Automatic Performance Model Synthesis From Hardware Verification Models

Robert H. Bell, Jr., Matyas Sustik, David W. Cummings, Jonathan R. Jackson
IBM Systems and Technology Group
Austin, Texas
{robbell, sustik, dwcummin, jrj1}@us.ibm.com

ABSTRACT

Performance models are typically written by hand for a new model or assembled piece-meal from the prior simulation code of an old model. In either case, many man-months of work may be required to write the new model and validate design details against a prior or current design. In reality, the majority of information about the performance of the design already exists in the design structure of either the old hardware model or the new model or both.

To harvest this information and eliminate the significant duplicate coding and validation efforts, we propose that a performance model be automatically synthesized from a prior or current hardware design using a bottom-up, design-oriented approach. We demarcate the performance-critical boundaries of the design and perform backward-trace cone analysis to identify logic to include in the performance model. We then abstract specific components for design changes and expend modeling effort only on the few functions relevant to a particular design study. Engineering effort then becomes focused on workload selection and quality, defining and projecting new designs, and assessing design tradeoffs and sensitivities – the small set of tasks with the highest potential to improve design performance.

We present a case-study that shows that even the simplest proposed transformations on a high-performance IBM L2 cache design result in a simulation speedup of 3.9, with evidence that an order of magnitude speedup can be obtained using a few additional modeling abstractions.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques

General Terms

Performance, Design, Verification, Hardware Acceleration

Keywords

Benchmarking, Performance Modeling, Hardware Description Languages, Hardware Acceleration

ICPE '10, March 14-16, 2011, Karlsruhe, Germany.

1. INTRODUCTION

Performance models that are developed to assess computer performance in the early stages of design are traditionally written by hand or assembled piece-meal from prior code long before actual hardware is available. Development is generally *top-down*, starting with analysis of overall instruction flow and finishing with detailed microarchitectural tradeoffs. In a development environment, a new hardware design is usually based on an old prior design, and, likewise, the associated old performance model is enlisted for the new modeling effort, but only *after* the old model has been validated against the old hardware using a functional model compiled from a hardware description language (HDL) or by executing on a physical machine if it is available [4].

In general, validation with hardware alone is difficult due to limited performance monitor events, the coarse granularity of the logical functions expressed by events, or imprecise counts, so usually validation against an HDL functional simulation model is employed. Experience shows that instruction latencies and throughput through serial logic components of the system must be correctly understood and accurately modeled, which calls for direct compare with internal random logic macro (RLM) and logical unit signals, the details of which may only be available in a HDL functional model.

Validation of a hand-written performance model using an HDL model or hardware is a laborious and error prone process [5, 7, 4]. On a large, custom microprocessor design, many man-months may be expended validating elements of the performance model against the hardware. Hand-coded microbenchmarks [5] or synthetic codes [14] may be used to quickly validate specific instruction sequences. Automatic benchmark synthesis improves the process by reducing the number of instructions to be executed while assuring the representativeness of the codes [3, 4]. Hardware acceleration systems built from FPGA arrays or custom hardware may be used to speedup processor simulations [4]. Despite these techniques, HDL model simulation speeds may be orders of magnitude slower than native hardware execution [4].

In this paper, we show that additional simulation speedups may be obtained by reducing the size and complexity of the HDL functional model itself. The implication is that a useful performance model may be created by transforming a functional HDL model into a reduced model that executes more efficiently. This process of functional model *reduction* is a form of performance model synthesis in which the design logic and structures that are not performance-critical are identified and pruned from the synthetic performance model. This *bottom-up, design-oriented* performance model synthesis starts with the

