An Extended Tracing System for the COTSon Simulator

Marco Procaccini[1], Roberto Giorgi[1]

Department of Information Engineering and Mathematics, University of Siena, Via Roma 56, 53100 Siena, Italy

ABSTRACT

An important step before adopting a simulator is its validation, in order to determine how accurate the simulator is compared to a real machine. When validating a simulator, errors in simulating full applications can lead to misleading conclusions. In this paper, an extension of the existing tracing tool of the COTSon simulation framework is presented with the capabilities to collect information about the executed instructions from both timing and functional execution (e.g., timestamp, latency). The output trace has been compared with the assembly representation of the benchmark executed into the COTSon simulator to verify the correct execution flow. Thanks to the tracing tools, we were able to analyze the behavior of COTSon simulator during the execution of benchmarks and discover possible bottlenecks and optimization points.

KEYWORDS: Performance Evaluation, Computer Architecture, Computer Simulation

1 Simulation Framework

Research on high-performance computing architectures depends on accurate and flexible simulation to enable the development of future generations of computer systems. Simulators have become an integral part of the computer architecture research and design process. Since they have the advantages of cost, time, and flexibility, architects use them to guide design space exploration for analyzing the efficiency of new features not yet available in the market [GKP19b]. A simulator not only ensures the functional correctness but also may provide accurate timing information [PG19].

The HP-Labs COTSon simulator [AFF+09] is a full-system simulation from multi-core (e.g., 1000 cores [GBB+14]) to multi-node and the capability of network simulation, which makes COTSon a complete simulation environment [GKP19a,GKP19c,P+12,GSPF12]. COTSon is based on the functional directed simulation, which means that the functional part drives the timing part, and the two parts are completely separated both in the coding and during the simulation Figure 1. The functional model is very fast but does not include any architectural detail, whilst the timing model is an architectural-complete description of the system (and, as such, also includes the actual functional behavior).

The functional execution is driven and validated by the AMD SimNow virtualizer tool, which is proposed by AMD in order to test and develop their processors and platforms.

[1]E-mail: {procaccini,giorgi}@dii.unisi.it

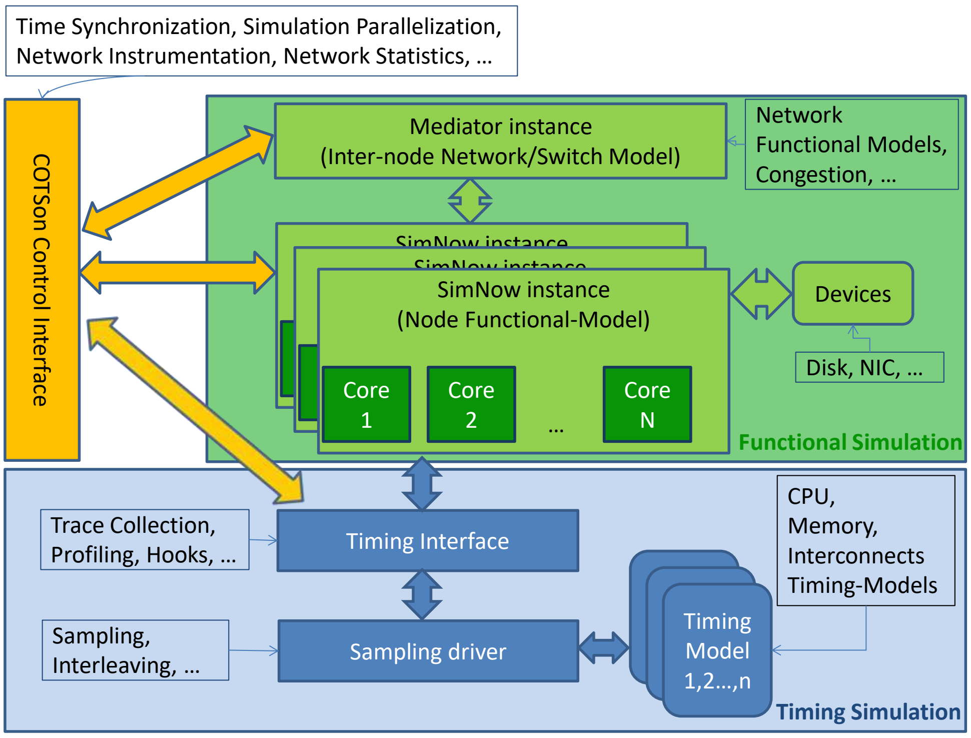

Time Synchronization, Simulation Parallelization, Network Instrumentation, Network Statistics, …
COTSon Control Interface
Mediator instance (Inter-node Network/Switch Model)
Network Functional Models, Congestion, …
SimNow instance
SimNow instance
SimNow instance (Node Functional-Model)
Core 1
Core 2
…
Core N
Devices
Disk, NIC, …
Functional Simulation
Trace Collection, Profiling, Hooks, …
Timing Interface
CPU, Memory, Interconnects Timing-Models
Sampling, Interleaving, …
Sampling driver
Timing Model 1,2…,n
Timing Simulation

Figure 1: The COTSon simulation framework architecture.

