SPRI: Simulator Partitioning Research Infrastructure

Zhuo Ruan Koy Rehme David A. Penry
Department of Electrical and Computer Engineering
Brigham Young University
Provo, UT, 84602

Abstract—**Using FPGAs as architectural simulation accelerators has been widely discussed in the computer architecture design community. We previously proposed a hybrid SW/HW simulation infrastructure named SPRI (Simulator Partitioning Research Infrastructure) which automatically partitions the general timing model into the software and hardware portions for simulation speedup, conforming to the set-based partitioning specification. The SPRI platform takes two main inputs—partitioning specification and the architectural model; it then produces a modified SW architectural binary and a HW-accelerated RTL description which can communicate with each other, called hybrid SW/HW co-simulator—the final output of SPRI. Various experiment cases have been also run through the SPRI infrastructure to test its partitioning functionality and API wrapper generation.**

I. INTRODUCTION

Today's simulators are too slow to simulate many-core chips. Reconfigurable hardware acceleration is believed to be a flexible way to help achieve fast and cycle-accurate simulation results. Previous efforts have required massive investments of time and effort in both RTL and simulator design and have produced “one-off” solutions. That is, when simulating different target architectures, the partitioning decision are made specific to the different simulator and the hardware has to be manually redesigned in hardware description language. Lots of previous work has been done to explore the hardware acceleration potential [1], [2]. However, none of them involve an automatic procedure of partitioning a target architectural model and generating its respective hybrid SW/HW co-simulator. Thus, the purpose of proposing SPRI infrastructure in WARP’2007, is to try to solve this problem. A more detailed description of the SPRI platform is presented as follows.

SPRI has three main blocks—partitioner, API generator and SystemC-to-VHDL synthesizer. Its block diagram is shown in Figure 1.

First, the partitioner takes the target timing model and builds the simulation database which contains the structural information of the target model; it partitions out from the simulation database those units that need to go to the hardware according to the input partitioning specification. Then the partitioner modifies the LLVM intermediate representation (IR) of the target model and make sure the replacement portion on the SW side can successfully talk with the HW portion.

Second, the API generator is in charge of creating the SW and HW wrappers. The SW wrapper is the set of functions which encapsulate the actual host-to-device APIs and thereby hides them from the user and the simulation codes. The HW wrapper is a template-based VHDL program that specifies the controlling logic and the signal connection between the outer communication channel and the internal components. The SW and HW wrapper are designed to act as such communication channel.

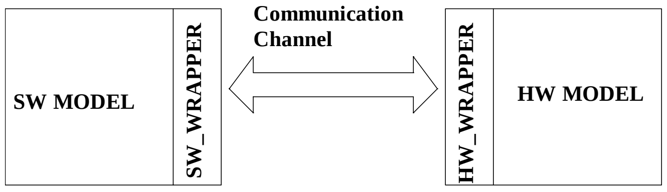

Figure 2. the SW/HW Cosimulator

Third, SPRI utilizes the Trident synthesizer as our SystemC-to-VHDL tool [7]. Because it uses the same LLVM intermediate representation (IR) also adopted by the partitioner, the trident tool can directly synthesize the result of the partitioner and the API generator.

The result of this process, as in Figure 2, is a modified software model with the necessary API wrappers to communicate with an equivalent hardware model placed in an FPGA. The API generator and the partitioner ensure that both parts will communicate with each other and keep data coherent on the partitioning boundary

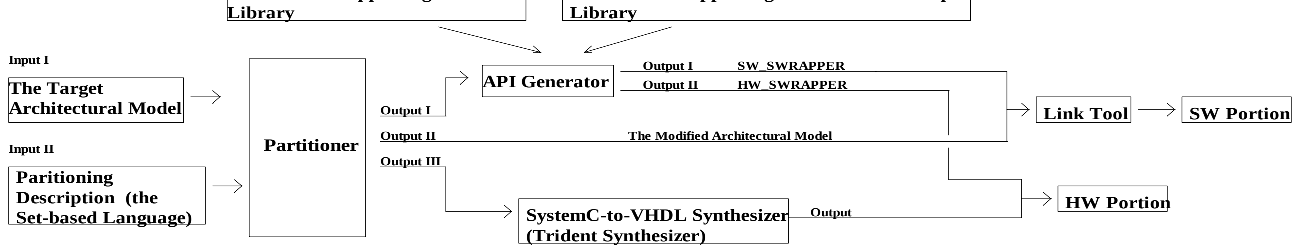
Multi-board-supporting Device API Library
Multi-board-supporting Device VHDL Template Library
Input I
The Target Architectural Model
Input II
Paritioning Description (the Set-based Language)
Partitioner
Output I
Output II
Output III
API Generator
Output I SW_SWRAPPER
Output II HW_SWRAPPER
The Modified Architectural Model
SystemC-to-VHDL Synthesizer (Trident Synthesizer)
Output
Link Tool
SW Portion
HW Portion

Figure 1. SPRI Block Diagram

II. SPRI IMPLEMENTATION STATUS

The SPRI infrastructure is currently under construction. The status on the development of these three SPRI blocks—partitioner, API generator and synthesizer are discussed below.

