An Infrastructure for HW/SW Partitioning and Synthesis of Architectural Simulators

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

Abstract

Many researchers are interested in using FPGAs to accelerate architectural simulation. Partitioning of the simulator between hardware and software is an important problem which has not been explored because of the enormous effort required to develop different RTL and communication infrastructure for each potential partition. We are developing a hybrid HW/SW simulation infrastructure which will provide tools for partitioning architectural simulators and synthesizing RTL for the hardware portions. This infrastructure will allow the community to explore and understand the partitioning problem and will eventually lead to automated partitioning algorithms.

Introduction and Motivation

Recently there has been much interest in accelerating architectural simulation through the use of FPGAs. Many efforts to do so have been presented in the last two meetings of the Workshop on Architectural Research using FPGA Platforms (WARFP). For all of these efforts, an important issue which must be addressed is *partitioning*: what portion of the simulation should be computed by reconfigurable logic and what portion should remain in software? The approach taken to partitioning affects both the speed of simulation and the size of the model which can be simulated.

There have been three main approaches to partitioning:

1. Place all simulation functionality into hardware, e.g. [8, 10, 14]. This approach creates a full system prototype, but requires a complete RTL-level design of the hardware and is limited by hardware capacity.
2. Partition the simulator into functional and timing portions, placing the timing portion in hardware and leaving the functional simulator in software[6]. This approach mimics the structure of several microarchitectural simulators, requires fewer hardware resources than the previous approach, and may reduce the design effort. A related partitioning has the software handling only system calls[4].
3. Partition the simulator based upon hierarchical structure in the simulator and allow hardware to communicate with the simulator. Such hybrid simulation was suggested in [7]; a hybrid simulation mode has been demonstrated[11] within the Liberty Simulation Environment (LSE)[13]. Hybrid simulation allows incremental implementation of hardware components.

No work has yet evaluated the relative effectiveness or the speed vs. capacity tradeoffs of different partitioning strategies; in part this is because researchers have had to manually partition and create RTL-level descriptions of the hardware portion of the simulator. This manual work is very time-consuming; indeed some projects announced in WARFP '05 still did not have complete hardware designs by WARFP '06. As a result, researchers do not have the human resources to explore partitioning.

In this talk we describe a hybrid simulation infrastructure under development which will support research into partitioning and HW synthesis of simulator components. The ultimate goal is to automatically partition a microarchitectural model, generating both the RTL required and communication code within the simulator to allow hybrid HW/SW simulation.

This infrastructure will have several components:

- A platform-independent “front-end” communication API which hides platform communication details from both the user and the generated simulator code.
- A partitioning tool which accepts user input about the partitioning to perform, provides feedback on the quality of the partitioning, automatically partitions simulator code into SW-implementation and HW-implementation portions, and inserts calls to the communication API into the SW-implementation code.
- A synthesis tool which generates RTL for the HW-implementation portion of the simulator.

We expect to use this infrastructure to investigate partitioning tradeoffs of speed vs. capacity and to develop automated partitioning algorithms which take into account communication costs and hardware capabilities. The infrastructure will be made available to the community.

Methodology and Status

Our work builds upon the hybrid hardware/software simulation described in [11]. In that work, we integrated PowerPC cores into a structural microarchitectural model of a chip multiprocessor. Structural microarchitectural models are an excellent design entry point because they allow rapid specification of accurate, detailed microarchitectural simulators within weeks[12] and because they provide a natural starting point for partitioning based upon either design hierarchy or functional/timing boundaries. We support the Liberty Simulation Environment[13], Unisim[1], and SystemC[2] for design entry.

The platform-independent communication API is designed to support a variety of partitioning approaches, thus it must support not simply transfer of signals between the hardware and software and control of hardware clocks, but also library calls and means for sharing data with the software simulation. The API hides implementation details from the user; one API call is as simple as "send data to the hardware," with the implementation being able to translate and route the data as needed. We present the API definition in the talk. We intend to demonstrate the platform-independence of the API by implementing the API for two platforms: the DRC Development System DS1000 system, and the BEE2 system[5]; the implementation is ongoing and extends work begun in [11].

