ISSN 2278 – 0211 (Online)

Performance Enhancement of HW/SW Co-simulation using Pipelined Processor Virtual Models

Ritika Chandan
M.Tech Student, Department of Information and Technology
Chandigarh Group of Colleges, Landran, Punjab, India
Sachin Majithia
Assistant Professor, Department of Information and Technology
Chandigarh Group of Colleges, Landran, Punjab, India
Vivek Sharma
VTI VPTeam, Intel Mobile Communications Pvt. Ltd., Bangalore, Karnataka, India

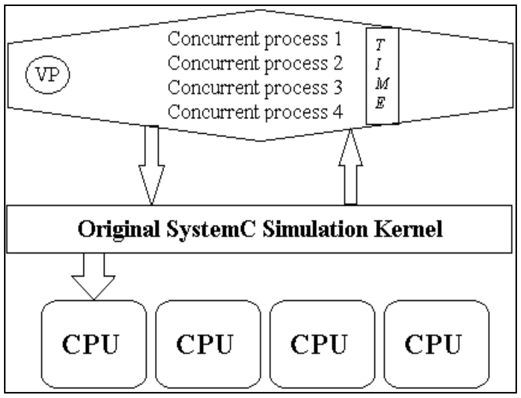
Abstract:
Mainstream processor virtual models do not implement pipelines in their design, as it is considered as an overhead in terms of code complexity and may reduce the performance due to large context switching that will happen in the simulation in effect to handle the notion of concurrent execution of the pipeline algorithm. This problem provides an interesting opportunity to evaluate the performance of a pipelined processor virtual model which is implemented using a multithreaded language and is free of the single thread context switching overhead present in current hardware description programming languages. The novelty of the pipelined processor model and its performance evaluation in different scenarios of loads is described in this paper, and it describes the execution process of the multithreaded SystemC language, which is used for the model's implementation. The limitations in current multicore implementation of simulation kernel which were faced during the implementation are also analyzed to provide scope for further research and development.

Keywords: *Performance Evaluation, Pipelining, SystemC modeling language, ARM Instruction Set Simulator, Virtual Platform, Parallel Architectures, VCD Analyzer, and Branch Delays*

1. Introduction
Multicore machines are a commonplace for current generation of computer systems, as these machines have the capability to perform multiple tasks concurrently and hence are more efficient when compared to single core machines but same cannot be said for virtual platforms and simulation tools, which are largely having a sequential simulation engine and doesn't support concurrent multithreaded simulation even when running on a multicore machine, which downgrades simulation performance for complex virtual platforms having multiple processes that requires concurrent notion of execution, as is in case of pipeline architecture, thus mainstream processor virtual models do not implement pipelines in their design. The language of choice for developing processor virtual models is C, C++, which all inherently don't implement multi-threaded language constructs. The increase of processing power and parallelism of host machines creates the need for faster yet accurate multithreaded virtual models and supporting simulation tools for virtual prototyping, supporting both functional verification and performance evaluation [1].Several industrial and academic frameworks appeared to help modeling, simulating and debugging these architectures. The SystemC hardware description languageis the effective backbone of these entire frameworks, describing hardware from RTLto Transactional Level Modeling (TLM) [1]. However, when it comes to simulate architectures containing concurrent processes that share the parallel notion of simulation time, even the simulation speed provided by the TLMis not enough. Unfortunately, the genuine SystemC simulation kernel is fully sequential and cannot exploit the processing power provided by these multi-cores host machine [1], and any performance gain expected from implementing the pipelines in the processor virtual model requires multicore execution of each pipeline process, which makes the necessity to use an alternative parallel simulation engine for the kernel, called SystemC-SMP [1].

