Rapid Architectural Design and Validation Using Program-Driven Simulations

Andrew Chien  Ali Dasdan  Rajesh K. Gupta  Ben Zhang

Department of Computer Science
University of Illinois, Urbana-Champaign
1304 W. Springfield Avenue, Urbana, Illinois 61801.

Abstract
We describe key features of a simulation environment, MORPHSIM, that allows us to quickly evaluate the effect of architectural changes on application runtimes. We discuss the effect of runtime interpretation, and describe how the MORPHSIM simulator is able to achieve significant speedup in architectural evaluation by minimizing the use of interpretation over a program-driven simulation.

This environment is currently being used to design a customizable system architecture built using reconfigurable hardware blocks [1]. We present early simulation results that highlight the utility of the simulation environment in system design.

1 Extended Summary

Recent availability of processor, memory, DSP, and special-purpose ‘cores’ have made it possible to assemble complex systems using these cores and using reprogrammable glue logic on a single chip or substrate [2, 3]. High-speed system-level simulation is a necessary prerequisite for correct and efficient design of these highly integrated systems. Since these systems use general-purpose computing elements, correct system functionality can often be tested by running existing target software applications. In order to test these applications, however, the system designer must be able retarget the application to the new machine’s architecture, and execute it along with the application-specific hardware assists and cores. This requires an integrated simulation environment where detailed architectural models consisting of hardware and software components can be built and simulated for performance on new and existing applications. Currently, large scale system design at the highest level of abstraction is done using the programming language C or C++ on simulators such as Tango [4] and MINT [5]. To obtain a meaningful evaluation of system designs, simulation speed is of utmost importance [6]. To improve simulation speed, tradeoffs against simulation accuracy, hardware and language assists have been tried [7, 8].

In general, almost any simulation framework that supports the notion of events and provides mechanisms to process these events, can be used to build simulatable system prototypes. This
is even more true of modeling reactive hardware systems. Indeed, most existing HDLs can be classified as source languages for a corresponding event-driven simulator [9]. An event-driven model is powerful enough to describe most hardware systems at any level of abstraction: from algorithms to gate-level circuits [10]. However, this generality also imposes a significant burden on the simulation efficiency due to the extra work (or overhead) needed in event maintenance and processing. Event processing often requires interpretation of event generation, propagation and disposition by the simulation model. “Interpretation” of an object in a simulation model refers to the evaluation of the object by a separate procedure that provides semantic interpretation of the object in the context of the simulation model. This can be done either statically (at compile time) or at runtime. Runtime interpretation is done by a (separate) module that holds the state of execution and it is invoked whenever the interpretation is needed. This is in contrast to a native execution where the host machine (machine running simulation) is used to hold the state of execution. Since each call to the interpreter may require a context-switch (to another simulated context for instance), the tradeoff between native versus interpretation is dictated by the cost of execution versus the cost of context switches (needed for interpretation). Mixed hardware/software simulations are slowed down due to their interface with an event-driven hardware simulator. Recent efforts in this direction (e.g., [11]) have demonstrated the utility of simulation backplanes in integrating various simulators. Though the achieved simulation speeds are not mentioned it is unlikely that hybrid machine simulations can be used to run moderately large application benchmarks which require simulation efficiencies upwards of 100,000 simulated cycles per second.

A critical bottleneck in achieving higher simulation speeds stems from the integration of hardware description language (HDL) models. The HDL simulations often require operation-level interpretation, that is, each operation (statement) requires a call to the interpreter. This is because, each signal assignment statement, in a language such as VHDL, can potentially generate an event, therefore, the system simulations are significantly slowed by frequent calls to the interpreter (or the event manager). An alternative is to build models that work with a cycle-based simulation. A cycle-based simulation, though not necessarily efficient in terms of work required, is often able to use native execution (against software interpretation) to achieve significant reductions in simulation time. Cycle-based simulation using conditional control flow in high-level programming languages has been used to demonstrate speedup in gate-level hardware simulations [12]. Since in these simulations the main simulation loop is based on per cycle, only synchronous systems can be simulated. Clearly, the circuit latency can not be used to improve simulation efficiency as in the case of event-driven simulations.

Our approach to system simulation uses a combination of interpretation and native code execution such that the native codes are encapsulated into non-event producing blocks and interpretation is done at a coarse-level. This is done by separating the two very different time-scales of concern related to system simulation and hardware verification as shown in Figure 1. Cycle-based system-level simulation is uses a modified program-driven simulator MINT [5] that performs the system simulation as the application executes. This simulator consists of a “front end” and a “back end.”
The front end represents interface to the application program, whereas the back end models the underlying micro-architecture. The front end handles the application program execution. When the program execution leads to generation of event (either through reference to modeled blocks in MINT or through directives embedded in the source program), the front ends sends the event to the back end. As an example, the MORPH design focus is on system interconnect and memory hierarchy. For this design, typical events are memory references and synchronization operations such as locks and barriers. When the operations associated with an event complete, the back end returns with a status signal to either block, abort or continue execution depending the event semantics. The underlying simulation library provides event processing and disposition. As an example, consider interconnection to a processor core with a peripheral on an on-chip system bus. A read event generated by the processor will be processed by executing a corresponding procedure call (such as sim_read()). The back end defines this function, and can do almost anything inside this function representing its handling by the corresponding dedicated hardware block. For instance it can call new functions, create new events, etc. This allows us to conveniently customize system simulations for specific hardware blocks. Due to the high-level synchronous nature of the system level simulations, the hardware blocks are modeled at the behavioral level while detailed implementations are only considered in hardware validation.