The partitioning tool accepts a user specification of the partitioning to be done. In the talk we present a compact, easy-to-use partition specification language and outline the range of partitions which are supported. Such partitions include both high-level partitions such as design hierarchies and functional vs. timing and low-level partitions based upon control flow or data flow. The partitioning tool will be based upon a C++ compiler; we plan to use extensions to LLVM[9] to parse elements of the simulator descriptions, perform data and control flow analysis, generate simulator code with calls to the communication APIs, and provide parsed input to the synthesis tool. The tool will provide feedback on the quality of the partitioning, reporting metrics such as the size of the partitions and the estimated communication bandwidth required.

The synthesis tool will be based upon existing C-to-gates synthesis technology; we plan to use the Trident C-to-gates framework[3]. This synthesizer is able to handle a reasonably large subset of C++ code. It also uses LLVM to deal with its IR, allowing us to easily integrate the partitioning and synthesis tools.

The completed infrastructure will allow the community to investigate partitioning and to understand the speed vs. capacity tradeoffs for different models and hardware. We intend to use the results of such investigation to develop automatic partitioning algorithms that take into account communication costs, hardware capacity, and resources such as memories or processors.

References

[1] UNISIM: UNIted SIMulation Environment home page. http://www.unisim.org.

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

[3] Trident: An FPGA compiler framework for floating-point algorithms. In *Proceedings of the International Conference on Field-Programmable Logic and Applications (FPL)*, pages 317–322, 2005.

[4] Arvind, K. Asanović, D. Chiou, J. C. Hoe, C. Kozyrakis, S.-L. Lu, M. Oskin, D. Patterson, J. Rabaey, and J. Wawrzynek. CRI: RAMP: Research accelerator for multiple processors - a community vision for a shared experimental parallel HW/SW platform. Technical report, Ramp Project, 2005.

[5] C. Chang, J. Wawrzynek, and R. W. Broderson. BEE2: A high-end reconfigurable computing system. *IEEE Design and Test*, 22(2):114–125, Mar/Apr 2005.

[6] D. Chiou, H. Sunjeliwala, D. Sunwoo, J. Xu, and N. Patil. FPGA-based fast, cycle-accurate, full-system simulators. In *Proceedings of the 2nd Annual Workshop on Architecture Research using FPGA Platforms*, 2006.

[7] E. S. Chung, J. C. Hoe, and B. Falsafi. ProtoFlex: Co-simulation for component-wise FPGA emulator development. In *Proceedings of the 2nd Annual Workshop on Architecture Research using FPGA Platforms*, 2006.

[8] J. D. Davis, L. Hammond, and K. Olukotun. A flexible architecture for simulation and testing (FAST) multiprocessor systems. In *Proceedings of the 1st Workshop on Architecture Research using FPGA Platforms*, 2005.

[9] C. Lattner and V. Adve. LLVM: A compilation framework for lifelong program analysis and transformation. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO)*, pages 75–86, 2004.

[10] E. Nurvitadhi and J. C. Hoe. Full-system architectural exploration sandbox. In *Proceedings of the 1st Workshop on Architecture Research using FPGA Platforms*, 2005.

[11] 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 Twelfth International Symposium on High-Performance Computer Architecture (HPCA)*, pages 29–40, February 2006.

[12] D. A. Penry, M. Vachharajani, and D. I. August. Rapid development of a flexible validated processor model. In *Proceedings of the 2005 Workshop on Modeling, Benchmarking, and Simulation*, June 2005.

[13] M. Vachharajani, N. Vachharajani, D. A. Penry, J. A. Blome, and D. I. August. Microarchitectural exploration with Liberty. In *Proceedings of the 35th International Symposium on Microarchitecture*, pages 271–282, November 2002.

[14] S. Wee, J. Casper, N. Njoroge, Y. Tesylar, D. Ge, C. Kozyrakis, and K. Olukotun. A practical FPGA-based framework for novel CMP research. In *Proceedings of the 15th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2007.