

Title of the working program

Functional verification of multi-core processors by combining software simulation with custom configurable hardware emulation techniques

Scientific Domain

Information Systems and Computer Engineering (Reconfigurable Hardware, Computer Architecture Verification, Computer Architecture Simulation, Design Automation)

Abstract

There is wide consensus that verification times for modern microprocessor designs have reached an unacceptable level. Two industry case studies from Intel and Motorola both cite time spent in verification as a major bottleneck to tapeout and estimates are that between 50% to 70% of engineering effort on a design is spent in verification, the majority of it in functional simulations.[1][2][3][4] We plan to improve the functional simulation performance of large multi-core computer processor designs, by using reconfigurable hardware (FPGAs). By identifying components that are performance bottlenecks in sequential simulations and emulating them in parallel on FPGA hardware, we can improve performance and give designers more flexibility to prototype and test portions of their design rapidly in hardware emulation. We will propose an interface between the emulator and simulator that reduces communications overhead and a partitioning algorithm maximizes the amount of parallelization between the hardware and software.

State of the Art

Many attempts have been made to improve functional simulation performance by using hardware emulation of the design.[5][6][7] These efforts have focused primarily on designs where the device under test (DUT) has already been designed for synthesis onto a FPGA while the behavioral testbench has been emulated in software. In some cases, researchers were able to report average performance increases of up to 5000 times by emulating portions of the testbench in hardware.[8] The disadvantages of this approach are that it is only feasible for designs that are coded within the constraints of an FPGA methodology and these techniques are unlikely to scale to larger designs. The designer is also forced to define new interfaces between the testbench and DUT to implement these solutions leading to additionally complexity and consequently errors.

Olukotun et al. propose an automatic technique for partitioning a processor design between emulation in hardware and simulation in software by estimating the performance of each block in hardware and in software and using an iterative algorithm to determine the optimum selection.[9] Using this technique, the authors were able to achieve a speedup of up to 2.8x over software only simulation. Although this is an interesting method, the partitioning is determined statically based on pre-calculated execution times before simulation and there is no adjustment of the partitions based on dynamic events. On a multi-core design it is difficult to determine before simulation which blocks of the design will be performance critical and a dynamic selection of partitions to execute in hardware based only the actual simulation should improve performance significantly. Stitt et al. have developed tools to support dynamic hardware/software partitioning, but these efforts have focused on profiling software benchmarks and not specifically on the simulation of hardware designs.[10]

Additionally, one of the factors that becomes critical as designs become larger, is the communication overhead between the software and hardware. Several partitioning techniques have been proposed to attempt to reduce this overhead without limiting testbench functionality by finding this interface automatically.[11][12] Researchers have found that as the number of configurable logic blocks increase, the amount of time spent in communication between hardware and software can increase to over half the simulation time.[9]

Finally, a new company GateRocket has recently released a product for “Device Native

Verification” of FPGAs. The product integrates an FPGA into the simulator allowing portions of the design to be emulated instead of simulated. This is the first product of its kind aimed at FPGA designers and claims to provide 10-100 times simulation acceleration. However, the product does not provide any kind of partitioning of the design (static or dynamic) and is also limited to designs that will fit in a single configuration of the FPGA.[4]

Objectives

In this proposal, we plan to investigate more aggressive techniques for using reconfigurable hardware to improve the functional simulation performance of multi-core microprocessor designs. To help us identify the critical blocks of the design for improvement, we will profile the execution of a non-trivial multi-core design in software simulation.

Briefly, the principal objectives of this proposal are:

- Research a mechanism for identifying performance critical blocks during simulation
- Consider how designs can be partitioned into the reconfigurable hardware to take maximum benefit of parallel execution
- Develop techniques for limiting the communication overhead between the hardware and software
- Prototype a method for automatically partitioning the design into these blocks
- Measure the scalability of performance with more reconfigurable hardware and larger designs
- Evaluate and study the impact of these proposals with the state of the art techniques

Detailed Description

Multi-core chips present designers with special challenges especially in the area of verification. Unlike traditional single-core processors, increasing performance is directly correlated to increasing complexity. As the number of cores is increased, software cannot adequately simulate designs through the millions of cycles needed for verification.[5] It is “common knowledge”[13][3] that at least half of engineering resources are spent in verification, the bulk of it in functional simulation “since simulation is the only practical mean which can handle the verification for all levels of a design, it is still the mainstream methodology used in industry by high-end microprocessor design projects.”[2] For example, to verify the Pentium 4 design, Intel used a cluster of “several thousand machines” because the full-chip simulation performance was 3-5 simulation cycles per second running on Pentium III based machines. By tapeout, they had verified 200 billion cycles in simulation which was equivalent to only two minutes of real CPU time on a 1GHz CPU![1] It is clear that current methodologies and verification tools will not be able to cope with plans for one-thousand core designs.[14]

Reconfigurable hardware (e.g. FPGAs) permits system designs that can be configured before or during execution. Using this technique, it is possible to improve the performance of functional simulation by migrating components of the microprocessor under test into hardware for emulation. The hardware can then be coupled to the software simulation through the Verilog VPI mechanism to invoke C/C++ functions to access the hardware directly during simulation in a hardware/software co-emulation environment to create a complete and optimized simulation. (See attached diagram). Additionally, emulation gives the designer the ability to have a very specific hardware emulation that more accurately models the actual device.

