Event-based Re-training of Statistical Contention Models for Heterogeneous Multiprocessors

Alex Bobrek
ECE Department
Carnegie Mellon University
Pittsburgh, PA 15213 USA
abobrek@ece.cmu.edu

JoAnn M. Paul
ECE Department
Virginia Tech
Blacksburg, VA 24061 USA
jmpaul@vt.edu

Donald E. Thomas
ECE Department
Carnegie Mellon University
Pittsburgh, PA 15213 USA
thomas@ece.cmu.edu

ABSTRACT

Embedded single-chip heterogeneous multiprocessor (SCHM) systems experience frequent system events such as task preemption, power-saving voltage/frequency scaling, or arrival of new events/data from the outside world. Traditionally, the designers model these events by explicitly coupling them to corresponding simulation events within environments such as SystemC. However, this approach places a burden on the designer to identify which events are important enough to be captured by simulation, resulting in an overly conservative selection of events to model. This work presents a technique for de-coupling of system events from simulation events, removing the burden from the designer to determine which events significantly affect the system performance model, while decreasing simulation runtime. Enabling this de-coupling is a prediction model that quantifies the magnitude of changes introduced by system events, and identifies those important enough to be considered simulation events. The prediction model is evaluated by over 4000 separate scenarios featuring dynamic event changes to processor speed, bus speed, application type, and application input data, finding that almost 70% of tested events impacted contention modeling by less than 10%. Without resorting to detailed simulation, the prediction model captures the system event effects to within 5% of actual measured error, while staying within 18% in 95% of all tests.

Categories and Subject Descriptors: C.4 [Performance of Systems]: Modeling techniques, Performance attributes; I.6.5 [Simulation and Modeling]: Model Development – *Modeling methodologies*

General Terms: Performance, Design

Keywords: Performance Modeling, Simulation, Statistical Contention Modeling, Heterogeneous Multiprocessors

1. INTRODUCTION

Single chip heterogeneous multiprocessor (SCHM) systems contain tens of heterogeneous processors, a variety of communication and memory architectures, and are often focused on certain workload classes. Tens to hundreds of times per second SCHMs experience *system events* such as preemptive scheduling, scaling frequency/voltage or shutting down parts of the chip to save power, and responding to outside events and data. To model the performance effects of these events, designers manually identify and couple each system event to a *simulation event* within a system simulation environment such as SystemC. However, this one-to-one coupling of system and simulation events places the burden on the designer to identify which system events are important enough to be represented in the simulation. Conservatively specifying too many events reduces simulation performance, while specifying too few affects accuracy.

CODES+ISSS'07, September 30–October 3, 2007, Salzburg, Austria.

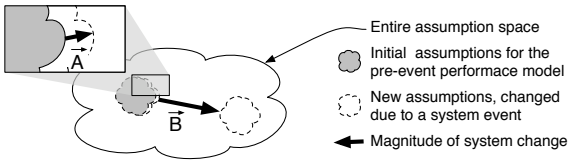

Figure 1: Change in model assumptions due to system events.

This paper presents a technique for de-coupling of system events from simulation events, removing the burden from the designer to determine which events significantly affect the system performance model, while decreasing simulation runtime. By monitoring the assumptions made by the performance model prior to the system event, and comparing them to the assumptions' state post-event, our approach estimates the introduced model error. Figure 1 shows a simple 2D view of the assumption space with a small gray area representing the acceptable assumption region for the current performance model. Two consecutive system events are illustrated, with their changes shown by magnitudes of vectors A and B. These vectors are proportional to the modeling error expected due to the change in the system; because of the smaller deviation in assumptions, system event A is expected to have a smaller impact on model error than B. The key challenge tackled by this paper is how to quantify the magnitude of vectors like A and B by isolating the parameters reflecting the changes in model assumptions. Identifying the magnitude of system changes allows the impact of a system event to be defined, allowing large magnitude events like B to be coupled to a simulation event.

As a basis for this work, we will focus on the statistical contention modeling presented in [3], which uses short training runs of a cycle-accurate (CA) simulation to train a high level model for estimating contention within SCHMs. This statistical approach captures application and system-specific information during training and is able to apply it to estimate contention within 1% of CA simulation while operating 40X faster. However, when a system event results in a significant deviation in model assumptions (large vector magnitude in Figure 1), a simulation event must be specified, resulting in re-training of the statistical model via detailed simulation.

