Simulation of Hybrid Computer Architectures: Simulators, Methodologies and Recommendations

Pranav Vaidya and Jaehwan John Lee
Department of Electrical and Computer Engineering
Purdue School of Engineering and Technology
Indiana University-Purdue University Indianapolis

***Abstract*— In the future, high performance computing systems may consist of multiple multicore processors and reconfigurable logic coprocessors. Industry trends indicate that such coprocessors will be socket compatible to microprocessors and will be integrated on existing multiprocessor motherboards without any glue logic. Due to these trends, it is likely that such hybrid computing machines will be a breakthrough for various High Performance Computing (HPC) applications. It is essential to investigate the computer architecture of such hybrid computing machines that utilize reconfigurable logic coprocessors as application accelerators in a HPC system. Simulation can be used to aid this architectural research and guide design space exploration. In this paper, we first present a representative architecture for future hybrid computing machines. Next we present a survey of existing simulators and simulation methodologies for simulation of components of hybrid computing systems. Finally, we present some of the challenges and recommendations to encourage research in hybrid computing machines and their simulators.**

***Index Terms*— Simulation, modeling of hybrid computer architectures, simulation of multiprocessor systems, simulation of FPGAs**

I. Introduction

Two major trends are evident in the computing industry. Firstly, physical limitations of frequency scaling has led to major microprocessor manufacturers pushing for integration of multiple processor cores in a single chip. Secondly, novel computing fabrics such as reconfigurable devices are prominently being used for application acceleration. It is quite likely that these two trends will merge and hybrid computing machines made up of several processors and Reconfigurable Logic (RL) coprocessors will become commonplace. Commodity multiprocessor server platforms containing multiple processor cores and reconfigurable coprocessors [1]–[3] are indications of this trend. These machines offer high performance computation beyond the limitations of Von Neumann machines.

It is imperative to investigate the system architectures of such hybrid computing machines and understand any associated issues with design of such machines. Such investigation can be undertaken by using computer architecture simulation. Computer architects have long utilized simulators to guide the design space exploration and validate the efficacy of proposed architectural enhancements. In addition to traditional challenges such as trade-offs between simulation fidelity and speed, hybrid computing simulators face unique challenges in the form of lack of open source architectures, lack of open source synthesis, configuration and debugging tools. Furthermore, the variation in the reconfigurable logic coprocessor architectures make the design space exploration of hybrid computing architectures truly challenging.

Here, we first define several terms that will be used in this paper. We define a simulator designer as an individual responsible for designing the simulator. We also define a simulation designer/performer as an individual that leverages the simulator to perform simulation. Additionally, we follow the definition of simulation techniques and methodologies as described in [4] by Yi and Lilja. They define simulation methodology as a general term to describe how the simulator is constructed and simulation technique as the approach used by the simulation designer/performer to perform simulation such as using reduced input sets and microbenchmarks. The design decisions associated with the simulation methodology are usually made by the simulator designer while the design decisions associated with the simulation techniques are usually made by the simulation designer/performer.

The design decisions associated with the simulation methodology have direct consequences on the speed and fidelity of simulation. Here, fidelity of the simulation refers to the degree to which the simulated system models the real system. Any design decision associated with the simulation methodology should ensure that the simulation methodology is:

1) Efficient: The simulation methodology should be able to utilize greatly, if not completely, the capabilities of the simulation host. In this case, a simulation host refers to the computing system used to perform the simulation. Dynamic binary translation and parallel simulation are some of the examples of increasing the efficiency and the speed of simulation.
2) Elegant: The chosen simulation methodology should be easily understandable and extensible. This typically involves choices such as choosing an existing simulation language and/or a well validated simulation kernel. Hardware designers exercise this choice frequently where hardware designs are typically simulated using languages such as VHDL [5] and Verilog [6]. Recently, SystemC [7] has also become a popular option in hardware simulation.
3) Deterministic and Reproducible: The simulation should be able to produce identical results given identical initial conditions. Popular simulation language kernels are Sequential Discrete Event Simulators (SDES) because it is relatively easy to ensure determinism in SDES. Simulation kernels like SystemC ensure determinism by modeling concurrent activities in the simulation as user-level threads managed via cooperative multitasking. If concurrent activities are modeled as kernel-level threads, then non-determinism is introduced into the simulation as scheduling of kernel-level threads is seldom available to applications such as the simulation kernels.

