T2R2 東京科学大学 リサーチリポジトリ
Science Tokyo Research Repository

論文 / 著書情報
Article / Book Information

題目(和文)	
Title(English)	A Fast and Accurate Statistical Bus Model for Efficient Performance Estimation of Shared Bus Based MPSoC Architectures
著者(和文)	SHAFIQFARHAN
Author(English)	Farhan Shafiq
出典(和文)	学位:博士(学術), 学位授与機関:東京工業大学, 報告番号:甲第10566号, 授与年月日:2017年3月26日, 学位の種別:課程博士, 審査員:一色 剛,上野 修一,髙橋 篤司,原 祐子,中原 啓貴,伊藤 和人
Citation(English)	Degree:Doctor (Academic), Conferring organization: Tokyo Institute of Technology, Report number:甲第10566号, Conferred date:2017/3/26, Degree Type:Course doctor, Examiner:,,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Powered by T2R2 (Science Tokyo Research Repository)

A Fast and Accurate Statistical Bus Model for Efficient Performance Estimation of Shared Bus Based MPSoC Architectures

FARHAN SHAFIQ

A thesis submitted in conformity with the requirements for the degree of

Doctor of Philosophy

Specialty: Computer Engineering

Advisor: Tsuyoshi Isshiki

Department of Communication and Computer Engineering
Tokyo Institute of Technology

© Copyright by Farhan Shafiq 2016

A Fast and Accurate Statistical Bus Model for Efficient Performance Estimation of Shared Bus Based MPSoC Architectures[1]

Farhan Shafiq

Doctor of philosophy

Department of Communication and Computer Engineering

Tokyo Institute of Technology, Japan

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 and require a lot of development work. 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. In this research we propose a statistical based model to predict the performance cost of arbitration related stalls, on specific bus architectures and specific applications. The proposed model uses workload trace of the actual applications

[1]

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 research presents bus models capturing single-blocking, burst bus-transfers, and multi-blocking bus behaviors. Single Blocking Model (SBM) captures single interfering bus masters for each bus request at a time, Burst Blocking Model (BBM) captures multiple back-to-back bus transfers on a single interfering bus masters for each bus request at a time while Multi Blocking Model (MBM) captures multiple bus transfers on multiple interfering masters for each bus request at a time. Performance estimation experiments are performed for multiple benchmarks using two different architectures i.e. four processing elements connected via a shared bus and eight processing elements connected via a shared bus. The estimation results are compared against a simulation based estimation technique for accuracy and speed. Moreover, the speedup benefit as opposed to simulation techniques is reported

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Prof. Tsuyoshi Isshiki for giving me the opportunity to work on a very interesting research topic and for the continued support, guidance and direction during the course of my PhD study at Tokyo Institute of Technology. I would also like to extend my gratitude to Prof. Hiroaki Kunieda, Dr. Dongju Li and all other members of Kunieda Isshiki laboratory who contributed their time and efforts to help me perform my research activities during my stay at the laboratory. Specially, I would like to extend my utmost gratitude to Ms. Miwa Tashiro and Ms. Yumiko Kondo for helping with every procedure during my stay at the laboratory. I wish to thank all the current and ex- members of Kunieda-Isshiki laboratory specially Dr. Arif Ullah Khan, Mr. Amr Fathy Eid Yehia, Dr. Surachai Thongkaew, Dr. Hao Xiao, Dr. Hsuan-Chun Liao, Mr Anurag Singh and Mr Farhan Nawaz for their ideas and guidance during my research.