The contribution of this paper is a quantitative approach that, by comparing the distributions of key shared resource access statistics, can determine whether an SCHM system event should be considered a simulation event without resorting to detailed CA simulation. Figure 2 illustrates the simulation advantages gained by identifying which system events are important simulation events. Timeline A repre-

sents a situation where every event is treated as a simulation event, requiring re-training of the statistical contention model (shown as dark blocks). Since re-training must be done at the CA level, the penalty for model switching is significant. However, by identifying important simulation events (thick lines in Timeline B),the number of re-trainings can be reduced, increasing overall simulation performance without significantly affecting accuracy.

In the experiments presented in this paper, featuring over 4000 separate SCHM scenarios with dynamic event changes to processor speed, bus speed, application type, and application input data, we found that almost 70% of system event changes tested had less than 10% impact on the statistical contention model accuracy. By observing the changes in key statistics of shared resource access patterns, we identify the remaining 30% of simulation events with high impact on contention, ensuring that only the events that matter to the overall accuracy are simulated in detail. This prediction model can assess system change impact to on average within 5% of actual measured error, while staying within 18% in 95% of all tests.

This paper will first give the necessary background on the statistical contention model and its simulation framework. We will review access attribute-based contention modeling introduced in [3], and show how statistics related to those attributes can be used to evaluate which system-changing events affect contention the most. Finally, we will describe the experiments featuring system changes due to runtime events, while drawing conclusions about their effect on contention and the resulting accuracy of the statistical contention model.

2. STATISTICAL CONTENTION MODELING

The purpose of this section is to provide some background on the statistical contention modeling approach introduced in [3] by describing the underlying simulation framework along with the access attribute-based model used to capture contention behavior.

2.1 Simulation Framework

The statistical contention model is based on the Modeling Environment for Software and Hardware (MESH) framework. MESH [4] allows designers to answer questions about how the numbers and types of processors and communications resources, the scheduling decisions, and the software tasks (mapping and complexity) affect the overall performance of SCHM systems. MESH increases simulation performance by executing application code natively on the host platform to capture data dependent execution, but emulates target system performance through annotations inserted within the code. This approach, called *execution-driven simulation with cross-profiling* or *back-annotation of target information*, has been commonly used for traditional multiprocessor simulation [6], as well as simulation of SCHM systems [1] [4].

Figure 3 shows a simple application control flow graph executing natively on the host simulator. At the end of each basic block, an annotation is inserted estimating the *target* processor performance. The annotation, shown within a box in the detailed area of Figure 3, is not a simple delay

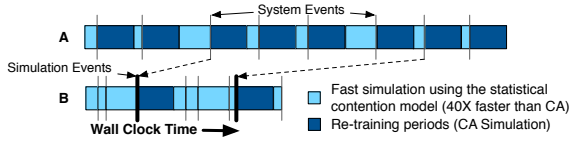

Figure 2: Performance impact of identifying crucial simulation events from a group of system events.

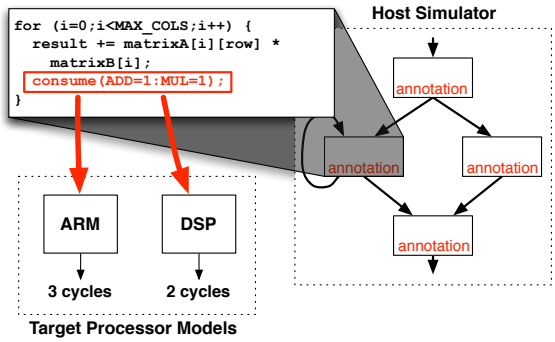

Figure 3: Execution-based simulation with back-annotation of target system information.

value, but instead captures the computational complexity of the natively-executing code. In this example, each loop iteration is annotated as one add and one multiply operation, which are then in turn "consumed" by hardware resources to determine timing. This level of indirection allows one set of annotations to capture code performance on multiple heterogeneous processors, speeding up the design exploration process. Note that annotations do not have to contain instruction types, and can be made at different levels of abstraction.

