DOM: A Data-dependency-Oriented Modeling Approach for Efficient Simulation of OS Preemptive Scheduling

Peng-Chih Wang, Meng-Huan Wu, and Ren-Song Tsay
Department of Computer Science
National Tsing Hua University
HsinChu, Taiwan
{pengchih_wang, mhwu, rstsay}@cs.nthu.edu.tw

***Abstract*—Operating system (OS) models are widely used to alleviate the overwhelmed complexity of running system-level simulation of software applications on specific OS implementation. Nevertheless, current OS modeling approaches are unable to maintain both simulation speed and accuracy when dealing with preemptive scheduling. This paper proposes a Data-dependency-Oriented Modeling (DOM) approach. By guaranteeing the order of shared variable accesses, accurate simulation results are obtained. Meanwhile, the simulation effort of our approach is considerably less than that of the conventional Cycle-Accurate (CA) modeling approach, thereby leading to high simulation speed, 42 to 223 million instructions per second (MIPS) or 114 times faster, than CA modeling as supported by our experimental results.**

Keywords-OS modeling; preemptive scheduling; simulation

I. INTRODUCTION

Conventionally, instruction-set simulators (ISS) are widely adopted to run specific OSs for software simulation, but the achievable simulation speed is only few million instructions per second (MIPS). For efficient system-level simulation, the concept of OS modeling has been proposed recently. The idea is to abstract routine OS operations into an efficient model and simulate only the application part at the fine-gained level.

However, current OS modeling approaches are unable to maintain both simulation speed and accuracy. In a modern OS system, the preemptive scheduler dynamically alters the interleaving between software tasks. Different interleaving sequences may introduce different execution results. Hence, it is critical to simulate the effect of preemptive scheduling correctly when validating software.

To model detailed preemption behavior correctly, mostly the *cycle-accurate* (CA) modeling approach is adopted. At each cycle, the CA approach swaps between simulated tasks and the OS model. Nevertheless in reality, the overheads caused by frequent swaps severely degrade the simulation performance.

In contrast, coarse-grained approaches swap at the boundaries of software program *basic blocks* or *functions*. Although these coarse-grained approaches greatly reduce swapping overheads, the preemption behavior cannot be simulated at the correct time point. As a result, the delayed handling of preemption can lead to an inaccurate interleaving sequence, and the simulation results may be false-negative, in which error is not detected as a result of incorrect simulation.

To deal with this issue, this paper proposes a data-dependency-oriented modeling (DOM) approach. From our observations, different interleaving sequences between software tasks may produce different results. Nonetheless, as long as the interleaving sequences have the same shared variable access order, i.e., the same data-dependency, their execution results will be identical. Therefore, to speed up simulation performance, the basic idea of DOM is to minimize the frequency of task swapping while guaranteeing the data-dependency between simulated software tasks rather than a fixed task interleaving sequence as in the CA modeling approach.

Since the number of shared variable accesses is considerably smaller than the number of simulated cycles, DOM allows faster simulation due to fewer swapping overheads. In addition to being fast, the proposed approach also offers same accurate simulation results as CA modeling approaches do.

The experimental results show that our approach can generate not only results of the same accuracy as those from CA approaches but also have a simulation speed of at least 42 MIPS, which is 22 times faster than the CA modeling approach and 3 times faster than the basic-block-level modeling approach.

The remainder of this paper is organized as follows. Section II describes related work. The proposed data-dependency-oriented modeling approach is elaborated in section III. Section IV explains how to implement DOM OS models in SystemC. Section V demonstrates our experimental results. Finally, a conclusion is given in section VI.

II. RELATED WORK

The Instruction-set simulator (ISS) technique [1-3] has been widely used for software simulation. Although it can accurately simulate the preemptive scheduling of target OS, its poor performance (only a few MIPS) is not practical for full system simulation.

Although some dynamic compiled ISS techniques allow high software simulation performance (approximately a hundred MIPS), such as QEMU [4], which uses a binary translation approach, their implementation is complex and difficult to integrate into HW/SW co-simulation frameworks such as SystemC.