Last but not the least; I would like to thank my family who extended their emotional support and encouragement when I needed it the most and helped me through the highs and lows of my research and my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1 Motivation and background

Over the past few decades, semiconductors have gone through a fierce evolution following the Moore's law. Recently chips are being fabricated at 1x, 1y and 1z nm process nodes on the high-end and 2x, 2y and 2z nm process nodes for middle-end products with billions of transistors on a chip. The industry is exploring the trend of further continuation of Moore's law termed "more Moore" while also exploring the "more than Moore" trend of integrating multiple die in one packaged chip as System in Package. This huge advancement in performance and reduction in size has afforded designers the possibility to put more and more resources on a single chip. A single System on Chip (SoC) can integrate multiple processor cores, Graphic Processing Units (GPU), Application Specific Instruction-set Processors (ASIP), Digital Signal Processors (DSP), and hardware accelerators. For example, recently, leading smart phones SoCs or "Application Processor SoC" integrate multiple functionalities on a single chip including multi-core CPU (Central Processing Unit), GPU (Graphic Processing Unit), DSP (Digital Signal Processor), display, multimedia support, camera support, connectivity, security, location and sensor core etc. Moreover, multicore CPU can be found in most high and middle-end SoCs e.g. SnapDragon821 SoC from Qualcomm [1] and Apple`s A10 SoC [2] have a quad-core CPU, Samsung`s Exyos8 [3] has an octa-core CPU, while MediaTek Helio X20, X25 and X30 [4] have a Decacore CPU. With the availability of such complex and advanced SoCs the design and implementation of such systems becomes ever more complex and extremely time consuming using the conventional methods. With the end of Moore`s law on the horizon, more than ever, there is a need to fill the "design gap" between the available resources on a chip and development productivity by employing highly optimized design at the system level, both on MPSoC architecture as well as application software. The truly heterogeneous nature of Multi Processor System on Chip (MPSoC) means architects face a larger design space, bigger extent of design decisions, complexity and bigger tradeoffs. Along with the design of processors and application specific processors, GPU and DSP etc, the complexity of communication

architecture also poses a challenge for designers. Communication architectures require a very detailed and careful consideration during the design phase in order to meet performance, power and latency constraints.

Recent commercial Electronic System Level (ESL) design tools aim to enhance the productivity of design efforts [5]. Some of these efforts are focused on providing a software development platform, before the detailed architecture of the MPSoC is fixed. Such a "virtual platform" enables designers to optimize the architecture of the MPSoC and the software concurrently, through fast system level simulations. However, these design processes require a great degree of manual designs and iterations, in particular, these software must be written, somewhat correctly and completely including device drivers and OS models etc in order to drive the system-level MPSoC simulations. To improve the system simulation speed, multiple techniques have been developed including abstract hardware models such as SystemC TLM2.0 [6] and advanced Instruction Set Simulator (ISS) technologies [7] [8]. However, the increasing number of cores on current MPSoCs causes long simulation times and as a result make the architecture exploration and performance estimation process a long and time consuming process. Moreover, during the course of software and architecture refinement, there is a risk of bug insertion which could make the exploration and estimation process further time consuming and complex.

1.2 Statistical workload modeling

Another effective method for design space exploration is statistical workload models. Statistical workload models can enable fast exploration of a wide design space. However, getting the appropriate workload model is highly application dependant and usually apply to data traffic but are difficult to generate accurate computation load models. In case of communication centric MPSoC subsystems, statistical workloads have been used for performance estimation of communication architecture such as shared bus architectures, namely Core-Connect from IBM [9], AMBA from ARM [10], SiliconBackplane from Sonics [11], STBus from STMicroelectronics [12] and more recently, Network On Chips NOCs [13] [14].