Target system information within annotations and processor models can be extracted from a CA simulation of the target processor, inferred by the compiler, or be manually inserted by the designer. Because most of the instructions are executing natively on the host system, this approach results in about a two orders of magnitude speedup over CA simulation and can be very accurate ($< 1\%$ error) depending on the quality of annotations. More detail on the MESH simulator, including its features, accuracy, and performance is available in [4].

2.2 Access Attribute Models

Although promising, the back-annotated simulation strategy experiences problems when simulating concurrent shared resource (SR) accesses. Since annotations containing SR accesses may have their timing affected by contention, only blocks with timing localized to a single processing element can be accurately annotated. Therefore, once an SR access is reached, an annotation must be inserted, forcing an expensive context switch to the other processors to determine contention. Solving this problem, the access attribute-based statistical model summarizes the impact of SR access contention at a high level of abstraction, thus removing the negative effects of frequent SR accesses on simulator performance.

Figure 4 illustrates the statistical contention model's role within the MESH simulation via a timeline of two concurrent applications running on separate processors while sharing memory. Multiple consecutive accesses to the shared resource are assumed to be serviced with no contention, speeding up execution. Once a designer-inserted annotation is reached, the distribution of skipped accesses during the preceding block is included with other annotation information. By combining SR information from multiple concurrently

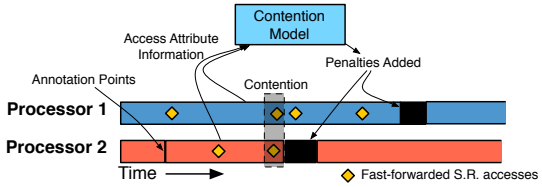

Figure 4: Fast-forwarding of shared resource accesses while estimating contention delays.

executing threads, the MESH simulator calculates the three access attributes used to estimate contention:

- **Average Requested Utilization (ρ)**: Captures the overall busyness of the SR.
- **Access Balance (B)**: Quantifies how much requested utilization for each thread varies with regard to the average.
- **Thread Concurrency (T)**: Average number of threads making SR accesses.

A key distinction between annotation information and access attributes must be made: annotations contain information only about SR accesses within single thread of execution; they can be created from a compiler or a single threaded simulation without the need of knowing how the application interacts with others. Access attributes, on the other hand, summarize the SR access behavior for multiple concurrent threads, capturing the current interleaving of applications within an SCHM system. Complete details about how access attributes are calculated from annotation information is included in [3].

Once the simulation reaches an annotation point, the three access attributes are passed back to the statistical contention model, which estimates contention during the block, and applies it after the block in a form of a penalty (black boxes in Figure 4). The statistical contention model trains the relationship between the three access attributes and the expected contention delay by sampling a cycle-accurate representation of the system. Tests have shown that a representative CA simulation of about 10 mil. cycles is enough to statistically describe the contention delay - access attribute relationship for most systems.

Nonparametric regression (e.g. curve fitting) is used to describe the contention-attribute relationship of the following format:

$$DPT = f_1(\rho) + f_2(B) + \beta \times T$$

where DPT represents contention delay per unit time, and $f(\cdot)$ are the nonparametric fits to the ρ and B access attributes. In general, as average requested utilization (ρ) or the concurrency level (T) increase, so does the DPT. Higher values of balance (B) decrease contention delay, since they indicate that the system is unbalanced, i.e. one of the threads makes the majority of S.R accesses, making contention less likely.

Thus, by leveraging the information gathered during training, the contention model is capable of estimating contention during large blocks of annotated MESH execution to within 1% of the CA simulation. Skipping large amounts of SR accesses allows the MESH simulator to perform 40X faster than CA simulation, while training the contention model takes a trivial section of the overall runtime. More detail on the access attribute-based statistical contention model, including how the three access attributes were selected and how the model was created and tested, is included in [3].

