An Activity-Sensitive Contention Delay Model for Highly Efficient Deterministic Full-System Simulations

Shu-Yung Chen[1] , Chien-Hao Chen[2] and Ren-Song Tsay[3]
National Tsing Hua University, Department of Computer Science
Hsinchu, Taiwan
{sychen[1], chchen[2], rstsay[3]}@cs.nthu.edu.tw

***Abstract*—As modern systems are integrating exceeding number of components for better performance and functionality, early full-system simulation tools have become essential for validating complex concurrent system interaction activities. In the past decades, many useful timing-accurate system simulation tools have been developed; however, we find that even for the most efficient techniques, more than 90% of overhead occurs when simulating shared devices, such as buses. Instead of adopting the constant-delay model that compromises accuracy or using the time-consuming precise scheduling approach, we propose in this paper an effective system activity-sensitive contention delay model that can dynamically capture runtime contention situations and system configuration changes. To verify the idea, we construct an analytical bus delay model and integrate that into a system simulation tool. The experimental results show 20 to 80 times performance improvement over the scheduling-based bus model on full-system simulations and the estimated timing difference is less than 3%.**

I. INTRODUCTION

As increasingly more components are integrated into system designs for better performance or functionality, the concurrent interactions among components are exponentially more frequent and complicated through shared resources such as buses and memory components for message exchange. Since the concurrent interactions can only be verified at system level, effective and efficient techniques for deterministic full-system simulations are extremely critical for system verification and debugging.

In fact, it is most desirable at the early system design stage to verify and debug the correctness of planned software applications. Mostly, the challenges are to check atomicity and race-condition bugs. To achieve the purpose, it requires knowing *precise* shared-data-access order from the resultant concurrent executions of the applications and being able to deterministically reproduce execution results.

To produce precise shared-data-access order, we need to include in the system model not only regular components, such as processors, but also shared resource components, such as buses and memories. However, we find that the traditional scheduling-based models for shared resources can take up more than 90% of full-system simulation time [11]. We hence propose in this paper an efficient activity-sensitive shared resource modeling approach for effective system concurrent behavior verification.

To illustrate our idea, we take the multi-core system shown in Fig. 1 as an example. In general, a multi-core system contains multiple CPU cores and shared resources, such as buses or memories. To simplify later discussion, we further classify shared resources into Data Storage Device (DSD) and Data-Pass-Through Device (DPTD).

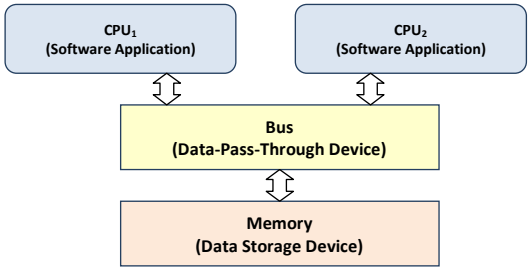

Figure 1: A sample multi-core system with two CPU cores and shared resources, here a bus and a memory, for data exchange among CPU cores.

Note that in practice each DSD serves at a time only one data access, read or write, according to requests issued. Contending accesses are executed according to scheduling results and some accesses can be delayed due to contention. A critical requirement for deterministic full-system simulations is that the access order to shared data has to be maintained to avoid causality violations. Therefore, synchronization algorithms [1] are necessary to maintain deterministic simulation results.

Unlike DSD, the function of DPTD is to transmit data between components without affecting data content. Although DPTD does not directly affect shared data causality violations, its data-pass-through time of each request indirectly affects the shared data access order. Therefore, an efficient and accurate method for calculating data-pass-through time is the key to produce high performance deterministic full-system simulations.

In general, the data-pass-through time of a DPTD includes both the basic data process time and the contention delay. Here we use the term contention in a general sense which can be bus contention or cache miss. The basic data process time is the process time of the request assuming no contention and it can be extracted by analyzing DPTD's behavior protocol in static time. Conversely, the contention delay is mainly resulted from dynamic system activities, which can only be detected at runtime. As a result, it is highly desirable to be able to efficiently calculate a deterministic contention delay for an effective DPTD model. However, the prior modeling methods, scheduling-based model [2-6] and the constant-time model [7, 8], cannot fully meet designer's need either because of low simulation speed or insensitivity to system configurations.

