A Fast and Highly Accurate Statistical Based Model for Performance Estimation of MPSoC on-Chip Bus

FARHAN SHAFLIQ, TSUYOSHI ISSHIKI, DONGJU LI, HIROAKI KUNIEDA
(farhanshafliq, isshiki, dongju, kunieda)@vlsi.ce.titech.ac.jp
Dept. Communications and Computer Engineering, Tokyo Institute of Technology, Tokyo, Japan.

Abstract—While Multiprocessor System-On-Chips (MPSoCs) are becoming widely adopted in embedded systems, communication architecture analysis for MPSoCs becomes ever more complex. There is a growing need for faster and accurate performance estimation techniques for on-chip bus. In this paper, we present a novel statistical based technique that makes use of accumulated "workload statistics" to accurately predict the "stall cycle counts" caused due to bus contention. This eliminates the need to simulate arbitration on every bus access, resulting in substantial speed-up. It is assumed that each Processor in the system has a distinct fixed priority, and arbitration is based on priority. We verify accuracy of our proposed model against results achieved by cycle accurate simulation. Two kinds of traffic is used for experiments. Synthetically generated traffic as well as traffic from real-world application is used to verify the bus model. We report an accuracy with an error range of 0.1% - 5% for the synthetic traffic as well as achieving a speedup of 7x on average. For the real traffic, we use a limited “single blocking” bus model and report results accordingly. (Abstract)

Keywords— bus; statistical model; performance prediction; arbitration.

I. INTRODUCTION

System-On-Chips are rapidly moving towards multiprocessor architectures (MPSoCs). With the increasing complexity of the MPSoCs and on-chip communication architecture under short time-to-market, there is a growing need for faster and accurate design techniques. There is also a growing demand for a highly optimized design at the system-level, on MPSoC architecture as well as application software.

The performance and efficiency of MPSoCs depends heavily upon the efficiency of it’s interconnect. Therefore, there is a need for faster and better techniques for analysis and optimization of on-chip interconnect. Currently, a lot of MPSoCs use bus based communication architectures such as Core-Connect from IBM, AMBA from ARM, SiliconBackplane from sonics etc[1]. Whereas MPSoCs with many more Processing Blocks are employing Networks-on-chip (NoCs) as the communication infrastructure.

In this paper we present a novel performance estimation technique for on-chip bus based architecture. Our approach includes using a statistical based model to accurately predict the stalls caused due to bus contention for a given application. The rest of the paper is organized as follows: section II presents related works. Section III gives a comparison of the simulation model vs the statistical based model. Section IV explains some key concepts that will be used in the rest of the document. Section V and Section VI, being the central parts of the document; explain a simple and a more complex mathematical bus model, respectively. Section VIII shows the experiments and results. Section IX gives a brief conclusion.

II. RELATED WORKS

In this section, we review existing performance evaluation approaches of on-chip communication architectures. Some approaches focus on simulation based performance estimation. As shown in a previous work in [2]. Simulation approaches usually tend to be more accurate but are very costly on simulation time. Commercial tools employ simulation at various abstraction levels [3]. Work in [4] uses static performance estimation techniques. Static techniques are not able to capture the dynamic nature of bus contention. [5] Employ semi Markov process model for estimating performance of multiple-bus communication architecture. Work done in [6] uses generalized semi Markov process (GSMP) model and report an accuracy of 84% compared to simulation. Our proposed approach is different to previous works in that (a) compared to simulation methods its significantly faster (b) our approach uses actual recorded workloads of an application as input hence enabling the model to capture actual dynamic behavior of every application under study which results in the calculation being tailored to every application under study. As opposed to previous approaches, our model does not assume any specific traffic pattern and can be used for several different kinds of applications and SoC configurations.