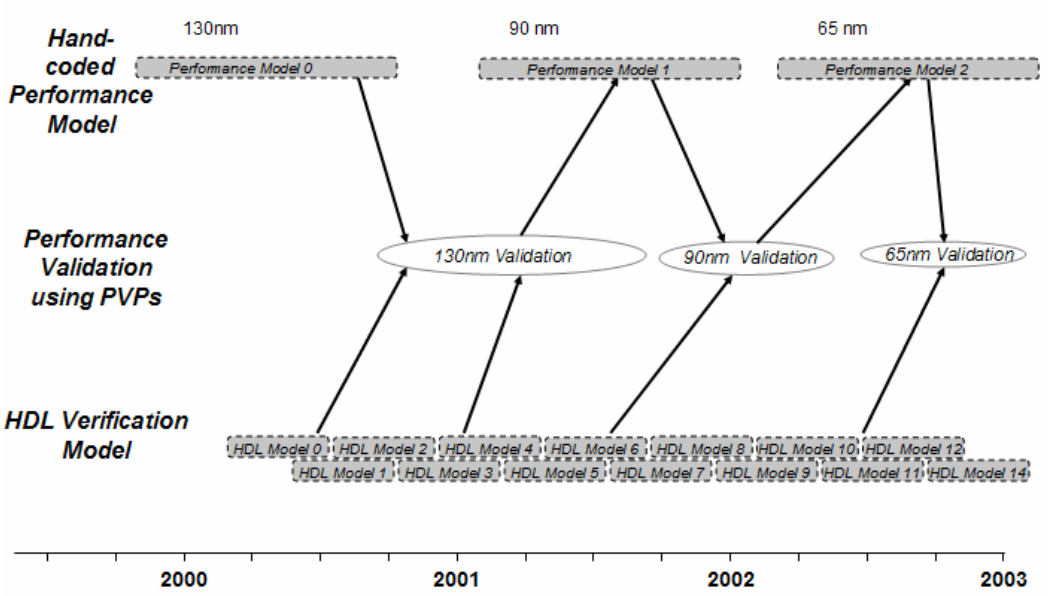

Figure 1 : Typical Hand-coded Performance Model Development Timeline

low-level logic blocks of a fully-functional design and ends with a smaller control network that is capable of obtaining representative performance running real workloads. Also available are techniques specific for modeling data-dependent design performance and design space exploration.

Significant benefits accrue from such a famework. Since HDL models are frequently required for functional verification as the design development proceeds, *every* verified HDL model is potentially a more accurate performance model, in lockstep with the design itself, and simulation of the synthesized model – with only a few limitations – may proceed on the same simulation platforms as the design verification, consolidating verification and simulation resources. Automatic model synthesis relaxes the need for hand-coded performance models which, in turn, eliminates or reduces laborious and expensive model validations against hardware. Engineering effort then becomes focused on workload selection and quality, defining and projecting new designs, and assessing design tradeoffs and sensitivities – the small set of tasks with the highest potential to improve design performance.

Essentially, we suggest that the HDL hardware model and the performance model, including model compilation, analysis and simulation, are two sides of the same coin. To our knowledge, this is the first case study to combine a functional verification environment with performance model synthesis.

The rest of this paper is organized as follows. Section 2 describes the performance model synthesis approach. Section 3 presents our case study, experimental results and current status. Section 4 presents related work, and the final sections present conclusions, future work and references.

2. PERFORMANCE MODEL SYNTHESIS

To be automatic and current, a performance model must be synthesized from the most recent HDL model, the same model required for functional verification of the design.

Figure 1 shows a typical performance model development effort for a real-world billion-transistor microprocessor. The writing of the initial performance model (version "0") begins before the first HDL model (also version "0") has been compiled from the design and verified. Once both performance and HDL models are in a relatively stable state, microbenchmarks are executed on performance model 0 and on HDL model 0, and any performance differences are resolved. Meanwhile, HDL models 1 through 3 have been generated by the design team as they check in code to fix bugs and add function, prompting more validations. There is a large lag between the start of the validation process and the perception that the performance model is accurate and can project design tradeoffs correctly. If the lag is too long, the final design changes may go into the design before the performance model is ready to assess them.

