HARNESSING JAVA FOR NOVEL CHIP MULTIPROCESSOR ARCHITECTURE SIMULATIONS

Horsnell, M. J. (horsnell@cs.man.ac.uk)
Advanced Processor Technologies Group, University of Manchester, Manchester, UK

Key words to describe the work: Simulation, Java, Chip Multiprocessor (CMP), Serialisation, Object-Oriented

Key Results: A simulation platform for microprocessor architectures written in Java. The simulation platform allows simulation at varying levels of architectural-model accuracy defined through object-oriented components, allows simulation state saving through Java language features, and can be *hot-started* to significantly decrease simulation time in complex operating environments, and increase productivity when debugging architectural components, the simulation platform, or compiler generated code targeted at the architecture.

How does the work advance the state-of-the-art?: The ability for a complex simulation platform to save state using regular Java language constructs, means that the effort required to save and *hot-start* the simulation platform is negligible allowing significant time savings for simulations in complex operating environments, for example Java code execution on top of a Virtual machine running on the simulator.

Motivation (problems addressed): As the number of transistors on a chip continues an upward trend towards Giga-scale integration, new design paradigms need to be considered to ensure efficient use of the increase in resources. Chip-multiprocessors are considered a forerunner amongst these paradigms. However, the design space for these architectures is vast, and simulation is the only viable approach to evaluating such architectures. The overall motivation of this work is the ability for increased simulation productivity using features of the Java language applied to microprocessor simulation. The work will help to assess possible designs for the next generation of microprocessors.

Introduction

A core topic of microprocessor research at present is the issues associated with Giga-scale integration[1] int architectures anticipated to feasible by 2013 [1]. These architectures provide an approximately 5-fold increase over current transistor budgets of today's largest microprocessors, however with this additional budget come new challenges in the design of next generation microprocessors.

The majority of current microprocessors designs employ increasing amounts of available transistors in building more dynamically scheduled, super-scalar and deeper pipelined uni-processors, in order to exploit more instruction level parallelism (ILP). The application of such design paradigms to future processors is likely to counter problems[2][3] such as wire-delay limitations, diminishing benefits from further ILP exploitation, and increased design time.

An alternative design approach, whereby multiple processing cores are integrated on a single die, chip-multiprocessor (CMP), is a viable forerunner. CMPs can reduce design time through component replication, overcome wire-delay limitations as the circuit is partitioned decreasing the critical length of global clock distribution, and can exploit thread level parallelism and speculative execution orthogonally to ILP in order to increase the overall throughput of the microprocessor.

1 Giga-Scale Integration (GSI) – integrated circuits containing upwards of $1x10^9$ transistors.

The design space for CMPs however is vast. Standard evaluation of processor pipelines, caches and memory configurations is supplemented by coherence, hierarchy, topology and scalability options which must also be assessed. The only viable approach to such assessment is through simulation.

Simulation is an essential tool in ensuring both functionality and performance of a new microprocessor design. The execution time of such CMP simulations can increase exponentially with the number of components in the system. As with any such simulation there is a trade-off between speed and accuracy. In some cases functional simulations may allow a novel feature to be evaluated, however cycle-accurate simulation may be necessary when testing the functionality of new coherence protocols or work distribution schemes where accurate timing is required. Compromises in scaling workloads and machine parameters can also be considered[4].

This paper presents several advantages of using Java as a language for constructing simulation tools for evaluating different chip multiprocessor architectures, in particular the use of *hot-starting* to remove constant simulation warm-up time overheads in complex operating environments.

Novel chip multiprocessor simulation using Java

The simulator being developed as part of this work advances the work carried out on other simulators[5] [6][7], by producing a platform which can be

reconfigured to test different architectural structures using previously built components, and also allows simulations to occur in parallel across distributed resources[8]. Recently the simulator has been extended to allow more efficient simulation through the application of several Java language features.

Variable Model Accuracy using Inheritance - As previously discussed there is often a need to run simulations at varying levels of accuracy. The simulation platform has been extended to provide a functional processor model, in addition to the cycle-accurate model produced previously. The functional model removes the complexities associated with pipeline hazards and delays, resulting in an approximately 10 times speed-up. This model was achieved through unification of code into a `Processor` superclass, from which both `CycleAccurate` and `Functional` are simply subclasses, with benefits coming from reuse of the common code and the ability to dynamically change the accuracy of the processor during the simulation if a critical stage needs to be evaluated with greater accuracy.

