An Accurate and Fast Trace-aware Performance Estimation Model For Prioritized MPSoC Bus With Multiple Interfering Bus-Masters

Farhan Shafiq1,a) Tsuyoshi Ishiki1,b) Dongju Li1,c)

Abstract: Accurate and fast performance estimation methods for modern and future multi-core systems are the focal point of much research due to the complexity associated with such architectures. The communication architecture of such systems has a huge impact on the performance and power of the whole system. Architects need to explore many design possibilities by using performance estimation techniques at early stages of design to make design decisions earlier in the design cycle. While software developers need to develop and test applications for the target architecture and gather performance measurements as early in the design cycle as possible. Full system simulation techniques provide accurate performance values but are extremely time consuming. Static analysis techniques are fast but cannot capture the dynamic behavior associated with shared resource contention and arbitration. Moreover, synthetic traffic patterns have been used to analyze the communication architecture, however, such patterns are not realistic enough. We propose a statistical based model to predict the dynamic cost of bus arbitration on the performance of a bus architecture. The proposed model uses workload trace of the actual applications and benchmarks to capture the real application traffic behavior. Statistics on the traffic patterns are collected and input to the analytical model which calculates performance values for the communication architecture under consideration. By knowing the performance measures, designers can avoid over and under-design of the communication architecture. This paper builds up on a previously developed performance estimation model. The previous work modeled single and burst bus-transfers, however only one interfering bus master at a time for each blocked bus request was considered. The proposed, improved accuracy model considers multiple interfering masters for each blocked request hence improving the estimation accuracy especially for traffic intensive applications and many PE architectures. Experiments are performed for two different architectures i.e., 4 processing elements connected via a shared bus and 8 processing elements connected via a shared bus. Results show no significant difference in accuracy compared to previously developed model, for low traffic applications SPARSE and ROBOT however notable accuracy improvement for traffic intensive applications. Maximum estimation error is reduced from 1.75% to 0.6% for FPPP and from maximum 13.91% to 8.8% for FFT on the 4PE architecture. On the 8PE architecture, maximum estimation error is reduced from 11.8% to 2.7% for the FPPP benchmark. Moreover simulation speed-up for the proposed technique over simulation method is reported.

Keywords: MPSoC shared bus, bus contention model, performance estimation, statistical model, multiple bus masters

1. Introduction

The rapid adoption of multiprocessor architectures by System-On-Chips means architects face a larger design space, bigger extent of design decisions, complexity and bigger tradeoffs. Along with processing performance, the performance of communication architecture also plays a very important role in the overall performance of the design. Communication architectures require a very detailed and careful consideration during the design phase in order to meet performance, power and latency constraints. To achieve this, the designer not only needs to understand the design but also the target applications and the corresponding application traffic behaviors of each application for the SoC to be designed.

Currently, a lot of multi processor System on Chips (mpsoc) use bus based communication architectures due to their simplicity and popularity such as Core-Connect from IBM[1], AMBA from ARM[2], SiliconBackplane from Sonics[3], STBus from STMicroelectronics[4] etc. Whereas MPSoCs with many more Processing Blocks are increasingly employing Networks-on-chip (NoCs) as the communication infrastructure[5], [6]. In this paper, we focus on bus-based communication architecture.

Most commonly, Simulation is used to study the performance of a subject architecture running a number of target applications. There are several simulation methodologies with different levels of accuracy and speed tradeoffs. At one end of the spectrum static analysis techniques are used to predict performance of an architecture resulting in very quick simulation estimation however, lack the level of accuracy needed for a complex mpsoc design. “Full system simulators”, on the other hand, model each hardware component and run full OS and applications providing the...
highest accuracy however very long simulation times that are not suitable for rigorous and iterative estimation.

In this paper we try to address this issue by presenting a novel performance estimation technique for on-chip bus based architecture. Our approach uses a statistical based model to accurately predict the dynamic stalls caused due to bus contention for a given application. Statistical models usually assume that bus requests are distributed evenly throughout the whole execution duration of an application however; this assumption is a source of inaccuracy since bus access behavior of actual applications is time varying[7] and cannot be modeled this way. To overcome this issue we assume that workload statistics for computation as well as bus traffic on each processing element are known. This assumption complies well with a workload simulation technique for example as presented in Ref.[8]. The estimation technique works such that, for an arbitrary window of “T” cycles, histograms on all bus-workloads and computational-workloads on each PE for a given application are provided to the prediction model which calculates the bus contention stall for each PE. The resulting stall cycle counts are added to the overall cycle counts on each PE. Unlike a simulation approach, which requires arbitration simulation on every bus request, our proposed technique runs once every “T” cycles. Since the value of “T” can be chosen to be big or small in accordance with the total number of cycles, the time required for performance estimation does not increase drastically with increasing number of bus workloads. Moreover, unlike other researches in this area, that solely focus on bus-architecture design space exploration, we aim to use this model to enable application developers to perform “bus-performance aware” application optimization and partitioning and provide a better understanding of the bus performance of an application on a target architecture. Once application optimization is satisfactory and a suitable set of bus architectures and mapping are finalized, simulation techniques can be used hence resulting in a considerably shorter simulation time.

This research builds on a previous performance estimation model which took into account single and burst transfers on only one interfering bus master at a time for each blocked bus request[9]. The proposed, improved accuracy model considers multiple interfering masters for each blocked request hence improving the estimation accuracy especially for traffic intensive applications and many PE architectures. The previously developed models are named Single Blocking Model (SBM) and Burst Blocking Model (BBM) while the proposed model in this paper is named Multi Blocking model (MBM). The difference between the three is detailed in Section 4.3.4.

The rest of the paper is organized as follows: Section 2 gives an account of related works and our contribution, Section 3 gives an overview of the proposed simulation flow, Section 4 explain the previously developed Single Blocking Model (SBM) and Burst Blocking Model (BBM), and Section 5 describes the proposed Multi-Blocking Model (MBM). Section 6 gives a summary of experiments and results. Section 7 outlines the future works and Section 8 concludes the paper.

2. Related Works

Related works cover the literature associated with performance estimation techniques for MPSOC bus architectures.

There have been a few approaches to address the need for performance estimation of an mpsoc shared bus. Some researchers have focused on static techniques for communication architecture performance estimation. Knudsen et al. presented a high level estimation model for communication throughput for a given protocol assuming pipelined transfers[10]. Yen and Wolf propose to estimate the communication delay using the worst-case response analysis of the real-time scheduling[11]. Daveau et al. considered static information like maximum bandwidth of channel and bandwidth of processing elements to estimate performance of interconnect between processing elements[12]. Nandi and Marculescu use continuous-time Markov process technique for performance measurement[13]. Drinic et al. used the profiled statistics of inter-core traffic for core-to-bus assignment[14]. Thepayaswan and Doboli propose a simulated annealing approach[15]. Cho et al. proposed analytical performance model for AMBA 2.0 AHB single and hierarchical shared bus architectures, assuming bus slaves do not introduce any waits[16]. However these techniques are not able to model bus contention due to its dynamic nature and hence these techniques are not suitable for today’s MPSOC.

Simulation based approaches have been more popular for performance estimation. Simulation is performed at various abstraction levels. Loghi et al. used Cycle Accurate models written in SystemC to explore AMBA2 and STBus communication architectures for MPSO Cs[17]. Sermeria et al. used PA-BCA models (modeled in systemC) to improve simulation speed over CA models[18]. Kalla et al. executed traces of component behavior on a Pin Accurate Bus Cycle Accurate simulator and achieved speed up over CA simulation model[19]. Caldri et al. used transaction based bus cycle accurate approach to model AMBA2 using function calls for read/writes using SystemC 2.0. Capturing communication systems using TLMs has been tried due to its standardization[20], [21]. Ogawa et al. created another T-BCA model variant for the AMBA AHB bus architecture using C as the modeling language[22]. Ariyamparambath et al. annotated ATLM models with bus-protocol-specific timing details[23]. Viaud et al. proposed TLM/T abstraction level[24]. Schriner et al. report a quantitative analysis of speed-accuracy tradeoff of TLM, using the advanced high-performance bus (AHB) as a test case, at different abstraction levels[25]. Beltra me et al. proposed using multiple levels of abstraction for communication architecture exploration, with the ability to dynamically shift between BCA, untimed TLM and timed TLM abstractions to improve simulation speed[26]. A simulation-based method gives accurate estimation results but pays too heavy a computational cost with increasing number of bus requests and simulation iterations. FPGA based simulation has been proposed[27], [28], [29] to speed-up simulation, however, implementing the architecture on an FPGA in early design phase is usually not possible.