978-3-9810801-7-9/DATE11/©2011 EDAA

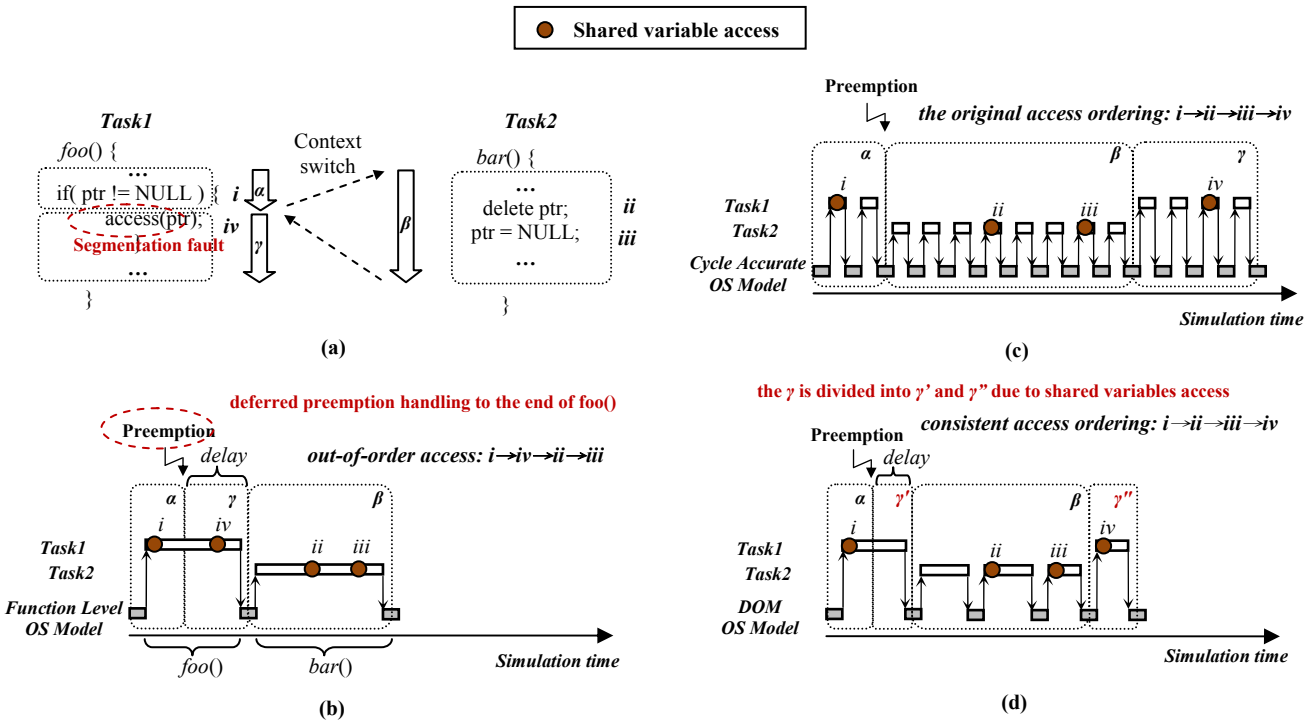

Figure 1: (a) An illustration of an interleaving sequence which results in program segmentation fault; (b) The interleaving and access order of shared variables following the function-level modeling of OS preemptive scheduling; (c) The interleaving and access order of shared variables following the cycle-accurate modeling of OS preemptive scheduling; (d) The interleaving and access order of shared variables following the proposed data-dependency oriented modeling of OS preemptive scheduling.

To improve the performance of integration in a co-simulation framework, abstract OS modeling approaches [5-10] are proposed. However, as far as we know, current high-speed OS modeling approaches cannot accurately simulate the effect of preemptive scheduling and may produce inaccurate software execution results.