Similarly, the design decisions associated with the simulation techniques have direct consequences on the accuracy and validity of simulation. Accuracy and validity of simulation refers to the degree to which the workload used during simulation reflects the true workload of the real system. Yi and Lilja [4] cite several simulation techniques such as reduced input set simulation techniques, truncated execution simulation techniques, processor warm-up approaches and sampling simulation techniques as the popular simulation techniques. Due to space limitation, we concentrate more on the simulation methodology that can be useful in simulation of hybrid computing machines.

Hybrid computing machines consist of multiprocessors and RL coprocessors. Hence, it is essential to identify the simulators, simulation methodologies and techniques used for simulating multiprocessors and RL coprocessors. This work surveys these facets of a hybrid computing system. The remainder of the paper is structured as follows. In section II, we present a representative architecture of future hybrid computing machines. This enables us to identify the main components that the simulators should simulate to a certain degree of fidelity. In section III, we present an overview of existing simulators, simulation methodologies and approaches that may be useful in simulating the hybrid computing system. Section IV presents the challenges and limitations of current simulation methodologies, and section V presents some recommendations for improving research in hybrid computing architectures and simulators.

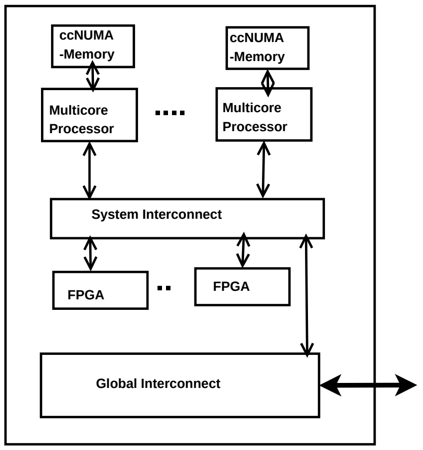

Fig. 1. A Single Node In A High Performance Hybrid Computing System.

II. A Representative Computer Architecture For Hybrid Computing Machines

In this section, we present a representative computer architecture for hybrid computing machines that we believe will be common in a High Performance Computing (HPC) environment. Figure 1 shows the most likely system architecture of a single node in a high performance hybrid computing system.

As shown in Figure 1, a single node of the hybrid HPC system will consist of several complex out-of-order issue RISC/CISC multicore processors and Reconfigurable Logic (RL) coprocessors. These coprocessors will be socket compatible to processors and hence will be integrated on existing motherboards without any glue logic. The processors and coprocessors will be interconnected through uniform chip-to-chip and board-to-board interconnects like Hypertransport [8]. To ensure scale-up as well as speed-up, it is quite likely that the most prevalent memory architectures of a single node in these hybrid computing machines will be cache-coherent Non-Uniform Memory Access (ccNUMA) [9]. The machines will have multiple levels of caches and main memory sizes of several gigabytes, if not terabytes [10].

Field Programmable Gate Arrays (FPGAs) [11]–[15] will be the most commonly used RL coprocessors. These FPGAs will be made up of hundreds of thousands of simple logic blocks such as Configurable Logic Blocks (CLBs). It is quite likely that with better fabrication processes, such FPGAs will have millions of CLBs. Other variations of FPGAs such as coarse-grained FPGAs [16] may also be used to reduce configuration times. Furthermore, these devices will support Run-Time Reconfiguration (RTR) and Run-Time Partial Reconfiguration (RTPR) so that the reconfigurable coprocessors can be used as multiplexed shared resources. We consider the ability of processors to configure and control these custom coprocessors as a distinguishing characteristic of these hybrid computing machines as compared to System-on-Chip (SoC). In SoC designs, the hardware modules are pre-configured to perform a specified function. In a hybrid computing machine, the RL coprocessor is used as either a shared or dedicated resource to perform several functions in hardware.

The DS2004 system from DRC Computer Corp. [2] is reviewed here as an example of the suggested representative architecture. This system is based on a Tyan Thunder K8QSD (S4882) 4-way motherboard with four processor sockets. It supports up to four AMD Opteron Model 875 dual core processors, 12GB ECC DDR, an Nvidia 7300GT PCI Express video card, one 160GB SATA hard drive and one or two DRC Reconfigurable Processor Units (RPUs). The DRC RPU provides a tightly coupled RL coprocessor with direct access to DDR memory and any adjacent Opteron processor at full HyperTransport [8] bandwidth and low latency. The RPU is controlled via an RPU manager, which allows FPGA configuration over HyperTransport. This system is capable of hosting ccNUMA operating system namely Linux (64-bit) Ubuntu 6.x. It is an indication of the growing trend towards integration of RL coprocessors with multicore processors in the industry. Other competitive vendors [1], [3] offer similar platforms.