To overcome this difficulty, a hybrid approach (between a static estimation and a simulation approach) has been developed by
Lahiri et al. [30]. They used some static analysis to group the traces and apply a trace-driven simulation with the trace groups however their approach converges to trace driven simulation as the memory traces become larger. Kim et al. proposed another hybrid performance estimation approach based on queuing analysis [31]. However, static queuing techniques are inherently insufficient to handle complex bus protocol features. Moreover, due to the use of FSM, the number of events/state transitions can explode with increasing PEs. Kawahara et al. propose a simulation method that takes memory access contention into account for evaluation of the execution time of an application program [32]. However, the analysis is not based on “actual trace” of a program rather UML or state-chart of the program is simulated which results in longer simulation time. Moreover the experiments are performed for only two bus-masters.

2.1 Multi Blocking Model Contribution

Multi Blocking Model (MBM) provides improved accuracy over the previously developed SBM and BBM. The SBM assumes that, when blocked, a bus request will be granted as soon as the current bus master releases the bus. The BBM builds on the SBM by including burst transfers on the current bus master when the current bus master has a higher priority than the blocked bus request. This model can capture the bus performance with reasonable accuracy when application traffic is not intensive and number of bus masters is limited. However, when number of PEs or application traffic increases, a single low priority PE can be blocked by multiple high priority bus masters at a time. A bus request on a low priority PE is blocked by a higher priority PE until the higher priority PE releases the bus. However, during the time while the low priority PE was blocked, another high priority PE may issue a bus request. Once the current bus master releases the bus, the higher priority bus wins bus arbitration and the bus request on the lower priority PE gets blocked again. This kind of blocking can happen indefinitely. We call this kind of blocking by multiple interfering bus masters as “Multi Blocking”. The BBM does not model Multi Blocking. The proposed Multi Blocking Model accounts for multi blocking behavior and hence improves on estimation accuracy as opposed to BBM.

3. Overview of Proposed Simulation Flow

Let us explain the simulation flow of our proposed technique for performance estimation. Although the model can be used with any schedule aware simulator where the computational and bus workloads on each PE are known, we use our bus model as an addition to the trace driven workload simulator presented in Ref. [8]. Figure 1 shows an overview of the performance estimation flow in a trace driven workload simulation. A partitioned application program’s execution trace on a given set of input data is computed, through source-level instrumentation and native code execution and encoded as branch bit-stream. Moreover, workload model for the application is generated. The branch bit-stream is then used to steer workload models inside the trace driven workload simulator for a specific target MPSoC architecture model. Readers are directed to Ref. [33] for detailed reading. However whenever there is a bus access, the bus stall due to contention cannot be calculated from the workload models and on every bus request, arbitration must be simulated to resolve any bus contention. We propose to eliminate this simulation part and replace it with a statistical prediction model. A comparison of the simulation and prediction method will be shown below. Either way, at the end of the workload simulation, an estimation on the cycle counts is produced. Depending on the performance numbers, application optimization or partitioning improvements can be performed and the above steps can be repeated to estimate bus performance. This results in a very fast performance estimation framework. We aim that the estimation on application cycle-counts using our technique can work well with the Tightly Coupled Thread model [34] which enables programmer to specify system level partitioning directly on reference C code.

3.1 Arbitration Simulation Technique in a Trace Simulator

In the simulation technique, arbitration must be simulated to resolve any bus contention on every bus access. There needs to be a global scheduler inside the trace simulator kernel that dispatches a processor on processor queue to the workload 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 must be arbitrated by cycle-accurate bus simulation that leads to a huge computational load. Working principles of the simulation based bus model is illustrated in Fig. 2. In the trace simulator presented in Ref. [8] a Program Trace Graph (PTG) is used for efficient trace retrieval. The PTG consists of nodes and edges such that each PTG-node represents function-start, function-end, branch or call. A PTG-edge connects two PTG-nodes and carries attributes about the program execution information, including cycle count between the two PTG-nodes it connects. On the trace simulator,
the workload corresponding to each PTG-edge \( e_n \) at processor \( P_{E_i} \) \((i = 0, 1, \ldots)\) contains at most a single leading “bus workload” \( B_i[e_n] \), representing memory access instruction that generates bus traffic, which is followed by normal “computation workload” \( C_i[e_n] \), representing normal instruction executions. Computation workload \( C_i[e_n] \) denotes the number of execution cycles on an instruction stream contained in an edge \( e_n \). Bus workload \( B_i[e_n] \) denotes the number of bus access cycles including accurate bus setup cycles and data transfer cycles, but does not include bus stall cycles which can only be obtained by actual bus simulation. When an edge \( e_n \) contains a leading bus workload \( B_i[e_n] \), the trace simulator kernel triggers the bus arbitration simulation, and if the bus request is not granted due to occupied bus, bus stall cycle \( D_i[e_n] \) obtained from the bus arbitration simulation is added to the processor’s simulation clock.

3.2 Statistical Based Technique in a Trace Simulator

Figure 3 illustrates the bus statistics used in stall cycle prediction that are collected during normal trace simulation. At processor \( P_{E_i} \), computation-workload \( C_i[e_n] \) and bus workload \( B_i[e_n] \) on PTG-edge \( e_n \) are simply accumulated on processor’s simulation clock, where bus arbitration simulation at each bus workload is not performed. Statistics of \( C_i[e_n] \) and \( B_i[e_n] \) are collected at \( P_{E_i} \) within a predefined bus prediction interval \( T \) (cycles). All computation workloads within two consecutive bus workloads are merged as a single interval workload \( L_i \) (cycles) where histogram \( h_{li} \) for \( L_i \) and \( h_{bi} \) on all bus workloads \( B_i \) are collected. Figure 4 illustrates the bus-stall prediction flow. For every prediction-window of \( T \) cycles, Statistics \( (N_i, h_{li}, h_{bi}) \) are collected at each processor \( P_{E_i} \) and used to compute the expected bus stall cycles per request \( E[D_i] \). Total bus stall cycle count during the bus prediction interval is predicted as \( E[D_i] \cdot N_i \), which is added to the processor’s simulation clock, where \( N_i \) is the total number of bus workloads within the prediction window \( T \).

4. Single Blocking Model and Burst Blocking Model

4.1 Overview

The single blocking model (SBM) assumes that when a bus request on \( P_{E_i} \) is blocked by \( b_j \) it will for sure be granted after maximum \( B_j \) cycles. SBM does not model the case where, (1) there is burst traffic on a higher priority PE or (2) for \( P_{E_i} \) there are two or more higher priority PEs. Added to the SBM, the Burst Blocking Model (BBM) models (1) i.e. burst traffic on a higher priority PE. This paper presents Multi-Blocking Model, which in addition to SBM and BBM, also models (2) such that bus workloads on higher priority PEs can block a request on \( P_{E_i} \) indefinitely. The following section introduces some basic concepts from SBM, BBM and the MBM.

4.2 Key Concepts

4.2.1 Computation Workload

Computation workload \( L_i \) denotes the number of execution cycles between two successive bus accesses on a Processing Element \( P_{E_i} \), where \( i \) indicates the priority of a PE. Let \( N_i \) be the total number of computation workloads on \( P_{E_i} \).

4.2.2 Bus Workload

Bus workload \( B_i \) denotes the number of bus cycles between two successive computation workloads on \( P_{E_i} \). Total number of bus workloads on \( P_{E_i} \) are same as \( N_i \).