III. COMPARING SIMULATION MODEL WITH STATISTICAL MODEL

In a simulation based approach, on every bus access, a global scheduler dispatches a processor on processor queue to the simulator where the processor queue is sorted by processor’s simulation clock, or in the case of a tie, by processor’s priority on the bus access. Multiple bus requests are arbitrated by cycle-accurate bus simulation that leads to a huge computational load on the simulation. As shown in figure 1 the statistical based approach works such that for a predefined window of “T” cycles of simulation, arbitration delay on the bus is ignored. During these T cycles, histograms on all bus workloads $h_{Bl}$ and computation workloads $h_{Cl}$ are collected. These statistics are used by a mathematical model to calculate the expected arbitration delay per request $E[D_i]$. The total expected delay, $E[D_i] \times N_i$ is added to the total number of cycles.
IV. KEY CONCEPTS

A. Computation Workload

Computation workload $L_i$ denotes the number of execution cycles between two successive bus accesses on a Processing Element $PE_i$ where $i$ indicates the priority of a PE. Let $N_i$ be the total number of computation workloads on $PE_i$.

B. Bus Workload

Bus workload $B_i$ denotes the number of execution cycles between two successive computation workloads on $PE_i$. Total number of bus workloads on $PE_i$ are same as $N_i$.

C. Request Probability and average interval workload

Let $\lambda_i$ be the probability that a bus request $r_i$ occurs at each cycle on $PE_i$. Probability that interval $L_i$ equals $n$ (cycles) is given as:

$$\Pr(L_i = n) = \lambda_i (1 - \lambda_i)^{n-1} \quad (n \geq 1)$$

Here, expectation of interval $L_i$ ($E[L_i]$) is given as:

$$E[L_i] = \sum_{n=1}^{\infty} \Pr(L_i = n) \cdot n = \frac{1}{\lambda_i}$$

Hence request probability is given as:

$$\lambda_i = \frac{1}{E[L_i]}$$

D. Request inactivation Probability

On each occurrence of bus workload $B_j$, probability that request $r_i$ does not occur within the duration of $B_j - 1$ cycles on $PE_i$ is called request inactivation probability $Y_{ij}^r$.

V. MATHEMATICAL MODEL SINGLE BLOCKING MODEL

from here on we will use following terminologies throughout this document, $r_i$ : a request event by processor $PE_i$, $b_j$ : a bus event (with arbitrary length $B_j$) at processor $PE_j$, $blk_{ij}$ : a blocking event by bus event $b_j$ on request $r_i$, $b_{ij}(k)$ : a bus event with length $B_j = k$ at processor $PE_j$, $blk_{ij}(k)$ : a blocking event by bus event $b_j(k)$ on request $r_i$, $t_{ij}(k)$ : time difference between the first cycle of $b_j(k)$ and request $r_i$. $E[D_{ij}]$: Expectation of bus stall per request at processor $PE_i$, $E[D'_{ij}]$: Expectation of bus stall per request at processor $PE_j$.

First we introduce a simple bus model such that, (a) bus requests on all PE are generated randomly with probability $\lambda_i$ (b) Blocked requests must be granted after the current workload on the bus finishes.

The Binary value $\alpha_{ij}$ describes the fixed priority relationship between processors $PE_i$ and $PE_j$.

$$\alpha_{ij} = \begin{cases} 0 & \text{priority}(PE_i) < \text{priority}(PE_j) \\ 1 & \text{priority}(PE_i) > \text{priority}(PE_j) \end{cases}$$

As shown in Figure 2 request blocking condition of request $r_i$ by bus event $b_j(k)$ is given as follows:

$$a_{ij} \leq t_{ij}(k) \leq k - 1$$

A. Bus stall expectation equation

First, bus event $b_j$ that potentially blocks the request $r_i$ is further categorized into two subcases namely, (a) Consecutive bus event $bb_{ij}$: bus event $b_j$ at processor $PE_j$ follows immediately after (with no interval) a bus event at processor $PE_i$. Let $S_{ij}$ be the probability of event $bb_{ij}$ i.e. $S_{ij} = \Pr(bb_{ij})$. (b)Non-consecutive bus event $\overline{bb}_{ij}$: bus event $b_j$ at processor $PE_j$ follows a bus event $b_k$ at processor $PE_k$ after one or more cycles. Clearly:$S_{ij} = \Pr(bb_{ij}) = 1 - \Pr(bb_{ij}) = 1 - S_{ij}$. Here, among $N_j$ occurrences of event $b_j$ on processor $PE_j$, there are $N_j \cdot S_{ij}$ occurrences of event $bb_{ij}$ and $N_j \cdot (1 - S_{ij})$ occurrences of event $\overline{bb}_{ij}$.

