Hierarchical Simulation of a Multiprocessor Architecture*

Marius Pirvu, Laxmi Bhuyan and Rabi Mahapatra
Department of Computer Science
Texas A&M University
College Station, TX 77843
{pirvum, bhuyan, rabi}@cs.tamu.edu

Abstract

When proposing new architectural enhancements, it is also important to account for the hardware complexity. To achieve this goal, we propose to model the new design in a hardware description language (HDL), synthesize the HDL code, and infer a realistic clock cycle which will be used in subsequent simulations. For accurate results, we develop a two-level hierarchical simulation technique, where an execution driven simulator (RSIM) and an HDL simulator (Verilog-XL) are coupled together to evaluate an entire system. We detail the simulation process and show its impact on the design of an interconnect switch architecture for CC-NUMA multiprocessors.

1 Introduction

One of the problems with the research in the computer architecture field is that most of the architecture simulators overlook the hardware complexity issue and thus, their estimates might be exaggerated. In this paper we demonstrate that logic complexity can indeed affect the clock cycle of a design and hence, the performance of the simulated system. In our opinion, the effect of hardware complexity must be reflected somehow in our simulation results. To achieve this goal we advocate the use of hardware description languages (HDL) together with synthesis. However, since usually we are interested only in a subsystem, it is more advantageous (from the simulation time point of view) to describe in HDL only the targeted component, and let the rest of the system be modeled in a high level programming language. Such a strategy combines the simulation at the architectural level (the C code) with simulation at the circuit level (the HDL code) and hence, will be called a two-level hierarchical simulation.

The benefits of HDL simulation are many fold: First, it shows that the system can be effectively built. This is important because, sometimes, architectural improvements that look good on paper are very difficult to be built in hardware. HDL reveals what is feasible and what is not. Second, it increases the confidence in the design and simulation results. Third, when comparing two designs we must always take into account the effect of hardware complexity on clock cycle. By synthesizing the HDL code we can have a fairly accurate idea of the clock cycle attainable by that hardware. We can identify critical paths and modify the design as to improve its frequency of operation. Moreover, while simulation alone typically focuses only on performance, synthesis directly allows us to consider other factors as well, such as chip area and dissipated power. These factors are important in estimating the design's cost, price/performance ratio, applicability, and marketability.

In this paper we show how a two-level hierarchical simulation can be used in the context of an execution driven simulator. We detail the simulation process, analyze its performance and pinpoint the factors that limit the simulation speed. As case study we use the design of a superpipelined switch for CC-NUMA multiprocessors. For a thorough evaluation, the new design is implemented in Verilog and synthesized. The inferred clock frequency is fed back to our execution driven simulator (RSIM) to analyze the effect of the proposed design on execution time of parallel applications. By incorporating this new switch architecture we show substantial performance improvements through RSIM, but obtain only modest improvements when we synthesize the hardware. Thus, we demonstrate that, for an accurate evaluation, synthesis is a required step, since it effectively brings out the influence of hardware complexity on clock cycle and implicitly on the design's performance.

The research outlined in the paper is somewhat related to papers in the areas of hardware/software co-simulation and HW/SW codesign [7, 1, 6, 4] in the sense that new software interfaces are developed to coordinate the simulation process running on two different machines. However, it must be emphasized that our aim is to design and improve a general purpose computer architecture or a part of it, as opposed to developing HW/SW architectures for some specific target applications. Hence, the underlying evaluation tool in our case is a basic execution driven simulator that can measure the performance of many different applications.

The rest of the paper is organized as follows: Section 2 introduces our design methodology and details the hierarchical simulation approach. Section 3 describes the proposed switch design. Section 4 evaluates the design and demonstrates the importance of hierarchical simulation for accurate simulation results. Finally, Section 5 concludes the paper.