4.2.3 Request Probability and Average Interval Workload

Request Probability is the probability \( \lambda_i \) that a bus request \( r_i \) occurs at each cycle on \( P_{E_i} \). A burst transfer request is generated on \( P_{E_i} \), with probability \( \mu_i \) and probability that interval \( L_i \) equals \( n \) (cycles) is given as:

\[
Pr(L_i = n) = \begin{cases} 
\mu_i & (n = 0) \\
(1 - \mu_i) \lambda_i (1 - \lambda_i)^{n-1} & (n \geq 1) 
\end{cases}
\]

Here, expectation of interval \( E[L_i] \) is given as: \( E[L_i] = \sum_{n=0}^{\infty} Pr(L_i = n) \cdot n = \frac{\lambda_i}{\mu_i} \), therefore,

\[
\lambda_i = \frac{1 - \mu_i}{E[L_i]} \tag{1}
\]

and burst request probability \( \mu_i \) is obtained from the collected statistics during the bus prediction interval \((N_i, h_{li}, h_{bi})\):

\[
\mu_i = Pr(L_i = 0) = \frac{h_{li}(0)}{\sum_{m} h_{li}(m)} = \frac{h_{li}(0)}{N_i}
\]

4.2.4 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 \( P_{E_i} \) is called request inactivation probability \( y_{ij} \).

4.2.5 Merged Bus Workload

When a bus request on \( P_{E_i} \) is blocked due to a bus workload on \( P_{E_j} \) (such that priority of \( P_{E_i} \) is higher than priority of \( P_{E_j} \)), the occurrence of a zero interval workload on \( P_{E_i} \) can block \( P_{E_i} \) continuously for multiple bus workloads. Effectively, from the perspective of \( P_{E_i} \), the bus workloads are merged into one continuous workload. This bus workload is termed merged bus workload \( B_{ij} \) from here on.

4.2.6 Effective Request Inactivation Probability

Similar to merged bus workload, the probability that request \( r_i \) does not occur within the duration of the merged workload \( B_{ij} \) on \( P_{E_i} \) is called merged request inactivation probability \( y_{ij}^* \).
4.3 Mathematical Model SBM and BBM

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 \( PE_j \), \( blk_j \): a blocking event by bus event \( b_j \) on request \( r_i \), \( t_j(k) \): a bus event with length \( B_j = k \) at processor \( PE_j \), \( blk_j(k) \): a blocking event by bus event \( b_j(k) \) on request \( r_i \), \( t_j(k) \): time difference between the first cycle of \( b_j(k) \) and request \( r_i \), \( bb_j \): bus event \( b_j \) at processor \( PE_j \) follows immediately after (with no interval) a bus event \( b_i \) at processor \( PE_i \), \( E[D_{ij}] \): Expectation of bus stall per request at processor \( PE_i \).

4.3.1 SBM

For SBM The overall “bus stall” expectation, \( E[D_{ij}] \) on all bus events \( b_j \) is given as,

\[
E[D_{ij}] = \begin{cases} 
Q_j\left(E[B_j] - U_{ij}\left(1 - \frac{V_{ij}}{\lambda_i}\right)\right) & (\alpha_{ij} = 0) \\
Q_j\left(E[B_j] - \frac{1}{\lambda_i}\right) & (\alpha_{ij} = 1) 
\end{cases}
\]

Such that,

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

\( v_{ij} = (1 - \lambda_i)y_{ij} \) and \( y_{ij} = \sum_k f_{bj}(k)(1 - \lambda_i)^{k-1} \), \( U_{ij} = S_{ij} + (1 - \lambda_i)(1 - S_{ij}) \) and \( S_{ij} \) is the probability of event \( bb_{ij} \) (observed at \( PE_j \)) i.e., \( S_{ij} = \Pr(bb_{ij}) \). Moreover, \( \lambda_i \) is calculated using Eq. (1) with value of \( \mu \) assumed to be 0 as SBM does not model zero interval bus requests.

\( Q_{ij} = N_i/N_j \) is the bus event count ratio. Although \( N_i \) and \( N_j \) 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 result 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

\[
G_i = E[L_i] + E[B_i] + E[D_i]
\]

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

\[
Q_{ij} = \frac{E[L_i] + E[B_i] + E[D_i]}{E[L_j] + E[B_j] + E[D_j]}
\]

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

4.3.2 BBM

On the processor with higher priority, the occurrence of a zero interval workload in effect merges successive bus workloads. The expectation of the merged bus workloads is given as:

\[
E[B'] = \frac{E[B]}{1 - \mu}
\]

While the merged request inactivation probability \( y'_{ij} \) is given as,

\[
y'_{ij} = \sum_k f_{bj}(k)(1 - \lambda_i)^{k-1} = \frac{(1 - \mu_i)y_{ij}}{1 - \mu_j(1 - \lambda_i)y_{ij}}
\]

Moreover, when \( \alpha_{ij} = 0 \), since \( PE_i \) will only see \((1 - \mu_i)N_j \) effective bus workloads, the bus event count ratio \( Q_{ij} \) in this case is rewritten as:

\[
Q_{ij}' = Q_{ij}(1 - \mu_i) \quad (\alpha_{ij} = 0)
\]

The bus stall expectation is calculated as,

\[
E[D_{ij}] = \begin{cases} 
Q_j\left(E[B_j] - U_{ij}\left(1 - \frac{V_{ij}}{\lambda_i}\right)\right) & (\alpha_{ij} = 0) \\
Q_j\left(E[B_j] - \frac{1}{\lambda_i}\right) & (\alpha_{ij} = 1) 
\end{cases}
\]

where

\[
U_{ij}' = (1 - \mu_i)S_{ij} - (1 - \lambda_i)(1 - S_{ij})
\]

Note that when \( \mu_j = 0 \) for all PEs, the BBM is reduced to SBM.

4.3.3 Calculation Flow

Detail of the bus stall delay calculation flow is described below. Here, let \( PEs \) be the set of processor indices.

1. Coefficient calculations: from the new sets of bus workload statistics \((N_i, h_{ij}, h_{bi})\) obtained during the bus prediction period \( T \), interval workload expectation \( E[L_i] \), request probability \( \lambda_i \) and bus workload expectation \( E[B_i] \) are calculated.

2. Initial bus stall delay values: \( \forall i \in PEs, E[D_i] = 0 \)

3. Iterative refinement: calculations for \( E[D_i] \) are repeated until all bus stall delays \( E[D_i] \) converges. Value of \( E[D_i] \) is updated on each iteration.

At the end the calculated values of \( E[D_i] \) are added to the simulation clock of corresponding PE.

4.3.4 Multi-blocking Behavior (Limitation of SBM and BBM)

We define multi-blocking behavior as the event where a single bus request on \( PE_i \) is blocked consecutively by bus-workloads on multiple higher priority PEs. Assuming that a request \( r_i \) on \( PE_i \) is blocked by \( PE_j \) and that there is at least one more bus master \( PE_l \) such that \( \alpha_{ij} = 0 \) and \( \alpha_{il} = 0 \), then there is a possibility that immediately after \( PE_j \) releases the shared bus, a bus-workload on \( PE_l \) wins the bus and \( r_i \) is blocked for another bus-workload on \( PE_l \).

Note that this consecutive blocking can happen indefinitely. Figure 11 shows the multi-blocking behavior on a shared bus. This behavior has not been modeled in SBM or BBM. This limitation introduces inaccuracies as the number of PEs increases and or the
target application traffic becomes intensive. This research focuses on modeling the Multi Blocking Behavior.

5. Multi Blocking Model

The Multi Blocking Model (MBM) captures multi blocking behavior as defined in the previous section. A comparison of the modeled bus stall by SBM, BBM and MBM is shown in Fig.5. As shown, SBM only captures single blocking behavior, BBM also captures the potential extended stall by burst blocking behavior and MBM captures multi blocking behavior as well.

First we define some basic terms.