2. Problem Statement
Mainstream processor virtual models do not implement pipelines in their design, mainly because the hardware description languages used to develop the processor virtual models don't implement the support for utilizing the resources available in a multicore host

workstation, hence losing out the ability to have a true concurrent execution, thus it is considered as an overhead in terms of code complexity to implement a heavy context switching logic like that of pipeline on a sequential simulator and in fact it may even reduce the overall simulation performance.
This problem provides an interesting opportunity to evaluate the performance of a pipelined processor virtual model when it is implemented using a multithreaded language and simulated on a multicore supported simulation kernel, which is free of the single thread context switching overhead. SystemC-SMP is used to implement and evaluate the performance improvement of concurrent execution algorithm of pipelines in a SystemC based RISC processor virtual model, this simulation engine is a work in progress version of genuine SystemC kernel, having multicore execution and is different from original SystemC as shown by Fig.1 and Fig.2.



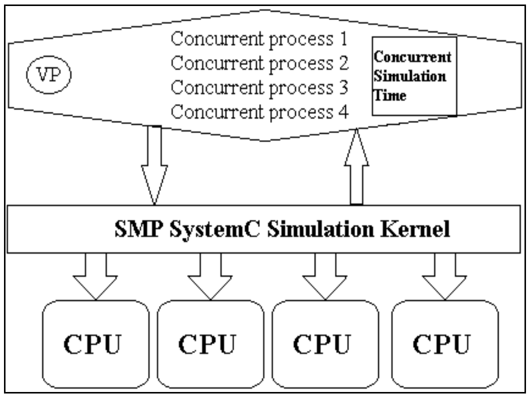


Figure 1: Simulation with Only Notion of Concurrency & Figure 2: True Concurrent Simulation

3. Proposed Solution

Implement the 3 stages of pipeline in the processor virtual model and simulating it on SystemC-SMP kernel improve the performance of the virtual system without any context switching overhead that is present when the pipeline is described in the sequential SystemC kernel. SystemC-SMP kernel exploit the underlying multi-core architecture of the host work station as it execute independent simulation processes on individual cores concurrently. 3-stage pipeline consists of fetch, decode and execute independent processes. To implement the logic of 3-stages, SystemC method process is used. Three independent SystemC method processes - fetch method, decode method, execute method are used. The concurrent algorithm for pipeline is simple to implement, at the time of simulation start, only fetch method is allowed to be executed by adding it in the runnable queue of the SystemC SMP kernel. Both decode method and execute method has been removed from the runnable queue by the use of don't initialize function. When executed fetch method creates the TLM-generic payload, which is used to interact with the rest of the virtual system and fetches the instruction from the memory virtual model, to be functionally accurate fetch operation, the address field of the payload is assigned the value present in the program counter register of the processor model. Before the execution of fetch finishes, a timed notification to the init-decode-event is performed, with time argument as one clock cycle. After the timed notification of event, the method performs a call to the next trigger function with argument as one clock cycle time period, so that after the simulation time increases by one clock cycle, fetch method can be executed again using the dynamic sensitivity, this finishes the execution of fetch method and there is no other process schedule in the runnable queue, thus, SystemC-SMP kernel scheduler advances the simulation time by the smallest value which can trigger the next scheduled event, that in this scenario is one clock cycle time period, so at simulation time t = t + 1, fetch method and decode method are both started concurrently, and they will get executed on independent individual cores of the host machine. The decode method also performs timed event notification to init-execute-event and a next trigger call with argument as one clock cycle time period. At simulation time t = t + 2, fetch method, decode method, and execute methods are all started concurrently, and they all will also get executed on independent individual cores of the host machine.
When this processor model is compared to a sequential SystemC based no pipelined processor model, which has no overhead of pipeline logic, we get the following results for different branching condensation level in the executed code.

