Fast and Time-Accurate Cosimulation with OS Scheduler Modeling

YOUNGMIN YI ymyi@iris.snu.ac.kr
School of Computer Science and Engineering, Seoul National University, Shilim-dong, San 56-1, Kwanak-gu Seoul, Korea

DOHYUNG KIM dbkim@iris.snu.ac.kr
School of Computer Science and Engineering, Seoul National University, Shilim-dong, San 56-1, Kwanak-gu Seoul, Korea

SOONHOI HA sha@iris.snu.ac.kr
School of Computer Science and Engineering, Seoul National University, Shilim-dong, San 56-1, Kwanak-gu Seoul, Korea

Abstract. Hardware/Software cosimulation is the key process to shorten the design turn around time. We have proposed a novel technique, called virtual synchronization, for fast and time accurate cosimulation that involves component simulators running concurrently and interacting with each other. In this paper, we further extend the virtual synchronization technique with OS modeling for the case where multiple software tasks are executed under the supervision of a real-time operating system. The OS modeler models the RTOS overheads of context switching and tick interrupt handling as well as preemption behavior. While maintaining the timing accuracy to an acceptable level below a few percents, we could reduce the simulation time drastically compared with existent conservative approaches by removing the need of time synchronization between simulators. It is confirmed with a preliminary experiment on a multimedia example that consists of four real-life tasks.

Keywords: cosimulation, virtual synchronization, OS modeling.

1. Introduction

A complex embedded system includes software modules and hardware modules that are mapped to heterogeneous components such as microprocessor, DSP, ASIC, FPGA and so on. Software modules are taking more and more roles in embedded systems and it is crucial to enable software development concurrently with the hardware development to reduce the design time. This can be achieved by hardware/software cosimulation in the hardware/software codesign methodology that uses a virtual prototype.

Cosimulation can be used to evaluate several candidate architectures in the architecture selection stage of which the goal is to find out an optimal architecture for a given application. A popular cosimulation technique in this stage is to use host code execution with delay annotation. A task is not cross-compiled to target code but compiled and executed as host code and its timing information is estimated a priori and annotated. Data exchange between processing components is modeled and performed at the transaction level. Since static timing estimation cannot take into account any run-time variation of performance, timing accuracy of cosimulation is
limited. Recently, there have appeared many research results ([3], [4], [7], [16]) and commercial tools ([17], [18]) that are considering this level of abstraction.

Cosimulation can also be used in the verification of designed system where timing accuracy is the key requirement. In this stage, time-accurate instruction set simulator (ISS) and RTL simulator are usually used for software component and for hardware component, respectively [1], [17], [18]. As a result, cosimulation involves multiple component simulators running concurrently and interacting with each other. It is known that time-accurate cosimulation is much slower than delay-annotated transaction level cosimulation by some orders of magnitude [20]. Since a time-accurate cosimulation may also be used for the last stage of design space exploration such as fine-tuning of communication architecture and memory systems, it is desirable to speed up the time-accurate cosimulation speed, which is the main concern of this paper.

There are two main causes for low performance of time accurate cosimulation: one is slow simulation speed of each component simulator, and the other is time synchronization overhead between component simulators. There have been several researches to speed up time-accurate cosimulation. Compiled ISS [2] was proposed to boost the simulation speed of software simulator. It achieves high simulation speed by removing the overhead of instruction decoding at run time. Instead, it translates each target instruction directly to one or more host instructions at compile time. But it has a drawback that it lacks of adaptability to the modification of the architecture or the compiler. Also, it is difficult to apply this approach to the dynamic program such as OS model. Moreover, time synchronization overhead remains between component simulators.

Recently, we have proposed the virtual synchronization technique for time-accurate cosimulation to reduce the simulation time and the time synchronization overhead simultaneously [8]. The virtual synchronization technique eliminates the need for time synchronization between component simulators at all: synchronization appears accomplished only when events are exchanged. Also it improves the performance of an individual simulator by removing the overhead of redundant local clock advancement.

However, there is a critical constraint on the simulated tasks to apply the virtual synchronization technique. The task execution model that virtual synchronization assumes is that the execution results of the task do not depend on the arrival times of input events but only on the arrival order of them. Once a task is executed with a given set of input samples, it assumes completed without interruption.

In this paper, we extend the virtual synchronization technique to the case where multiple software tasks are executed under the supervision of a real-time operating system (RTOS) in a processor. Virtual synchronization cannot be directly applied in this case since an RTOS does not satisfy the execution model that virtual synchronization assumes: if the input samples for a task with higher priority arrives at the processor, the RTOS preempts the current task execution in the middle. On the other hand, it is a big burden to run the RTOS itself on the processor ISS. The proposed approach runs only application tasks on the ISS and models the RTOS in the cosimulation backplane to achieve faster cosimulation still preserving timing accuracy to an acceptable level below a few percents.