To illustrate the issue, an example is shown in Figure 1(a), in which a particular interleaving sequence, $\alpha \rightarrow \beta \rightarrow \gamma$, leads to a program segmentation fault. Ideally, after checking the null pointer, *Task1* is supposed to successfully access the non-null pointer, but the OS scheduler unintentionally preempts *Task1* right after the checking and switches to *Task2*. Subsequently, *Task2* deletes the given pointer, so that when *Task1* resumes its execution, the invalid pointer access will crash the program. Undoubtedly, designers would like to reveal this type of error while validating software applications with an OS model.

To achieve high-speed simulation, the swapping overheads must be reduced. An intuitive overhead reduction idea is to limit task swapping at the boundaries of functions or basic blocks of simulated software tasks. However, since preemption may occur at any time point, the effect of preemption cannot be captured accurately with proper ordering mechanisms.

Based on a function-level OS model [9] that swaps executions at function boundaries, Figure 1(b) shows the simulation result of the case in Figure 1(a). The preemption handling is deferred until the end of the function *foo*(), thereby leading to different task interleaving, $\alpha\gamma \rightarrow \beta$. In this case, the program execution no longer crashes due to this incorrect simulation sequence, and designers will miss this bug, making it a false negative result. Similarly, a basic-block-level OS model [8] may incur the same issue.

Instead, a cycle-accurate modeling approach [7] that swaps between simulated tasks and the OS model at each cycle, as shown in Figure 1(c), can accurately produce correct results due to the same interleaving sequence, $\alpha \rightarrow \beta \rightarrow \gamma$, as Figure 1(a). However, as discussed before, the CA modeling approach would have massive swapping overheads and hence poor simulation performance.

III. Data-Dependency-Oriented Modeling (DOM)

As discussed before, the efficient function-level OS model leads to inaccurate simulation result and the accurate cycle-accurate OS model results in massive simulation overhead. In order to be both efficient and accurate in simulating the preemption behaviors, we introduce a data-dependency-oriented (DOM) approach.

Based on our observations, although task interleaving sequences can differ due to varying preemption event points, execution results are actually determined by their shared variable access order, i.e., data-dependency. In fact, as long as the interleaving sequences have the same data-dependency, their execution results are identical.

Previous studies [11-13] conclude that consistent data-dependency can be guaranteed if the execution order of shared variable accesses is maintained. The authors of [11-12] show that software applications on multi-core system can be simulated accurately and efficiently by effectively maintaining the

data-dependency. Based on the data-dependency concept, the authors of [13] further provide an efficient and accurate coarse-grained-level software application simulation approach.

In order to maintain the same data-dependency of the original execution, the proposed DOM swaps between the OS model and simulated tasks at each shared variable access point. The swapping of executions maintains data integrity influenced by the task interleaving sequence. In fact, only shared variable accesses can affect data integrity. For better understanding, we use the example in Figure 1 to demonstrate how the DOM approach works.

As shown in Figure 1(b), when applying function-level modeling, the delay of preemption handling incurs the accesses to the shared variables, *ptr*, i.e., *i→iv→ii→iii*.

However, the cycle-by-cycle execution diagram of the two interleaving tasks is shown in Figure 1(c), in which the accesses to the shared variables, *ptr*, are marked as *i, ii, iii,* and *iv* according to their accessing order. Hence, the data-dependency ordering is violated in function-level modeling since the out-of-order access to *ptr*.

In contrast as illustrated in Figure 1(d), DOM can maintain the same access ordering, i.e., *i→ii→iii→iv* as the CA execution by synchronizing (or swapping) at the shared variable access points. Although the delayed preemption handling in DOM still alters the interleaving between *Task1* and *Task2* from $\alpha\rightarrow\beta\rightarrow\gamma$ to $\alpha\gamma'\rightarrow\beta\rightarrow\gamma''$, the out-of-order portions β and γ' have no shared variable access. Therefore, their execution order does not affect the execution result, and an accurate simulation is obtained.

In summary, the proposed DOM not only offers the same simulation results as CA modeling but also largely reduces swapping overheads. Hence, it is capable of a fast and accurate simulation of the preemption effect.