1.2.1 Trace-driven workload modeling

In order to address the challenge of performance estimation through workload modeling, *Trace-driven workload model* has been proposed [15] as a way of precisely capturing data-dependant behavior of a software application to achieve near-cycle-accurate performance estimation. Program execution trace of an application on a given input data set is computed, through source-level instrumentation and native code execution, prior to the trace-driven workload simulation and efficiently encoded as branch bitstream. This branch bitstream is then used to steer the workload models in the form of program trace graphs generated for the target processor inside a parameterized MPSoC performance estimation frame-work. Accurate coarse-grain workloads are automatically generated directly from the application source code by using the precise execution profile information obtained from the same branch bitstream. Figure1-1 shows a program trace graph (PTG) where each PTG-node represents either function-start, function-end, branch or call in the application code. While a PTG-edge carries attributes about the program execution information, including cycle count between the two connecting nodes. The *branch bitstream*, which is simply a sequence of branch condition bits, that record the branch outcome (true or false) with a single bit in the order of program execution. This branch bitstream together with the PTG is used to drive the actual execution trace of a program and generate the PTG-edge sequence. The total cycle counts are calculated by summing the cycle-count values of each edge in the generated edge sequence while the cycle count on each edge is dependent on the target processor. *Branch bit-stream* can be used together with the PTG to generate the PTG-edge sequence. Starting at the start of the PTG traversing through the edges, each edge is added to the PTG-edge sequence until a branch node is reached. On reaching a branch node, the corresponding branch bit is read from the branch bit-stream to determine which direction (in the shown figure, true is represented by blue colored edge and false is represented by red colored edge) to traverse next. Depending on the value (1 or 0) the corresponding edge is added to the edge sequence. The entire PTG is traversed in this fashion, such that a bit-stream bit is read on reaching each branch node to determine the next traversal edge until the “main-end” is reached. The total cycle counts on a PTG-edge sequence are simply calculated by

summing the cycle counts on each individual edge. For example in Figure1-1, total cycle count on the PTG edge sequence "$e_0e_1e_7e_9e_2e_3e_3e_4e_5e_7e_8e_6$" is calculated as follows.

$$T = c(e_0) + c(e_1) + c(e_7) + c(e_9) + c(e_2) + c(e_3) + c(e_3) + c(e_4) + c(e_5) + c(e_7) + c(e_8) + c(e_6)$$

Where, $c(e_i)$ is the cycle count on PTG-edge e_i.