2 Hierarchical Simulation Methodology

To obtain a better simulation accuracy and increase the confidence in a proposed hardware enhancement we advo-

*This research was supported by a grant from the Texas Advanced Technology program

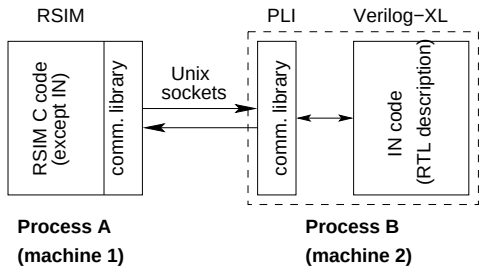

Figure 1. Interaction between RSIM and Verilog-XL

cate the following design process:

1. Implement the proposed architectural enhancements in a high level language and use an execution driven simulator to quantify their impact on performance.
2. If the architectural modifications look promising, isolate the subsystem that contains them and describe it in a hardware description language.
3. Synthesize the HDL code and derive a realistic clock cycle for the design. Study the influence of hardware complexity on clock cycle and, if possible, refine the design as to improve the clock cycle.
4. Feed the obtained clock cycle back to the execution driven simulator and re-evaluate the performance using hierarchical simulation.

In the following we will show how the hierarchical simulation can be carried out in the context of a execution driven simulator. For exemplification we will use RSIM [9], even though any other execution driven simulator could have been used.

2.1 Combining RSIM with Verilog-XL Simulator

The first step to be accomplished in our simulation methodology is to decouple the subsystem to be improved from the original simulator. Because we want to test a new switch design idea we chose the interconnection network (IN) as the target for the HDL implementation. The entire IN structure has been described in Verilog HDL and then compiled and tested using Verilog-XL from Cadence Design Systems, Inc. The next step is to define a communication protocol between the two major parts of our simulation strategy: Verilog-XL, which simulates the IN, and RSIM, which simulates the rest of the system. These parts must be run as separate processes because it is not possible to invoke a Verilog module from a C program. The two processes, RSIM and Verilog-XL, can interact with each other using any standard form of UNIX interprocess communication. We preferred to use UNIX sockets due to their versatility: This solution gives us freedom on the machines to use for simulation and creates the premises (at least theoretically) for speeding up the simulation process by executing two processes in parallel on two different machines (see Figure 1). The HDL code does not interact directly with RSIM, but through a communication library written in C. Calls to the communication routines are done using the Programming Language Interface (PLI), a standard feature in Verilog.

The communication and synchronization model of the two processes is as follows: For each clock cycle only one pair of messages is sent. The Verilog module communicates the status of all its input ports and informs RSIM about the packets that reached their destinations. RSIM reads in these data and informs the Verilog simulator about the new packets that are waiting to enter the interconnect. Upon the reception of this message, Verilog-XL applies the data to the corresponding registers and generates a new clock cycle which in turn will trigger the simulation for one time step.

Since socket communication can add a serious overhead to the entire simulation process, we take some steps as to reduce this communication overhead. For instance, we use the UDP protocol instead of the heavyweight TCP. The disadvantage is that UDP does not provide reliable communication and therefore, to handle occasional network errors (like packet dropping or duplication), we enhance our communication library with a thin layer of error detection based on sequence numbers attached to messages.

The exchanged messages are rather small (around 50 bytes), so the Ethernet bandwidth cannot be a concern. Nevertheless, the latency of a message can seriously degrade the speed of our distributed simulation process. Therefore, we tried to minimize the number of messages as much as possible. To achieve this objective we employ three techniques. First, we pack the information for/from all the switches in the IN into one block of data, and send this structure over the network using a single message. Second, we always piggyback acknowledgments on other useful messages. Third, we simulate the IN only when there is at least one packet in it. If no packets are present, the network simulation step is skipped.

	Processor+memory		Interconnect	
Case	Comm.	Simul.	Comm.	Simul.
Rsim	-	1.26	-	0.08
Rsim+Verilog	4.60	1.26	1.16	4.70

Table 1. Time per simulation step for different simulation strategies. All results are in ms.