Furthermore, events $bb_{ij}$ and $\overline{bb}_{ij}$ are annotated as $bb_{ij}(k)$ and $\overline{bb}_{ij}(k)$ to denote the specific bus workload length $k$ of $b_j(k)$. Also, let $S_{ij}(k) = \Pr\left(bb_{ij}(k)\right)$ and $\overline{S}_{ij}(k) = \Pr\left(\overline{bb}_{ij}(k)\right)$ be the probabilities of event $bb_{ij}(k)$ and $\overline{bb}_{ij}(k)$. If we assume that bus events $b_j(k)$ with different lengths $k$ are generated randomly with probability $f_{bb}(k)$, then $S_{ij}(k)$ and $\overline{S}_{ij}(k)$ are given as:

$$S_{ij}(k) = S_{ij} \cdot f_{bb}(k)$$

$$\overline{S}_{ij}(k) = \overline{S}_{ij} \cdot f_{bb}(k) = (1 - S_{ij}) \cdot f_{bb}(k)$$

Figure 3 illustrates the blocking probabilities and bus stall delays on event $bb_{ij}(k)$ with specific values of $t_{ij}(k)$. Here, the probability of $t_{ij}(k) = n$ on event $bb_{ij}(k)$ is given as:

$$\Pr(t_{ij}(k) = n|bb_{ij}(k)) = \lambda_i \cdot (1 - \lambda_i)^{n-1}$$

Whereas bus stall length is given as:

$$D'_{ij}(k|t_{ij}(k) = n, bb_{ij}(k)) = k - n$$

Then the conditional bus stall delay on event $bb_{ij}(k)$ are:

$$E[D'_{ij}(k|bb_{ij}(k)) = k \lambda_i - 1 + (1 - \lambda_i)^k \lambda_i$$
Where the effective ranges of \( \alpha_{ij} = 0 \) and \( \alpha_{ij} = 1 \) are:

\[
\begin{align*}
\lambda_i & : \lambda_i(1-\lambda_i)^n \quad (\alpha_{ij} = 0) \\
\lambda_i(1-\lambda_i)^{n-1} & : \lambda_i(1-\lambda_i)^n \quad (\alpha_{ij} = 1)
\end{align*}
\]

(8)

Next, the overall bus stall delay expectation \( E[D_{ij}(k)] \) on all bus events \( bb_{ij}(k) \) are derived as follows:

\[
E[D_{ij}(k)] = E[D'(ij)(k) bb_{ij}(k)] \Pr(bb_{ij}(k))
\]

(11)

\[
E[D_{ij}(k)] = Q'_i E[D'(ij)(k)]
\]

(12)

Where \( Q'_i = \frac{N_i}{N'_i} \). Here \( D'(ij)(k) \) is the probability that \( bb_{ij}(k) \) occurs among \( N'_i \) bus events on processor \( PE_i \). On the other hand, \( D_{ij}(k) \) is the probability that \( bb_{ij}(k) \) occurs among \( N_i \) bus events on processor \( PE_i \). Since \( R_{ij}(k)N_i = R'_i(k)N'_i \), the factor \( Q'_i \) is introduced to convert the two probabilities observed on processors \( PE_i \) and \( PE'_i \).

Over all \( E[D_{ij}(k)] \) for \( \alpha_{ij} = 0 \) becomes:

\[
E[D_{ij}(k)] = Q'_i \left( \frac{k\lambda_i - 1 + (1 - \lambda_i)^k}{\lambda_i} \cdot S_{ij}(k) \right)
\]

\[
+ \left( \frac{k + 1}{\lambda_i} \right) \lambda_i(1 - \lambda_i)^k \cdot \mathcal{S}_{ij}(k)
\]

\[
= Q'_i f_{B_j}(k) \left( \frac{k\lambda_i - U_{ij}(1 - (1 - \lambda_i)^k)}{\lambda_i} \right)
\]

Finally, the overall “bus stall” expectation \( E[D_{ij}(k)] \) on all bus events \( b_j \) (with arbitrary length) are:

\[
E[D_{ij}(k)] = \sum_k E[D_{ij}(k)]
\]