5.1 Key Terms

5.1.1 Effective Bus Workload

When a bus request on PEi is blocked due to a bus workload on PEj (such that priority of PEj is higher than priority of PEi), the occurrence of a zero interval workload on PEj or a bus workload on PEk (such that priority of PEk is also higher than priority of PEi) can block PEi continuously for multiple bus workloads. This kind of blocking can happen in different combinations on all high priority PEs. Effectively, from the perspective of PEi, the bus workloads are merged into one continuous workload. This merged bus workloads is termed effective bus workload from here on. Note that the effective bus workload on each PEj will be different for each PEi, as opposed to the merged workload defined for BBM which is the same for any lower priority PE. Hence it is denoted as Bj, i.e. the effective bus workload on PEj from the perspective of PEi. Calculation of effective bus workload is one of the key steps in the MBM model.

5.1.2 Effective Request Inactivation Probability

Similar to effective bus workload, the probability that request rj does not occur within the duration of the effective bus workload Bj on PEi is called effective request inactivation probability Yij.

5.2 Mathematical Model

Derivation of the model follows in this section. For simple reading, first the derivations are done with the assumption that there are two higher priority PEs, both of which can generate burst traffic as well as show multi blocking behavior. At the end, the general form of the mathematical equations for n number of higher priority PEs is reported.

5.2.1 Probability Mass Functions

As noted above, the effective bus workload on each PEj will be different for each observer PEi. Let’s first derive the equations for PEj such that there are two higher priority PEs PE0 and PE1. Below defines some terms that will be used in the derivation.

Cji: Probability of event bji, i.e. bj immediately follows bj (observed at PEj)

1 − (Cji + Cj'): probability that neither bj nor bj immediately follows bj.

fji(m, k): “scaled” probability mass function of m + 1 merged bus events (bj or bji) starting with bj and terminating with bj.

fj′(k): Probability mass function of all merged bus workloads (bj or bji) starting with bj and terminating with bj.

E[Bj(m)]: Expectation of m + 1 merged bus events (bj or bji) starting with bj and terminating with bj, such that PEi is lower priority than PEj and PEk.

E[Bj′]: Expectation of all merged bus workloads (bj or bji) starting with bj and terminating with bj, such that PEi is lower priority than PEj and PEk.

Yij: Request inactivation probability of request rj with probability λj on the effective bus workload Bj.

Yij′: Partial Request inactivation probability of request rj with probability λj on the merged bus workload starting with bj and terminating with bj.

5.2.1.1 Expression for fj′(m, k)

Now, assuming two higher priority PEs PE0 and PE1, first expressions for fj′(m, k) are derived

\[
\begin{pmatrix}
 f_{00}(0, k) & f_{00}(0, k) \\
 f_{01}(0, k) & f_{11}(0, k)
\end{pmatrix} = \begin{pmatrix}
 f_{00}(k) & 0 \\
 0 & f_{11}(k)
\end{pmatrix}
\]

\[
\begin{pmatrix}
 f_{00}(1, k) & f_{00}(1, k) \\
 f_{01}(1, k) & f_{11}(1, k)
\end{pmatrix}
= \sum_{n} \begin{pmatrix}
 f_{00}(n) & 0 \\
 0 & f_{11}(n)
\end{pmatrix} \begin{pmatrix}
 C_{20} & C_{21} \\
 C_{20} & C_{21}
\end{pmatrix} \begin{pmatrix}
 f_{00}(0, k-n) & f_{00}(0, k-n) \\
 f_{01}(0, k-n) & f_{11}(0, k-n)
\end{pmatrix}
\]

\[
\begin{pmatrix}
 f_{00}(m, k) & f_{00}(m, k) \\
 f_{01}(m, k) & f_{11}(m, k)
\end{pmatrix}
= \sum_{n} \begin{pmatrix}
 f_{00}(n) & 0 \\
 0 & f_{11}(n)
\end{pmatrix} \begin{pmatrix}
 C_{20} & C_{21} \\
 C_{20} & C_{21}
\end{pmatrix} \begin{pmatrix}
 f_{00}(m-1, k-n) & f_{00}(m-1, k-n) \\
 f_{01}(m-1, k-n) & f_{11}(m-1, k-n)
\end{pmatrix}
\]

where

\[\begin{pmatrix}
 C_{20} & C_{21} \\
 C_{20} & C_{21}
\end{pmatrix} = \begin{pmatrix}
 C_{00} & C_{10} \\
 C_{01} & C_{11}
\end{pmatrix}\]

5.2.1.2 Probability Mass Function of All Merged Bus Workloads

Next the probability mass function of all merged bus workloads is derived as,

\[
\begin{pmatrix}
 f_{00}(k) & f_{00}(k) \\
 f_{01}(k) & f_{11}(k)
\end{pmatrix} = [H] \sum_{m=0}^{\infty} \begin{pmatrix}
 f_{00}(m, k) & f_{00}(m, k) \\
 f_{01}(m, k) & f_{11}(m, k)
\end{pmatrix}
\]

where

\[\begin{pmatrix}
 1 & 0 \\
 0 & 1 - (C_{00} + C_{01})
\end{pmatrix}\]

5.2.1.3 Scaling Factor Fj′(m)

The scaling factor Fj′(m) is derived as,

\[
\begin{pmatrix}
 f_{00}(m) & f_{00}(m) \\
 f_{01}(m) & f_{11}(m)
\end{pmatrix} = \sum_{k} \begin{pmatrix}
 f_{00}(m, k) & f_{00}(m, k) \\
 f_{01}(m, k) & f_{11}(m, k)
\end{pmatrix}
\]

\[
= \sum_{k} \sum_{n} \begin{pmatrix}
 f_{00}(n) & 0 \\
 0 & f_{11}(n)
\end{pmatrix} \begin{pmatrix}
 C_{20} & C_{21} \\
 C_{20} & C_{21}
\end{pmatrix} \begin{pmatrix}
 f_{00}(m-1, k-n) & f_{00}(m-1, k-n) \\
 f_{01}(m-1, k-n) & f_{11}(m-1, k-n)
\end{pmatrix}
\]

\[
= \sum_{n} \begin{pmatrix}
 f_{00}(n) & 0 \\
 0 & f_{11}(n)
\end{pmatrix} \begin{pmatrix}
 C_{20} & C_{21} \\
 C_{20} & C_{21}
\end{pmatrix} \begin{pmatrix}
 f_{00}(m-1, k-n) & f_{00}(m-1, k-n) \\
 f_{01}(m-1, k-n) & f_{11}(m-1, k-n)
\end{pmatrix}
\]

\[= [C_{2}]^{n} \begin{pmatrix}
 f_{00}(0) & f_{10}(0) \\
 f_{01}(0) & f_{11}(0)
\end{pmatrix} = [(C_{2})^{n}
\]

where,
5.2.2 Effective Bus Workload Expectation
Recall that $B_{ij}$ is effective bus workload on $PE_j$ as observed by a bus request on $PE_i$. Let $E[B_{ij}]$: expectation of effective bus workload $B_{ij}$.

$$E[B_{ij}] = \sum_k f_{B_{ij}}(k) \cdot k$$

Figure 6 A, B, C and D show the multiple merging fashions on two higher priority PEs for effective bus workloads, $m = 0, 1, 2$ and 3 respectively.