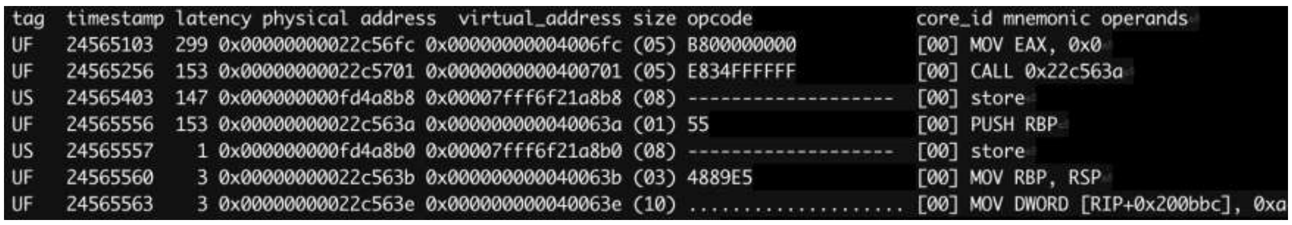
COTSon executes its functional model into the SimNow virtual machine, running and testing the execution of the functional model (green part of Figure 1). The timing model is a formal specification that defines a custom behavior of a specific architectural or micro-architectural component, in other terms the timing model defines the architecture itself [AFF+09]. The timing model behavior is specified by modeling in C/C++ statements the steps performed by the functional part and associating them with the estimated latency (blue part of Figure 1).

The functional and timing model communicate each other through event queues. These event queues decouple the generation of events by the functional simulator and their processing by the timing models. The event queues implement a timing feedback mechanism (e.g., instruction throughput), which periodically adjusts the speed of the functional simulator for reflecting the timing speed. The feedback mechanism aims at limiting the functional simulator's divergence with respect to the timing simulator.

2 The Extended Tracing System

The COTSon simulator provides a tracings tool through which is possible the extraction of information from the functional execution performed into the SimNow virtual machine. In order to validate both functional and timing execution and analyze the effects on the simulation produced by the communication between functional and timing models, we decided to extend the existing tracing system of the COTSon simulator.

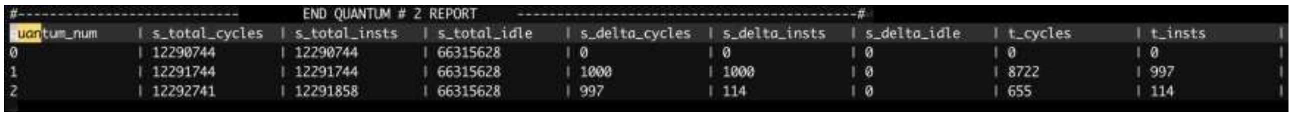
The extended tracing tool combines the information of the instructions produced during the functional execution (e.g., virtual address, physical address) with the behavior of the timing model Figure 2. The list of executed instructions is now ordered by the timestamp provided by the timing model, through which is possible the validation of the timing execution. Additional information like the latency computed by the timing model for each instruction gives us the possibility of highlighting possible improvements in the design process (e.g., reducing the long-latency instructions with different architecture configurations). Moreover, a report of statistics is generated for the tracking the behavior of the functional and timing execution at the end of the COTSon intervals named "quantum". In fact, COTSon makes the SimNow virtualizer tool run for a custom interval named "quantum" specified by the user. Each quantum produces a stream of executed instructions, which are sent to the respective CPU timing models by using event queues following the “feedback directed" approach [AFF$^+$09]. The tracing system collects statistics useful for understanding and analyzing how the functional execution is modeled by timing model during each quantum Figure 3.



```
tag  timestamp latency physical address  virtual_address size opcode              core_id mnemonic operands
UF   24565103  299 0x00000000022c56fc 0x00000000004006fc (05) B800000000          [00] MOV EAX, 0x0
UF   24565256  153 0x00000000022c5701 0x0000000000400701 (05) E834FFFFFF          [00] CALL 0x22c563a
US   24565403  147 0x000000000fd4a8b8 0x00007fff6f21a8b8 (08) ------------------- [00] store
UF   24565556  153 0x00000000022c563a 0x000000000040063a (01) 55                  [00] PUSH RBP
US   24565557    1 0x000000000fd4a8b0 0x00007fff6f21a8b0 (08) ------------------- [00] store
UF   24565560    3 0x00000000022c563b 0x000000000040063b (03) 4889E5              [00] MOV RBP, RSP
UF   24565563    3 0x00000000022c563e 0x000000000040063e (10) ................... [00] MOV DWORD [RIP+0x200bbc], 0xa
```

Figure 2: Example of output produced by the tracing system tool for the COTSon simulator with a Recursive Fibonacci benchmark.