For the aforementioned representative architecture, it is crucial to note that any simulator for such architecture should be able to simulate the following components:

1) Multicore processors and caches.
2) Reconfigurable logic coprocessors.
3) System interconnects and global interconnects: It is crucial to model system interconnects to a certain degree of fidelity. This is essential as any HPC involves both computation and communication.
4) Run-Time Reconfiguration (RTR) and Run-Time Partial Reconfiguration (RTPR): A hybrid computing machine simulator should be able to model RTR and RTPR to simulate the RL coprocessor as a shared resource.
5) Memory modules: It is quite likely that in the future, each node of a hybrid HPC system will have ccNUMA memory access architecture.

III. Simulation and Co-Simulation Research Work

In this section, we present an overview of the popular simulators and simulation methodologies that will be useful for modeling and simulating the components of hybrid computing machines.

A. Simulators for Chip Multiprocessors

As can be seen in Figure 1, a node in most of the future hybrid computing systems will contain multiple multicore processors. Hence, we only survey the popular simulators that simulate multiprocessors and chip-multiprocessors. Popular uniprocessor simulators such as SimpleScalar [17] are not reviewed here as they do not model such multiprocessing systems.

1) RSIM: Rice Simulator for ILP Multiprocessors: RSIM is the Rice Simulator for ccNUMA, ILP multiprocessors. It was developed and released to public in 1997 [18], [27]. Key RSIM features include support for out-of-order issue, register renaming, branch prediction and nonblocking caches. RSIM also supports user-configurable parameters such as cache sizes and latencies, flit size and delay, as well as instruction window size [4], [18]. RSIM was different from most other simulators in that it modeled the ILP features of a multiprocessor system. RSIM's research showed that disregarding the ILP-level features of a multiprocessor system resulted in the overestimation of the execution time by as much as 132 percent.

RSIM's simulation methodology was derived from YACSIM [19]. YACSIM is a process-oriented, discrete-event simulator developed as part of Rice Parallel Processing Testbed. YACSIM supported user-level multithreading to represent multiple processes. Thus, each process in RSIM runs in a user-level thread and the simulation kernel manages the scheduling of these threads. As a result, RSIM does not take advantage of multiprocessing simulation hosts. RSIM utilizes execution driven simulation techniques to simulate applications compiled and linked for Sparc/V9/Solaris. RSIM uses standard Sparc compilers and linkers at all optimization levels. However, it lacks support for 64-bit integer and quad-precision floating-point operations. Furthermore, it lacks support for standard libraries and applications that rely on conventional Sparc traps. To overcome this limitation, RSIM provides standard C library to support applications.

2) Virtutech Simics - A Full System Simulation Environment: Virtutech Simics is a commercial full-system simulator that can simulate multiprocessor systems with enough accuracy to boot unmodified operating systems [38]. Simics executes unmodified binaries from an ISA perspective and provides a timing interface to user modules. For example, instruction fetch by the simulator is forwarded to the cache modules to stall the execution of instruction for an arbitrary number of cycles [20], [21].

As of Simics 2.0, Simics supports a Micro-Architectural Interface (MAI). Using MAI, the user module can determine when an instruction passes through the microprocessor pipeline such as fetch, decode, execute and commit phases. Using the timing interface provided by Simics, the user module can also support detailed timing modeling. Simics supports checkpointing as a useful simulation technique. This allows the user to run the application to a specific point of interest and save the state of the simulated machine to disk. This technique can reduce simulation time since application initialization phase is run only once. This has important consequences for commercial benchmarks where such initialization or warmup phase can require a significant amount of time, even requiring weeks of simulation [38], [43]

Simics is one of the most popular simulators used in the academia and industry to model entire computer systems and even distributed computing systems. Simics toolset has been used in the academia to develop the Wisconsin General Execution-driven Multiprocessor Simulator (GEMS) [20]. GEMS leverages the full-system functional simulation infrastructure of Simics to drive a set of timing simulator modules for modeling the timing of the memory system and microprocessors. Other projects which have used Simics are Vasa [21] and SimFlex [24]. VASA [21] is a highly configurable multiprocessor simulation package for Simics. Vasa includes models of multilevel caches, store buffers, interconnects, memory controllers and detailed complex out-of-order SMT/CMP processors. It also supports two additional, less detailed simulation modes which run up to 287 times faster than the detailed simulator. SimFlex [24] is a simulator package for Simics developed at the Carnegie Mellon University that leverages the statistical sampling of the inputs to reduce the simulation time of a chip multiprocessor system.