Hot-starts using Serialisation - Java, as a high-level object-oriented language allows almost automated serialisation of classes. When applied to a simulation platform this allows a rapid method by which the complete state of the simulation can be saved into a binary format with very little effort by the programmer. This binary file can later be reloaded to allow *hot-starting* of simulations.

The JAMAICA architecture[9] which is currently simulated on the platform presented, is being targeted by a complete port of the IBM Jikes Research VM[2]. This VM is bootstrapped into the simulator and initialises itself such that standard Java class files can be run on top of the VM on top of the JAMAICA architecture, used for research into the benefits of dynamic compilation on a CMP architecture[10]. This complex operating environment, takes approximately 2 billion clock cycles to get to the point where the simulator loads the Java test file.

Simulation Phase	Cycles	Execution Time (s)	
		Cycle-Accurate (~1.5 MIPS)	Functional (~12 MIPS)
Bootstrapping	~ 2.0×10^9	1333	166
Test file Execution.	~ 2.0×10^6	1.33	0.16

Table 1 Execution time spent in simulation phases

Table 1 shows approximate time taken to run the bootstrapping of the VM compared to the time taken to run a simple test class. If a batch of tests need to be run, it can be seen that the ability to *hot-start* (which takes less than 3 seconds) on the simulator considerably reduces the total time spent in simulation as it removes the bootstrapping phase.

Hot-start can also aid in complex debugging using the simulation platform, where tracking a bug in either the architectural model, the simulator or the executing code requires a large period of execution prior to the execution of the bug. The simulated state prior to the erroneous execution can be saved and resumed many times, until the problem is located without having to repeat the initial simulation phase. In the case of running a VM on top of the simulator where dynamic compilation may cause different in-lining of code and loading of classes into different areas in memory, exact state replication can also be achieved using this saved state.

Conclusion

Simulation is a necessary tool in the evaluation of the large design space of future GSI microprocessors. Application of some features in Java, and other high-level object-oriented languages can provide considerable benefits when requiring efficient productivity from simulation. In particular *hot-starting* can be used to resume simulation for efficient batch testing and debugging in complex environments.

Bibliography

[1] D. Burger, J.R. Goodman. *Billion-Transistor Architectures*, IEEE Computer, Sept. 1997, pp 46-49.

[2] M Bohr. *Interconnect scaling - the Real Limiter to High Performance ULSI.* In Proceedings of the International Electron Devices Meeting, pages 241 244. IEEE, IEEE Press, 1995.

[3] D. Matze. *Will Physical Scalability Sabotage Performance Gains?* IEEE Computer, pages 37-39, September 1997.

[4] D. Culler, J. Singh with A. Gupta. *Parallel Computer Architectures - A Hardware/Software Approach*, Morgan Kaufmann Publishers, 1999, pp 233-237.

[5] M. Chidester and A.George. *Parallel Simulation of Chip-Multiprocessor Architectures*, ACM Transactions on Modelling and Computer Simulation, 12(3), July 2003, pp 176-200.

[6] D. Burger and T. Austin. *The SimpleScalar Tool Set, Version 2.0*, University of Wisconsin-Madison, Computer Sciences Department Technical Report 1342, June 1997.

[7] C. Hughes, V. Pai, P. Ranganathan, S. Adve. *Rsim: Simulating Shared-Memory Multiprocessors with ILP Processors*, IEEE Computer, February 2002, pp 40-49.

[8] M. Horsnell. *Cycle-Accurate Distributed Chip Multiprocessor Simulation*, Proceedings of PREP2004,

[9] G. Wright. *A single-chip multiprocessor architecture with hardware thread support*, Ph.D. Thesis, University of Manchester, January 2001.

[10] I. Watson, A. Dinn, C. Kirkham et al. *An Investigation of Dynamic Compilation for Parallelism in a Single-Chip Multiprocessor.* Technical report, University of Manchester, 2003.

2 VM – Virtual Machine (here referring to a Java VM).