IV. Implementation of DOM OS Model in SystemC

This section explains how the proposed DOM OS model can be incorporated into SystemC [14]. Although SystemC is an IEEE standard and has been widely applied to early-stage system simulation, it does not explicitly support task preemption [15]. In the following, we give details on how to employ the proposed OS model for preemptive scheduling in SystemC.

A. Simulation Diagram

The SystemC engine adopts a concurrent simulation strategy: the executable HW/SW tasks on SystemC are executed concurrently. Nevertheless, in the target single-core system, the software tasks are in fact sequentially executed.

The proposed simulation flow incorporating the DOM OS model is illustrated in Figure 2. In this simulation flow, the proposed DOM OS model manages the execution order of executable software tasks while the SystemC engine still manages the scheduling of hardware tasks.

When a software task is executable, the SystemC engine invokes our DOM OS model. Afterwards, DOM will first compute the time to next shared variable access by calling the *wait()* function, which will execute operations of the task and submit control back to SystemC for hardware simulation.

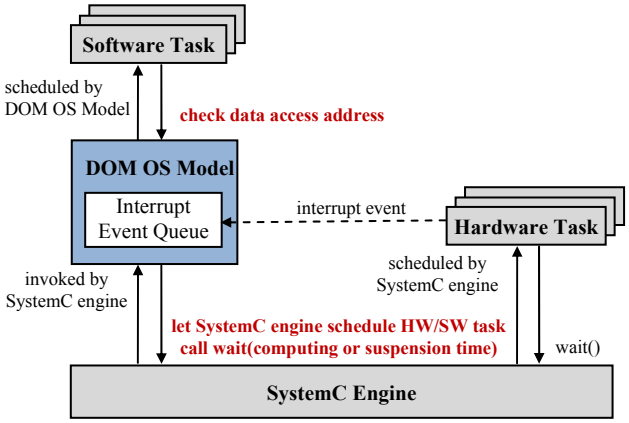

Figure 2. The simulation flow incorporating DOM OS model in SystemC.

In reality, the executing software can be preempted due to interrupts. For example, when a hardware task needs to interact with a specific software task, it will send an interrupt to notify the processor. The processor will suspend the current software task to invoke the corresponding interrupt service routine (ISR). Afterwards, if necessary, the OS scheduler performs context switch to execute the specific software task. Otherwise, it just resumes the suspended software task.

In SystemC, an interrupt is modeled as a SystemC event. When an interrupt event is issued by a hardware task, our OS model will record it in an *Interrupt Event Queue*. Afterwards, the DOM OS model will check the Interrupt Event Queue and consider the preemption effect when it is invoked.

When encountering a preemption event, the DOM OS model will request the SystemC engine by calling wait function to mimic the time delay for the preempted software task and trigger the preempting software task to execute. Otherwise, it will just resume execution of the current software task.

In order to mimic the time delay (or suspension time) of the preempted task, the DOM OS model has to perform timing adjustment, which is elaborated in the following section.

B. Timing Adjustment for Time-Sharing Effect

Because the SystemC engine is mainly designed to simulate the concurrent behavior of hardware tasks, the timing behavior of different tasks can appear to overlap in the perspective of simulated time. To illustrate the behavior, Figure 3 demonstrates the simulation results following the example in Figure 1(a). Here, software *Task1* starts first and then software *Task2* is launched at t_2. Following the concurrent behavior, the SystemC engine simulates their overlapped executions in Figure 3(a).

However, in a time-sharing multi-tasking OS system, software tasks are cooperatively performed one at a time. The execution of the two software tasks in the target system should be as depicted in Figure 3(b). Therefore, when using SystemC to simulate the timing behavior of software tasks, the tasks will be confused by the overlapped simulated time.