Eventually, the performance team codes the model for the next pass design. When the performance model and HDL design again reach stable states, another validation effort is undertaken, again paralleling several iterations of new designs. Eventually, code changes stabilize and validations become easier, but each

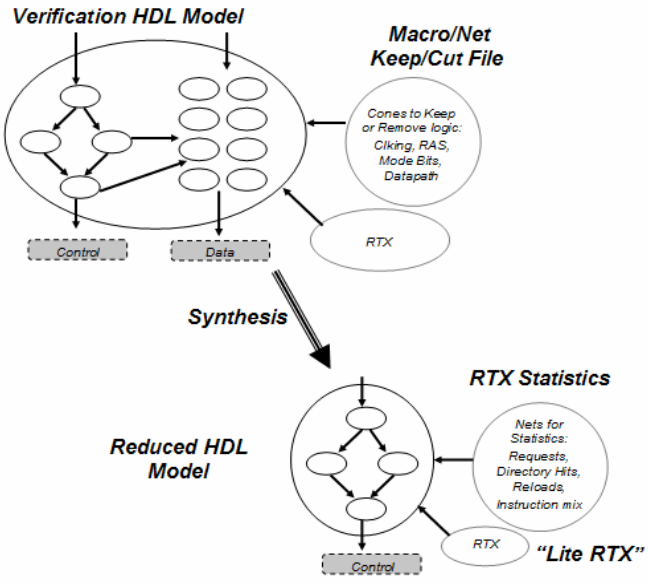

Figure 2: Performance Model Synthesis Overview

effort is still a lengthy process. The *design-oriented* approach described below merges these efforts such that each verified HDL model may be automatically processed into an equivalent performance model. Note that design verification is required before manufacture, and therefore a performance model generated from a verified model is practically already validated.

2.1 Model Synthesis Overview

Figure 2 shows an overview of performance model synthesis. The verification model in the upper left of Figure 2 is a complex jumble of logic blocks and associated RLM interfaces that have been compiled from an HDL design and are ready to be simulated in a verification environment [10], labeled "RTX". We then verify that real workloads execute on the design in the verification framework.

A design can be perceived from its several dimensions of functionality, but the two most significant aspects of a design are its control and data paths. The throughput of a modern machine is generally determined by the control paths independent of the data being used. In addition to the data path, many other structures such as those related to reliability, availability and serviceability (RAS), test and bring up, debug switches, distributed clock buffers, and error checking and handling have no impact on machine performance and can be removed for the purposes of building the synthetic performance model. In addition, depending on the type of performance model desired, such as a basic model or specific unit test model, we may remove large chunks of logic, eg. whole units, accelerator functions, I/O units, non-cacheable instruction units, coherency and synchronization operation logic, bus and memory prediction structures, etc. In the case of a memory-subsystem only model, entire cores of a chip multiprocessor may be pruned out.

We manually maintain a file of Keep/Cut nets for the design, or, equivalently, annotate the HDL nets as necessary, so as to define the set of nodes that will be preserved in or pruned from the design. To enable high-level abstraction and design studies (discussed below), we preserve the RLM and unit boundary names in the final compiled HDL model, unless the entire RLM can be removed in the synthesis process. As shown in Figure 2, nets are also annotated for use by RTX control input drivers, statistics collection, and output checker verification.

Figure 3 shows a simple example of the synthesis process. In this case the Keep/Cut file or net annotations specify *cuts* at

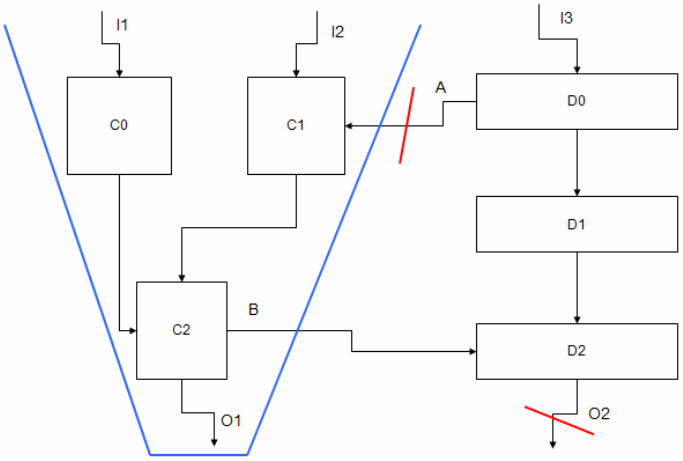

Figure 3: Cut and Keep sets example. Datapath signals A and O2 are in the Cut set. Control signal O1 is in the Keep set. After transformation, the control paths C0, C1 and C2 along with control inputs I1 and I2 remain, and the datapath, D0-D2 and I3 are gone.