2.2 Performance of the Simulation Technique

In the following we analyze the timing of the hierarchical simulation method and compare it to the original RSIM. Statistics regarding the simulation times for the two approaches are presented in Table 1. We show only average time per simulation step because this factor is more relevant and it directly translates into total simulation time by multiplying it with the corresponding number of simulation steps. The results are broken down into time needed for communication and time needed for actual simulation. All timings are given in milliseconds and were collected using the real time clock of UltraSPARC machines. Profiling with `gprof` was not considered because it was not able to capture the waiting time in a `recv_msg` system call. The machines used for experiments were equipped with UltraSPARC II processors running at 300 MHz.

First, we can see that in the original RSIM the interconnect simulation takes only 6% of the total simulation time: 0.08 ms compared to (1.26+0.08) ms. Thus, the main overhead is the simulation of the processors and it is unrealistic to hope that the overall simulation time can be improved by simulating the interconnect on another machine.

The second line from Table 1 reveals that the Verilog simulation of the IN is quite costly in terms of simulation time: 4.7 ms. One reason is, of course, the level of detail at which the simulation is performed. Another reason is the age of our Verilog simulator. HDL simulators tend to improve significantly from one version to another, [1] so the IN simulation time could be much better. However, we still have to pay a non-negligible communication overhead: 1.16 ms. We broke down this overhead and found that the `send_msg` operation takes 0.16 ms while the `read_msg` operation is responsible for 1 ms (this value includes the time Verilog-XL is waiting for RSIM to respond). All in all, the two-level hierarchical simulation increases the overall simulation time about five times.

Ideally, when RSIM issues the `recv_msg` call, the message from the IN simulator has already arrived at destination and waits in a system buffer to be retrieved. This sequence of actions would ensure that the communication and IN simulation overhead were completely overlapped with RSIM simulation overhead. However, due to large IN simulation and communication delays, when the message from the HDL simulator reaches its destination, RSIM is already waiting for it. After a minimal processing, RSIM responds by sending another message and then resumes simulation. Thus, the communication overhead for RSIM is composed by two system calls plus the waiting time in the `recv_msg` routine. On the other hand, the overhead for the HDL simulator is composed of a round trip network delay plus two operating system calls.

3 Case Study: Switch Design for CC-NUMA Multiprocessors

In a CC-NUMA environment message latency is crucial and hence, for good performance, our switch must have a short fall-through latency. Fall-through latency is defined as the time needed by a unit of data to traverse a switch in the absence of contention. Contemporary switches like SGI Spider [3] or the router used in the Mercury Interconnect Architecture [8] have a fall-through latency of about 40 ns. We will use a Spider-like switch as starting point and try to improve it.

Typically, interconnect switches have a link width smaller that the *flit* size, which is the unit of data at which control flow is maintained and arbitration is performed. Thus, a flit is transmitted over the wire as a burst of n *phits*. The common approach in such circumstances is to start processing the incoming data only after a whole flit has been assembled at the input port. In Spider, the core is clocked at 100 MHz while the links operate at 400 MHz. The flow of data is the following (see Figure 2(a)): First, four phits of 16 bits each are assembled in a receiving register. Once the 64 bit flit is synchronized with the core's clock, it is read from the receive buffer into the core. This task takes 17.5 ns on average [3]. Next, arbitration is performed (10 ns) followed by crossbar transmission (10 ns). Finally, the flit must be serialized over the wire and thus, 2.5 ns are lost to transfer the first phit from the core into a transmitting register. The remaining three phits follow in a pipelined fashion. Hence, the fall-through latency is 40 ns (see Figure 2(a)).

[1] For instance, according to Cadence, their improved simulator, NC-Verilog, is at least eight times faster than the latest version of Verilog-XL

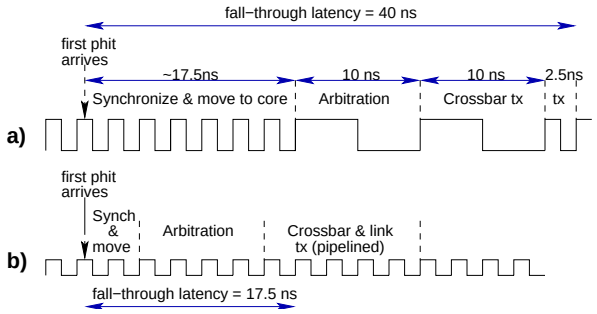

Figure 2. Comparison between the timing for a Spider-like switch (a), and a superpipelined switch design (b)