Therefore,

\[
E[D_{ij}] = \begin{cases} 
Q'_i \left( E[B_j] - U_{ij}(1 - V'_{ij}) \right) & (\alpha_{ij} = 0) \\
Q'_i \left( E[B_j] - \frac{1 - V'_{ij}}{\lambda_i} \right) & (\alpha_{ij} = 1)
\end{cases}
\]

(12)

Where, \( Y'_{ij} = \sum_k f_{B_j}(k) (1 - \lambda_i)^k \) and \( V'_{ij} = (1 - \lambda_i)Y_{ij} \)

B. Consecutive bus-event probability

As shown in Figure 4 Let \( bb_{ij}(k) \) be an event where a bus event \( b_j \) (with arbitrary length) immediately follows a bus event \( b_i \). This event \( bb_{ij}(k) \) occurs on the following two cases, (a) When \( blk_{ij}(k) \) occurs (bus event \( b_i(k) \) blocks a request \( r_j \)) (b) When request \( r_j \) occurs immediately after bus event \( b_i \). Let \( S_{ij} = \Pr(bb_{ij}) \). Due to space constraints the derivation is omitted, however \( S_{ij} \) is given as:

\[
S_{ij} = \begin{cases} 
Q'_i (1 - U_{ij} V'_{ij}) & (\alpha_{ij} = 0) \\
Q'_i (1 - V'_{ij}) & (\alpha_{ij} = 1)
\end{cases}
\]

(13)

C. Expressions for calculating bus stall

Using values of \( S_{ij} \) and \( \lambda_i \) from equations 13 and 2 respectively equation 11 becomes,

\[
E[D_{ij}] = \begin{cases} 
Q'_i \left( E[B_j] - (E[L_{ij}] - 1) (1 - V'_{ij}) \right) - (1 - V'_{ij})(1 - V'_{ij}) & (\alpha_{ij} = 0) \\
Q'_i \left( E[B_j] - E[L_{ij}] (1 - V'_{ij}) \right) & (\alpha_{ij} = 1)
\end{cases}
\]

Let us now discuss the bus event count ratio \( Q'_i = \frac{N_i}{N'_i} \). Although \( N_i \) and \( N'_i \) are the actual bus event counts observed during the bus predication interval, we need to take into account the fact that these bus event counts resulted while we ignored the bus stall delays, and therefore this will lead to
inaccuracy if used directly. In order to include the bus stall delay effects in the bus event count ratio, we define the average bus access interval \( G_i \) as

\[
G_i = E[L_i] + E[B_i] + E[D_i] \tag{16}
\]

Where \( E[L_i] \) is the interval workload expectation, \( E[B_i] \) is the bus workload expectation, and \( E[D_i] \) is the bus stall delay expectation:

\[
E[D_i] = \sum_{j \in S} E[D_{ij}] \tag{17}
\]

Then the bus event count ratio \( Q_{ij} \) is calculated as

\[
Q_{ij}' = \frac{E[L_i] + E[B_i] + E[D_i]}{E[L_i] + E[B_i] + E[D_i]} \tag{18}
\]

Here, \( E[D_i] \) and \( E[D_j] \) are the predicted bus stall delays that will be calculated by iterative method.

VI. MATHEMATICAL MODEL (MULTI BLOCKING MODEL)

Now we introduce a more complex bus model such that, requests may be blocked by an infinite number of bus workloads. This model can be applied to “N” number of PEs such that zero interval probability \( \mu_i > 0 \). For this model we define two kinds of bus-workload. (a) “Raw bus workload” refers to the actual bus workload on each PE, as opposed to (b) “effective bus workload” which refers to a merged bus workloads. Merged bus workload as observed on PE \( p \) by PE \( q \) refers to the following situations (a) After termination of a bus workload on PE \( p \), a request is immediately generated with probability \( \mu_j \) on PE \( p \) such that \( \alpha_j = 0 \). (b) After termination of a bus workload on PE \( p \), a bus workload immediately starts on PE \( q \) with probability \( \mu_j \) such that \( \alpha_j = 0 \). Then the probability of \( L_i = n \) is:

\[
\Pr(L_i = n) = \left\{ \begin{array}{ll}
\frac{\mu_i (1 - \mu_i)^n}{(1 - \lambda_i)(1 - \lambda_i)^n - 1} & (n > 1) \\
0 & (n = 0)
\end{array} \right.
\]