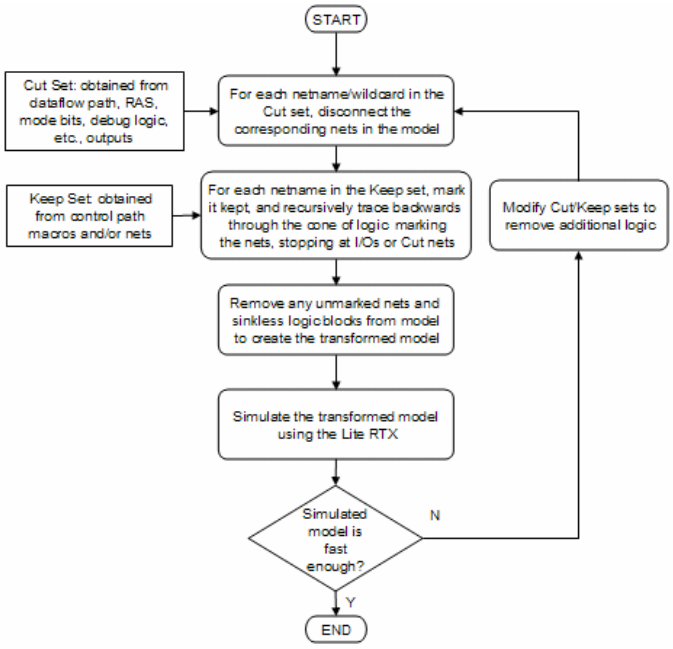

Figure 4: Model Reduction and Simulation Process

points A and O2, and a *keep* for output O1. The process then starts at the kept outputs and carries out a backwards cone trace to inputs or cut nodes, marking as *kept* each node encountered. When the cone analysis is finished, all unmarked nodes are removed from the model.

Figure 4 shows a flowchart of the overall synthesis process. Analogously to how design logic is removed in the cone analysis, much of the RTX simulation environment may also be removed. The "Lite-RTX" no longer contains drivers and checkers for the datapath, RAS, error handlers, bring up, etc., and therefore is much smaller and faster.

If the synthesized performance model executes too slowly, the Keep/Cut file may be augmented to further reduce the performance model and RTX environment and thereby speed up simulation run time more. Units or specialized functions may be targeted, or nest or even unit performance models synthesized. When the Lite-RTX monitors a chip model, it may do very little other than check inputs and drive control grants for a memory behavioral or check for the proper control signals on unit interfaces as execution-driven workload instructions exercise the model control paths. For a unit or nest model, it may provide input driver stimulus from a performance verification pattern (PVP) file or random pattern generator and check control signals on outputs. All of these functions would exist already and separately for the control interfaces in a properly-partitioned verification environment [10]. In any case, the performance of the synthesized performance model – in the form of the cycle-accurate statistics monitored by the RTX - is equivalent to the actual design performance at the desired level of model size and complexity, *without* the need for additional laborious simulations and validations.

2.2 Model Synthesis for Throughput Studies

Figure 5 shows an example of a trace-driven synthesized performance model for a core-nest design. Since the data paths have been removed from the model by the synthesis process, the instruction trace consisting of addresses without any data feeds