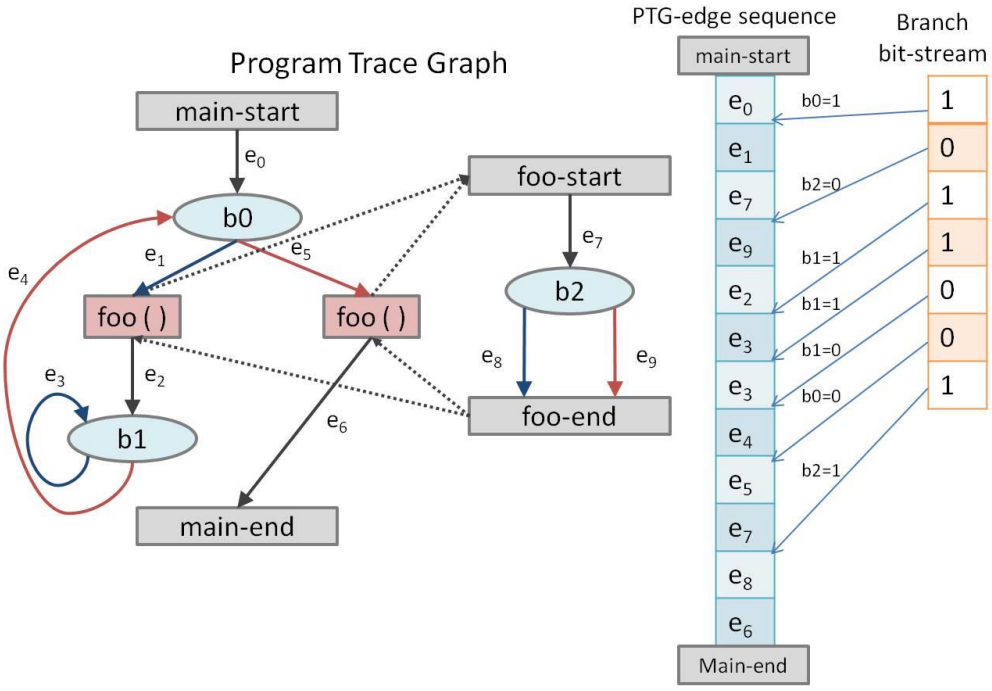

Figure 1-1 : Program Trace Graph (PTG) and branch bit-stream

The *Trace-driven workload model* is very useful in fast performance estimation and shows a clear advantage over the ISS simulation based performance estimation methods. However, for bus traffic it has to resort to simulation techniques to resolve bus arbitration. The trace-driven bus workload simulation works such that, bus traffic generation is triggered by processor communications, access to memories and IPs. The bus model implemented in the trace simulator is driven by the global scheduler inside the trace simulator kernel that dispatches processor on processor queue to the workload simulator. Several standard arbitration schemes are supported for example fixed priority, round-robin and QoS guaranteed. In case of priority based arbitration, 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. In this model, bus requests are arbitrated by cycle-accurate bus simulation that leads to a huge computational load on the entire trace simulator.

1.3 Research contributions

In this research we present 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 [16] and cannot be modeled accurately 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 as detailed in 1.2.1. 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. The proposed bus-model can replace the bus traffic simulation model in the trace-driven workload simulator such that bus requests do not need to be arbitrated by cycle-accurate bus simulation hence saving a huge computational load on the entire trace simulator.

In summary our research contributions are as follows;

i. Present a methodology to enable bus performance estimation much faster than simulation based performance estimation techniques while achieving much higher accuracy than static analysis based estimation techniques.

ii. Develop a mathematical model for estimation of MPSoC bus stall using traffic statistics.

iii. Use developed mathematical model together with a trace-driven workload simulator enabling fast bus performance estimation in a trace-driven workload simulation environment.

1.4 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. While some research has been focused on static analysis to estimate bus performance others focused on simulation based techniques.

1.4.1 Static techniques

Static techniques for communication architecture performance estimation are attractive due to its speed. Static analysis techniques are faster and require less effort compared to simulation techniques. Static methods attempt to determine the performance of a system through analysis using closed form expressions that capture system performance as a function of parameters. The following are some notable works for static analysis based performance estimation techniques. Knudsen et al presented a high level estimation model for communication throughput for a given protocol assuming pipelined transfers [17]. They estimate burst time and data packet splitting or joining time at the interface. Yen and Wolf propose to estimate the communication delay using the worst-case response analysis of the real-time scheduling [18]. Daveau et al. considered static information like maximum bandwidth of channel and bandwidth of processing elements to estimate performance of interconnect between processing elements [19]. Nandi and Marculescu use continuous-time Markov process technique for performance measurement [20]. Drinic et al used the profiled statistics of inter-core traffic for core-to-bus assignment [21]. Thepayasuwan and Doboli propose a simulated annealing approach [22]. 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 and

request and address phases occur in the same cycle [23]. Latency is calculated using parameters such as, number of data items to be transferred, number of masters of the bus, burst size, probability of single mode transfers on shared bus, probability of pipelined transfers on the bus etc. However static techniques of performance estimation have several limitations as follows;

i. Most static techniques are unable to account for non-deterministic traffic generation by the bus masters for example memory access delays.
ii. Require several functionality dependent assumptions.
iii. Static techniques cannot capture delays incurred due to dynamic factors, for example, stalls incurred due to traffic congestion. Hence these techniques are not suitable for a thorough performance estimation of today's MPSoC.

1.4.2 Simulation based techniques

Simulation based approaches employing various levels of abstraction have been more popular for performance estimation. Several modeling abstractions are used by designers namely algorithm, Transaction Level Modeling (TLM), Transaction-Bus Cycle Accurate (T-BCA), Pin Accurate-Bus Cycle Accurate (PA-BCA), CA and Register Transfer Level (RTL) listed in order of decreasing abstraction. Given the implementation and modeling complexity as well as effort and time demands of complete modeling of MPSoC architecture, higher levels of abstraction are used in various works trading off some level of accuracy for less design effort and complexity and increased simulation speed. Loghi et al. used Cycle Accurate models written in SystemC to explore AMBA2 and STBus communication architectures for MPSoCs [24]. Sermeria et al. used PA-BCA models (modeled in SystemC) to improve simulation speed over CA models [25]. Kalla et al. executed traces of component behavior on a Pin Accurate Bus Cycle Accurate simulator and achieved speed up over CA simulation model [26]. 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 [27] [6]. Ogawa et al. created another T-BCA model variant for the AMBA AHB bus architecture using C as the modeling language [28]. Ariyamparambath et al. annotated ATLM models with bus-protocol-specific timing details [29]. Viaud et al.

proposed TLM/T abstraction level [30]. 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 [31]. Beltrame 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 [32]. 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 [33] [34] [35] to speed-up simulation.