3. SYSTEM EVENT PRUNING

Deviations in SR accesses patterns due to system events can invalidate the assumptions set by the statistical contention model during the training process and introduce significant modeling errors. Therefore, every significant deviation from the trained model assumptions must result in statistical re-training, a computationally expensive process. Using a *prediction model* that estimates the contention modeling error in presence of system events (shown as vector magnitudes from Figure 1), only the events with a significant impact on model error are followed by re-training, allowing the contention model to retain its simulation speed advantage in presence of frequent system events. This section will describe how the necessary statistics for contention model error prediction are collected and how they are used to create the prediction model.

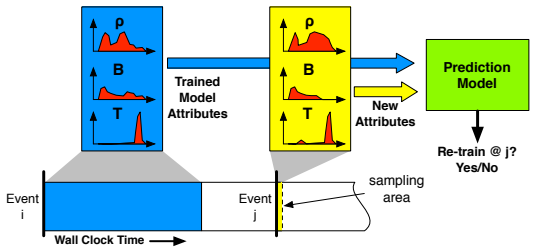

Figure 5: Comparing access attributes to determine need for re-training.

3.1 Collecting Comparison Statistics

Any change in SR access distributions is reflected within the distributions of ρ, B, and T attributes. Therefore, by comparing the populations of access attributes collected from the baseline (i.e. well-trained) model, to the populations after the system event, it is possible to capture the magnitude of system change.

Figure 5 illustrates this concept. At the start of the simulation (event i), the initial contention model is trained (dark box on the timeline). During this period, samples of ρ, B, and T access attributes are collected, and their distributions measured (illustrated as the small probability density function graphs in Figure 5). When event j is reached, the MESH simulator continues monitoring the same access attributes for a period following the event (shown as dashed border box). Note that this sampling can be performed much quicker in high speed simulation mode than during detailed training, as illustrated by the relative sizes of the colored boxes in Figure 5. Upon the completion of the sampling area, the access attributes for both the training model and the newly acquired data are passed to the prediction model.

3.2 The Prediction Model

Much like the contention model itself, the prediction model uses regression to estimate contention model error based on the populations of ρ, B, and T. Since there are many statistical methods to compare two populations of data, we considered a multitude of different options. For all values of ρ, B, and T, we compared common statistics such as medians and standard deviations. Additionally, we looked at the number of outliers that the new data had outside the range defined by the trained model. Finally, we ran the Kolmogorov-Smirnov (KS) test which is used to determine whether the shapes of two (non-normal) distributions differ significantly.

To capture these relationships, we ran over 4000 different situations (described in the Experiments section) in which a system is slightly changed from the baseline, either by changing the speed of one processor, changing the speed of a shared bus, or changing the application and/or its input data. For each of these situations, the error of the contention model due to mis-training was found by comparing the results to a detailed CA simulation.

Unlike the contention model itself, which uses a nonparametric regression technique, we found that a simpler linear regression is sufficient to describe the relationship between model error and access attribute deviations. Although we tried over 10 different combinations of medians, standard deviations, KS-tests, and outliers for all the access attributes, the two statistics that showed the best correlation to model error were the Average Requested Utilization Difference (D_ρ) and the KS Test of Utilization/Balance Ratio (KS_{ratio}). Table 1 gives more details about these two prediction model explanatory variables (i.e. variables used to

Table 1: Prediction Model Explanatory Variables

Model Parameter	Definition	Comments
Average Requested Utilization Difference (D_ρ)	$D\rho = \|median(\rho_T) - median(\rho_N)\|$	Captures the overall busyness of the shared resource.
KS Test of Utilization/Balance Ratio (KS_{ratio})	$KS_{ratio} = KS(\frac{\rho_T}{B_T}, \frac{\rho_N}{B_N})$	Quantifies how much requested utilization for each thread varies with regard to the average.

explain model error) and shows how they were calculated.

Intuitively, the D_ρ variable captures how the system change has impacted the overall utilization of the SR. For example, any system event that results in slowing down one of the processors would be reflected in the D_ρ variable, since the median of the ρ access attribute would be lower in the new data than the baseline. The KS_{ratio} variable captures more subtle changes in the distributions of both ρ and B access attributes, especially as they relate to each other. For example, a system change that substitutes one application for another would be reflected in the KS_{ratio} variable because the changed access patterns affect the distributions of the collected ρ and B access attributes.