Simics methodology involves simulating a multiprocessor system by simulating each processor in a round-robin fashion. Each processor is simulated for a given number of cycles controlled via a variable called *cpu-switch-time*. This variable allows the coarseness in thread interleaving to be scaled. However, adjusting *cpu-switch-time* to large value can have significant effect when simulating multithreaded applications with contended locks [21]. As a result, derived simulators such as VASA typically set the value of this variable to one. Other simulators in the academia use a similar round-robin simulation of each processor [22] to simulate a multiprocessor system. While Simics can be customized using the APIs that it provides, it does not expose its simulation methodology as it is a commercial software. On the other hand, simulators such as GxEmul [22], [23] are open source and simulate the processors at instruction set level.

3) Wisconsin Wind Tunnel II (WWT II): WWT II [25] differs from the above two simulators in that it is a parallel, discrete-event, direct-execution simulator that can be run across a wide range of platforms, such as desktop workstations, a SUN Enterprise server, a cluster of workstations, and a cluster of symmetric multiprocessing nodes.

WWT II simulates a parallel, ccNUMA system on various parallel systems connected using Myrinet [26]. It uses Synchronized Active Messages (SAM) to communicate between the host nodes for parallel simulation. Analytical modeling has been used to approximate the performance of WWT II for a variety of system sizes. WWT II uses direct execution and parallel hosts as the simulation methodology to speed up the execution. Direct execution executes an instruction of a target machine by directly executing it on the host system. Only operations unavailable on the host platform are simulated by the host platform. Direct execution typically runs orders of magnitude faster than pure interpreted software simulation [25]. Furthermore, WWT II performs parallel simulation by exploiting the parallelism inherent in the target parallel computer to achieve speed-ups of up to 5.8X. However, this approach does not allow changes in the processor models and other architectural parameters such as issue widths, speculative memory accesses and out-of-order execution [29].

WWT II uses SAM as its programming model for communication and synchronization operations. Since SAM runs only on the SPARC architecture, WWT II is not portable to other architectures.

4) Parallel Trace Driven Simulation approaches: Other approaches to parallel simulation of computer architectures include [30]–[35]. All of these approaches use parallel trace driven execution to speed-up simulation of benchmarks such as SPEC CPU 2000 [37]. A given benchmark application is executed concurrently on multiple instances of the simulator initialized with different configurations. Though such an approach increases throughput of simulation, it does not reduce the simulation time of a single simulation as demonstrated by [29].

5) Parallel Simulation of Chip-Multiprocessor Architectures: Research by Chidester *et al*. [29] targeted the simulation of Chip Multiprocessors (CMP) by performing parallel simulation of tightly coupled CMPs (which share L2 caches) on a distributed host system consisting of commercial-off-the-shelf (COTS) workstations. These workstations were connected by a high-speed network.

The simulation methodology used by Chidester *et al*. involves cycle-accurate simulation of the processors and L1 caches. They used the parallel, event-driven simulation built using the Message Passing Interface (MPI) [36] to model communication between L1 cache and the shared L2 cache. Using this approach, simulation speed-ups of up to 5X were obtained.

B. Simulators for Reconfigurable Logic

1) Levels of Abstraction in modeling custom logic: Due to the large complexity of hardware designs today, most simulations are done at various levels of abstraction. Gajski and Cai [28] explain the various levels of abstraction used in system models. They identified that system functionality/computation and communication can be developed independently of one another and refined at each subsequent stage.

As seen in Figure 2, Gajski and Cai have classified the following levels of abstraction:
i) Model I: Model I represents an untimed system architectural model. This model is typically used to specify the functionality and communication of the system and its subsystem without any attention paid to the timing of the interfaces. This model is used to verify the correct functioning of the system and system interconnects.
ii) Model II: Model II represents the Component Assembly Model (CAM). The CAM is used to integrate the empirical understanding of computational time into the model. However, data transfer between components is still untimed.
iii) Model III: Model III represents the Bus Arbitration Model (BAM) or transactional model. In this model, the information about each cycle of the bus is accurately modeled.
iv) Model IV: Model IV represents the Bus Functional Model (BFM). In this model, each signal transition of the bus is modeled as a single event. As a result, communications are timing accurate.
v) Model V: Model V represents the cycle accurate computation model. However, the timing is approximately timed. This model emphasizes communication at transaction level.
vi) Model VI: Model VI represents the register-transfer level model. In this case, both the communication and computation are modeled accurately. This model closely represents the actual hardware and is typically used for automatic synthesis to gates.