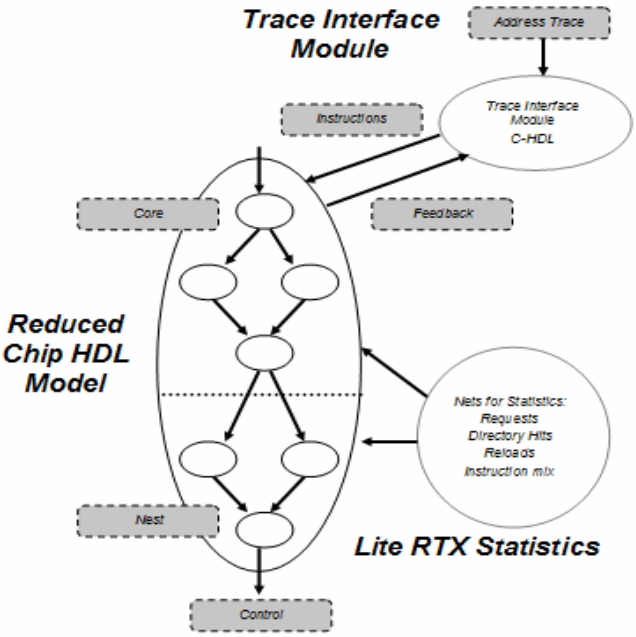

Figure 5: System Throughput Performance Model Overview

naturally into the control functions of a throughput model consisting of a reduced core and memory subsystem. The Lite-RTX supports a DIMM memory behavioral and tracks system performance but otherwise is required to do very little, which improves simulation time. The Trace Interface Module formats the trace into model-readable addresses and opcodes, and may be written in high-level C-code or in HDL itself for compilation and linkage into the performance model. The IBM HDL simulation model permits linkage of VHDL compiled code with high-level C-compiled code as long as the interface signal definitions match and clocking semantics are understood [10].

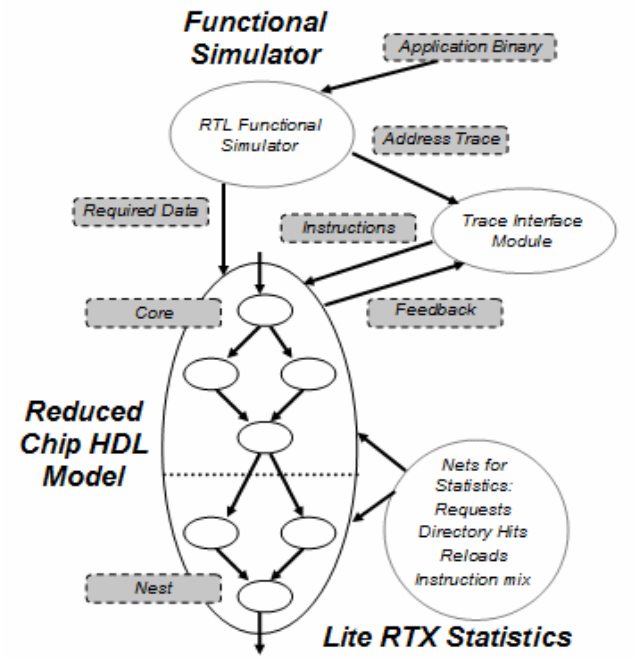

Figure 6: System Performance Model with Data

2.3 Extensions for Data-Dependent Models

In modern processor designs, there are several cases in which system performance is dependent on the particular data set being processed. Examples include value prediction, bus power reduction techniques, barriers, locking and synchronization of multiprocessor threads. These structures can still be studied using the reduced HDL model.

Figure 6 gives an example of extending the model to support data-dependent performance simulation. A fast, register transfer level (RTL) simulation of the application binary is executed in parallel with the synthetic performance model. The RTL simulator either interfaces to the trace formatter or formats the instructions for the model itself, and it also supplies the required data for instructions that are data-dependent. In the PowerPC ISA, for example, the core HDL that compares a cached data value with a reservation register value for a STCX instruction would be provided with the data associated with the instruction address, so that the correct locking behavior would ensue. In the case of value prediction, a datum retrieved from memory could be compared to a value previously predicted for an execution unit calculation, which could lead to instruction rollbacks and predictor updates. In the throughput case, data is not needed to project performance, but for accurate multi-threaded simulation, synchronization and locking may come to dominate performance [8].

2.4 Design Space Exploration

With a framework as described above, an accurate performance model can be synthesized from a functional HDL model. But in addition to projecting the performance of a design, performance models are required to carry out design space explorations. Figure 7 depicts a reduced HDL model with one RLM abstracted from low-level HDL to high-level C-code, with a final transformation of the C-code to new design functionality for a design study. As alluded to earlier, it is a good idea to generously annotate the boundaries of units or logic function in the HLD in order to enable design studies. The performance engineer writes a high-level C-code replacement for a unit or function, compiles and links it with the rest of the compiled C-HDL complex, and quantifies its performance by