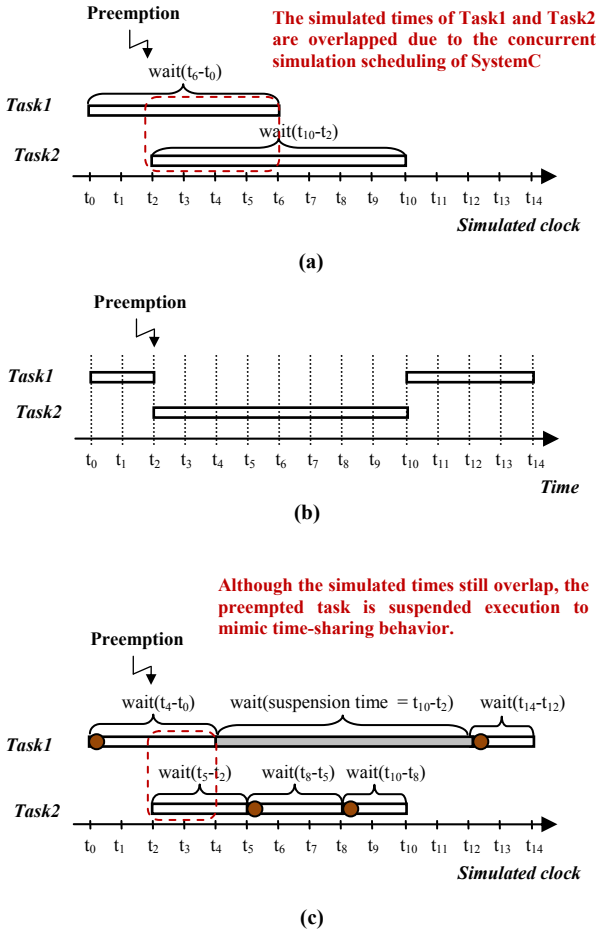

Figure 3: (a) The concurrent simulation result from the aspect of simulated clock; (b) The true time-sharing behavior of the target system; (c) The time-sharing behavior from the concurrent simulation by adopting timing adjustment mechanism of DOM OS model.

To resolve the issue, our OS model first ensures correct time-sharing behavior and then adopts the timing adjustment mechanism [9-10] by adjusting the simulated task end time to be the same as that in the target system.

To simulate the time-sharing behavior, the DOM OS model lets the preempted software tasks wait further the suspension time as shown in Figure 3(c). Note that this figure is depicted from the aspect of simulated clock which indicates the simulated time of the target. In contrast, simulation time implies the time to simulate on the host.

Figure 4 illustrates in detail how the mechanism works by checking the Interrupt Event Queue. If there is a preemption event, the preempted software task will wait the suspension time which is the computing time of preempting software task execution. Otherwise, the executing software task will resume execution at the next shared variable access. The checking of preemption will be recursively executed until there is no interrupt in Interrupt Event Queue.

The purpose of having suspension time is to mimic the time-sharing behavior by forcing the preempted software task

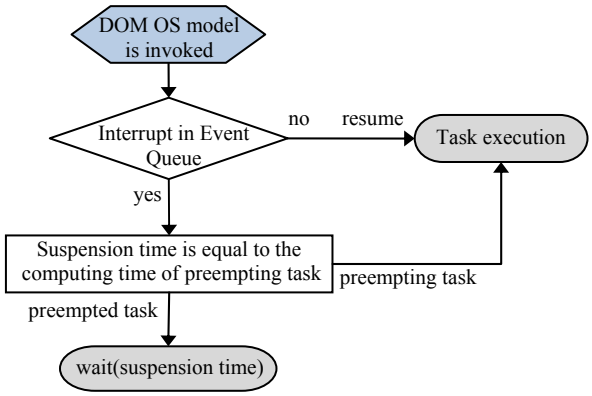

Figure 4. The timing adjustment flow of software tasks in DOM OS model.

to suspend program execution. During suspension time, the preempted task does nothing and consumes only time, depicted as the shaded segment in Figure 3(c). Although tasks still overlap with the execution from t_2 to t_4 due to the SystemC engine, the end time of simulated clock will be the same as time-sharing with the timing adjustment.