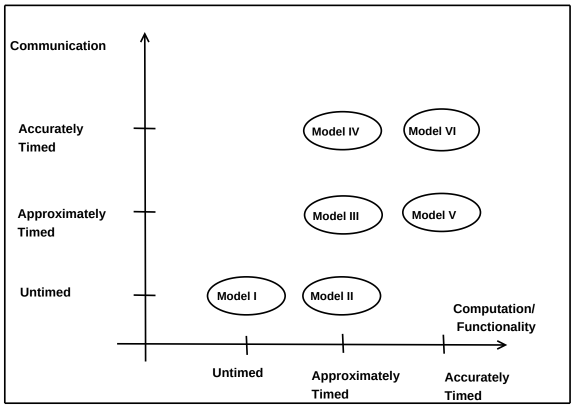

Fig. 2. Abstraction Level Of Models. Courtesy of Cai and Gajski [28]

2) Traditional Simulation/Co-Simulation Approaches and Limitations: Compton *et al*. provided an extensive survey of reconfigurable computing systems and software [39]. However, they did not consider the ability of processors to configure and partially reconfigure RL coprocessors as the defining characteristics of hybrid computing machines. We feel as described in section II that these abilities are key features of hybrid computing machines.

Typically, hardware designers have designed and validated hardware models (Models I-VI) using vendor-specific tools and hardware design languages such as VHDL, Verilog and SystemC. Tools such as ModelSim [40]–[42] are used to perform functional simulation, static timing analysis and timing simulation of the hardware designs. These simulators use the knowledge of cell and routing primitives of the actual device to perform simulations.

Most FPGA design suites assume that hardware design simulated using behavioral and timing simulation will work in actual hardware as intended. However, hardware designs validated using timing simulation may not work on the actual device due to several problems. For example, third party implementation tools may have inferred, places and routed the designs differently than what was specified.

These design suites also assume that the hardware design being synthesized is the only design resident on the reconfigurable device. Such an assumption is valid for most embedded systems which use reconfigurable devices to implement SoC designs. However, these assumptions may not be valid for hybrid computing machines where the RL coprocessors may be multiplexed across multiple applications. As a result, the RL coprocessor may be configured to support several hardware functions. Hence partial reconfiguration is an essential characteristic of such machines. Most of these simulators do not have support for reconfigurable design concepts such as partial reconfiguration. As a result, simulation and co-simulation approaches using traditional hardware design flow is of limited use to the simulation of hybrid computing machines.

3) VTSim - A Virtex-II Device Simulator: VTSim [44], [45] was a discrete-event simulator written in Java that modeled all the hardware resources present in a Virtex-II FPGA [11]. VTSim provided a virtual FPGA device which was compatible to existing Xilinx tools. Using VTSim, the designers could access all the resource values in the virtual FPGA such as flip-flop and look-up table values or values on a routed wire. VTSim was a bitstream level simulator that took the bitstream file (.bit extension) generated from the Xilinx tool chain to simulate the hardware designs.

VTSim was useful in reconfigurable designs as it was able to read and modify bitstreams used to configure and reconfigure the virtual FPGA device. Furthermore, VTSim was integrated into the JHDLBits [47] design suite allowing simulation in Java Hardware Description Language (JHDL) or as a standalone tool.

Unfortunately, this simulator was never released to the public because the permission to release this simulator was never granted by Xilinx, the vendor for Virtex II devices.

4) VirtexDS - A Virtex Device Simulator: Virtex Device Simulator (VirtexDS) [46] was a device level simulator for Virtex-II Pro devices [11] from Xilinx. It was released as part of the Xilinx JBits 2.8 SDK [11]. This simulator was similar to VTSim that simulated Virtex FPGA devices. VirtexDS provided a software model of the FPGA device for the entire Virtex family of FPGAs. It supported run-time configuration and run-time partial reconfiguration that could be controlled through the JBits 2.8 environment. VirtexDS allowed for existing tools such as the BoardScope [48] debug tool to interface directly to the simulator without any modification.

Subsequently, Xilinx released JBits 3.0. However, it did not release a device level simulator. During our survey, we found no device level FPGA simulators available in the industry or academia for research purposes.