As described above, Spider implements a *flit switching* technique: it waits for a whole flit to arrive before taking any actions. We believe that this is the main impediment to achieving a lower fall-through latency. By increasing the core frequency four times and starting data processing as soon as the first phit enters the switch we can reduce the synchronization period and begin the arbitration process sooner. We can also improve fall-through latency by overlapping the transmission over the outgoing link with the transmission through the crossbar. To compensate for the higher core frequency, all the data paths inside our switch are four times narrower. For fairness, we also assume that arbitration cannot be sped-up, thus requiring four core cycles. Figure 2(b) shows that by implementing this *superpipelining* we can almost halve the fall-through latency of the switch, provided that the link frequency is not affected.

4 Experimental Results

4.1 Experimental Test-Bed

The architecture we simulate is a CC-NUMA multiprocessor with 16 nodes connected by a 4×4 mesh. Each node is composed of a processor, two levels of cache, a memory module, a network interface and a switch. Given that the cell library used for the switch synthesis corresponds to an older process technology (0.38 μm), for fairness, we scaled down the frequency of our processor to 250 MHz. Other than this, the CPU is 4-way superscalar and employs branch prediction, out-of-order speculative execution, non-blocking reads and a release consistency shared memory model.

Both L1 and L2 caches are write back, write allocate, with a line size of 64 bytes. The L1 data cache is 16 KB, two way set-associative and has an access time of one processor cycle. The L2 cache is 128 KB, four way set-associative, and fully pipelined with an access time of 8 processor cycles. The latency to read an entire cache line from DRAM memory is 24 processor cycles (96 ns). The bus connecting the various components is a split transaction bus with a width of 128 bits and clocked at 83 MHz.

We use five benchmark applications: FWA, GS, TC, EM3D and NBF. FWA implements the Floyd-Warshall algorithm that computes the shortest path between any two nodes in a directed graph. GS (Gramm-Schmidt) computes an orthonormal basis for a set of N-dimensional vectors. TC computes the transitive closure of a directed graph using

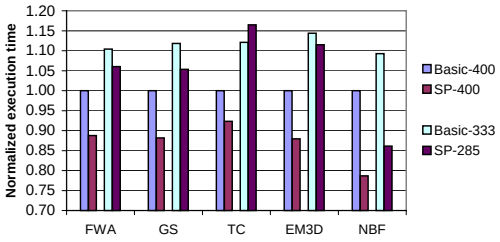

Figure 3. Switch performance comparison

Warshall's algorithm. EM3D [2] models the propagation of electromagnetic waves through objects in three dimension. NBF (the Non-Bonded-Force kernel) is a molecular dynamics simulation taken from the GROMOS benchmark [5].

4.2 Synthesis Insight

To derive the clock cycle for our designs we synthesized the HDL code using Synopsys Design Compiler. The cell library used for synthesis was LCA500K from LSI Logic, library which corresponds to a 0.38 μm process technology. The entire synthesis process was performed automatically by the compiler. The synthesis of the basic switch indicates that the attainable core frequency is about 85 MHz which corresponds to a 340 MHz link frequency. Previously, we assumed that our superpipelined switch design will not degrade the link frequency. However, the synthesis reveals that the clock cycle is 3.5 ns long, yielding a 285 MHz link frequency.

Figure 3 compares the performance of four switch variations. The first two represent designs where we ignore the hardware complexity. `Basic-400` is the Spider-like design which, like Spider, runs at 400 MHz link frequency. `SP-400` is our proposed superpipelined design assumed to work at 400 MHz as well. For the last two configurations we used feedback from synthesis: `Basic-333` is the Spider-like design which runs now at only 333 MHz link frequency as indicated by the synthesis tool, while `SP-285` represents our improved superpipelined design running at 285 MHz.