To have an appropriate DPTD model for accurate and fast deterministic full-system simulations, we propose an activity-sensitive data-pass-through device contention delay model. The key

978-3-9815370-2-4/DATE14/©2014 EDAA

idea is to dynamically estimate the contention delay according to the runtime activities on the DPTDs and generate an efficient activity-sensitive contention delay analytical model. According to the dynamically changing activities on a DPTD, the associated contention delay and final data-pass-through time of each data access can be accurately estimated. With this efficient and effective contention delay analytical model, full-system simulations can perform one to two orders faster, and hence become feasible and practical.

To verify our idea, we construct a bus activity-sensitive contention delay model of a FIFO bus design and integrate the model onto a shared-variable-synchronization-based multi-core simulation tool developed by Wu et al. [1]. The experimental results show that our approach performs 20 to 60 times faster for full-system simulations with bus models and the timing error rates are only within 2.3% to 2.8%.

The rest of the paper is organized as the following. In Sec. 2, we briefly review related work and then present the proposed bus activity-sensitive contention-delay model in Sec. 3. In Sec. 4, we discuss how to determine activity sampling period for accurate and deterministic delay calculation. Finally, we present experimental results in Sec. 5, and conclude in Sec. 6.

II. RELATED WORK

The scheduling-based method [2-6] and the constant-time method [6, 7] are two main approaches for data-pass-through device deterministic contention delay modeling. The main idea of the scheduling-based approach is to schedule every data request one-by-one and the choice of timing granularity for basic data-process-time models leads to different simulation speed and accuracy.

The cycle-accurate (CA) based basic data-process-time modeling approach [2] is inherently timing accurate and deterministic as it captures every system activity at each cycle. However, the accumulated overhead from computations at each cycle severely degrades simulation performance and makes it impractical for full-system simulations.

For performance improvement yet retaining timing accuracy, cycle-count-accurate (CCA) approach [5, 6] is proposed based on the observation that computation can be done at each transaction boundary instead of at each cycle boundary. According to the behavioral protocol of a DPTD, each data request is divided into a series of atomic transactions, which cannot be interrupted or preempted. The CCA approach preserves cycle-accurate information and reports simulation performance one order faster than the CA approach.

In contrast to the CCA approach, the cycle-approximate (CX) approach [4] simply treats each read/write request as one *atomic* transaction and disregards internal true atomic sub-transactions. With the approximation, the CX method further reduces the number of scheduling points and improves simulation performance while deriving imprecise data process time.

Although all above methods can capture dynamic system activities and produce deterministic delay model with scheduling at each data request point, the issue is that the number of data requests is huge and thus the scheduling overhead can severely affects system simulation performance. Clearly, a better scheduling modeling method is critical for improving simulation performance.

Instead of performing actual scheduling, the constant-time method assumes a fixed delay, usually derived from statistical data, for each DPTD request. Without scheduling overhead, the simulation speed is fast. As an example, Hwang et al. [7] adopts a list of pre-specified memory access delay and branch penalty delay values and performs fast runtime simulations. Although the constant-time approach is fast, it cannot capture dynamic system activities, such as the contentions resulted from active components at runtime, and cannot truly represent system concurrent behaviors and related bugs. For example, as discussed in [9], the behavior of video decoder is dependent on if frame data to be decoded are ready on time, and in this case constant-time model simply cannot capture runtime dynamics and so miss this issue.

In summary, all existing approaches mentioned above either sacrifice simulation speed or ignore dynamic system behaviors. To resolve the issue, we propose in this paper an activity-sensitive dynamic contention delay analytical method that can accurately captures dynamic system concurrent behaviors. Details are discussed in the following sections.

III. BUS ACTIVITY-SENSITIVE CONTENTION DELAY MODEL

Since bus is a common DPTD used in system designs, we construct a bus activity-sensitive contention delay model to demonstrate our proposed idea. Nevertheless, the proposed idea can be adapted and applied to other type of DPTDs for system simulation performance improvement.

First, we observe that the bus data-pass-through time of each data request has two portions. The first portion is the basic data process time for actual data transmission of a given read or write request and the second portion is the contention delay caused by concurrent accesses to the bus as shown in Fig. 2.

Next, we will discuss how to efficiently model the basic data process time and contention delay and have a complete bus timing model.

