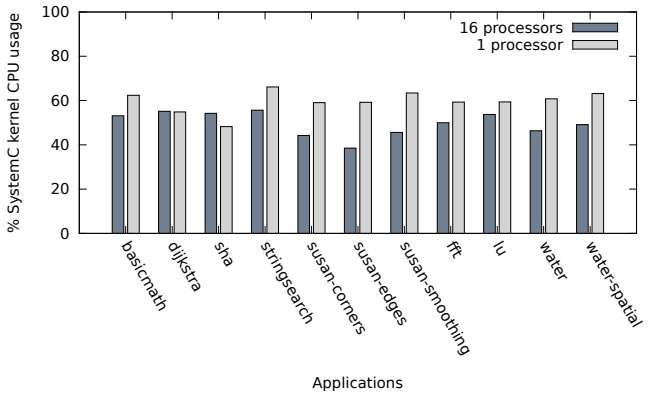


Fig. 6. SystemC kernel CPU usage over 16-cores and single-core PowerPC platforms.

behavior holds for the other applications, we summarized the results in Figure 7 that shows the average CPU use for all applications in the 16 cores platform. **SystemC kernel** refers to the simulation kernel CPU usage; **Processor** refers to the processor behavior CPU usage; **TLM peripherals** refers to the use of CPU by TLM peripherals (`router`, `memory`, and `lock`); finally, **Others** refers mainly to the use of operating system kernel (basically for input and output file access).

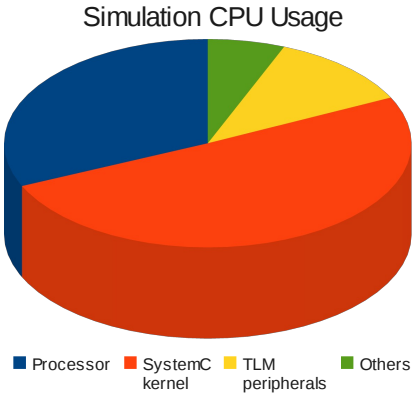


Fig. 7. Average percentage of CPU usage for all applications on a 16-core platform.

Next, we evaluated the four processor models using all available applications. Figure 8 shows a stacked bar graph detailing the average CPU usage. As with the PowerPC graphs, the SystemC kernel uses around 50% of all processing resources during the simulations. The other 30% are consumed by the processor model behavior, where we can notice the difference in implementation complexity and optimization among all four models.

The SystemC simulation is non-preemptive, i.e., a SystemC

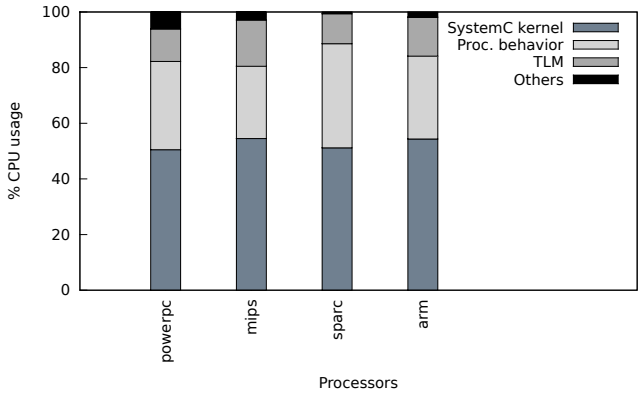


Fig. 8. Average percentage of CPU usage for all applications on 16-core platforms, using powerpc, mips, sparc, and arm processors.

thread runs until finished or until it makes a `wait()` function call; when a `wait()` is called, there will be a context switch between SystemC threads and it requires the state of the simulation threads to be properly stored for later retrieval. Since the SystemC kernel is sequential, it is necessary that the ArchC processor model calls `wait()` after the execution of each instruction, or a small amount of them. This leads to a large number of context switches between SystemC threads.

In order to gauge the impact of large amount of context switches, we counted the number of dynamic calls to `wait()` in our simulations, varying the number of processors. We divide ten applications of ParMiBench [11] and SPLASH-2 [17] benchmarks in two groups, according to the increase in number of instructions performed as the number of processors increases. In **Group 1** we include applications that do not have a significant variation in the amount of instructions performed as the amount of processors increased; we put in **Group 2** the applications that vary significantly the computational load as the amount of processors. This increase is caused by scalability of applications (week or strong scaling) or by the significant increase in communication between processors. The experimental results are available in Figures 9(a) and 9(b).

The applications in Figure 9(a) do not increase the work size when divided into more cores. This makes the total processing time almost stable with the same amount of SystemC context switches. Since we executed a version of ArchC that calls one wait for every instruction, this graph also represents the total number of instructions executed. The slight increase in the number of instructions, as the number of cores is increased, represents the extra overhead to divide the data working set into more cores.