C. Identifying Shared Variable Access

This section discusses how shared variables are identified by shared allocation routine in our implementation. The purpose of the identification is to mimic the data allocation mechanism of the OS. In addition to the preemptive scheduling, our OS model also provides the required shared allocation routine for software task implementation. A software task can use the routine to allocate shared data once the allocation function is called. Afterwards, the address of the shared data is then recorded for simulation use.

As shown in Figure 5, when a simulated software task issues a memory access, we check whether the target address is of shared data. If it is, the DOM OS model will compute the time from last shared variable access and annotate the time to the *wait()* function to invoke the SystemC engine for simulation.

In this way, the DOM OS model can guarantee the data-dependency by maintaining the temporal order of shared variable accesses.

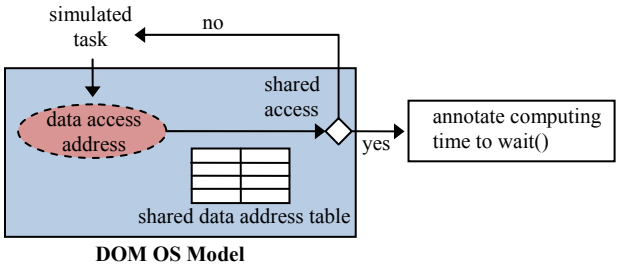

Figure 5. Identifying shared variable access during run-time in a DOM OS model.

V. EXPERIMENTAL RESULTS

To verify the proposed DOM OS model, we performed an experiment to demonstrate the simulation speed and another one to confirm the simulation accuracy. The proposed OS model is implemented and integrated into the SystemC kernel.

During simulation, the DOM model is invoked to process preemption events at shared variable access points. We followed the timing annotation ideas proposed in [16-18] and dynamically annotate the computing time of each task.

A. Simulation Speed

Table I summarizes the simulation speed benchmark results for a few OS modeling approaches using SPLASH-2 parallel programs [19]. The *barnes, lu, ocean, fft*, *fmm* and *radix* benchmarks are implemented into 2, 4 and 8 tasks. Each task is executed in turn when the target processor is periodically notified by a timer interrupt.

TABLE I. THE SPLASH-2 EXPERIMENTAL RESULTS

		Simulation Speed (MIPS)		
Bench Programs	**Shared Access Rate**	**CA**	**BB**	**DOM**
barnes_2	7.46%	2.1	16.8	53.0
barnes_4	7.77%	2.0	15.9	48.9
barnes_8	8.36%	1.9	15.4	42.1
lu_2	3.14%	2.2	18.3	73.5
lu_4	3.16%	2.0	17.6	71.5
lu_8	3.26%	1.9	16.2	66.6
ocean_2	1.81%	2.1	18.4	97.5
ocean_4	1.82%	2.0	17.1	95.8
ocean_8	3.36%	1.8	15.8	60.2
fft_2	0.93%	2.1	18.2	140.3
fft_4	1.16%	2.0	17.6	130.2
fft_8	1.12%	1.9	16.4	128.2
fmm_2	0.71%	2.1	18.4	170.4
fmm_4	0.47%	2.0	17.1	179.4
fmm_8	0.36%	1.9	16.4	182.4
radix_2	0.08%	2.2	18.6	223.1
radix_4	0.10%	2.1	17.5	217.1
radix_8	0.09%	1.9	16.7	221.9

Among all cases, the simulation speed of CA OS model is consistently at around 2 MIPS. As for the basic-block-level (BB) OS model, in average each basic block contains 3 to 6 instructions [20] and hence the simulation speed is improved to be around 18 MIPS.

In contrast, the DOM OS model swaps between the OS model and executing tasks only at each shared variable access, and hence the simulation speed is further raised up to the range of 42 to 223 MIPS.

The variation of the simulation speed actually depends on the density of shared variable access. Simulation speed is lower for cases of higher shared variable access rate. Particularly, the radix example has only few shared variable accesses, so its simulation on the DOM OS model is faster than other benchmarks.