IV. Challenges and Limitations for Hybrid Computing Machine Simulators

From our survey, we found the following limitations and challenges that the hybrid computing simulators and machines face:

1) *Current limitations for simulating multicore processors*: Most simulators in the industry and academia such as RSIM and Simics are built using Sequential Discrete Event Simulators (SDES). Even hardware simulation languages and kernels use SDES for functional and timing simulation. These simulators do not take advantage of the parallel computation facilities that are becoming available even at the desktop computing level. With the advent of multicore processors, these kernels should use the parallel computational facilities that current simulation hosts offer. WWT II and other parallel discrete event simulation approaches show that speed-ups can be obtained from parallel simulation of computer architectures without compromising on the fidelity of the simulator. However, these simulators have been built using specialized programming models for distributed computing such as SAM and MPI. While SAM is not portable, MPI suffers serious performance degradation on multicore shared memory architectures as it maps each node of computation to an OS process. Hence, one of the main challenges in simulating multicore processors is balancing portability of the simulator with the ease of using and extending the simulator. This challenge can be solved by identifying and using a good programming model for multicore and cluster simulation hosts. The key idea behind such a programming model should be exploiting local multiprocessor as well as cluster computing power. We state such a computing model in section V.
2) *Challenges in Simulating Reconfigurable FPGAs*: The challenges in simulating reconfigurable logic devices are greater than that of traditional processors. Reconfigurable devices typically have closed architectures, closed bitstreams, and even more so there is a lack of open source development tools, compilation-to-gates tools, verification and synthesis tools. Furthermore, there are no standard APIs for configuring and communicating with these reconfigurable coprocessors in a hybrid computing machine. It is reasonable to understand that the industry would most likely not release open architectures and tools due to the inherent financial gains associated with such tools. In another aspect, as the granularity of FPGA devices increases towards fine-grained architectures, it would be extremely inefficient to simulate these devices using SDES.
3) *Challenges in Simulating the Hybrid Computing System*: Most simulation kernels do not support multiple models of computation in the simulator. Different models of computation may be advantageous to model the various components of the hybrid computing system. For example, while Synchronous Data Flow Graphs (SDFG) may be advantageous to model streaming devices such as DSPs, Parallel Discrete Event Simulation (PDES) may be advantageous to model multicore processors. Hence, research and further exploration of such multi-model simulation kernels [52] should be encouraged.

V. Recommendations

Based on the aforementioned observations, we make the following recommendations for fostering research in the area of hybrid computing systems and hybrid computing simulators.

Recommendation 1: Use of Parallel Simulation Techniques for Current Simulation Hosts

It is essential to note that as hybrid computing machines are growing more complex, the simulation hosts are also becoming more powerful. Over the last few years, even desktop computers with two or more processors/processor cores have become available to the general public [50], [51]. Simulator designers should take note of this, and research simulators using Parallel Discrete Event Simulation (PDES) should be investigated.

As identified in the challenges, the choice of programming model is a key challenge for developing simulators for multicore simulation hosts. It is also essential that to ensure scalability of the simulation host, such programming model should be seamlessly extensible to a cluster of computers. Streaming programming models based on well established process calculi such as Communicating Sequential Processes [49], [53] may be the solution to this issue. These programming models may be more flexible and faster than SAM and MPI for both multicore and other cluster simulation hosts. Such programming models can model physical processes as nano-threads, user-level threads and kernel-level threads. Thus, the designer is flexible in choosing the appropriate granularity of threads according to the level of communication between the modeled components. However, we did not find any simulator that is built using streaming languages. Developing simulators using such programming paradigms should be pursued.

Recommendation 2: Open FPGA Architecture and Open Source FPGA Design Tools

FPGA industry is a multi-million dollar industry. Device vendors have invested greatly in their proprietary architectures and FPGA design suites. However, it would be beneficial to both the academia and industry if a consortium similar to OpenFPGA consortium [54] is established for the development of open source FPGA architectures and design tools for hybrid computing machines. This would be both financially and intellectually beneficial to the industry and academia. For example, traditionally it is assumed that the RL coprocessors are dedicated for a single application. It would be beneficial if these RL coprocessors are used as multitasking shared resources. To make this happen, the detailed layout of the application on the RL coprocessor should be known beforehand. Hence, with open source FPGA architectures, a detailed architectural model of the RL coprocessor can be used to perform intelligent compilation and synthesis for these shared coprocessors. Additionally, required research impetus can be accelerated using open source reconfigurable coprocessor architectures in HPC applications. Furthermore, such an endeavor can create the required engineers and scientists who are exposed to reconfigurable computing internals.

VI. Summary

In this paper, we have summarized some of the simulators and simulation methodologies that are likely to be useful in the simulation of hybrid computing machines made up of multiple multicore processors and reconfigurable logic coprocessors. It would be beneficial if the simulator methodologies fully utilize the computational power offered by the simulation hosts. Research into developing simulators built around the concept of Parallel Discrete Event Simulation (PDES) and/or streaming language paradigms such as Communicating Sequential Processes (CSP) [53] should be encouraged.