Using the two explanatory variables described above, the best regression format describing the variable to model error relationship was found to be:

$$\sqrt{M_{error}} = D_\rho + KS_{ratio}$$

Since M_{error} variance increases with higher values of D_ρ and KS_{ratio}, a square root transformation of M_{error} variable is used to even out the variance across the entire range of explanatory variables. Transformations of response variables in this manner are common in the regression model building process when increasing variances are encountered.

Figure 6 shows the collected model error data and its relationship to one of the explanatory variables, D_ρ. Note that as the utilization difference between the trained and the new system increases, so does the range of model errors (i.e. the variance increase that prompted the square root transformation). The circles in Fig. 6 show what the regression model described above would predict each of the errors to be (i.e. every collected model error has a corresponding predicted error). Note that the prediction model captures the upward error trend as D_ρ increases, but cannot capture all the variance in data found for high D_ρ values. Although usage of the KS_{ratio} variable helps (seen as widening of the fit as D increases), the prediction model describes only about 70% of the variance in the observed model errors. This is not unexpected, since D_ρ and KS_{ratio}, cannot capture all the behavior resulting in increases in model error.

The prediction model described in this section enables the MESH simulator to selectively re-train only the system events with potentially large impact on contention modeling error. Although the prediction model does not capture all the variables affecting model error, the results in the Experiments section will show that it can make the correct decision on which system events to ignore in the great majority of cases.

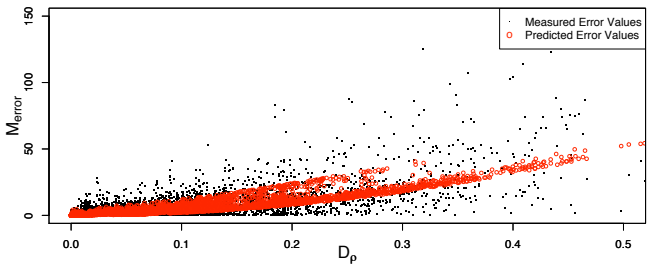

Figure 6: Measured vs. predicted model error as fit with relation to the D_ρ explanatory variable.

4. PRIOR WORK

The SCHM performance modeling space is populated with many simulation and analytical approaches. Approaches operating at CA level explicitly define a simulation event for every cycle, while the Transaction Level Modeling (TLM) [5] [9] approaches explicitly define simulation events at transaction boundaries. Both of these approaches operate at a high level of detail and can easily capture impacts of system events, albeit at the cost of long simulation times, interfering with designer's ability to rapidly transverse the SCHM design space.

Analytical contention models, such as queuing theory-based models [2], are popular in modeling data network contention and have been used in embedded systems as well. However, they assume purely exponential access inter-arrival times, an assumption that breaks down when loop-based applications create SR accesses with more periodic patterns. Queuing models are not designed to respond to temporal changes in the system, instead, they are optimized to find average throughputs and wait times. Thus they are not well suited to capture impacts of system events.

Statistical simulation and statistical sampling approaches [7] [10] extrapolate commonly recurring program behavior by statistical sampling in a way similar to our contention model. However, these approaches focus on estimating system throughput, and are less appropriate for capturing system response to outside stimuli, crucial for modeling embedded systems. These statistical sampling approaches do monitor the quality of their statistical estimate by periodically reverting to detailed CA simulation, similar to our re-training approach. However, this "re-training" is done at regular intervals, and performed after important system events like our approach. Therefore, the simulation (re-training) events are not pruned at all, but are explicitly and periodically defined, just like CA simulation but at a higher level of abstraction.

By introducing a method to estimate the impact of runtime system events on contention, our work selects system events of importance, reducing the overhead of model re-training, and allowing a high-level contention model to operate in a dynamic SCHM environment. To our knowledge, our model is the only contention modeling approach operating above the CA/TLM level that can adjust to system changes during execution.