1.4.3 Hybrid techniques

Hybrid approaches aim to reduce the design effort and simulation time while maintaining accuracy.

Hybrid approach between a static estimation and a simulation approach has been developed by Lahiri et al. [36]. Initially a co-simulation of the system is performed with the communication described in an abstract manner as events or abstract data transfers. An abstract set of traces are extracted from the initial co-simulation that contain necessary and sufficient information about the computations and communications of the system components. The system designer then specifies a communication architecture by selecting a topology, mapping the abstract communications to paths in the communication architecture, and finally customizing the protocol used for each channel. The traces extracted in the initial step are represented as a *Communication Analysis Graph* (CAG), and an analysis of the CAG provides an estimate of the system performance, as well as various statistics about the components and their communication. In this technique, the initial use of static analysis to accumulate the traces is somewhat similar to our approach however, in the second phase of performance estimation, trace simulation is performed to resolve arbitration. Due to this, their approach converges to trace driven simulation as the memory traces become larger and does not provide a significant simulation speed-up.

Kim et al. proposed another hybrid performance estimation approach based on queuing analysis [37]. They present a static performance estimation method that is based on the queuing model and makes use of memory traces and task execution schedule information. Their approach is similar to ours in the use of program traces as input while, in contrast with our approach, a queuing model together with a state-transition FSM is used to estimate bus stalls due to contention. They propose to use this static estimation approach to prune the design space and then using simulation-based approach for a smaller design space. They report up to 13% estimation errors even for synthetically generated traffic. Hence use of the simulation method becomes necessary at some point in the design space exploration. Moreover, static queuing models are not able to capture advanced bus features like split-transaction, out-of-order transaction etc. because it affects the service rates used in the queuing model. This aspect limits the applicability of queuing analysis to advanced bus-systems. In contrast mathematical expressions for analytical models are not based around service rates and with due consideration during derivation, such characteristics can be modeled. For instance by introducing a memory access delay factor in the model such that different values are used for different kinds of bus-transactions. 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 during the system architecture design in an early phase of development [38]. However, in contrast to our approach, the analysis is not based on “actual trace” of a program rather UML or state-chart of the program is used which results in inaccuracy while the UML execution also introduces longer simulation time. Moreover the model applicability is shown for only two bus-masters. For a two bus-master system an error of 3% is reported as opposed to the well under1% errors achieved through our approach for two PE-systems.

1.5 Thesis organization

The rest of the thesis is organized such that, chapter 2 gives an overview and comparison of trace driven workload simulation flow using simulation-based bus model or using the proposed estimation based model. Chapter 3, 4 and 5 develop the statistical estimation

models called Single Blocking Model, Burst Blocking Model and Multi-Blocking Model respectively. Chapter 6 reports the experiments and results while chapter 7 presents conclusion and future works.

2. TRACE DRIVEN WORKLOAD SIMULATION FLOW WITH STATISTICAL BASED BUS MODEL

This chapter gives a detailed overview of the trace driven workload simulation flow and highlights the difference between simulation based and statistical based bus models in the overall simulation flow.

The trace driven workload simulation works as follows;

1. Initially original sequential code of an application program and the corresponding set of inputs are put through native execution taking full advantage of the fast native execution and branch bit-stream is recorded. This stage is decoupled with the partitioning which is a very attractive feature because a single branch bit-stream can drive multiple work-load models that have been derived from multiple different partitioning.
2. Application partitioning is performed on the same application program by inserting annotations in the sequential code. The previously obtained sequential branch bit-stream is decomposed into thread branch bit-stream.
3. Next, the thread branch bit-stream and target processor and target bus model are used with Program Trace Graph (PTG), eventually obtaining the workload model.
4. This workload model is used by the Trace Driven Workload Simulator together with the target MPSoC architecture to obtain a performance estimation. Multiple iterations involving MPSoC optimization, processor and bus optimization, and application optimization are performed for a vigorous and detailed design space exploration. Figure 2-1 shows an overview of the simulation flow.