A. *Basic Data Process Time*

To efficiently model basic data process time, we adopt the systematic method proposed by Lo et al. [5]. The idea is to pre-analyze for each type of bus request the finite state machines (FSM) of the communication protocol between the master and slave components. Then merge and compress the two FSMs according to the synchronization protocol automata into a new FSM. The compressed FSM only retains the states which are affected by the signal issued from external components. With the remaining states merged into one super state, we have the desired basic data process

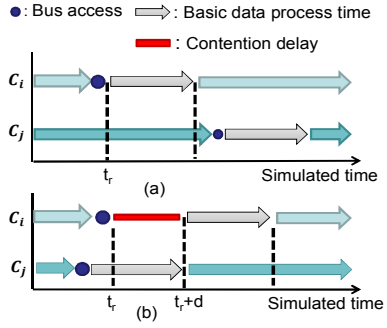

Figure 2: An illustration of data-access scenario without and with contention. (a) The bus takes a basic data process time to transmit the data requested by the component C_i, if there is no contention; (b) If contention occurs, the component C_i has to wait an extra contention delay d before its data can be processed.

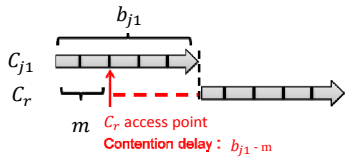

Figure 3: An illustration of contention delay calculation when the data request of a component C_r overlaps with the data request of another bus-using component C_{j1}.

time pre-analyzed and annotated. Details of the algorithm can be found in [5].

Usually the external signal considered is the bus grant signal and sometimes interrupt or preempt signal if sophisticated bus designs are considered. Essentially, the actual activation time of the external signal determines at runtime when the request can start actual transmission, whereas whose basic data process time is pre-analyzed as discussed previously. The difference of the actual issuing time and starting time of a data request is the contention delay of the request. We will next discuss how to analyze the contention delay and then construct a bus activity-sensitive contention delay model.

B. Contention Delay Estimation

To effectively and efficiently estimate contention delay, we propose a contention delay analytical model by analyzing all possible contention scenarios and then derive an accurate analytical formula that can work under different system configurations and adapt to dynamic bus utilization rates.

In the following analysis, we assume that we are to estimate the contention delay of a component C_r which is to issue a bus request and the bus already has n active master components using the bus. We start from the case that $n = 1$ and assume that the bus-using component is C_{j1}.

Since a bus can serve only one request at a time, the later bus request has to wait until the earlier one completes its work as illustrated in Fig. 3. We assume that the component C_r issues a request to the shared bus m cycles after the component C_{j1} takes the bus. Suppose that the basic data process time of C_{j1} is b_{j1}, then the component C_r has to delay $b_{j1} - m$ cycles to actually process its bus request.

Then we introduce the dynamic bus utilization rate P of a component C_{j1} on a bus as the percentage of time used for actual data transmission in a period T. In other words,

$$P = \frac{\sum_{k-th\ request\ of\ C_{j1}\ in\ T} b_k}{T} \quad (1)$$

Note that the bus utilization rate of a component depends on how often the corresponding master component issues requests and hence it should not include component idling time caused by contention delays. Therefore, when estimating the bus utilization rate from what was observed on the bus, we should deduct the total contention delays from the period T for accurate utilization rate estimation. In other words,

$$P = \frac{\sum_{k-th\ request\ of\ C_{j1}\ in\ T} b_k}{T - \sum_k d_k} = \frac{\sum_k b_k}{T'} \quad (2)$$

where d_k is the contention delay of the k-th request of C_{j1} in the period T and $T' = T - \sum_k d_k$. Moreover, each type of request has its own bus utilization rate P and causes different contention effects.

For simplicity, we assume that C_{j1} issues only one type of data requests with the basic data process time b_{j1} and bus utilization rate p_{j1}. Then, the probability of the later bus request from C_r conflicts with that of C_{j1} is p_{j1}.

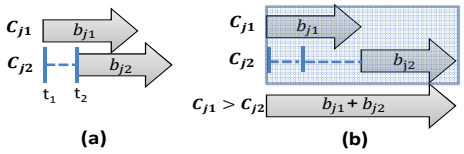

Figure 4: (a) Assume that two bus-using components, C_{j1} and C_{j2} have overlapped requests. (b) Treat the two overlapped requests of C_{j1} and C_{j2} as a single request with the equivalent basic data process time being $(b_{j1}+b_{j2})$.

Since the time unit is one cycle, C_r may contend with C_{j1} at b_{j1} different possible time points and the contention delay depends on where C_r's request hits C_{j1}'s. If the request of C_r occurs at m cycles after the request of C_{j1}, the contention delay of C_r will be $(b_{j1} - m)$ as shown in Fig. 3.