In Figure 9(b), all application threads have a fixed amount of work to be done. In this case, when we increase the number of cores, there is an increase in the number of wait calls and thus in the total number of instructions.

To demonstrate that the simulation performance decreases when we increase the number of processors in multicore platforms, we also measured the execution time of simulation platforms 1, 2, 4, 8 and 16 processors. Figures 10(a) and 10(b) show the results. We notice that the simulation time increases for both application groups, although the increase for the weak scaling group (Group 2) is higher.

VI. Analysis

The experimental results show that the simulation kernel takes about 50% of simulation time, on platforms varying from 1 to 16 cores, using four different models of processors, and this fact exposes the SystemC kernel as the major bottleneck that causes delays in the simulation and the best opportunities to improve simulation performance. The large number of `wait` function call in the simulations, required to simulate multiprocessor systems in a sequential simulation kernel, shows that decreasing the number of context exchange among SystemC processes can result into considerable improvement, without requiring changes in the design.

One approach to optimize the simulation speed is to reduce the number of `wait` calls through distribution of the simulation modules into the real processor while simulating platforms. Each SystemC thread is mapped as an operating system thread and can proceed independently of the others until a explicit synchronization point but the SystemC kernel restricts this behavior by serializing them. A optimistic scheme of synchronization between threads in a discrete event-based simulation will be able to obtain great improve of performance. This is possible in the version 2.3 of SystemC, which has a temporal decoupling mechanism using the TLM 2.0 standard.

Another approach is to optimize the scheduling algorithm. Originally, the SystemC scheduler has a queue of ready process and selects one of these process to execute in a single-core environment. In order to exploit temporal and spatial locality and to not degrade the performance of the cache memory system, we will implement and verify different scheduling policies.

VII. Conclusion

This paper has presented a representative set of experiments on virtual platform simulators modelled in SystemC. The experimental results show that the simulation kernel takes about 50% of simulation time, on simulations of multiprocessor systems, given the large number of context switches in the simulation; hence, we have provided an effective set of evidence that the effort to speed-up SystemC simulators through time reduction of scheduling and the CPU use by the kernel can obtain good improvements.

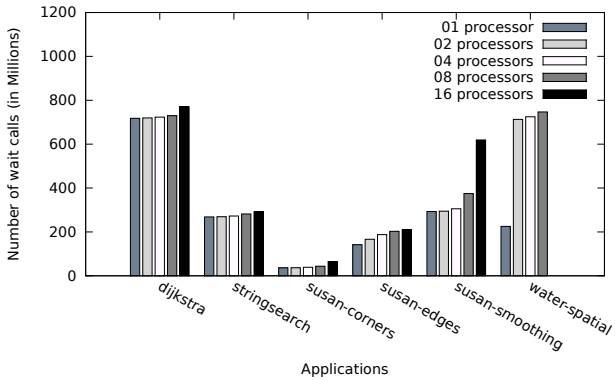
Acknowledgement

We would like to thank the reviewers for their comments and suggestions on the original manuscript. This work was partially funded by CNPq and CAPES.

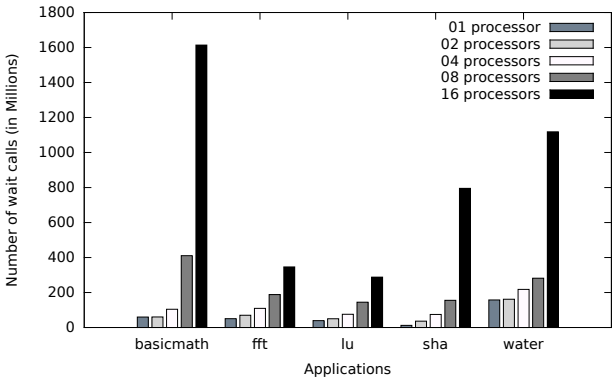
References

[1] IEEE 1666^{TM} Standard SystemC Language Reference Manual. http://standards.ieee.org.

[2] OProfile: A System Profiler for Linux. http://oprofile.sourceforge.net/news/.

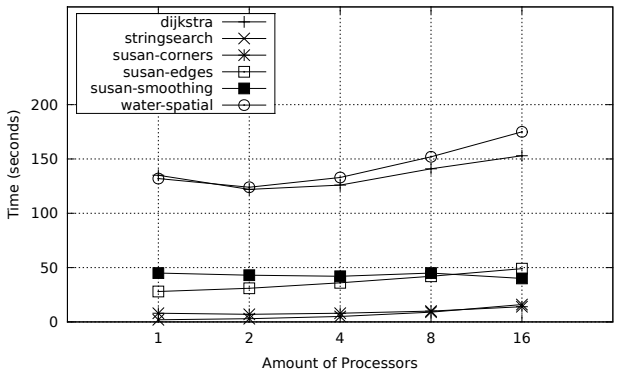


(a) Applications from Group 1, where the computational load is not increased with the number of processor.