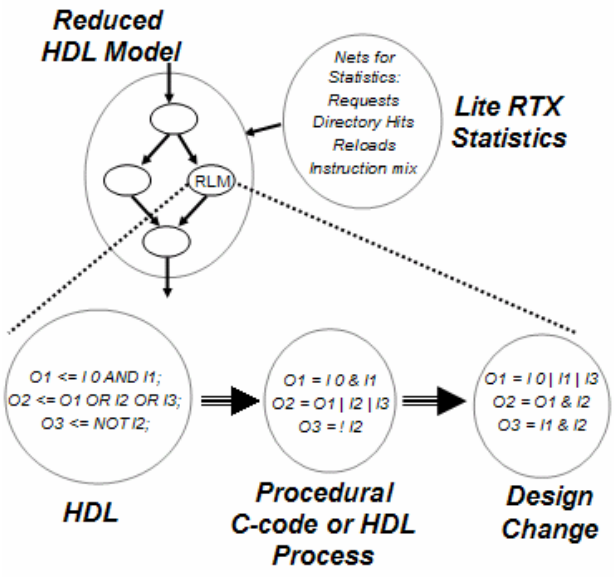

Figure 7: Model Synthesis with Design Space Exploration

simulating workloads on it. He then replaces the high-level function with the new design and compares the performance.

2.5 Model Validation Elimination

As described above, performance model synthesis virtually *eliminates* the validation effort. In the traditional methodology, the performance engineer must validate the old performance model against the old hardware, update the model for the new design, project performance, and, when available many months later, hand-validate the new hardware against the new model. In the new methodology, the performance model is synthesized such that either old or new model is automatically valid versus the target design. Synthesis based on an old design and model can be carried out in order to confirm a correct synthesis methodology prior to synthesizing the new model.

For design studies, the engineer abstracts and changes just the few units or functions necessary for a particular design study. RLM boundaries rarely change, so that as new designs are generated, new performance models can be synthesized and similarly modified by linking in previously-written high-level code. Whereas in the past the vast majority of effort and resources went to hand-code and/or validate a base performance model for an old or new design, now the performance engineer puts effort *only* into workload selection and quality, defining and projecting new designs, and assessing tradeoffs. This results in a more productive performance engineer focused on the tasks with the most impact on design improvement.

3. MODEL SYNTHESIS CASE STUDY

The performance model synthesis process was applied to the L2 unit of the POWER6 microprocessor. The POWER6 L2 is a 4 MB, 8-way set-associative, store-through, unified data and instruction cache consuming a significant portion of the 341mm^2 of the POWER6 chip die [9]. It contains 32 read-claim machines, 8 castout machines, 8 bus snoop machines and runs at half the frequency of the high-performance 5GHz+ frequency core [9]. The L2 possesses well over one hundred unique RLMs and custom logic blocks that comprise the L2 directory and cache control, L3 castout control, bus snoop logic, error correction and detection logic, distributed clock buffers, datapath control, and coherency management and control.