Therefore, when the later requester C_r accesses the bus, its probability of hitting the request of C_{j1} is p_{j1}. Additionally, since the contention of C_r to C_{j1} can occur at any point in the basic data process time b_{j1}, the average contention delay is $\sum_{m=0}^{b_{j1}-1}(b_{j1} - m)/b_{j1}$. To summarize, we have the estimated contention delay d_r of C_r as

$$d_r = p_{j1}\frac{\sum_{m=0}^{b_{j1}-1}(b_{j1}-m)}{b_{j1}} = p_{j1}\left(\frac{b_{j1}+1}{2}\right) \quad (3)$$

In case that C_{j1} has more than one type of data requests each with different basic data process time, we can simply analyze each type separately using the method discussed above because that the requests from a same master component cannot have overlaps by themselves. We then sum up the estimated contention delay contributed from each type to have the total estimated contention delay as shown in the following equation

$$d_r = \sum_{k=1}^{\#\ types\ of\ C_{j1}}\left[p_{j1}^k\left(\frac{b_{j1}^k+1}{2}\right)\right] \quad (4)$$

Nevertheless, for simplicity we assume each master component issues only one type of requests. We now extend the above solution to the case that $n=2$ and assume the two bus-using master components are C_{j1} and C_{j2}.

For this case, we only need to consider the scenario that C_{j1} and C_{j2} have overlap as shown in Fig. 4(a); otherwise, we may use Equation (2) to estimate the contention delay of C_r with C_{j1} or C_{j2} separately. For the overlapping C_{j1} and C_{j2}, we simply model them as one single request as shown in Fig. 4(b) and then apply Equation (2) to calculate the contention delay of C_r on this merged request.

In the following we elaborate how to calculate the corresponding parameters of the equivalent single request. First, the equivalent basic data process time, denoted as b', is simply the summation of the basic data process time of two requests, or

$$b' = b_{j1} + b_{j2} \quad (5)$$

For the equivalent bus utilization rate calculation, we do the calculation in two parts by first considering the scenario that C_{j1} is overlapping with C_{j2} and the request of C_{j1} is ahead of the request of C_{j2}, denoted as $j1>j2$ as shown in Fig. 4(a). The case that $j1<j2$ can be similarly analyzed.

According to the definition of bus utilization rate, the probability of C_{j1} using the bus at t_1 is p_{j1}, and the conditional probability of t_1 being the starting cycle of the bus request of C_{j1} is (p_{j1}/b_{j1}) since the request length is b_{j1} cycles. Likewise, the probability of C_{j2} starts at t_2 is (p_{j2}/b_{j2}). For C_{j2} to overlap with C_{j1}, t_2 can range from t_1 to $t_1+(b_{j1}-1)$ with b_{j1} possibilities. Since t_1 is equally likely to be any time point in the active period T_{j1}, and it ranges from 0 to $T_{j1}-1$ with T_{j1} possibilities, therefore the bus utilization of the merged request for $j1>j2$, is

$$p_{j1>j2} = \frac{1}{T_{j1}}[b_{j1} \cdot \left(\frac{p_{j1}}{b_{j1}} \cdot \frac{p_{j2}}{b_{j2}}\right)]T_{j1} = \frac{p_{j1} \cdot p_{j2}}{b_{j2}} \quad (6)$$

Similarly, $p_{j1<j2} = p_{j1} \cdot p_{j2}/b_{j1}$. Therefore, the equivalent bus utilization rate of the merged request is

$$p' = p_{j1>j2} + p_{j1<j2} = \left(\frac{1}{b_{j1}} + \frac{1}{b_{j2}}\right) \cdot (p_{j1} \cdot p_{j2}) \quad (7)$$

Therefore, the total contention delay of C_r corresponding to the merged request is

$$d_r = \mathrm{p}'\left(\frac{b'+1}{2}\right) = \left(\frac{1}{b_{j1}} + \frac{1}{b_{j2}}\right) \cdot (p_{j1} \cdot p_{j2}) \cdot \left(\frac{b_{j1}+b_{j2}+1}{2}\right) \quad (8)$$

Now we discuss the contention delay caused by three overlapping bus-using active requests from C_{j1}, C_{j2} and C_{j3}. Note that independent of how these three overlapping requests are ordered, the equivalent single-request basic process time is

$$b' = b_{j1} + b_{j2} + b_{j3} \quad (9)$$

We now consider the case that C_{j3} is ordered the last, denoted as *L_j3*. Then we find that both cases *j1>j2>j3* and *j2>j1>j3* with C_{j3} ordered last have the same bus utilization rate, i.e.