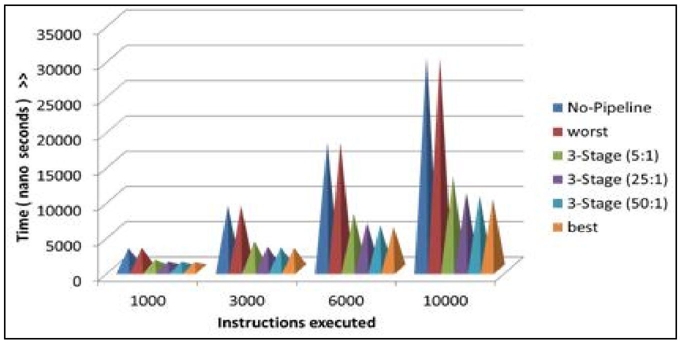


Figure 3: Pipelined processor model vs No pipelined processor model

4. Methodology

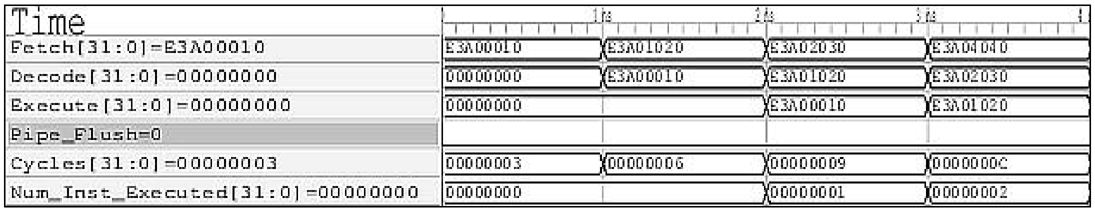
SystemC-SMP simulation kernel is used for simulation of 3-stage concurrent pipeline. Features of SMP-SystemC kernel include concurrent execution of methods using TBB multi-processor C++ class library. TBB extensions are the industry recommended methodology for implementation of concurrent program logic. Base SystemC is a sequential simulator which provides a notion of concurrency to the simulated processes but in actuality executes only one process at a time and uses one host process thread to execute its tasks on a single core, even on a multi-core work station. This results in large context switching overhead for concurrent notion processes. For the simulation of pipeline implemented in this paper, base SystemC simulation kernel is modified to implement TBB functions in its evaluation phase to provide the multi-core execution support for simulation processes. SystemC has two simulation processes – sc method, sc thread in SystemC SMP kernel, out of which only the method process is able to execute on multi core work station concurrently. Salient features of SystemC SMP kernel are:

- Uses TBB standard multi core C++ extension library.
- Does not change the stack of SystemC, keeping the simulator backward compatible.
- No shared variables are changed in kernel and hence it is safe from race conditions.

As the SMP SystemC is a work in progress project, it has the following limitation:
1. Only method simulation process can be executed concurrently.

To analyze the execution of the instructions on the ISS, we also require a value change trace tool - GTKWave, which creates and updates a graph with the values of all signals which it is tracing at the exact time of change event, As the ISS has provided signals to monitor the phases of instruction being executed, the instruction value is also displayed in the graph every time any change occurs.

For the 3 stage pipelined , the graph is given in Fig. 7, which displays that the instruction 0xE3A00010 being fetched at 0ns, at 1ns it goes into decode mode, and a new instruction 0xE3A01020 starts its fetch phase at the same time, at 2ns instruction-1 goes into execute phase, instructions-2 goes in decode phase and a new instruction-3 0xE3A02030 starts its fetch phase, and the processor virtual model continues the concurrent phases of individual instructions, providing the necessary concurrency of a three stage pipelined real processor in the corresponding virtual model.


```
Time                                      1ns       2ns       3ns
Fetch[31:0]=E3A00010                 E3A00010  E3A01020  E3A02030  E3A04040
Decode[31:0]=00000000                00000000  E3A00010  E3A01020  E3A02030
Execute[31:0]=00000000               00000000            E3A00010  E3A01020
Pipe_Flush=0
Cycles[31:0]=00000003                00000003  00000006  00000009  0000000C
Num_Inst_Executed[31:0]=00000000     00000000            00000001  00000002
```
Figure 4: Three Stage ISS execution value change dump trace graph

5. Conclusion

Analysis of the value change dump graphs generated by the GTKWave trace dump tool Fig. 4, shows that the pipelined processor virtual model executes the instructions concurrently in all three stages, and verifies the implementation of 3 stage pipeline in the processor virtual model, using the results given by the trace tool and the simulation execution times observed as given in Fig. 3, we can conclude that the newly implemented 3 stage pipelined processor virtual model had improved the performance and time efficiency of every instruction executed, and thus giving a functionally accurate base for simulation of processor intensive software at a higher speed, which may not be possible to execute otherwise until the time of silicon tape-out, this results in saving the expensive cost in terms of software readiness delays which happen due to slow processor virtual models simulating instructions sequentially.