Note that Table I does not include results from the function-level OS model, since the execution of the SPLAH-2 programs cannot terminate on it. This is because a dead-lock situation occurs if there is an execution loop within a function call and the termination condition relies on a shared variable to be set by other tasks, since no other task can be activated if the current task does not terminate but instead waits for another task to provide the termination value. Therefore, the function-level OS model is not suitable for preemptive OS modeling purposes.

Furthermore, to show that the shared variable access rate greatly influences the simulation performance, we adopt the *producer-consumer* program of ADPCM encoder and decoder from MiBench [21]. The ADPCM is a signal encoding program. In the design, there is a FIFO shared by both the encoder and decoder tasks. The encoder produces data into the FIFO and then the decoder consumes the FIFO data. Upon a half-filled FIFO, a controller sends an interrupt to trigger the encoder task to work. The application has a very high shared variable access rate, but the simulation speed based on the DOM OS model can still achieve 40.1 MIPS, compared to 9.6 MIPS on the BB model and 1.7 MIPS on the CA OS model.

B. Simulation Accuracy

To verify the simulation accuracy, we further adopt the *WatchDog Timer* benchmark and compare the final simulation results from the BB OS model and the DOM OS model with that of the CA OS model. The *WatchDog Timer* is a widely used for deadlock detection. The principle operation of the *WatchDog* benchmark is to periodically test a shared register (i.e., WTCNT) value. If the register value is tested to be zero, the multi-tasking OS system determines that it is in a deadlock and will reset the system.

To demonstrate the accuracy of the DOM OS model, we implement the WatchDog, which consists of two tasks. One task counts down the value of the shared register one by one and then checks the value at the end of each specified timeout period. The other task assigns a non-zero value to the register when it is triggered by a timer interrupt. The result is erroneous if the simulated value of the register is different from that of the CA OS model. The error rate is computed by dividing the number of erroneous results by the total number of register value checks. We then vary the timeout period and control the frequency of timer interrupt to create different tests. As shown in Figure 6, the proposed DOM OS model is always 100% accurate, matching the CA OS model, while the error rate of the BB OS model ranges from 56% to 66% as the time-out period becomes shorter.

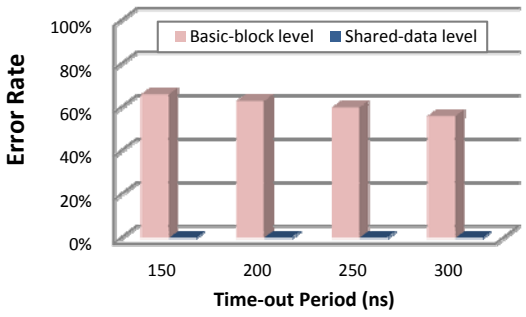

Figure 6: The simulation accuracy of the WatchDog Timer with different OS modeling approaches.

VI. CONLUSION

In this paper, we have presented and demonstrated the DOM approach for the simulation of OS preemptive scheduling. By maintaining the data-dependency between the software tasks, we can accurately simulate the preemption effect. Moreover, the proposed OS model is implemented to enable preemptive scheduling in SystemC. The experimental results show that the preemptive scheduling simulation using our modeling approach can perform 114 times faster than the CA OS model while maintaining the same accuracy.

ACKNOWLEDGMENT

This work was supported by National Science Council (Grant No. NSC99-2221-E007-114-MY3) and the specification of Andes ISA was provided by Andes Technology.

REFERENCES

[1] M. Rosenblum, S. Herrod, E. Witchel, and A. Gupta, "Complete Computer System Simulation: The SimOS Approach", *IEEE Parallel and Distributed Technology: Systems and Applications*, 3(4): pp. 34–43, 1995.

[2] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, B. Werner, "Simics: A Full System Simulation Platform", *IEEE Computer*, 35(2): pp. 50-58, 2002.

[3] T. Austin, E. Larson, D. Ernst, "SimpleScalar: An Infrastructure for Computer System Modeling", *IEEE Computer*, 35(2): pp. 59-67, 2002.