5.2.2.1 Expectation of $m + 1$ Merged Bus Events
Expectation of $m + 1$ merged bus events is calculated as,

$$E[B_{B_00}(m)] = E[B_{B_10}(m)] = \sum_k f_{B_{B_00}}(m, k), f_{B_{B10}}(m, k)$$

$$E[B_{B11}(m)] = \sum_k f_{B_{B11}}(m, k)$$

$$= [C_2] \cdot E[B_{B00}(m-1)] + E[B_{B11}(m-1)]$$

$$+ E[B_{B00}(0)] 0 \quad E[B_{B11}(0)] \quad 0$$

5.2.2.2 Expectation of All Merged Bus Workloads
Expectation of all merged bus workloads is given as,

$$E[B_{B_{00}}] = E[B_{B_{10}}] = \sum^\infty_{m=0} E[B_{B_{00}}(m)], E[B_{B_{10}}(m)]$$

$$E[B_{B_{11}}] = \sum^\infty_{m=0} E[B_{B_{11}}(m)]$$

$$= [H_2] \cdot \sum^\infty_{m=0} [C_2] \\ E[B_{B00}(m-1)] E[B_{B11}(m-1)]$$

$$+ E[B_{B00}(0)] 0 \quad E[B_{B11}(0)] \quad 0$$

$$= [H_2] (1 - C_2)^{-1} \cdot E[B_{B00}] + E[B_{B11}] \cdot (1 - [C_2])^{-1}$$

Where 5.2.4 shows that,

$$\lim_{m \to \infty} ([C_2]^m = 0$$

The overall effective bus workload $E[B_{ij}]$ is simply calculated by summation of the partial bus workloads $E[B_{ij}]$ for all $l < i$,

$$E[B_{ij}] = E[B_{ij}'] + E[B_{ij}^*]$$

Moreover as shown in Section 5.2.4 the value of $([C_2]^m$ decays with increasing value of $m$, therefore the assumption above does not introduce significant inaccuracy. However, in some cases when the probability that neither $b_i$ nor $b_l$ immediately follows $b_j$ calculated as $1 - (C_{ij} + C_{lj})$ is significantly small then the assumption of infinite bus event count becomes a significant source of inaccuracy. This kind of traffic pattern results in starvation in low priority PEs and will be covered in detail in Section 6.2.

5.2.3 Calculation of $[1 \quad 1][H_2] (I - [C_2])^{-1}$
Let $a = 1 - C_{00}, b = -C_{10}, c = -C_{01}, d = 1 - C_{11}$ then

$$(I - [C_2])^{-1} = \left[ \frac{d}{ad - bc} - \frac{b}{ad - bc} \right]$$

The product $[1 \quad 1][H_2] (I - [C_2])^{-1}$

$$= \left[ \frac{d}{ad - bc} \right] - \frac{b}{ad - bc}$$

$$= \frac{1}{ad - bc} (ad - bc)$$

5.2.3 Effective Request Inactivation Probability
Partial Request inactivation probability is calculated as,

$$\begin{bmatrix} Y_{200} \\ Y_{210} \end{bmatrix} = \sum_{k=0}^{\infty} f_{00}(m, k), f_{10}(m, k), f_{01}(m, k), f_{11}(m, k) \cdot (1 - \lambda)^k$$

$$= [H_2] \sum_{m=0}^{\infty} f_{00}(m, k), f_{10}(m, k), f_{01}(m, k), f_{11}(m, k) \cdot (1 - \lambda)^k$$

$$= [H_2] \sum_{m=0}^{\infty} f_{200}(m), f_{210}(m), f_{201}(m), f_{211}(m) \cdot (1 - \lambda)^k$$

$$= \begin{bmatrix} Y_{200}(m) \\ Y_{210}(m) \end{bmatrix} \begin{bmatrix} Y_{200}(0) \\ Y_{210}(0) \end{bmatrix} = ([V_2][C_2]^m \begin{bmatrix} y_{20} \\ y_{21} \end{bmatrix}$$
Such that, \([V]_2 = \begin{bmatrix} v_{20} & 0 \\ 0 & v_{21} \end{bmatrix}\), hence
\[
\begin{bmatrix}
Y_{200} & Y_{210} \\
Y_{201} & Y_{211}
\end{bmatrix} = [H_2] \sum_{m=0}^{\infty} ([V]_2 [C]_2)^m \begin{bmatrix} y_{20} \\ 0 \\ 0 \\ y_{21} \end{bmatrix}
= [H_2] (I - [V]_2 [C]_2)^{-1} \begin{bmatrix} y_{20} \\ 0 \\ 0 \\ y_{21} \end{bmatrix}
\]
where Section 5.2.4 shows that,
\[
\lim_{m \to \infty} ([V]_2 [C]_2)^m = 0
\]
The overall Request inactivation probability is simply calculated by summation of the partial request inactivation probabilities, \(Y_{ij} = Y_{ij1} + Y_{ij2}\) therefore,
\[
[Y_{20} \ 0 \ 0 \ y_{21}] = [1 \ 1][H_2] (I - [V]_2 [C]_2)^{-1} \begin{bmatrix} y_{20} \\ 0 \\ 0 \\ y_{21} \end{bmatrix}
\]
5.2.4 \(\lim_{m \to \infty} ([C]_2)^m = 0, \lim_{m \to \infty} ([V]_2 [C]_2)^m = 0\)
Recall that,
\(C_{ij}\): Probability of event \(bb_{ij}\), i.e. \(b_i\) immediately follows \(b_j\) (observed at \(PE_i\)),
\(1 - (C_{ij} + C_{ji})\): probability that neither \(b_i\) or \(b_j\) immediately follows \(b_j\).
And \([C]_2 = \begin{bmatrix} C_{00} & C_{01} \\
C_{01} & C_{11} \end{bmatrix}\)
And \([V]_2 = \begin{bmatrix} v_{20} \\ 0 \\ v_{21} \end{bmatrix}\)
First of all, we know that practically speaking all probabilities \(C_{ij} < 1\), moreover, \((C_{ij} + C_{ji}) < 1\), assuming that the probability that neither \(b_i\) or \(b_j\) immediately follows \(b_j\) is greater than zero.
\(\text{Let } \begin{bmatrix} a & b \\
c & d \end{bmatrix} \text{ represent } [C]_2\)
Then, \(\begin{bmatrix} a & b \\
c & d \end{bmatrix}^2 = \begin{bmatrix} aa + bc & ab + bd \\
c(a + c) & bc + d(d) \end{bmatrix}\)
Since \((aa + bc) < (a + c) < 1\), \((ac + cd) < (a + c) < 1\) and \((ab + bd) < (b + d) < 1\), \((bc + d) < (b + d) < 1\)
So we establish that all values of the resulting matrix are < 1.
Moreover, as value of \(m\) increases for \(\begin{bmatrix} a & b \\
c & d \end{bmatrix}^m\), the probability shrinks by factors \(1 - (a + c)\) and \(1 - (b + d)\). Intuitively we can deduce that \(\sum_{m=0}^{\infty} ([C]_2)^m\) converges to 0. Hence we conclude,
\[
\sum_{m=0}^{\infty} ([C]_2)^m = (I - [C]_2)^{-1}
\]
Similarly, \([V]_2 [C]_2 = \begin{bmatrix} v_{20} C_{00} & v_{20} C_{10} \\
v_{21} C_{01} & v_{21} C_{11} \end{bmatrix}\)
As per above representation, \([V]_2 [C]_2 = \begin{bmatrix} v_{20} a & v_{20} b \\
v_{21} c & v_{21} d \end{bmatrix}\), also since
\(v_{20}, v_{21} < 1\) therefore, \(v_{20} a < a, v_{20} b < b, v_{21} c < c, v_{21} d < d\), and since \(\sum_{m=0}^{\infty} ([V]_2 [C]_2)^m\) converges to 0, therefore we can deduce that \(\sum_{m=0}^{\infty} ([V]_2 [C]_2)^m\) converges to 0, hence
\[
\sum_{m=0}^{\infty} ([V]_2 [C]_2)^m = (I - [V]_2 [C]_2)^{-1}
\]
Note that for all \(0 < C_{ij} < 1, 0 < C_{ji} < 1\) and \(1 - (C_{ij} + C_{ji}) > 0\);
Perron-Frobenius Theorem[35] can also be used to deduce that
\(\sum_{m=0}^{\infty} ([C]_2)^m\) converges to 0 i.e.
\[
\sum_{m=0}^{\infty} ([C]_2)^m = (I - [C]_2)^{-1}
\]
5.2.5 Consecutive Bus Event
The consecutive bus event probability i.e. \(Pr(bb_{ij})\) is calculated on two observers, \(PE\) and \(PE_i\), \(C_{ij}\); probability of event \(bb_{ij}\) observed at \(PE_j\) and \(S_{ij}\); probability of event \(bb_{ij}\) observed at \(PE_j\).
\(C_{ij}\) is calculated in different ways for \(i = j\) and \(i \neq j\).
For \(i = j\), event \(bb_{ij}\) happens when (1) burst transfer request arrives on \(PE_i\) with probability \(\mu_i\), and a higher priority request does not arrive, i.e. event \(bb_{ij}\) does not occur for \((l < i)\), therefore,
\[
C_{ji} = \left(1 - \sum_{l=0}^{i-1} C_{il}\right) \mu_i
\]
For \((i \neq j)\):
Event \(bb_{ij}\) happens when (1) request \(r_j\) is blocked by \(b_i\) i.e. event \(blk_{ji}\) happens or \(r_j\) immediately follows \(b_i\) and (2) Event \(bb_{ij}\) does not happen for \((l < j)\). (1) can be easily modeled as a \(blk_{ji}\) where bus event \(b(k)\) has 1 extended cycle.
Therefore, for \(\alpha_{ji} = 1\) can be calculated as,
\[
\sum_{i=0}^{k} \lambda_i(1 - \lambda_i)^k = 1 - (1 - \lambda_i)^k
\]
And for \(\alpha_{ji} = 0\) (1) becomes,
\[
(1 - U_{ji}^*(1 - \lambda_j)^k)
\]
Therefore, overall \(C_{ij}\) can be calculated by summing over all \(k\),
\[
C_{ij} = \left(1 - U_{ji}^* \right) \sum_{l=0}^{i-1} C_{il} \alpha_{ji} = 1
\]
where \(U_{ji}^* = (1 - \mu_i) S_{ji} = (1 - \lambda_j)(1 - S_{ji})\)
\(S_{ij}\) is observed at \(PE_i\) as opposed to \(C_{ij}\) which is observed at \(PE_i\). To reflect this difference the bus event count ratio \(Q_{ji} = \frac{S_{ij}}{N_i}\) is introduced.
\[
Q_{ji} = \left(1 - U_{ji}^* \right) \sum_{l=0}^{i-1} C_{il} \alpha_{ji} = 0
\]
5.2.6 n-PE Equations
Finally, we can write the equations for effective bus workload, and effective request inactivation probability for \(n\) number of higher priority PEs. For effective bus workload,
\[
[E(B_{0}) \ E(B_{1}) \cdots E(B_{n})] = [1 \ 1 \cdots 1][B]_n(I - [C]_2)^{-1}
\]
Such that,
\[
[B]_n = \begin{bmatrix} E(B_0) & 0 & \cdots & 0 \\
0 & E(B_1) & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & E(B_n) \end{bmatrix}
\]
\[ C_h = \begin{bmatrix}
C_{00} & C_{10} & \cdots & C_{n0} \\
C_{01} & C_{11} & \cdots & C_{n1} \\
\vdots & \vdots & \ddots & \vdots \\
C_{0n} & C_{1n} & \cdots & C_{nn}
\end{bmatrix} \]