5. EXPERIMENTS

This section presents a set of experiments featuring an SCHM system experiencing a variety of system event changes. The goal of these experiments is to determine how well the prediction model from Section 3 tracks the actual contention model error, while identifying system events with the most impact. The test system executes several concurrent applications running on ARM processors while contending for shared memory through a shared bus. Since applications are independent, the only contention in the system is contention for resources, not data. For the evaluation of the contention model, we assume that each application is running on its own processor, therefore, there is no contention for execution resources, only for shared memory blocks. The memory and bus models assume a constant service delay for each uncontended access. Only one outstanding memory access is allowed per processor, meaning that processors stall on con-

tention. The impact of caches is not modeled, although we believe that, with some adjustments, the statistical regression training approach could be used to model caches as well. Note that none of these assumptions are limitations of the MESH simulator, but are instead limiting factors chosen by us to isolate the statistical model's accuracy.

To capture a wide variety of design perturbations, access patterns, and contention levels, we ran tests with groups of 2, 3, 4, and 5 single-threaded applications executing concurrently. We selected several multimedia, encryption, compression, and signal processing applications from SPEC2000 and MiBench [8] benchmark suites: adpcm (adaptive differential pulse code modulation), FFT, jpeg, gzip, rijndael (encryption), rsynth (speech synthesis), and crc (cyclic redundancy check). To select these applications from the greater set of SPEC and MiBench benchmarks, each benchmark's memory access utilization and coefficient of variation were measured and used for the selection criteria, ensuring the widest variety of memory access patterns. The 7 applications, executing for all combinations for groups of 2, 3, 4 and 5 threads at a time, provide us with over 100 different test runs for each experiment data point.

System events and their impact on system behavior are modeled through five representative experiments:

- **Application Change**: Simulates a preemptive scheduling switch to a different application.
- **Data set Change**: Simulates change in external input; new data is delivered to a already running application.
- **Processor Speed Change**: Simulates voltage/frequency scaling, resulting in relative speedup or slowdown of one of the processors.
- **Bus Speed Change**: Simulates voltage/frequency scaling of the bus, resulting in relative speedup or slowdown of access time to memory.
- **Coprocessor Change**: Simulates shutting down a section of the chip by removing a floating point and/or multiply-accumulate unit.

Experiments summarized above capture a wide variety, albeit not all, of possible system-level events at a variety of data points. For example, the bus speed change experiment includes altering the memory access time from 50% to 150% at 10% intervals. Taking into account the five separate experiment types, their multitude of data points, and over 100 different application combinations for each data point, the experiments in this section include over 4000 separate SCHM operating situations. In the following sections we will discuss these individual system changes separately, analyze their impact on contention modeling, and evaluate the prediction model's ability to anticipate the modeling error. Due to the space limitations, we will focus most of our analysis on application change and data set change experiments while summarizing data from the remaining three experiments.

5.1 Application Change

To simulate a switch of one application with the another as a result of a scheduling decision, we replaced the adpcm and jpeg applications with others from the group of 7. Figure 7 shows the model error when the adpcm application is replaced by one of the other 6 applications labeled on the x-axis. Each data set is also separated according to the thread concurrency (i.e. 2th represents all data where 2 applications ran concurrently). The data is presented in a standard box plot format, where the thick line in the middle of the boxes represents the median of collected data, and the edges of the box represent the edges of the first and third quartiles. As the data shows, replacing adpcm with crc, gzip, or jpeg applications results in highest errors. This is not a surprising result since the adpcm benchmark has very

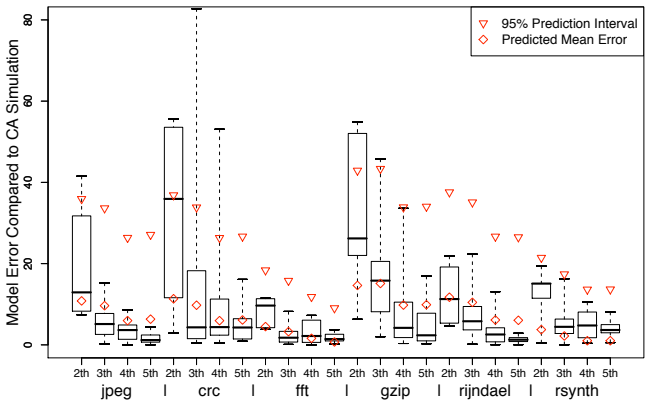

Figure 7: Model error as the adpcm application is swapped out by another application.

low memory utilization and burstiness, while crc, gzip, and jpeg access memory often and with frequent bursty patterns. Also, note that as the concurrency level rises, the model error decreases. This result is also intuitive, since a change in one application affects system operation more when only two applications are present vs. five.