[4] F. Bellard, "QEMU, a Fast and Portable Dynamic Translator", In *ATEC: Proceedings of the annual conference on USENIX Annual Technical Conference*, pp. 41-41, 2005.

[5] A. Gerstlauer, H. Yu, D. Gajski, "RTOS Modeling for System Level Design", In *DATE: Proceedings of the conference on Design, Automation and Test in Europe*, pp. 130-135, 2003.

[6] R. Moigne, O. Pasquier, and J. Calvez, "A Generic RTOS Model for Real-time Systems Simulation with SystemC", In *DATE: Proceedings of the conference on Design, Automation and Test in Europe*, pp. 82-87, 2004.

[7] I. Bacivarov, S. Yoo, and A. Jerraya, "Timed HW-SW Cosimulation Using Native Execution of OS and Application SW", In *HLDVT: Proceedings of the Workshop on High Level Design Validation and Test*, pp. 51-56, 2002.

[8] S. Yoo, G. Nicolescu, L. Gauthier, A. Jerraya, "Automatic Generation of Fast Timed Simulation Models for Operating Systems in SoC Design", In *DATE: Proceedings of the conference on Design, Automation and Test in Europe*, pp. 620-627, 2002.

[9] G. Schirner, R. Dömer, "Introducing Preemptive Scheduling in Abstract RTOS Models using Result Oriented Modeling", In *DATE: Proceedings of the conference on Design, Automation and Test in Europe*, pp. 122-127, 2008.

[10] Y. Yi, D. Kim, S. Ha, "Virtual Synchronization Technique with OS Modeling for Fast and Time-accurate Cosimulation", In *CODES+ISSS: Proceedings of the conference on Hardware/Software Codesign and System Synthesis*, pp. 1-6, 2003.

[11] M. Wu, C. Fu, P. Wang, and R. Tsay, "An Effective Synchronization Approach for Fast and Accurate Multi-core Instruction-set Simulation", In *EMSOFT: Proceedings of the conference on Embedded Software*, pp. 12–16, 2009.

[12] J. Chen, M. Annavaram, and M. Dubois, "Exploiting Simulation Slack to Improve Parallel Simulation Speed", In *ICPP: Proceedings of the international conference on Parallel Processing*, pp. 371-378, 2009.

[13] M. Wu, W. Lee, C. Chuang, and Ren-Song Tsay, "Automatic Generation of Software TLM in Multiple Abstraction Layers for Efficient HW/SW Co-simulation", In *DATE*: *Proceedings of the conference on Design, Automation and Test in Europe*, pp. 1177-1182, 2010.

[14] SystemC [online]. Available: http://www.systemc.org/home/.

[15] T. Grötker, "Modeling Software with SystemC 3.0", Synopsys Inc, *European SystemC Users Group Presentations*, 2002.

[16] Y. Hwang, S. Abdi, D. Gajski, "Cycle-approximate Retargetable Performance Estimation at the Transaction Level", In *DATE: Proceedings of the conference on Design, Automation and Test in Europe*, pp. 3-8, 2008.

[17] J. Schnerr, O. Bringmann, A. Viehl, W. Rosenstiel, "High-Performance Timing Simulation of Embedded Software", In *DAC: Proceedings of the conference on Design Automation Conference*, pp. 290-295, 2008.

[18] K. Lin, C. Lo, R. Tsay, "Source-level Timing Annotation for Fast and Accurate TLM Computation Model Generation", In *ASP-DAC: Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 235-240, 2010.

[19] S. Woo, M. Ohara, E. Torrie, J. Singh, A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations", In *ISCA: Proceedings of the international symposium on Computer architecture*, pp. 24-36, 1995.

[20] J. Hennessy and D. Patterson, *Computer Architecture: a quantitative approach*, 4th ed., 2007.

[21] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite", In *WWC-4: Proceedings of the Workshop on Workload Characterization*, pp.3-14, 2001.