And effective request inactivation probability,
\[
[Y_i]_j = \frac{1}{H}[I - [V_i]_j[C_h]^{-1}][Y_i]
\]
Such that,
\[
[V]_h = \begin{bmatrix}
v_0 & 0 & \cdots & 0 \\
0 & v_{in} & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & v_{in}
\end{bmatrix}, \quad [Y]_i = \begin{bmatrix}
y_0 & 0 & \cdots & 0 \\
0 & y_{in} & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & y_{in}
\end{bmatrix}
\]
\[
[H]_h = \begin{bmatrix}
1 - \sum_{i=0}^{n} C_{0i} & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0
\end{bmatrix}
\]

### 6. Experiments and Results

Performance estimation of MPSoC shared bus was performed using the MBM (multi blocking model). We used recorded traffic patterns for benchmark applications as presented by Liu et al. [36]. The experiments were performed for two traffic intensive applications namely (1) “SPEC95 Fpppp” which is a chemical program performing multi-electron integral derivatives. It consists of 334 tasks and 1145 communication links (2) “Fast Fourier Transform” with 1024 inputs of complex numbers. It consists of 16384 tasks and 25600 communication links. For comparison, prediction results of two low traffic applications, i.e. SPARSE Matrix Solver which is “Random sparse matrix solver for electronic circuit simulations” and ROBOT which is Newton-Euler dynamic control calculation for 6-degrees-of-freedom Stanford manipulator, are also shown. The benchmark applications are run on two different architectures (1) Consisting of four processing elements connected through a shared bus (2) Consisting of eight processing elements connected through a shared bus. Recorded traffic patterns of both applications for both architectures are fed into the trace workload simulator. Histograms are populated over the course of a predefined window of application clock. On the conclusion of every time window, the statistics are input to the mathematical model and expected bus stall is predicted. This predicted stall is then added to the clock of each PE as detailed in Section 3.2. The results are then compared with the simulation method.

#### 6.1 Estimation Error

Figure 7, Fig. 8, Fig. 9, Fig. 10 and Fig. 11 show the estimated average bus access interval comparison.
comparison of “average bus access interval” calculated using the bus simulation method, calculated using prediction by BBM and calculated using prediction by MBM methods. While Table 1 shows a closer comparison of estimation error between the BBM model and the MBM model.

### 6.1.1 Four PE Architecture

On a 4 PE architecture, BBM shows minimum 0.1% and maximum 1.75% estimation errors for FPPPP and minimum 2.2% and maximum 13.91% estimation errors for FFT. MBM, on the other hand, shows minimum 0.02% and maximum 0.6% estimation errors for FPPPP and minimum 0.6% and maximum 8.8% estimation errors for FFT.

### 6.1.2 Eight PE Architecture

On the 8 PE architecture, BBM shows minimum 0.005% and maximum 0.08% estimation errors for ROBOT benchmark. For the SPARSE benchmark, minimum 0.075% and maximum 1.5% estimation errors are shown. However, in case of FPPPP a minimum of 0.28% and maximum 11.8% error. MBM, on the other hand, shows minimum 0.005% and maximum 0.8%, minimum 0.003% and maximum 0.945%, minimum 0.098% and maximum 2.7% estimation errors for ROBOT, SPARSE and FPPPP benchmarks respectively.

From our observation we conclude that the BBM is quite suitable and accurate for applications that are not traffic intensive, however, for traffic intensive applications BBM shows an increasing estimation error. Table 1 shows a comparison of estimation error on each individual PE for the FPPPP benchmark. It is evident that the error increases for lower priority PEs when using BBM while MBM retains its accuracy.

### 6.2 The Curious Case of FFT8

Performance estimation values for the FFT benchmark for 8-PE architecture are very curious and show a huge estimation error. Both the BBM model as well as the MBM model over-estimate the bus stall for lower priority PEs. In the case of BBM the estimated cycle count is 50–60% greater than the simulated cycle count. While for MBM the over estimation is above 1000% as shown in Fig. 12. Intuitively speaking the BBM should always under-estimate the cycle count for lower priority PEs since it does not account for multi-blocking behavior. This observation led us to close inspection of the traffic pattern for this benchmark. Observations are performed using bus simulation method to check for starvation. Table 2 reports a record of workload completion on each PE. During the simulation, whenever any PE completes its last bus workload the number of completed bus workloads on each PE is logged at that point in the simulation. This point is termed “Check Point” (CP). “F” indicates that a PE has already finished all of its bus-workloads. Note that the three lowest pri-
priority PEs did not complete a single bus transfer until the three highest priority PEs had finished all their bus transfers. It’s evident from the observations that only a very negligible number of bus requests on the lowest priority PEs incurred any bus stall due to bus workloads on the highest priority PEs. Moreover, a significant portion of the bus workloads on all PEs used the bus when the effective number of bus masters competing for bus access was four, three, two or even only one. On the other hand, as shown in Fig. 3 and Fig. 4 the prediction model assumes a full crossbar bus during a prediction window, such that all PEs are assumed to perform bus transfers without incurring any stall. For every prediction-window of T cycles, the expected bus stall cycles per request, \( E[D_j] \) is calculated and total bus stall during the bus prediction interval \( E[D_j] \cdot N_j \) is added to the processor’s simulation clock, where \( N_j \) is the total number of bus workloads within the prediction window \( T \). In the case of starvation a lower priority PE does not get the bus until all the bus workloads on higher priority PE have been completed. Therefore, at least \( (N_j - 1) \) bus workloads on lowest priority PEs are wrongly predicted to incur stall due to highest priority PEs. Let’s look at a hypothetical example to understand this case. Figure 13 shows two PEs, \( PE_H \) and \( PE_L \) such that \( PE_H \) is higher priority. During the prediction window, assuming full cross bar, both PEs are assumed to have completed 10 bus transfers each and workload statistics are accumulated. The prediction model uses these statistics to calculate expected stall per bus request. However, when using a shared bus, the first bus request on \( PE_L \) gets blocked until \( PE_H \) finishes all its bus workloads as shown in Fig. 14. After which \( PE_L \) completes all its bus workloads without incurring any stall at all. The prediction model assumes that 10 requests are issued on each PE in the T cycles however, simulation shows that only 1 bus request is issued on \( PE_L \). The prediction model calculates the incurred bus stall assuming all 10 bus requests must be serviced before the next window starts. This adds a huge stall to all 10 bus requests however, in reality the remaining 9 requests will not be issued until the first bus request is serviced. In contrast, the workload completion record of FPPPP8 benchmark, as shown in Table 3, indicates absence of starvation, hence the prediction model is able to estimate with higher accuracy. A close look at the CP0 row and the PE7 column clearly shows the difference between the two traffic patterns.