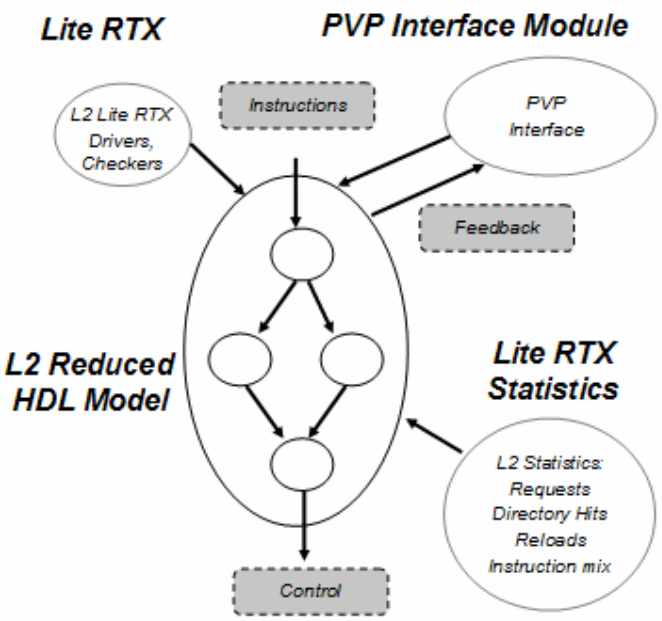

Figure 8: L2-Only Performance Model Synthesis Overview

3.1 Experimental Setup

The L2 VHDL compilation into an IBM HDL simulation model for design verification replaces the directory SRAM cells with high-level RTX behaviorals, which were retained for performance model synthesis. The chip RTX was already well partitioned into separate drivers and checkers for control and datapath logic, which simplified development of the L2 Lite-RTX as well as for mapping the unit and cache SRAM I/Os and structures to nodes in the Keep/Cut file for model reduction. Other nets internal to the L2 VHDL on datapath boundaries were identified, annotated in the VHDL before compilation if necessary, and placed in the Cut file. We focused on synthesizing a model for throughput studies, so structures and signals in the retained RLMs that rely on particular values for reservation logic or coherency logic were tied to non-controlling values using explicit assertions on annotated nodes.

The resulting Keep/Cut file was used for cone analysis and pruning of the compiled L2 design. Of the unique RLM classes in the design, about half operated only on the datapath and were completely removed in the synthesis process. Well over two-thirds of the total nodes in the original design were removed.

Figure 8 shows the resulting synthetic L2 performance model simulation environment. The Lite-RTX can drive random loads and store addresses into the control paths and check for the proper addresses and store ordering on the directory, cache, and bus interfaces, and it can drive the resulting grants and acknowledgements back into the logic. As an alternative, a unit PVP interface can drive specific sequences of strided addresses and bursty traffic onto the interfaces, with the Lite-RTX continuing to check ordering and drive expected return flow.

3.2 Experimental Results and Status

Experiments on random and PVP workloads show that the synthesized performance model achieves equivalent performance, including throughput, while experiencing a 3.9x speedup in simulation time versus the original full L2 hardware model. We hypothesize that an additional 2x speedup or more may be obtained by abstracting the latch-clocking framework and simulating latches logically, the so-called single-phase clock simulation that is supported in the IBM HDL simulation environment. For our experiments, we ran the simulation environment without this feature, simulating both up and down phases of clocks at the inputs of latches to accurately simulate various latch design styles, such as L1-only or L2-only cycle-stealing designs.

In addition, the current environment simulates both up and down edges of clocks and signals in time to correctly handle the special case of pulse generators. An additional speedup may be obtained by removing clock generators and simulating logical single-clock the design, also supported in the IBM HDL simulation environment.

Overall, current results indicate that an order of magnitude speedup over the base HDL model simulation is quite possible, even without high-level abstraction of logic function. With high-level abstraction as shown in Figure 7 and described in Section 2, simulation runtimes would improve further by consolidating RLMs, simplifying control functions, and removing duplication

of function in different parts of the design.

4. RELATED WORK

There has been much work on synthesizing hardware designs from high-level descriptions or co-design based on performance descriptions (eg. [11]) but not on the reverse process of synthesizing a performance model from a hardware design. However, the need for more automatic performance model synthesis from a design, especially for large system-on-chip designs, is well-established [2]. Darringer, et al. [2] recommend a top-down approach in which the system design is mapped into a performance model either at a low-level for accuracy or at a high-level in C++ for simulation speed.

Performance model functions can be mapped to design blocks to ease model validation. Pimentel et al. [13] calibrate the inter-component latencies in system-level simulation models based on static latency tables and dynamically perturb the system latencies to better match expected latencies.

Software and compiler studies can benefit from inferring processor performance at a high-level. Cavazos et al. [6] predict machine performance for compiler development using execution results of a small set of benchmarks operated on by some compiler transformations. Similarly, Augonnet et al. [1] calibrate high-level models used for estimating performance of code-scheduling algorithms in multi-processor pipelines. At a higher level, Nurmi et al. [12] schedule workflow tasks in a grid based on component simulation results on specific machines. These techniques are suitable for fast performance estimation at a static design point but are not appropriate for detailed processor design studies.

5. CONCLUSIONS AND FUTURE WORK

Most of the information necessary for accurate performance simulation and design studies already exists in old or new hardware models compiled for the purposes of functional verification. We propose that a performance model be automatically synthesized from a prior or current hardware design by demarcating the performance-critical boundaries of the design and performing cone analysis to identify the logic to include in the performance model, a bottom-up, design-oriented approach. We then abstract components for design changes and expend modeling effort on only the few functions relevant to a particular design study. Engineering effort focuses on workload selection and quality, defining and projecting new designs, and assessing design tradeoffs and sensitivities – the small set of tasks with the highest potential to improve design performance.