$$p_{\mathrm{j1>j2>j3}} = b_{j2}[b_{j1} \cdot \left(\frac{p_{j1}}{b_{j1}} \cdot \frac{p_{j2}}{b_{j2}}\right)]\frac{p_{j3}}{b_{j3}} = \frac{p_{j1} \cdot p_{j2} \cdot p_{j3}}{b_{j3}} \quad (10)$$

and

$$p_{\mathrm{j2>j1>j3}} = b_{j1}\left[b_{j2} \cdot \left(\frac{p_{j2}}{b_{j2}} \cdot \frac{p_{j1}}{b_{j1}}\right)\right]\frac{p_{j3}}{b_{j3}} = \frac{p_{j2} \cdot p_{j1} \cdot p_{j3}}{b_{j3}} = \frac{p_{j1} \cdot p_{j2} \cdot p_{j3}}{b_{j3}} \quad (11)$$

Therefore, we have the bus utilization rate p_{L_j3} of the case C_{j3} ordered last

$$p_{L\_j3} = (3-1)! \cdot \frac{p_{j1} \cdot p_{j2} \cdot p_{j3}}{b_{j3}} \quad (12)$$

Then the total contention delay caused by the three overlapping bus-using requests from C_{j1}, C_{j2} and C_{j3} is

$$d_r = (3-1)! \cdot \left(\frac{1}{b_{j1}} + \frac{1}{b_{j2}} + \frac{1}{b_{j3}}\right) \cdot (p_{j1}p_{j2}p_{j3}) \cdot \left(\frac{b_{j1}+b_{j2}+b_{j3}+1}{2}\right) \quad (13)$$

The same arguments can be extended to the general case of i overlapping bus-using master components. Without showing derivation details, the estimated contention delay d_r of the later contender C_r is

$$d_r = (i-1)! \cdot \left(\sum_{k=1}^{i}\frac{1}{b_{jk}}\right) \cdot \prod_{k=1}^{i} p_{jk} \cdot \left(\frac{1+\sum_{k=1}^{i} b_{jk}}{2}\right) \quad (14)$$

To sum up all possible scenarios for a contending component C_r with n bus-using components, we have the final estimated contention delay d_r for a given active component listed below, where the first summation term is the expected contention delay induced by the scenario of only one other contender and the second summation term shows the one with more than one contenders.

$$d_r = \sum_{k=1}^{n} p_{jk}\left(\frac{b_{jk}+1}{2}\right) + \sum_{i=2}^{n}\sum_{j_1=1}^{n}\sum_{j_2>j_1}^{n}\cdots\sum_{j_i>j_{i-1}}^{n} d_r(j_1,\ldots,j_i) = \sum_{k=1}^{n} p_{jk}\left(\frac{b_{jk}+1}{2}\right) + \sum_{i=2}^{n}\sum_{j_1=1}^{n}\sum_{j_2>j_1}^{n}\cdots\sum_{j_i>j_{i-1}}^{n}[(i-1)! \cdot \left(\sum_{k=1}^{i}\frac{1}{b_{j_k}}\right) \cdot \prod_{k=1}^{i} p_{j_k} \cdot \left(\frac{1+\sum_{k=1}^{i} b_{j_k}}{2}\right)] \quad (15)$$

With the above contention delay analytical function, we can efficiently construct an accurate bus activity-sensitive contention delay model without need of actual scheduling. When a component issues a bus request, we simply apply Equation (15), which considers the bus utilization rates and basic data process times of all other bus-using components, and immediately estimate the analytical contention delay by collecting activities from the others.

One critical issue to be discussed in the next section is that in fact the dynamic contention delay estimation depends on the accuracy of bus utilization rate which in turn strongly depends on the choice of sampling period.

IV. CHOOSING PROPER SAMPLING PERIOD

As discussed previously, the key of our approach is to know the runtime bus utilization rate accurately. To estimate the bus utilization rate, according to Equation (1) we have to first determine the activity sampling period T, which we assume starts at t_s and ends at t_e.