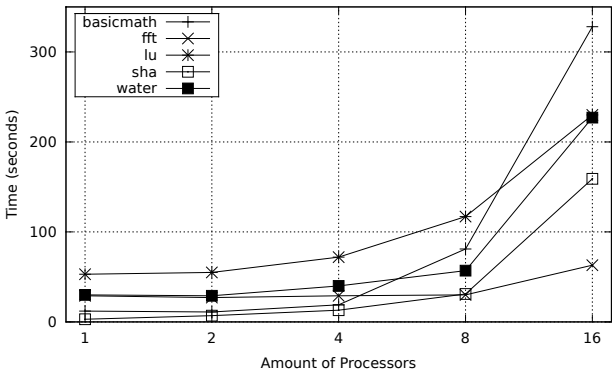


(b) Applications from Group 2, where the computational load is increased with the number of processor.

Fig. 9. Number of `wait()` calls executed by core's SystemC processes on multiprocessor platforms with 1, 2, 4, 8, and 16 cores.



(a) Simulation time of applications from Group 1.



(b) Simulation time of applications from Group 2.

Fig. 10. Simulation time on multiprocessor platforms with 1, 2, 4, 8, and 16 cores.

[3] The ArchC Architecture Description Language. http://www.archc.org/.

[4] TLM Transaction Level Modeling Library. http://www.systemc.org.

[5] BLACK, D., AND DONOVAN, J. *SystemC: From the Ground Up*. Kluwer Academic Publishers, 2004.

[6] CHOPARD, B., COMBES, P., AND ZORY, J. A Conservative Approach to SystemC Parallelization. In *Proceedings of the Workshop on Scientific Computing in Electronics Engineering*. May 2006, pp. 653–660.

[7] EZUDHEEN, P., CHANDRAN, P., CHANDRA, J., SIMON, B., AND RAVI, D. Parallelizing SystemC Kernel for Fast Hardware Simulation on SMP Machines. In *23rd Workshop on Principles of Advanced and Distributed Simulation*. July 2009, pp. 80–87.

[8] GALIANO, V., MIGALLON, H., D.PEREZ-CAPARROS, AND MARTINEZ, M. Distributing SystemC Structures in Parallel Simulations. In *Proceedings of SpringSim*. 2009.

[9] GUTHAUS, M. R., RINGENBERG, J. S., ERNST, D., AUSTIN, T. M., MUDGE, T., AND BROWN, R. B. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of IEEE 4th Annual Workshop on Workload Characterization, held in conjunction with The 34th Annual IEEE/ACM*. December 2001, pp. 03–14.

[10] HUANG, K., BACIVAROV, I., HUGELSHOFER, F., AND THIELE, L. Scalably Distributed SystemC Simulation for Embedded Applications. In *Proceedings of the International Symposium on Industrial Embedded System (SIES 2008)*. 2008, pp. 271–274.

[11] IQBAL, S. M. Z., LIANG, Y., AND GRAHN, H. ParMiBench: An Open-Source Benchmark of Embedded Multiprocessor Systems. In *IEEE Computer Architecture Letters, Vol. 9, No.2*. 2010, pp. 45–48.

[12] LEUPERS, R., AND TEMAM, O. *Processor and System-on-Chip Simulation*. Springer New York, 2010.

[13] MAIA, I., GREINER, A., AND PECHEUX, F. SystemC SMP: A parallel approach to speed up Timed TLM simulation. In *SoC-SIP-System on Chip-System in Package*. 2006.

[14] MELLO, A., AIND A. GREINER, I. M., AND PECHEUX, F. Parallel Simulation of SystemC TLM 2.0 Compliant MPSoC on SMP Workstations. In *Proceedings of Design, Automation and Test in Europe - DATE*. March 2010, pp. 606–609.

[15] RIGO, S., AZEVEDO, R., AND SANTOS, L. *Eletronic System Level Design*. Springer, 2011.

[16] SCHUMACHER, C., LEUPERS, R., D.PETRAS, AND HOFFMANN, A. parSC: Synchronous Parallel SystemC Simulation on Multi-Core Host Architectures. In *Proceedings of Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. October 2010, pp. 241–246.

[17] WOO, S. C., OHARA, M., TORRIE, E., SINGH, J. P., AND GUPTA, A. The Splash-2 Programs: Characterization and Methodological Considerations. In *Proceedings of the 22nd International Symposium on Computer Architecture - ISCA'95*. ACM, 1995, pp. 24–36.