### 6.2.1 Bus Starvation

In concurrent computing starvation is measured by the bound value of bypass such that if \( n \) processes are competing for access to a shared resource, then a process is deemed starved unless it gains access after being bypassed at most \( f(n) \) times by other processes for some function \( f \) [37], however the value of \( n \) is subjective and can be different for different applications. The term “bus starvation” here is used to define a situation where one or more PEs are starved of bus access such that on the starved PE, the number of serviced bus requests is less than 0.5% of the number of serviced requests within a time window. However this observation is based on our experiments on the FFT8 application and is purely subjective. Secondly, while calculating expectation of all merged bus workloads, the proposed model assumes infinite bus workloads. This assumption becomes a significant source of inaccuracy for traffic patterns that result in starvation. This is because of a very small value of the rate with which \( (C_j)_m \) shrinks. For \( PE \), this probability is \( 1 - \sum_{i=0}^{n} C_{ij} \). The values of \( \sum_{i=0}^{n} C_{ij} \) probabilities have been reported in Table 4. For the FFT8 example, we noted that when \( \sum_{i=0}^{n} C_{ij} \) becomes greater than 0.9 the traffic pattern starts to cause starvation i.e. as seen by lowest priority PE, \( b_i \) is immediately followed by a bus event on a higher priority PE at least 90% of the times.

### 6.2.2 Identifying Bus Starvation

Usually it is not straightforward for application developers to determine if an application exhibits bus starvation. Simulation is performed for specific bus architectures and specific task mappings to evaluate the bus performance and to identify potential

#### Table 2 Workload completion record on each PE for FFT8.

<table>
<thead>
<tr>
<th>Check points</th>
<th>PE0</th>
<th>PE1</th>
<th>PE2</th>
<th>PE3</th>
<th>PE4</th>
<th>PE5</th>
<th>PE6</th>
<th>PE7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CP0</td>
<td>F</td>
<td>30237</td>
<td>13087</td>
<td>1264</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CP1</td>
<td>F</td>
<td>F</td>
<td>21917</td>
<td>5343</td>
<td>530</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CP2</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>20125</td>
<td>8122</td>
<td>827</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CP3</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>25352</td>
<td>8902</td>
</tr>
<tr>
<td>CP4</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>20744</td>
</tr>
<tr>
<td>CP5</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>CP6</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>CP7</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Total 8W</td>
<td>41220</td>
<td>41080</td>
<td>40960</td>
<td>40860</td>
<td>40960</td>
<td>40960</td>
<td>40960</td>
<td>40960</td>
</tr>
</tbody>
</table>

#### Table 3 Workload completion record on each PE for FPPPP8.

<table>
<thead>
<tr>
<th>Check points</th>
<th>PE0</th>
<th>PE1</th>
<th>PE2</th>
<th>PE3</th>
<th>PE4</th>
<th>PE5</th>
<th>PE6</th>
<th>PE7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CP0</td>
<td>635</td>
<td>716</td>
<td>F</td>
<td>759</td>
<td>609</td>
<td>500</td>
<td>393</td>
<td>425</td>
</tr>
<tr>
<td>CP1</td>
<td>668</td>
<td>831</td>
<td>F</td>
<td>F</td>
<td>748</td>
<td>566</td>
<td>444</td>
<td>477</td>
</tr>
<tr>
<td>CP2</td>
<td>686</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>763</td>
<td>572</td>
<td>489</td>
<td>503</td>
</tr>
<tr>
<td>CP3</td>
<td>813</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>732</td>
<td>583</td>
<td>568</td>
</tr>
<tr>
<td>CP4</td>
<td>913</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>636</td>
<td>679</td>
</tr>
<tr>
<td>CP5</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>681</td>
<td>976</td>
</tr>
<tr>
<td>CP6</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>727</td>
</tr>
<tr>
<td>CP7</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Total 8W</td>
<td>940</td>
<td>880</td>
<td>720</td>
<td>940</td>
<td>920</td>
<td>780</td>
<td>720</td>
<td>780</td>
</tr>
</tbody>
</table>

#### Fig. 13 Completed bus workloads on \( PE_H \) and \( PE_L \) assuming full crossbar.

#### Fig. 14 Completed bus workloads on \( PE_H \) and \( PE_L \) using shared bus.
Table 4 Comparison of $\sum_{j=0}^{n} C_{j}$ on various traffic patterns.

<table>
<thead>
<tr>
<th>Application</th>
<th>C0</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
<th>C6</th>
<th>C7</th>
<th>C8</th>
<th>C9</th>
<th>starvation</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFT8</td>
<td>0.9999</td>
<td>0.9994</td>
<td>0.999</td>
<td>0.9978</td>
<td>0.9972</td>
<td>0.9944</td>
<td>0.9918</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFT8 2XBBW</td>
<td>0.9369</td>
<td>0.9119</td>
<td>0.8777</td>
<td>0.8776</td>
<td>0.9710</td>
<td>0.9506</td>
<td>0.933</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFT8 2XBBW 1/3XComp</td>
<td>0.9041</td>
<td>0.8598</td>
<td>0.8413</td>
<td>0.7946</td>
<td>0.7881</td>
<td>0.7206</td>
<td>0.6947</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FPPP8</td>
<td>0.8816</td>
<td>0.8965</td>
<td>0.884</td>
<td>0.8525</td>
<td>0.848</td>
<td>0.783</td>
<td>0.7144</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 15 Estimated Average Bus Access Interval comparison.

Table 5 Workload completion record on each PE for FFT8_manipulated.

<table>
<thead>
<tr>
<th>Check Points</th>
<th>PE0</th>
<th>PE1</th>
<th>PE2</th>
<th>PE3</th>
<th>PE4</th>
<th>PE5</th>
<th>PE6</th>
<th>PE7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CP0</td>
<td>F</td>
<td>35467</td>
<td>32882</td>
<td>26828</td>
<td>21536</td>
<td>11074</td>
<td>10765</td>
<td>623</td>
</tr>
<tr>
<td>CP1</td>
<td>F</td>
<td>F</td>
<td>38428</td>
<td>31709</td>
<td>26061</td>
<td>14385</td>
<td>6036</td>
<td>1393</td>
</tr>
<tr>
<td>CP2</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>34147</td>
<td>28288</td>
<td>16468</td>
<td>7808</td>
<td>2412</td>
</tr>
<tr>
<td>CP3</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>35181</td>
<td>22515</td>
<td>33297</td>
<td>6322</td>
</tr>
<tr>
<td>CP4</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>27669</td>
<td>18295</td>
<td>10460</td>
</tr>
<tr>
<td>CP5</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>CP6</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>CP7</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Total BW</td>
<td>41220</td>
<td>41080</td>
<td>40960</td>
<td>40880</td>
<td>40960</td>
<td>40900</td>
<td>40860</td>
<td>40820</td>
</tr>
</tbody>
</table>

bus starvation. However, in our experiments we relied on the results of proposed prediction model to identify starvation. The proposed prediction model, although hugely over estimating the actual stall, raised a red flag resulting in close inspection and identifying starvation. We feel that the proposed model could also serve as a tool to identify such cases of starvation so designers can examine the application and the architecture closely and tweak the application or the architecture accordingly to avoid bus starvation on any PE.