A simple and intuitive way is to take an execution trace and use the average bus utilization rate of the trace for data-process-time calculation. Another possibility is to use no sampling but have designers estimate the bus utilization rate based on experiences. Although these approaches cause little overhead, the fixed estimated values simply cannot capture dynamic bus activity variations. In practice, designers strongly desire to know the impact to system behavior from configurations and task assignment changes and the analysis can only be done by knowing the bus activity (or utilization rate) fluctuation caused by dynamic interactions among active system components.

Another heuristic is to set t_s to be time zero, the starting time of execution, and leave t_e float along with the latest execution end time in runtime as shown in Fig. 5(a). Although this approach can also produce a bus utilization rate for data process-time estimation, the sensitivity of bus utilization rate analysis is gradually diluted as the sampling period continues to increase along with the execution process. From the above, we conclude that the sampling period should be smaller and near current execution point in order to produce more accurate and sensitive bus utilization rate analysis. One way is to use a pre-defined time period t_p and set t_e to be the most current execution end time. Then t_s is always at a distance of t_p cycle before t_e as shown in Fig 5(b). Although this approach does generate more sensitive utilization rate, we need to keep all bus request information (such as access time) for analysis until both t_s and t_e are known and this process causes great runtime overhead.

To avoid the overhead, we first observe the fact that in system simulations, the execution of each simulated component is modeled as a series of transactions split by timing synchronization points and

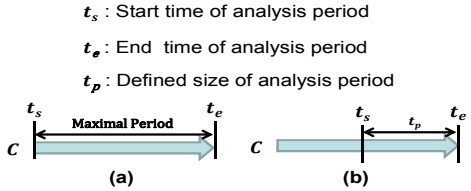

Figure 5: (a) Have the whole execution trace for bus utilization rate analysis. (b) Take a pre-defined sampling period t_p for bus utilization analysis.

the execution order of the synchronization points among different components has to be of a proper order to ensure the correctness of simulations [1]. Within the specified transactions, the execution order among different components will not affect the correctness of simulations. Therefore, a simulated component can freely execute until it hits a synchronization point. Then it waits for permission to start processing next transaction at the synchronization point. An idea is to set t_s to be the starting point of current transaction and dynamically change t_e to be the most current execution end time. Nevertheless, the issue of this approach is that t_e is not deterministic as the execution speed of the simulated component and the host OS scheduling may be different in each different run.

Our solution is to set t_s and t_e to be the starting time and end time of the last completed transaction in order to ensure the most current deterministic information. Note that since the transaction has been completed, we have known bus activity information and hence can derive deterministic while sensitive bus utilization rate.

V. EXPERIMENTAL RESULTS

To verify our idea, we adapt the shared-variable based multiple cores simulation tool developed by Wu et al. [1] and test on a host machine equipped with Intel Xeon® 2.67GHz quad-core, 4GB memory. In the following, we first show that our proposed bus model indeed produces deterministic results. We then compare performance and accuracy of our approach with accurate scheduling-based model. Third we compare the average contention delay in each transaction produced by each modeling approach. We finally test an MJPEG decoder [13] to demonstrate that our modeling approach can accurately reflect sensitive contention effect in full-system designs.

A. *Determinism Verification*

Because determinism is a basic requirement for simulations, we use a standard determinism test case from RACEY [10] to verify the determinism of our approach by checking the timing and the order of each synchronization point. For 10 different simulation runs, we always produce same execution results. Although the scheduling-based model also produces deterministic results, it runs 12 times slower than our proposed approach in average for this test case.

B. *Performance and Accuracy Comparison*

To demonstrate the performance and accuracy of our approach, we use the parallel computing test cases from SPLASH-2 [11]. As shown in Fig. 6, we compare the performance and accuracy of the average data process time with the scheduling-based bus model. For these cases, we assume non-preemptive first-in-first-out scheduling policy. For clarity, we adopt the following naming convention. The number after the name of the test case is the number of target cores; AS stands for our proposed runtime activity-sensitive model; ST is the static-trace-activity-based model which estimates contention delay with the activities analyzed from pre-executed trace in static time, where the pre-execution overhead is included for comparison, and CT is the constant-time model which uses an average data process time of all test cases analyzed from scheduling-based model.