The main conclusion that can be drawn from Figure 3 is that the feedback from synthesis cannot be neglected: The results collected prior to synthesis indicate that the superpipelined design improves execution time on average by 12%. However, when we consider the clock frequency limitations imposed by the hardware complexity, the average improvement drops to only 6% (SP-285 vs. Basic-333). The explanation for such a behavior is simple: `Basic-333` design offers a superior bandwidth due to its higher link frequency. When the network is lightly loaded pin-to-pin latency is the predominant factor, but when the network is congested bandwidth plays a key role. Applications usually alternate phases of light network load with phases of high traffic and therefore it is important to improve both the bandwidth and the latency.

Another important observation is that applications that do not stress the interconnect too much are more likely to benefit from the superpipelining approach. A case in point is NBF for which the relative improvement from `Basic-400` to `SP-400` is larger than in any other application. The reason is that other applications exhibit a significant blocking time for the IN packets, and blocking time cannot be directly reduced by superpipelining.

5 Conclusion

In this paper we proposed a two-level hierarchical simulation method meant at improving the simulation accuracy of execution driven simulators. The technique calls for the subsystem of interest to be implemented in a hardware description language and simulated using an appropriate tool (in our case Verilog-XL). The rest of the system is implemented in a high level language and simulated using an execution driven simulator (in our case RSIM). After incorporating the timing feedback from a synthesis tool, the two simulators are coupled using UNIX sockets, and run together. We found that the simulation time increases considerably, but the new technique is able to expose the effects of hardware complexity on performance.

As case study we compare two switch designs: a basic one, and a superpipelined design which reduces the fall-through latency of a switch. Without taking the hardware complexity into account, the new design exhibits important reductions in execution time of parallel applications. However, the synthesis showed that such modifications adversely affect the clock cycle and implicitly the link bandwidth. Consequently, the hierarchical simulation revealed that RSIM overestimated the benefits provided by superpipelining. Thus, we demonstrated that synthesis and the two-level hierarchical simulation are invaluable tools that allow a more accurate evaluation of a design.

References

[1] S. L. Coumeri and D. E. Thomas. A simulation environment for hardware-software codesign. In *International Conference on Computer Design*, pages 58–63, Oct. 1995.

[2] D. Culler, A. Dusseau, S. Goldstein, A. Krishnamurthy, S. Lumetta, T. Eicken, and K. Yelick. Parrallel programming in Split-C. In *Proceedings of Supercomputing '93*, pages 262–273, Nov. 1993.

[3] M. Galles. Spider: A high-speed network interconnect. *IEEE Micro*, pages 34–39, January/Febrary 1997.

[4] T. Givargis and F. Vahid. Incorporating cores into system-level specification. In *International Symposium on System Synthesis (ISSS)*, pages 43–48, December 1998.

[5] W. Gunsteren and H. Berendsen. GROMOS: GROningen MOlecular Simulation software. Technical report, Laboratory of Physical Chemistry, University of Groningen, 1988.

[6] T. Hopes. Hardware/software co-verification, an IP vendors viewpoint. In *Proceedings of the International Conference on Computer Design*, pages 242–246, October 1998.

[7] K. Keutzer. Hardware-software co-design and ESDA. In *Proceedings of the 31st Conference on Design Automation*, pages 435–436, June 1994.

[8] A. Mu, B. Chia, S. Kondapalli, C. Koo, J. Larson, L. Nguyen, R. Sastry, Y. Satsukawa, H.-C. Shih, T. Wicki, C. Wu, K. Yu, and X. Zhang. A 285 MHz 6-port plesiochronous router chip with non-blocking cross-bar switch. In *1996 Symposium on VLSI Circuits: Digest of Technical Papers*, pages 136–137, 1996.

[9] V. S. Pai, P. Ranganathan, and S. V. Adve. RSIM: An Execution-Driven Simulator for ILP-Based Shared-Memory Multiprocessors and Uniprocessors. In *Proceedings of the Third Workshop on Computer Architecture Education*, February 1997.