The partitioner is capable of reading the partitioning specification and extracting the structural information from the simulation database for the API generator. The partitioner is also capable of revising the software model and interfacing with the API generator. The hardware portion has been separated and we are currently in the process of interfacing these portions with the Trident synthesizer. It is planned that the partitioner will support different-level partitioning tasks including instance level, process level, function level and data-layout level, even though it only supports instance-level and process-level partitioning presently. For instance, if inputting such partitioning description as (1),

"tohw [DLX::ALU] | [DLX::IFID_Register]" (1)

the ALU and IFID_register instances of the DLX processor model will be synthesized into hardware and the respective SW and HW wrappers will be automatically generated. Also, SPRI can send only the rising-edge process of the IFID_Register instance together with the ALU into hardware when using (2).

"tohw [DLX::ALU] | [DLX::IFID_Register::RiseProcess %func%]" (2)

The second part—API generator, produces the SW and HW wrappers correctly. We brought in the concept "communication group" to simplify the data transfer control between the host and the accelerator. Each architectural unit in the HW has its own communication group which gathers the data that needs transferring between the SW and HW portions. The cross-boundary communication can be completed by simply matching the communication group number and its buffer space on both the SW and HW sides. Actually, the SW/HW data transfer job is done automatically, since both the SW buffer and the HW buffer will update themselves and stay coherent with each other when the data values on the partitioning boundary are changed. We are planning to build up multiple libraries for the API generator. They will provide different actual device communication APIs and various HW wrapper templates in order to support different simulation platforms, including BEE2, DRC 1000 system and other DRC systems.

Lastly, we are in the process of updating the Trident synthesizer's LLVM front-end and interfacing with the other parts of SPRI. With such change, the input of the synthesizer will be switched to the LLVM IR of the partitioned-out units instead of the original C/C++ program.

Since SPRI is designed for researching the simulator partitioning and different partitioning decision results in different SW and HW wrappers, three test cases have been performed on the simple DLX architectural model using the DRC Development System 1000:

1) No inter-component communication;

2) Inter-component communication;

3) Clock-triggered components.

Generally, the SPRI infrastructure has already obtained the basic capability to partition a general architectural model with various input partitioning specifications and also automatically generate a hybrid SW/HW co-simulator.

We are considering adding performance evaluator or counter into the infrastructure. This will provide the evaluation of a specific partitioning. With this support, SPRI could be extended to automatically test different partitionings, reconfigure the FPGA platform and evaluate the effectiveness of the resulting simulator at each trial. The goal is to find out the most efficient partitioning for a given simulator and thereby largely reduce the architects' workload on the procedure of repeatedly partitioning, simulating and evaluating.

III. CONCLUSION

It is hoped that the completed SPRI infrastructure can help the community evaluate different partitioning decisions and accelerate architectural simulation without requiring heroic simulator design. In the long term, SPRI has a potential to be applied in other fields such as embedded system design, general application acceleration, and self-updating FPGA reconfiguration by partitioning the target application and being integrated with automatic software and hardware co-scheduling.

REFERENCES

[1] D. A. Penry, D. Fay, D. Hodgdon, R.Wells, G. Schelle, D. I.August, and D. Connors. Exploiting parallelism and structure to accelerate the simulation of chip multi-processors. In Proceedings of the 12th International Symposium on High-Performance Computer Architecture, pages 29–40, February, 2006.

[2] Derek Chiou, Dam Sunwoo, Joonsoo Kim, Nikhil A. Patil, et el. FPGA-Accelerated Simulation Technologies (FAST): Fast, Full-System, Cycle-Accurate Simulators. In Proceedings of the 40th IEEE/ACM Symposium on Micro-architecture, pages 249-261, Dec., 2007.

[3] DRC Computer. http://www.drccomputer.com/

[4] J. D. Davis, L. Hammond, and K. Olukotun. A flexible architecturefor simulation and testing (FAST) multiprocessor systems. In Proceedings of the 1stWorkshop on Architecture Research using FPGA Platforms, 2005.

[5] N. Dave, M. Pellauer, Arvind, and J. Emer. Implementing a Functional/Timing Partitioned Microprocessor Simulator with an FPGA. In Proceedings of the Workshop on Architecture Research using FPGA Platforms, held at HPCA-12, Feb. 2006.

[6] IEEE Std 1666-2005: IEEE Standard SystemC Language Reference Manual. IEEE, 2005.

[7] Tripp J.L., Peterson K.D., Ahrens C., et el. Trident: an FPGA compiler framework for floating-point algorithms. In Proceedings of the International Conference on Field Programmable Logic and Application, pages 317-322, Aug. 2005.