And, expectation of interval \( E[L_i] \) is given as:

\[
E[L_i] = \sum_{n=0}^{\infty} n \Pr(L_i = n) = \frac{1 - \mu_i}{\lambda_i} \quad \lambda_i = \frac{1 - \mu_i}{E[L_i]} \tag{20}
\]

In the above mentioned two cases, as shown in Figure 5, a bus request \( r_i \) on the processor with lower priority sees a merged bus workload that potentially blocks its request. Note that merging can in fact occur consecutively as well.

**Figure 5: Merged bus workload**

A. Calculation of merged busworkload

Bus request \( r_i \) sees a merged bus workload on higher priority PEs. Let’s define, “\( C_j \)”: Probability that \( b_j \) immediately follows \( b_j \) (observed at PE \( p \)). “1 - (\( \sum_{i<j} C_j \))”: Probability that \( b_j \) is not followed by \( b_j \) or a bus workload on a higher priority PE compared to PE \( p \). The probability mass function for raw bus-workload \( f_{B_j}(k) \) is divided as:

\[
\{ (1 - (\sum_{i<j} C_j))f_{B_j}(k) \}, \quad \{ (C_jf_{B_j}(k)) \}
\]

Notice the effective bus-workload, effective request inactivation probability and effective event count ratios values are used. Also, instead of using the actual event count ratio we will use an effective bus event count such that \( Q_{ij} = Q_{ij}'K_{ij} \) and

\[
K_{ij} = \left\{ \begin{array}{ll}
1 & (\alpha_{ij} = 0) \\
1 - \sum_{t<i} S_{ij} & (\alpha_{ij} = 1)
\end{array} \right.
\]

\[
w_{ij} = \{ (1 - \mu_i)_{S_{ij}} u_{ij} \} = \left\{ \begin{array}{ll}
(1 - \mu_i)(1 - S_{ij}) & (i > j) \\
1 - S_{ij} & (i < j)
\end{array} \right.
\]

Here, note that \( C_j \) is defined against \( N_j \) bus-workloads on \( PE_j \) as opposed to \( S_{ij} \) that is defined against \( N_i \) bus-workloads on \( PE_i \). Let \( E[B_{ij}] \) be expectation of effective bus workload \( B_{ij} \). Let \( E[B_{ij}]' \) be expectation of all merged bus-workloads starting with \( b_j \) and terminating with \( b_j \). Due to space constraints we omit the derivation. However, for “n” number of higher priority PEs,

\[
\begin{bmatrix}
E[B_{00}] & \ldots & E[B_{0n}]
\vdots & \ddots & \vdots
0 & \ldots & E[B_{nn}]
\end{bmatrix} = \left( I - C_n \right)^{-1} (25)
\]

Finally, the expectation of effective bus workload \( E[B_{ij}] \) is,

\[
E[B_{ij}] = \sum_{l<i} E[B_{ij}]' \tag{26}
\]

B. Calculation of Request inactivation probability

Here we show the expressions for calculating the effective request inactivation probability. Let \( Y_{ij} \) be the effective request inactivation probability on PE \( p \) over a merged bus-workload starting on PE \( q \) and ending on PE \( k \). For “n” number of higher priority PEs,

\[
\begin{bmatrix}
Y_{00}' & \ldots & Y_{0n}'
\vdots & \ddots & \vdots
Y_{in}' & \ldots & Y_{nn}'
\end{bmatrix} = H_n (I - V_n C_n)^{-1} y_{n}' \tag{26}
\]

Where

\[
V_n' = \begin{bmatrix}
V_{00}' & \ldots & 0 \\
\vdots & \ddots & \vdots \\
0 & \ldots & V_{nn}'
\end{bmatrix}
\]

Finally, the effective request inactivation probability is,

\[
Y_{ij} = \sum_{l<i} Y_{ij}' \tag{27}
\]

C. Expressions for calculating bus stall

The overall bus stall is given by using equation (12) with some differences.