The challenges of this technique are two-fold. First, identifying which portions of the design are good candidates for emulation and secondly designing an interface between the software and hardware that maximizes parallelization while limiting communications overhead. We plan to identify blocks of the microprocessor design as candidates for emulation using the software profiling tools built into the commercial simulators (Synopsys VCS or Cadence NC-Verilog). We

will measure which parts of the design are most active and therefore most critical to performance under a variety of different testbench stimuli. Finally, using this information we will develop a mechanism to dynamically identify during simulation which blocks of the design are being executed at any time. For prototyping our solution, we plan to begin testing with a simple RISC processor. Possible choices include the LEON2 or LEON3 processor from Gaisler Research designed under contract from the European Space Agency or the OpenRISC core from OpenCores. [15][16] After initial testing on these simpler processors, we propose to use the open source OpenSPARC T1 processor from Sun Microsystems as the device under test. This processor is a complete implementation of a 64-bit, multithreading, multi-core processor conforming to the UltraSPARC 2005 specification. Performance will be measured for different configurations of the design from 1-8 cores.

We expect that the critical performance blocks of the design will change depending on the testbench being applied to the device under test (DUT). For example, during memory intensive code the DUT would be most active in the memory controller, but when running a mathematical testbench the floating point would be actively exercised while the memory controller was mostly inactive. Our investigation will need to determine if a generic design specific partition will be sufficient to handle both of these cases or whether a more dynamic partition that changes depending on the expected testbench stimulus will be necessary. In the latter case, a method of determining the expected critical blocks from analyzing the testbench or previous profiling data will be necessary. Profiling data can be gathered from the built-in profilers in the Verilog simulator or from testbenches developed to monitor performance of the design.

Once we have identified which components are candidates for moving to reconfigurable hardware for performance and based on the testbench stimuli, we will design a method for automatically determining a partition for these blocks. Care will be taken to explore what size partitioning should be used for maximum benefit between executing critical blocks and limiting the bandwidth between the hardware and software. An interface between the hardware and software that limits communications overhead and maximizes parallelization will be proposed. This will be achieved by using already proposed state of the art techniques to move portions of the testbench into the hardware emulator.

Finally, we will keep our methodology applicable to real-world designs that contain modules that may be difficult or impossible to synthesize into hardware. These blocks will be maintained in software simulation. Examples include attached C code that monitors the internal state of the design, debugging modules that inject errors or cause other events, registers that are forced to certain values to bypass lengthy initialization processes, states of the design that are randomly initialized to increase verification coverage, time delays to simulate external transactions or other components added for verification or debugging.

The following are the most important tasks to complete the plan:

- a) Research optimization techniques for using hardware to optimize functional simulation
- b) Research methodologies for partitioning designs to reduce communications overhead
- c) Benchmark and profile a freely available non-trivial multi-core processor to identify candidate blocks for optimization
- d) Experiment with techniques to partition the multi-core design
- e) Develop techniques to interface the software simulator with the hardware
- f) Compare the quality of the results developed in relation to known techniques
- g) Evaluate the developed techniques with the final implementation of an FPGA system
- h) Write thesis

Bibliography

1. Bob Bentley, "Validating the Intel Pentium 4 Microprocessor," *Design Automation Conference*, pp. 244-248, 2001.
2. Jen-Tien Yen and Qichao Richard Yin, "Multiprocessing design verification methodology for Motorola MPC74XX PowerPC microprocessor," *Design Automation Conference*, pp. 718-723, 2000.
3. F. Ozguner and D. Marhefka and J. DeGroat and B. Wile and J. Stofer and L. Hanrahan, "Teaching Future Verification Engineers: The Forgotten Side of Logic Design," 2001.
4. Jack Horgan, "Device Native Verification of FPGAs - GateRocket," 2007.
5. Bauer, J. and Bershteyn, M. and Kaplan, I. and Vyedin, P., "A reconfigurable logic machine for fast event-driven simulation," *Design Automation Conference*, pp. 668--671, 1998.
6. Kudlugi, M. and Hassoun, S. and Selvidge, C. and Pryor, D., "A transaction-based unified simulation/emulation architecture for functional verification," *Design Automation Conference*, pp. 623--628, 2001.
7. Kim, Young-II and Kyung, Chong-Min, "Automatic translation of behavioral testbench for fully accelerated simulation," *IEEE International Conference on Computer Aided Design*, pp. 218--221, 2004.
8. Henftling, R. and Zinn, A. and Bauer, M. and Zambaldi, M. and Ecker, W., "Re-use-centric architecture for a fully accelerated testbench environment," *Design Automation Conference*, pp. 372--375, 2003.
9. Olukotun, K.A. and Helaihel, R. and Levitt, J. and Ramirez, R., "A software-hardware cosynthesis approach to digital system simulation," *IEEE Micro*, 1994.
10. Stitt, G. and Lysecky, R. and Vahid, F., "Dynamic hardware/software partitioning: a first approach," *Design Automation Conference*, pp. 250--255, 2003.
11. Kim, Young-II and Kyung, Chong-Min, "TPartition: testbench partitioning for hardware-accelerated functional verification," *IEEE Design & Test of Computers*, 2004.
12. Kim, Young-II and Yang, Wooseung and Kwon, Young-Su and Kyung, Chong-Min, "Communication-efficient hardware acceleration for fast functional simulation," *Design Automation Conference*, pp. 293--298, 2004.
13. Steven P. Levitan, "DAC - Verification Continues to Dominate Program," 2007.
14. David Yeh, "Design Automation Conference 2007: SPECIAL SESSION: Thousand-Core Chips," 2007.
15. J. Gaisler, "The LEON Processor User's Manual," August 2001.
16. Damjan Lampret, "OpenRISC 1200 IP Core Specification," September 2001.