6. References

1. Aline Mello , Isaac Maia , Alain Greiner , and Francois Pecheux, 2010 , “Parallel Simulation of SystemC TLM 2.0 Compliant MPSoCon SMP Workstations”.
2. Mitesh J. Limachia, 2Dr. Nikhil J. Kothari; 2011 “Modelling of ARM Cortex- M3 processor Core using SystemC ” Dept. of Electronics & Communication, D. D. University, Nadiad, Gujarat, India.
3. K.L. Man; 2005 “AN OVERVIEW OF SystemC FL ” Formal Methods Group, Department Of Mathematics and Computer Science, Eindhoven University of Technology the Netherlands.
4. Ed Harcourt, James Perconti; “A SystemC library for specifying pipeline abstractions” Department of Computer Science,1St. Lawrence University, Canton, NY, United States, Department of Computer Science, Northeastern University, Boston, MA, United States
5. Christiensen C. Arandilla, Joseph Bernard A. Constantino, Alvin Oliver M. Glove Anastacia P. Ballesil-Alvarez, Joy Alinda P. Reyes; 2010 “High-Level Implementation of the 5-Stage Pipelined ARM9TDM Core” Intel Microprocessors Laboratory, Microelectronics and Microprocessors Laboratory, Electrical and Electronics Engineering Institute, University of the Philippines - Diliman
6. SystemC Language reference manual 2.0
7. ARM Technical reference manual.
8. ARM Limited, ARM9TDMI Technical Reference Manual, Document No.DDI–0180A, 2000.

9. ARM Limited, Performance of the ARM9TDMITM and ARM9E-STM cores compared to the ARM7TDMITM core, White Paper. September 2000.
10. Azucena, N., et al., High Level Implementation of an ARM7 Microprocessor with Multi-Core Capabilities, University of the Philippines, Diliman, Department of Electrical and Electronics Engineering, March, 2006.
11. Furber, S., ARM System-on-Chip Architecture, 2nd Edition, USA: Addison-Wesley Professional, 2000.
12. Henessy, J.L. and D.A. Patterson, Computer Organization and Design:The Hardware and Software Interface, 3rd edition, San Francisco, California, Morgan Kaufmann Publishers, Inc, 2004.
13. Luna, A., et al, Implementation of the Complete ARM7TDMI-S Instruction Set on a Debug-Capable Core, University of the Philippines, Diliman, Department of Electrical and Electronics Engineering, October, 2008.
14. P. Schaumont, D. Ching, I. Verbauwhede, "An interactive codesign environment for domain-specific coprocessors," ACM Transactions on Design Automation for Embedded Systems, January 2006.
15. W. Qin, S. Rajagopalan, S. Malik, A Formal Concurrency Model Based Architecture Description Language for Synthesis of Software Development Tools, ACM 2004 Conference on Languages, Compilers, and Tools for Embedded Systems, June 2004, pp. 47-56.
16. Bastian Haetzer, Martin Radetzki; 2013"SystemC Transaction Level Modeling with Transaction Events"Embedded Systems Engineering Group University Stuttgart Pfaffenwaldring 5bD-70569 Stuttgart, Germany.
17. M. Burton, J. Aldis, R. Guenzel, and W. Klingauf, "Transaction Level Modeling: A reflection on what TLM is and how TLMs may be classified," in Proc. of the Forum on Specification and Design Languages (FDL '07), 2007.
18. Standard SystemC Language Reference Manual, IEEE Standard 1666-2011, IEEE Computer Society, January 2012.
19. S. Pasricha and N. Dutt, On-Chip Communication Architectures: System on Chip Interconnect, ser. Systems on Silicon. Elsevier Science, 2010.
20. S. Pasricha, N. Dutt, and M. Ben-Romdhane, "Extending the transaction level modeling approach for fast communication architecture exploration," in Proc. of the