Red diamonds in Figure 7 shows the median error estimated by the prediction model. Additionally, the top 95% prediction interval is shown by a downward pointing triangle. Where the median predicted error tells the designer what average error can be expected from a design change, the 95% prediction interval suggests an approximate upper bound for the error estimate. The prediction model is on average within 5.01% actual error, while 95% of all errors are below 12.79%. More importantly, the model is effective at separating the low error substitutions like adpcm-fft from high error substitutions like adpcm-gzip.

5.2 Data Set Change

To simulate the effect of data set changes on contention, this experiment replaces the input data sets of two data-dependent applications from the benchmark set: gzip and jpeg. Figure 8 shows the model error when input data for the gzip application is changed. In this case, the gzip application was trained with a long text file of a Charles Dickens' classic "A Christmas Carol". Data labeled "great exp.", "moby dick", and "jane eyre" are alternative data sets that feature long text files of English text. The "bmp img" is a bit mapped (uncompressed) image, while "random" and "zero" provide a completely randomized and completely uniform data sets. Since the gzip application uses Huffman trees for compression, the frequency of encountered tokens significantly impacts the behavior of the application. Therefore, data sets with a single token (zero) or evenly distributed tokens (random) will produce a much different responses than when using English text. Figure 8 reflects this observation, while the prediction model nicely distinguishes the random and zero data sets from the text-based ones. Due to relatively low model errors associated with data set changes, the prediction model was very accurate in estimating contention model error in this experiment, staying within the average of 1.84% while 95% of the cases were predicted with about 6.21% error.

5.3 Other Experiments

To capture effects of frequency scaling on contention, the processor speed change experiment consists of a group of processors operating at approximately the same clock frequency, while the operating frequency of one processor is varied. As expected, increasing deviation from the trained baseline increases model error, while the error is higher for systems with less concurrency since change to one proces-

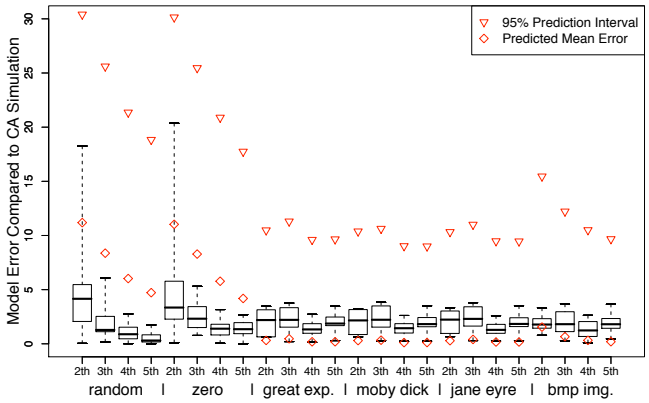

Figure 8: Model error as input data for the gzip application is changed.

sor represents a higher fraction of the system. Doubling the operating frequency of one of the processors produced an average 17% error for the 2 processor situation and an average 7% error for the 4 and 5 processor situations. Increasing processor frequency to 300% of the original resulted in 25% error for the 2 processor case and 10% error otherwise. The prediction model was effective in predicting this error to within an average of 3.65%.

Scaling the bus speed affects latency in the path to memory equally for all processors in the system. To capture this effect on contention model error, bus speed was reduced to 50% and increased to 150% of the trained value. Since memory latency change has a much higher effect on the system as a whole than scaling the speed of one processor, the contention model error was appropriately higher: scaling down to 50% speed introduced an average of 40% error, while speeding the bus up to 150% introduced 18% error. Due to higher errors encountered, the prediction model was less effective at predicting bus speed model error, estimating it to within an average of 8.37%.

By shutting down specialized hardware functional units, SCHM systems can save power at the expense of lowering performance for certain types of workloads. The coprocessor change experiment examines these cases by comparing systems with and without floating point and multiply-accumulate (MAC) units. For the floating point test, the experiment focuses on FFT and jpeg applications which both use floating point arithmetic. Since jpeg's DCT algorithm can take advantage of MAC functionality, jpeg application is used to evaluate the MAC unit addition. These tests produced surprisingly low contention modeling errors of about 5%. However, although the addition of a floating point and a MAC unit changed the contention patterns significantly, only a relatively small number of instructions was affected. Therefore, for most of the runtime, the baseline-trained contention model was sufficient.