6.2.2.1 FFT8_manipulated

To demonstrate this feature, the FFT8 traffic was manipulated to find a starvation-less configuration. Assuming a 2x increase in the bus bandwidth and computation speed reduced by 1/3x, the resulting system showed a much improved bus performance, and faster application execution despite a reduced computation speed. The prediction results were accurate with about 5% prediction error. Figure 15 shows a comparison of estimated and simulated average bus access interval while Table 5 shows a record of workload completion on each PE. This experiment was performed to demonstrate the usability of the prediction model in identifying starvation cases however further detailed work on more such benchmarks has not been performed and remains one of the potential research areas for future work.

6.3 Simulation Speedup

Next we compare the simulation times using the proposed estimation technique as opposed to the simulation technique. The simulation time for the “simulation based prediction method” consists of two components, $T_{sim\_bus}$ and $T_{sim\_que}$, where $T_{sim\_bus}$ is the time it takes to simulate the bus access itself, while $T_{sim\_que}$ is the time it takes to maintain the arbitration queue on the arrival or granting of every new bus request. $T_{sim\_bus}$ can be calculated as,

$$T_{sim\_bus} = \sum_{all \ PEs} \left( \sum_{N} t_{bus} \right)$$

where, $t_{bus}$ is the time it takes to simulate one bus access and “N”, is the total number of bus workloads on a PE.

$T_{sim\_que}$ has to be calculated on each individual arrival or grant of a bus request as the time required to maintain the queue would be different depending on the number of PEs in the queue. Over-all

$$T_{sim} = \sum_{I} \left( T_{sim\_bus} + T_{sim\_que} \right)$$

Here “I” is the number of times a benchmark/application is executed. This will be discussed in detail at the end of this section. On the other hand, the simulation time for the proposed “prediction” technique can be calculated as,

$$T_{pred} = \sum_{W} t_{m}$$

And Speed-up is simply calculated as,

$$\text{Speedup} = \frac{T_{sim}}{T_{pred}}$$

The expression for $T_{sim}$ clearly shows that $T_{sim}$ will increase as the length of simulation increases. On the other hand, the expression for $T_{pred}$ shows that it increases with the increase in the number of prediction time-windows $W$. Since the length of the prediction window can be adjusted depending on the size of simulation such that the total number of windows “W” does not increase drastically. This results in only a slight increase in $T_{pred}$ as the simulation data length increases. As a result the speedup ratio increases as “I” increases.

Figure 16 and Fig. 18 report $T_{sim}$ for each individual PE with an increasing value of “I” for the FPPP8 and FFT8 benchmarks respectively. Experiments were performed for different values of I i.e. “I = 1, 10, 100, 500 and 1000”. Moreover, Fig. 17 and Fig. 19 report a comparison between total $T_{sim}$, total $T_{pred}$ and the resulting speedup ratio for both benchmarks. As evident from the shown graphs, for shorter simulations, $T_{sim}$ is low however, the longer the simulation continues the value of $T_{sim}$ increases drastically while $T_{pred}$ increases very slightly.

For the experiments an increasing value of “I” was chosen in order to increase simulation data length to show that proposed
method is robust enough for any length of input data. The data length generated by 1000 iteration is long enough to demonstrate that the proposed method will not be affected adversely as simulation length increases as reflected in Fig. 17 and Fig. 19. This highlights the benefit of using proposed model for thorough and iterative performance estimation that involves running an application multiple times and involves multiple cycles of performance estimation, application tuning and design space exploration.

6.4 Calculation Optimization and Model Scalability

In order to reap the speed-up benefits of proposed technique, we try to optimize calculation of different values used by the mathematical model. First, the values that are independent of the iteratively calculated value $Q_{ij}$, are calculated outside the iterative loop and only the values dependent on $Q_{ij}$ are updated every iteration. Secondly, the matrices $[B]_n$, $[V]_n$, $[Y]_n$, $[H]_n$ are all diagonal matrices hence special optimized multiplication functions are implemented to reduce any useless calculations. Another point of discussion for proposed model is scalability. Given the matrix multiplication and inversion calculations involved in calculating “effective bus workload expectation” $E[B_{ij}]$ and “effective request inactivation probability” $Y_{ij}$, we expect that the calculation will become more complicated as number of PEs and as a result the matrix size increases. Especially the matrix inverse operation could be a speed bottleneck. In future work, an approximate model that can limit the size of matrices while maintaining accuracy will be developed in order to make the current estimation model scalable to any number of PEs.

7. Future Work

Further development of our proposed technique has a couple of directions.

As discussed before, in future work, an approximate model that can limit the size of matrices will be developed in order to make the current estimation model scalable to any number of PEs. Secondly, multiple arbitration schemes such as, TDM/Round Robin, Lottery based and Least Recently Granted will be modeled. Furthermore, we aim to augment the proposed model with a cache model that can predict the effects of cache performance on the overall performance of a bus-architecture for any specific application program.

8. Conclusion

This paper presented an analytical model to predict arbitration stalls for a shared bus in a multi processor system on chip architecture. A previous Burst Blocking Model (BBM) was extended to account for multiple interfering bus masters. We call this model, Multi Blocking Model (MBM). The developed model was tested mainly on an 8-PE architecture with a shared bus and corresponding results and comparisons were presented accordingly. We conclude that the previously submitted, Burst Blocking Model is accurate enough for low traffic applications even on 8PE systems however for traffic intensive applications the estimation results become inaccurate and hence use of the proposed model i.e. Multi Blocking Model becomes necessary.

Fig. 16 “Tsim” for each PE for FPPPP8, with increasing “I”.

Fig. 17 Comparison between “Tsim” and “Tprd”, and Speedup.

Fig. 18 “Tsim” for each PE for FFT8, with increasing “I”.

Fig. 19 Comparison between “Tsim” and “Tprd”, and Speedup.
References


Farhan Shafiq received his B.E. degree from National University of Sciences and Technology, Pakistan, and M.E. degree from KTH, Sweden, in 2006 and 2009 respectively. Since 2010 he has been studying for his Ph.D. degree in the Department of Communications and Computer Engineering, Tokyo Institute of Technology. His interests include MPSoC architecture, MPSoC interconnect modeling and simulation, MPSoC programming, Embedded systems, HW/SW co-design and VLSI architecture.

Tsuyoshi Isshiki has received his B.E. and M.E. degrees in electrical and electronics from Tokyo Institute of Technology in 1990 and 1992, respectively. He received his Ph.D. degree in computer engineering from University of California at Santa Cruz in 1996. He is currently a Professor at Department of Communications and Computer Engineering in Tokyo Institute of Technology. His research interests include MPSoC programming framework, high-level design methodology for configurable system, bit-serial synthesis, FPGA architecture, image processing, fingerprint authentication algorithms, computer graphics, and speech synthesis. Dr. Isshiki is a member of IEEE CAS, IPSJ and IEICE.
Dongju Li received her Ph.D. degree in Electrical and Electronics from Tokyo Institute of Technology in 1998 and currently an Associate Professor at Department of Communications and Computer Engineering, Graduate School of Science and Engineering, Tokyo Institute of Technology. Her current research interests include embedded algorithm for fingerprint authentication, fingerprint authentication solution for smart phone. Dr. Li is a member of IEEE CAS and IEICE since 1998.

(Recommended by Associate Editor: Kotaro Shimamura)