There exists an inherent trade-off between simulation speed and simulation accuracy. However, many simulation approaches target simulation speed by compromising the fidelity of simulation. Such a trade-off is acceptable for development of systems software; however it can result in the overestimation of execution speeds in some cases. With the current industry trends towards chip multiprocessing, it is essential that simulators model such systems with sufficient fidelity. As a result, as part of our recommendations, we have suggested that further research into parallel simulation of chip multiprocessors be pursued.

In addition, to foster further development into Reconfigurable Logic (RL) coprocessors, we have suggested that both the industry and the academia join hands to come up with open source FPGA architectures and programming tools. Currently, there exist no open source simulators that support run-time reconfiguration and run-time partial reconfiguration. Research into device level FPGA simulators would be greatly useful to both academia and industry and thus should be pursued. We predict with high confidence that such research will provide great impetus in developing open source compilation and synthesis tools. This will further the integration of such RL coprocessors with applications spanning from embedded systems, general purpose computing to High Performance Computing (HPC).

References

[1] Celoxica RCHTX System, http://www.celoxica.com/products/rchtx/default.asp, visited Mar 2007.

[2] DRC Computer Corporation, http://www.drccomputer.com/, visited Mar 2007.

[3] XtremeData Inc, http://www.xtremedatainc.com/, visited Mar 2007.

[4] J. Yi and D. Lilja, "Simulation of Computer Architectures: Simulators, Benchmarks, Methodologies, and Recommendations," IEEE Trans. on Computers, vol. 55, no. 3, pp. 268-280, Mar. 2006.

[5] VASG: VHDL Analysis and Standardization Group, http://www.eda.org/vhdl-200x/, visited Mar 2007.

[6] IEEE Verilog Standardization Group, http://www.verilog.com/IEEEVerilog.html, visited Mar 2007.

[7] SystemC Community Website, http://www.systemc.org/, visited Mar 2007.

[8] Hypertransport Consortium, http://www.hypertransport.org/index.cfm, visited Jan 2007.

[9] NUMA, HyperTransport, 64-Bit Windows, and You http://developer.amd.com/article_print.jsp?id=8, visited Dec 2006

[10] Performance Guidelines for AMD Athlon 64 and AMD Opteron ccNUMA Multiprocessor Systems, http://www.amd.com/us-en/assets/content_type/white _papers_and_tech_docs/40555.pdf, visited Dec 2006.

[11] Xilinx Corporation, http://www.xilinx.com/, visited Jan 2007.

[12] Altera Corporation, http://www.altera.com/, visited Jan 2007.

[13] Actel Corporation, http://www.actel.com/, visited Jan 2007.

[14] Lattice Semiconductor Corporation, http://www.latticesemi.com/, visited Jan 2007.

[15] QuickLogic Corporation, http://www.quicklogic.com/, visited Jan 2007.

[16] E. Mirsky and A. DeHon, "MATRIX: A reconfigurable computing architecture with configurable instruction distribution and deployable resources," IEEE Symposium on FPGAs for Custom Computing Machines, pp. 157-166, 1996

[17] T. Austin, E. Larson and D. Ernst, "SimpleScalar: An infrastructure for computer system modeling," Computer, vol. 35, no. 2, pp. 59-67, 2002.

[18] V. Pai, P. Ranganathan and S. 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.

[19] J. Jump, YACSIM Reference Manual. Rice University, version 2.1.1 edition, 1993, www.owlnet.rice.edu/ elec428/yacsim/yacsim.man.ps, visited Mar 2007.

[20] M. Martin *et al*., "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," SIGARCH Comput. Archit. News, pp. 92-99, 2005.

[21] D. Wallin, H. Zeffer, M. Karlsson and E. Hagersten, "VASA: A Simulator Infrastructure with Adjustable Fidelity," Parallel and Distributed Computing and Systems, 2005.

[22] P. Vaidya and J. Lee, "Design Space Exploration of Multiprocessor Systems with Multicontext Reconfigurable coprocessors," In Proceedings of Engineering of Reconfigurable Systems and Algorithms, ERSA'07, pp. 51-60, June 2007.

[23] GxEmul, http://gavare.se/gxemul/, visited Jan 2007.

[24] T. Wenisch *et al*., "SimFlex: Statistical Sampling of Computer Architecture Simulation," IEEE Micro special issue on Computer Architecture Simulation, vol. 26, no. 4, pp. 18-31, Jul/Aug 2006.

[25] S. Mukherjee *et al*., "Wisconsin Wind Tunnel II: A Fast and Portable Parallel Architecture Simulator," In Workshop on Performance Analysis and Its Impact on Design, June 1997.