Finally, the distorm disassembler tool [Dis] has been updated with the latest version for supporting the translation of the latest version of the x86_64 instruction set. The distorm gives us the possibility to have the mnemonic representation of an instruction, which makes possible the comparison of the executed instruction with the assembly language of the benchmark executed into the COTSon simulator.



```
#--------------------------          END QUANTUM # 2 REPORT       -----------------------------------------#
quantum_num      | s_total_cycles | s_total_insts | s_total_idle | s_delta_cycles | s_delta_insts | s_delta_idle | t_cycles | t_insts |
0                | 12290744       | 12290744      | 66315628     | 0              | 0             | 0            | 0        | 0       |
1                | 12291744       | 12291744      | 66315628     | 1000           | 1000          | 0            | 8722     | 997     |
2                | 12292741       | 12291858      | 66315628     | 997            | 114           | 0            | 655      | 114     |
```

Figure 3: Report produced at the end of each quantum by the tracing tool system for the COTSon simulator. The report tracks information about the executed cycles and instructions into the Simnow virtual machine (s_total_cycles, s_total_insts, s_total_idle, s_delta_cycles, s_delta_insts) and into the timing model (t_cycles and t_insts).

3 Trace Analysis and Considerations

Thanks to the extension provided for the tracing tool of the COTSon simulator, it was possible to confirm the correct execution of both timing and functional model, which correctly process the flow of instructions contained in the binary representation of a benchmark. Based on the reports at the end of each quantum, it was possible to verify and analyze how timing execution affects functional execution. As we can see in the Figure 3, the number of executed instructions in the SimNow virtual machine (s_delta_insts) is adapted to the number of instructions executed into the timing models (t_insts) during the quantum. However, the tracing tool highlights a peak of executed instructions (end cycles) during the first quantum in both functional and timing model (Figure 3), which is caused by a lack of timing model information at the beginning of the execution. The peak of performance in the first quantum can produce harmful effects on the simulation results, such as a wrong analysis of the power consumption. In order to solve this issue, we are investigating in an initial warm-up phase, which can produce the necessary information for aligning the functional and timing model from the beginning of the simulation execution.

References

[AFF$^+$09] Eduardo Argollo, Ayose Falcón, Paolo Faraboschi, Matteo Monchiero, and Daniel Ortega. COTSon: infrastructure for full system simulation. *SIGOPS Oper. Syst. Rev.*, 43(1):52–61, 2009.

[Dis] Distorm Disassembler. https://github.com/gdabah/distorm.

[GBB$^+$14] R. Giorgi, R. Badia, F. Bodin, A. Cohen, P. Evripidou, P. Faraboschi, B. Fechner, G. Gao, A. Garbade, R. Gayatri, S. Girbal, D. Goodman, B. Khan, S. KoliaÃ¯, J. Landwehr, N. LÃª Minh, F. Li, M. LujÃ n, A. Mendelson, L. Morin, N. Navarro, T. Patejko, A. Pop, P. Trancoso, T. Ungerer, I. Watson, S. Weis, S. Zuckerman, and M. Valero. TERAFLUX: Harnessing dataflow in next generation teradevices. *ELSEVIER Microprocessors and Microsystems*, 38(8, Part B):976–990, 2014.

[GKP19a] R. Giorgi, F. Khalili, and M. Procaccini. Analyzing the impact of operating system activity of different linux distributions in a distributed environment. In *IEEE Euromicro Int.l Conf. on Parallel, Distributed, and Network-Based Processing*, pages 422–429, Pavia, Italy, Feb. 2019.

[GKP19b] R. Giorgi, F. Khalili, and M. Procaccini. Axiom: A scalable, efficient and reconfigurable embedded platform. In *IEEE Proc. Design, Automation and Test in Europe (DATE)*, pages 1–6, Florence, Italy, Mar. 2019.

[GKP19c] R. Giorgi, F. Khalili, and M. Procaccini. A design space exploration tool set for future 1k-core high-performance computers. In *ACM RAPIDO*, pages 1–6, 2019.

[GSPF12] R. Giorgi, A. Scionti, A. Portero, and P. Faraboschi. Architectural simulation in the kilo-core era. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS 2012), poster presentation*, pages 1–3, London, UK, Mar 2012.

[P$^+$12] A. Portero et al. Simulating the future kilo-x86-64 core processors and their infrastructure. In *45th Annual Simulation Symp. (ANSS12)*, pages 62–67, Orlando, FL, Mar 2012.

[PG19] Marco Procaccini and Roberto Giorgi. X86_64 vs aarch64 performance validation with cotson. In *HiPEAC ACACES-2019*, pages 261–264, Fiuggi, Italy, jul 2019. poster.