We provide a case-study showing that even the simplest proposed transformations on a high-performance IBM L2 cache design result in a simulation speedup of 3.9, with evidence that an order of magnitude speedup can be obtained using a few additional logical abstractions. Future work will focus on extending the methodology to nest and system models, and incorporating a functional simulator to project data-dependent performance.

6. REFERENCES

[1] C. Augonnet, S. Thibault and R. Namyst, "Automatic Calibration of Performance Models on Heterogenous ulticore Architectures," *Proceedings of the 3rd Workshop on Highly-Parallel Processing on a Chip*, August 25, 2009.

[2] J. A. Darringer, R. A. Bergamaschi, S. Bhattacharya, D. Brand, A. Herkersdorf, J. K. Morrell, I. I. Nair, O. Sagmeister, and Y. Shin, "Early Analysis Tools For System-On-Chip Design," *IBM. J. Res. & Dev.*, Vol. 46, No. 6, November 2002, pp. 691-707.

[3] R. H. Bell, Jr. and L. K. John, "Improved Automatic Testcase Synthesis for Performance Model Validation," *International Conference on Supercomputing (ICS)*, June 20-23, 2005.

[4] R H. Bell, Jr., R. R. Bhatia, L. K. John, J. Stuecheli, J Griswell, L. Capps, A. Blanchard, R. Thai, "Automatic Testcase Synthesis and Performance Model Validation for High-Performance PowerPC Processors," *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, March 19-21, 2006.

[5] B. Black and J. P. Shen, "Calibration of Microprocessor Performance Models," *IEEE Computer*, May 1998, pp. 59-65.

[6] J. Cavazos, C. Dubach, F. Agakov, E. Bonilla, M. F. P. O'Boyle, G. Fursin, and O. Temam, "Automatic Performance Model Construction for the Fast Software Exploration of New Hardware Designs," *International Conference on Compilers, Archtectures and Synthesis for Embedded Systems*, October 23-25, 2006.

[7] R. Desikan, D. Burger and S. Keckler, "Measuring Experimental Error in Microprocessor Simulation," *International Symposium on Computer Architecture (ISCA)*, 2001.

[8] C. Hughes and T. Li, "Accelerating Multi-Core Processor Design Space Evaluation using Automatic Multi-Threaded Workload Synthesis," *International Symposium on Workoad Characterization (IISWC)*, September 2008.

[9] H. Le, W. J. Starke, J. S. Fields, S. J. O'Connell, D. Q. Nguyen, B. J. Ronchetti, W. M. Sauer, E. M. Schwarz and M. T. Vaden, "IBM POWER6 Microarchitecture," *IBM J. Res. & Dev.*, Vol. 51, No. 6, November 2007, pp. 639-662.

[10] J. M. Ludden, et al., "Functional Verification of the Power4 Microprocessor and the Power4 Multiprocessor Systems," *IBM J. Res. & Dev.*, Vol. 46, 2002, pp. 53-76.

[11] P. Mishra, A. Kajariwal, and N. Dutt, "Rapid Exploration of Pipelined Processors Through Automatic Generation of Synthesizable RTL Models," *Proceedings of the 14th IEEE Workshop on Rapid Systems Prototyping*, 2003.

[12] D. Nurmi, A. Mandal, J. Brevik, C. Koelbel, R. Wolski and K. Kennedy, "Evaluation of a Workload Scheduler using Integrated Performance Modeling and Batch Queue Wait Time Prediction," *International Conference for High Performance Computing, Networks, Storage and Analysis (Supercomputing)*, November 2006.

[13] A. D. Pimentel, M. Thompson, S. Polstra, and C. Erbas, "Calibration of Abstract Performance Models for System-Level Design Space Exploration," *J. of Signal Processing Systems*, Vol. 50, No. 2, February 2008.

[14] R. P. Weiker, "Dhrystone: A Synthetic Systems Programming Benchmark," Communications of the ACM, Vol. 27, No. 10, October 1984, pp. 1013-103.