The compiled simulator (based on MINT) can interpret almost any program that runs natively, and can generate a events specific to any hardware block. The original design of MINT was used to generate events representing references to (cache) memory blocks. However, with appropriate definition of an event, the system simulator can be used to address references to any hardware block. This way the MORPHSIM simulation environment allows us to associate events and reduce runtime
interpretation to those only on the blocks that are under design. The back end is customized to reflect the underlying system micro-architecture. In particular, in MORPH design we are interested in (reconfigurable) hardware assists such as context size recognizer, get (prefetch) and put (forward) hardware blocks. For architectural blocks that can be identified in the compiled simulator, the applications do not need to be preprocessed or recompiled. (The original MINT simulator interprets the applications that are compiled and linked on a MIPS-based machine. Variants of MINT for other Intel and HP architectures are also commonly available.)

Hardware validation in MORPHSIM is done using traditional event-driven simulation on hardware blocks. The input to these simulations is derived from translation of C models of hardware into HardwareC blocks. These blocks are subject to presynthesis optimizations based on Timed Decision Tables [13]. (Currently this translation is done manually.) The hardware validation does not require on-line application executions. Instead, the results from application executions are used to create a test-bench for the event-driven hardware validation. This reduces the redundancy in detailed hardware simulations (for the purposes of hardware validation), while modeling its effect on application-level simulation statistics.

To test the usefulness of MORPHSIM, we have implemented the MORPH micro-architecture in the simulator back end that models the memory hierarchy of a uniprocessor. This hierarchy consists of a L1 data cache, a L2 data cache, and the main memory. The L2 cache can be configured, using a programmable a programmable interconnect, as either shared or private among on-chip functional blocks. Four application kernels and two complete application benchmarks [14] were compiled using standard C compiler for simulation on the MORPH architecture. Table 1 reports (sample results for a specific machine configuration) on the number instructions simulated, cycles used and CPU time on an Silicon Graphics power challenge machine.

Results indicate system simulation speeds from 300,000 to over a million instruction per second for machine models in high-level C programs. These simulations use only an abstract view of the hardware blocks and thus are limited in comparing detailed designs of these blocks. However, these

<table>
<thead>
<tr>
<th>Application Kernels</th>
<th>Instructions</th>
<th>Cycles</th>
<th>CPU (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jacobi</td>
<td>2,097,183,413</td>
<td>10,228,705,01</td>
<td>5859</td>
</tr>
<tr>
<td>SAXPY</td>
<td>22,001,056</td>
<td>74,508,776</td>
<td>39,18</td>
</tr>
<tr>
<td>Gather-scatter</td>
<td>32,751,056</td>
<td>313,913,456</td>
<td>98</td>
</tr>
<tr>
<td>Large-stride</td>
<td>14,023,561</td>
<td>14,588,60</td>
<td>18.63</td>
</tr>
<tr>
<td>Complete Applications</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MP3D</td>
<td>138,577,960</td>
<td>202,259,040</td>
<td>304</td>
</tr>
<tr>
<td>Sparse</td>
<td>115,729,491</td>
<td>439,555,171</td>
<td>380</td>
</tr>
</tbody>
</table>

Table 1: System-level Simulations on MORPH
simulations are very useful in making application-driven machine design decisions.

1.1 Conclusions

This extended summary presents results on an on-going project on high-speed system simulations that use reconfigurable hardware and CAD tools to customize machine micro-architecture for a given application statically or at runtime. The architectural design of this machine requires machine fast system simulations. We have addressed the issue of simulation efficiency by focusing on the event processing and the required “runtime interpretation” for correct system modeling. Our preliminary results using MORPHSIM indicate that significant reductions in simulation time are possible by decreasing the event-processing overhead and by reducing the runtime interpretation to the set of events that is directly influenced by the hardware design in question. This simulation customization can be done based on the notion of an event interface to a compiled simulator, which our case is built using a modified MINT multiprocessor simulator. Using MORPHSIM we are able to carry out program-driven simulations of new machine architectures that does not require program instrumentation or expensive trace-collection. We are currently developing application-specific hardware assists to be incorporated in the MORPHSIM micro-architectural models. The workshop presentation will include the results on the simulation efficiency comparisons against traditional discrete event simulations.

References