\[
E[D_{ij}] = \begin{cases}
Q_{ij} \left( E[B_{ij}] - (w_{ij} + u_{ij}) \frac{(1 - V_{ij})}{\lambda_i} \right) & (\alpha_{ij} = 0) \\
Q_{ij} \left( E[B_{ij}] - \frac{(1 - V_{ij})}{\lambda_i} \right) & (\alpha_{ij} = 1)
\end{cases} \tag{12}
\]

Notice that the effective bus-workload, effective request inactivation probability and effective event count ratio values are used. Also, instead of using the actual event count ratio we will use an effective bus event count such that \( Q_{ij} = Q_{ij}'K_{ij} \) and

\[
K_{ij} = \left\{ \begin{array}{ll}
1 & (\alpha_{ij} = 0) \\
1 - \sum_{l<i} S_{ij} & (\alpha_{ij} = 1)
\end{array} \right.
\]

\[
w_{ij} = \{ (1 - \mu_i)_{S_{ij}} u_{ij} \} = \left\{ \begin{array}{ll}
(1 - \mu_i)(1 - S_{ij}) & (i > j) \\
1 - S_{ij} & (i < j)
\end{array} \right.
\]
Some of the derivations in the mathematical model have been omitted due to space constraints.

VII. EXPERIMENTS AND RESULTS

Below we show how the model presented in our work will be used for performance prediction.

A. Input

The mathematical model uses “traffic workload statistics” as an input for its calculations. Histograms are maintained for “bus workload” and “computational workload”. While, an “SoC configuration file”, contains information about (a) Number of Processing Elements (b) Priority of every Processing Element (c) Number of buses (d) bus mapping (e) Memory. The bus mapping determines potential bus contention areas. Whereas the histograms are used by the mathematical model to predict the stall resulting from those contentions.

B. Traffic

For our experiment we use two kinds of traffic.

1) Synthetically generated traffic

We use a synthetic traffic generator that generates traffic with certain traffic characteristics as specified in a text file and can be changed easily by hand. The traffic characteristics include (a) Total number of traffic generators (PE) (b) ID of generator (c) Packet Length of generated traffic on each generator (d) Average interval between two successive packets on each generator and (e) total simulation time.

2) Recorded traffic patterns of real applications

We use a real-world JPEG encoder application to evaluate our prediction model in a real-world scenario. The reference program is made available by the Independent JPEG Group [7]. At the moment due to technical issues we are able to report results based on this application only for our single blocking model.

C. SoC configuration

For our synthetic traffic experiments we use three different SoC configurations with 2, 3, and 4 PEs respectively. Figure7 shows 3 SoC configurations. All configurations have a single shared bus that connects the PEs. Whereas for real application we use a 8-PE system.

D. Simulation flow

The simulation flow consists of three phases. (i) In the Input phase, histograms of traffic statistics are read by the simulation engine. (ii) In the prediction phase, for a window of T cycles (bus prediction interval), “traffic statistics” are used by the prediction model to predict total bus stall cycle count for the prediction interval and added to the processor’s simulation clock. (iii) Phase ii is repeated until the end of the traffic cycles is reached.

E. Bus stall calculated by simulation

For verification we compare our predicted stall cycle count with simulated stall cycle count. A comparison of simulation and prediction approach is reported in section III.

F. Results

The results from our mathematical model are compared against results from simulation for verification. This simulation model has been a part of a previous work in [2].

1) Result 1

first we performed experiments for “SoC config1”, that comprises of two PEs, using synthetically generated traffic. Table1 shows details of three different synthetically generated traffic on each PE such that PE(BW,CW) specifies the “ID”, “average Bus-workload” and “average Computation – workload” on each PE. “sim.Stall” reports the “bus stall per request” calculated by simulation while “prd.Stall” reports the “bus stall per request” predicted using the proposed prediction model. “error” reports the error in predicted values calculated by the formula “((sim.Stall- prd.Stall)/ sim.Stall”.

Figure7 shows a comparison of “sim.Stall” and “prd.Stall” for three different traffic patterns and “percent error” in prediction vs simulation. The error in this case is under “0.1%”.