5.4 Results Summary

Table 2 summarizes the prediction model accuracy. In addition to the mean prediction error for each experiment, the table shows the 95% percentile of the collected prediction errors, while the third column illustrates the percentage of tests that fell outside the prediction models' 95% prediction interval (triangles in Figures 7 and 8). For all experiments, the prediction model was on average within 5.43% of the actual model error, 95% of all predictions were within 18.87%, and 7.63% of the data was outside of the 95% interval provided by the prediction model. The percentage of values over the bound is higher than calculated due to a slight non-normality in the distribution of model error residuals after the fit, where a normal distribution is assumed when calculating prediction intervals.

Table 2: Prediction Model Accuracy Summary

Experiment	Mean Predict Err	Error 95th Percentile	Over Bound
Application Change	5.01%	12.79%	3.09%
Dataset Change	1.84%	6.21%	0.00%
PE Speed Change	3.65%	11.54%	12.10%
Bus Speed Change	8.37%	29.48%	5.80%
Coprocessor Change	3.85%	9.67%	0.00%
All	**5.13%**	**18.28%**	**6.08%**

6. CONCLUSIONS

This paper introduced an event-based re-training technique for statistical contention modeling within SCHM systems, allowing frequent system events to be pruned down to a set of simulation events that truly affect contention behavior. The backbone of this technique includes a prediction model for estimation of contention modeling error without relying on detailed CA simulation. The prediction model is on average within 5.13% of the actual contention values, allowing the simulator to select which system events do not have a significant impact on contention, removing around 50% of simulation overhead (or about 70% of the number of events) due to frequent contention model re-training.

By removing the burden of abstracting system events from the designer, this work enables a high level statistical contention model to simulate a broader range of SCHM scenarios. Although we showed that de-coupling system events from simulation events is beneficial for statistical contention models, any modeling method for dynamic SCHM systems operating above the CA level can improve its accuracy and speed by identifying the events that do not significantly impact model assumptions.

7. ACKNOWLEDGMENTS

This work was supported in part by an NSF Graduate Research Fellowship, SRC contract 2005-HJ-1312, and the National Science Foundation, under grants 0509193, 0607934, and 0606675. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

8. REFERENCES

[1] J. R. Bammi, W. Kruijtzer, L. Lavagno, E. Harcourt, and M. T. Lazarescu. Software performance estimation strategies in a system-level design tool. In *CODES '00*, pages 82–86, 2000.

[2] D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, 1992.

[3] A. Bobrek, J. M. Paul, and D. E. Thomas. Shared resource access attributes for high-level contention models. In *DAC '07*, pages 720–725, 2007.

[4] A. Bobrek, J. J. Pieper, J. E. Nelson, J. M. Paul, and D. E. Thomas. Modeling shared resource contention using a hybrid simulation/analytical approach. In *DATE '04*, pages 1144–1149, 2004.

[5] L. Cai and D. Gajski. Transaction level modeling: an overview. In *CODES+ISSS '03*, pages 19–24, 2003.

[6] R. Covington, J. Jump, and J. Sinclair. Cross-profiling as an efficient technique in simulating parallel computer systems. In *Computer Software and Applications Conference*, pages 75–80, 1989.

[7] L. Eeckhout, S. Nussbaum, J. E. Smith, and K. De Bosschere. Statistical simulation: adding efficiency to the computer designer's toolbox. *IEEE Micro*, 23(5):26–38, 2003.

[8] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *IEEE Workshop on Workload Characterization*, pages 3–14, 2001.

[9] S. Pasricha, N. Dutt, and M. Ben-Romdhane. Extending the transaction level modeling approach for fast communication architecture exploration. In *DAC '04*, pages 113–118, 2004.

[10] T. F. Wenisch, R. E. Wunderlich, and et. al. SimFlex: Statistical Sampling of Computer System Simulation. *IEEE Micro*, pages 2–15, July-August 2006.