[26] Myricom Page for Myrinet, http://www.myri.com/myrinet/overview/, visited Jan 2007.

[27] R Covington *et al*., "The Rice Parallel Processing Testbed," In Proceedings of the 1988 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pp. 4-11, May 1988.

[28] L. Cai and D. Gajski, "Transaction Level Modeling: an overview," Hardware/Software Codesign and System Synthesis, pp. 19-24, 2003.

[29] M. Chidester and A. George, "Parallel Simulation of Chip-Multiprocessor Architectures," ACM Trans. on Modeling and Computer Simulation, vol. 12, no. 3, pp. 176-200, July 2002.

[30] L. Eeckhout and K. De Bosschere, "Efficient Simulation of Trace Samples on Parallel Machines," Parallel Computing, vol. 30, no. 3, pp. 317-335, Mar. 2004.

[31] B. Falsafi and D. Wood, "Modeling Cost/Performance of a Parallel Computer Simulator," ACM Trans. on Modeling and Computer Simulation, vol. 7, no. 1, pp. 104-130, Jan. 1997.

[32] G. Lauterbach, "Accelerating Architectural Simulation by Parallel Execution of Trace Samples," Sun Microsystems Laboratory Technical Report TR-93-22, 1993.

[33] A. Nguyen, P. Bose, K. Ekanadham, A. Nanda and M. Michael, "Accuracy and Speed-Up of Parallel Trace-Driven Architectural Simulation," In Proceedings of Int'l Parallel Processing Symp., 1997.

[34] D. Poulsen and P. Yew, "Execution-Driven Tools for Parallel Simulation of Parallel Architectures and Applications," In Proceedings of Supercomputing, pp. 860-869, 1993.

[35] W. Wang and J. Baer, "Efficient Trace-Driven Simulation Methods for Cache Performance Analysis," ACM Trans. on Computer Systems, vol. 9, no. 3, pp. 222-241, Aug. 1991

[36] MPI Homepage, http://www-unix.mcs.anl.gov/mpi/, visited Mar 2007.

[37] SPEC CPU 2000, http://www.spec.org/cpu/, visited Mar 2007.

[38] P. Magnusson *et al*., "Simics: A full system simulation platform," Computer, vol. 35, no. 2, pp. 50-58, 2002.

[39] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software," ACM Comput. Surv. 34, pp. 171-210, 2002.

[40] Mentor Graphics, ModelSim. http://www.mentor.com/modelsim.

[41] Mentor Graphics, Hardware/Software Co-Verification:Seamless. http://www.mentor.com/seamless/, visited Jan 2007.

[42] Mentor Graphiscs, Seamless FPGA, http://www.mentor.com/products/fv/hwsw_coverification/seamless_fpga/, visited Jan 2007.

[43] W. Fu and K. Compton, "A Simulation Platform for Reconfigurable Computing Research," IEEE International Conference on Field Programmable Logic and Applications, Aug. 2006.

[44] J. Hunter, P. Athanas and C. Patterson, "VTsim: A Virtex-II Device Simulator," In Proceedings of Engineering of Reconfigurable Systems and Algorithms, ERSA'04, Jun 2004.

[45] J. Hunter, "*A Device-Level FPGA Simulator*," Masters Thesis, June 2004.

[46] S. McMillan, B. Blodget and S. Guccione, "VirtexDS: a Virtex device simulator," In Proceedings of SPIE, pp. 50-56, Oct 2000.

[47] A. Poetter, "*JHDLBits: An Open-Source Model for FPGA Design Automation*," Master's Thesis, Aug 2004.

[48] D. Levi and S. Guccione, "BoardScope: a debug tool for reconfigurable systems," In Proceedings of SPIE vol. 3526, pp. 239-246, Oct 1998.

[49] W. Thies, M. Karczmarek and S. Amarasinghe, "StreamIt: A Language for Streaming Applications," In Proceedings of the 2002 International Conference on Compiler Construction, Apr 2002.

[50] AMD Multicore Website, http://multicore.amd.com/, visited Mar 2007.

[51] Intel Multicore Website, http://www.intel.com/multi-core/, visited Mar 2007.

[52] J. Eker *et al*., "Taming heterogeneity–the Ptolemy approach" In Proceedings of the IEEE Special Issue on Modeling and Design of Embedded Software, vol. 91, pp. 127-144, Jan 2003.

[53] C. Hoare, "*Communicating Sequential Processes*," Prentice Hall International, 1985.

[54] OpenFPGA consortium, http://www.openfpga.org/, visited Mar 2007.