Table 1: Prediction results for 2PE system

<table>
<thead>
<tr>
<th>PE(BW,CW)</th>
<th>sim.Stall</th>
<th>prd.Stall</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0(6,5)</td>
<td>8.67556</td>
<td>8.67436</td>
<td>0.00013832</td>
</tr>
<tr>
<td>P1(20,9)</td>
<td>3.864428</td>
<td>3.864936</td>
<td>-0.00013455</td>
</tr>
<tr>
<td>P9(20,5)</td>
<td>11.476614</td>
<td>11.47577</td>
<td>7.35409E-05</td>
</tr>
<tr>
<td>P1(20,9)</td>
<td>17.998777</td>
<td>17.99827</td>
<td>2.81686E-05</td>
</tr>
<tr>
<td>P9(7,25)</td>
<td>0.672208</td>
<td>0.671632</td>
<td>0.00056878</td>
</tr>
<tr>
<td>P1(7,25)</td>
<td>1.638099</td>
<td>1.636371</td>
<td>0.001054881</td>
</tr>
</tbody>
</table>

Figure 7: Comparison between predicted and simulated stall for 2PE system

2) Result 2

Experiment2 uses “SoC config2” which comprises of 3PEs. Table2 shows details of three different synthetically generated traffic on each PE. Figure8 shows a comparison of “sim.Stall”
and “prd.Stall”. The prediction results are very accurate with a maximum error of around 3.3% as per given traffic patterns.

3) result3

In the next experiment we verify our model using “SoCconfig3” which comprises of 4PEs. Table3 shows details of the synthetically generated traffic, Figure9 compares “sim.Stall” and “prd.Stall”. The prediction results show an error of about 5.2% for stall predicted for the lowest priority PE. We compared execution times of the “simulation method” and “prediction method”. We were able to achieve on average 7x speedup. Note that the speed up results are shown for a simulation run for 800,000 cycles. For a simulation of 5000,000 cycles, simulating the bus arbitration takes in order of 1000ms. Given that the designer needs to evaluate multiple mapping, architecture configurations and tweak application accordingly, the overall speedup could be significant. Moreover, bus contention simulation increases with increasing bus requests whereas the time required for proposed prediction technique is the product of “number of time-windows” and “prediction time for each window”.

4) result4

Next we use a real-world application. Due to technical issues we are only able to report results for the “single-blocking request model” which only caters for two PEs at a time. We apply this model to a system with 8PEs. The predicted bus stalls, even though containing 13%-50% error, does help us get an over-all parallel record of the timing. Figure10 compares parallel record achieved using bus simulation against prediction model.

VIII. CONCLUSION

In this paper we introduced a novel stall prediction technique for MPSoC based on a statistical model. We used workload statistics to predict bus stalls. The effectiveness of our method is confirmed on a set of experiments on synthetically generated traffic. In the future we want to build on this work and perform more experiments on real-world applications.

Table 3: Prediction results for 4PE system

| PE(BW,CW) | sim.Stall | prd.Stall | error  
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P0(15,14)</td>
<td>1.725689</td>
<td>1.73589</td>
<td>0.0003262</td>
</tr>
<tr>
<td>P1(15,14)</td>
<td>2.6</td>
<td>2.599</td>
<td>0.00034615</td>
</tr>
<tr>
<td>P2(15,14)</td>
<td>4.485717</td>
<td>4.58</td>
<td>-0.01231522</td>
</tr>
<tr>
<td>P3(15,14)</td>
<td>3.6</td>
<td>3.241</td>
<td>0.01531628</td>
</tr>
<tr>
<td>P0(18,18)</td>
<td>3.346957</td>
<td>3.346</td>
<td>0.00017859</td>
</tr>
<tr>
<td>P1(11,25)</td>
<td>3.9584</td>
<td>3.958</td>
<td>0.00035589</td>
</tr>
<tr>
<td>P2(15,14)</td>
<td>9.2566</td>
<td>9.295761</td>
<td>0.01534097</td>
</tr>
<tr>
<td>P3(9,29)</td>
<td>15.0364</td>
<td>15.831</td>
<td>-0.05284596</td>
</tr>
</tbody>
</table>

REFERENCES