Note that CT has higher error rate because its average constant value simply cannot capture the behavior of test cases although some cases coincidentally have similar average contention delay value as that of the scheduling-based model. On the other hand, ST produces reasonable results due to analyzed activities before each execution but the fixed pre-analyzed activities still cannot truly reflect runtime contention behaviors as later discussed in Sec. 5.C. Moreover, the pre-analysis adds to ST extra simulation overhead and causes lower speed up compared to AS and CT. In contrast, because our proposed approach AS dynamically captures bus activities, it produces accurate

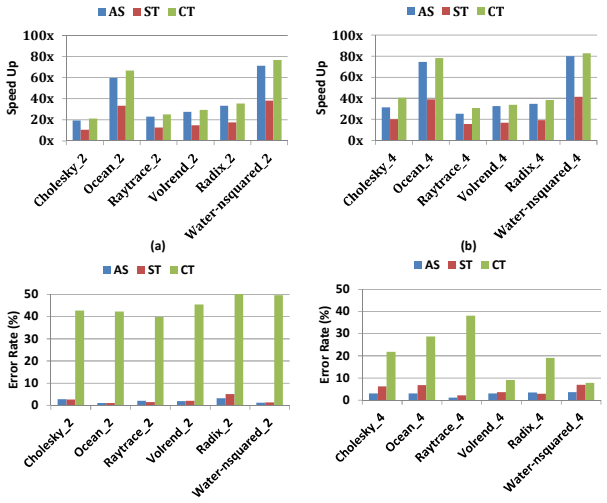

Figure 6: Comparisons on SPLASH-2 benchmark cases with the scheduling-based model. (a) Performance comparison of two-core cases; (b) Performance comparison of four-core cases; (c) Error rate comparison of two-core cases; (d) Error rate comparison of four-core cases.

results with low error rate and with no static analysis overhead. In summary, for two-core simulation cases our approach performs 20 to 72 times faster than the scheduling-based approach while the error rate remains within 2.3% and for four-core simulation cases performs 25 to 80 times faster and have error rate within 2.8%.

C. *Contention Delay Accuracy Comparison*

In order to verify how accurate our proposed model is in runtime, we show in Fig. 7 the average contention delay of each transaction separated by synchronization points. Note that some transactions contain only few requests and hence cause greater variations of contention estimation. However, since the number of requests is small, they also have little effect on the simulated time. Another note is that two-core cases have smaller contention delay compare to four-core cases because of fewer contention behavior. The other note is that at the very beginning of each simulation run both ST and CT insist overly high contention delay while the cold start actually presents very few contentions at the very beginning of bus operation. In contrast, our proposed activity-sensitive model accurately tracks this dynamic.

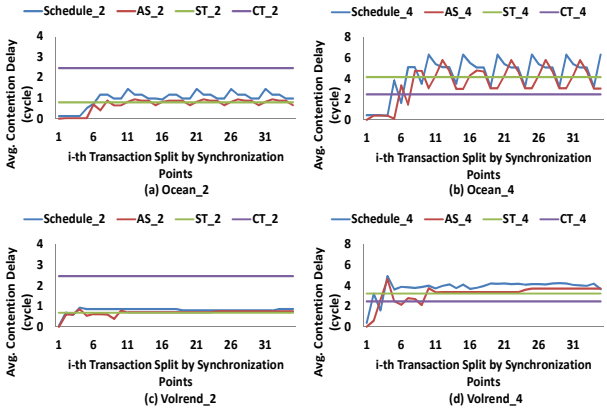

Figure 7: Compare the histogram of average contention delays along the synchronization points.

Note that the two-core cases produce higher error rate than the four-core cases. Mainly this is due the fact that the two-core configuration has fewer active components and smaller average contention delay. Therefore the difference from the estimation is amplified to become a larger error rate value. In general, our model closely tracks the scheduling-based model, which indicates that our approach can indeed accurately model dynamic contention delay based on the latest runtime system activities and effectively capture system configuration changes just as the scheduling-based model does.

D. *System Contention Effect Verification*

To show the effectiveness of our approach on true full-system simulations, we test on an MJPEG decoder [13], whose dynamic behavior is subject to contention situations as shown in Fig. 8. For this test case, one design concern is known exactly how bus contentions affect the decoding time of each frame and therefore subsequent decoding work. We hence compare three bus modeling approaches, scheduling-based, AS, and CT in different system configurations. Since the behavior of ST is similar to CT due to fixed data process time, we simply skip comparison of ST for clarity.

When the MJPEG operates on a system configuration with lower average bus utilization rates (1% and 26%) all modeling approaches report successful decoding of the specified number of frames, i.e. 10 in this case. However, when the average bus utilization rate increases to 73% at the 7-th second, greater contention in the system causes MJPEG fail to decode frames in time and it starts to abandon current frame and skip to decode next one. Note that in this case the constant-time model simply fail to capture this result; in contrast, our proposed activity-sensitive model accurately captures this phenomenon as the scheduling-based model does while our approach is 10.3 times faster than the scheduling-based model for this MJPEG test case.

In summary, our proposed runtime activity-sensitive model is almost as accurate as the scheduling-based model while with great performance improvement. Because of the runtime activity analysis, our proposed approach can easily reflect the effect of system configuration change or activity change from each active component.

VI. CONCLUSIONS

In this paper, we have proposed a new data-pass-through device modeling approach. By analyzing runtime bus activities, our proposed approach efficiently and accurately estimate activity-sensitive contention delay with results closely track with that of the scheduling-based model. The experimental results show that our approach can indeed produce deterministic, high performance and highly accurate full-system simulations. For future work, we will apply the proposed idea to other data-pass-through device such as cache system, which has more complex functional behaviors, to achieve more efficient full-system simulations.

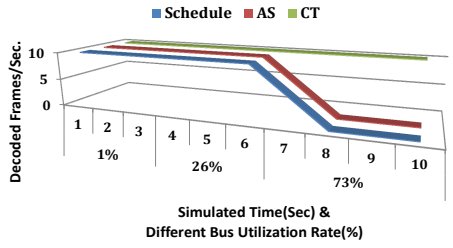

Figure 8: The number of decoded frames per second by decoder with dynamic changing average bus utilization rates in system.

VII. REFERENCES

[1] Wu, Meng-Huan, Wen-Chuan Lee, Chen-Yu Chuang, and Ren-Song Tsay. "Automatic generation of software TLM in multiple abstraction layers for efficient HW/SW co-simulation." In *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 1177-1182. European Design and Automation Association, 2010.

[2] Coware. www.synopsys.com

[3] Caldari, Marco, Massimo Conti, Massimo Coppola, Stephane Curaba, Lorenzo Pieralisi, and Claudio Turchetti. "Transaction-level models for AMBA bus architecture using SystemC 2.0." In *Proceedings of the conference on Design, Automation and Test in Europe: Designers' Forum-Volume 2*, p. 20026. IEEE Computer Society, 2003.

[4] Radetzki, Martin, and Rauf Salimi Khaligh. "Modelling Alternatives for Cycle Approximate Bus TLMs." In *FDL*, pp. 74-79. 2007.

[5] Lo, Chen Kang, and Ren Song Tsay. "Automatic generation of Cycle Accurate and Cycle Count Accurate transaction level bus models from a formal model." In *Design Automation Conference, 2009. ASP-DAC 2009. Asia and South Pacific*, pp. 558-563. IEEE, 2009.

[6] Pasricha, Sudeep, Nikil Dutt, and Mohamed Ben-Romdhane. "Extending the transaction level modeling approach for fast communication architecture exploration." In *Proceedings of the 41st annual Design Automation Conference*, pp. 113-118. ACM, 2004.

[7] Hwang, Yonghyun, Samar Abdi, and Daniel Gajski. "Cycle-approximate retargetable performance estimation at the transaction level." In *Proceedings of the conference on Design, automation and test in Europe*, pp. 3-8. ACM, 2008.

[8] Russell, Jeffry T., and Margarida F. Jacome. "Architecture-level performance evaluation of component-based embedded systems." In *Proceedings of the 40th annual Design Automation Conference*, pp. 396-401. ACM, 2003.

[9] Schirrmeister, Frank, Shay Benchorin, and Filip Thoen. "Using virtual platforms for pre-silicon software development." *White paper, Synopsys (2008)*.

[10] Xu, Min, Rastislav Bodik, and Mark D. Hill. "A flight data recorder for enabling full-system multiprocessor deterministic replay." In *ACM SIGARCH Computer Architecture News*, vol. 31, no. 2, pp. 122-135. ACM, 2003.

[11] Woo, Steven Cameron, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. "The SPLASH-2 programs: Characterization and methodological considerations." In *ACM SIGARCH Computer Architecture News*, vol. 23, no. 2, pp. 24-36. ACM, 1995.

[12] Cucinotta, Tommaso, and Dario Faggioli. "An exception based approach to timing constraints violations in real-time and multimedia applications." In *Industrial Embedded Systems (SIES), 2010 International Symposium on*, pp. 136-145. IEEE, 2010.

[13] Wallace, Gregory K. "The JPEG still picture compression standard." *Communications of the ACM* 34, no. 4 (1991): 30-44.