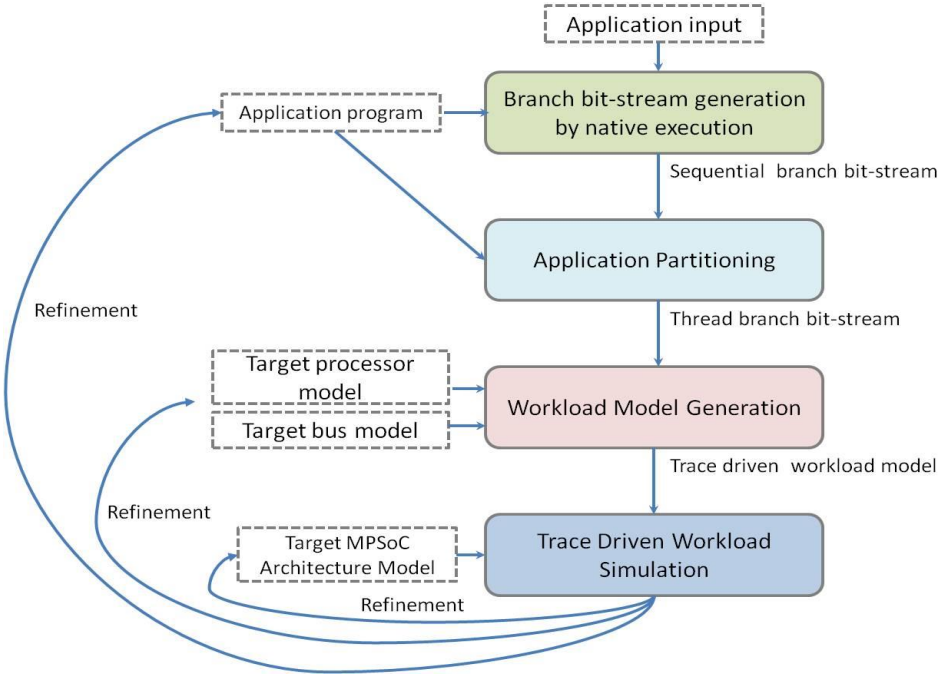

Figure 2-1: Overview of trace driven workload simulation flow

2.1 MPSoC architecture model

The MPSoC architecture modeled in Trace-driven workload simulation framework mainly consists of processing elements (PE) and the interconnection between PEs. A PE includes a processing unit, program memory, data memory and a communication processor capable of handling data transmit and receive operations with other PEs. While the interconnect architecture is modeled as a parameterized multi-bus. Each PE connects to a single bus on its output and to all buses on its input. This multi-bus model enables modeling of a wide variety of bus topologies from a single bus to full-crossbar. Figure 2-2 and Figure 2-3 show a simple block diagram of PE and the interconnect. The MPSoC architecture can be configured on a few parameters.

1. *Processor type:* Currently a single processor type is supported which is as implemented in a prototype MPSoC called TCT-MPSoC [39].
2. *Thread to processor mapping:* Threads are statically allocated to processors, where multiple threads can be allocated to a single processor. The simulator kernel directly handles the context switching of threads within the processors using non-preemptive

scheduling where the parameterized context switching overhead is also modeled.

3. *Bus Topology:* MPSoC interconnect is modeled as a parameterized multi-bus as explained above. This multi-bus model guarantees that any two processors have direct connectivity, which allows modeling a wide variety of bus topologies. PEs can be mapped to a bus in the MPSoC architecture model description.
4. *Communication channel:* communication setup time, data transfer rate and buffer depth can all be configured.

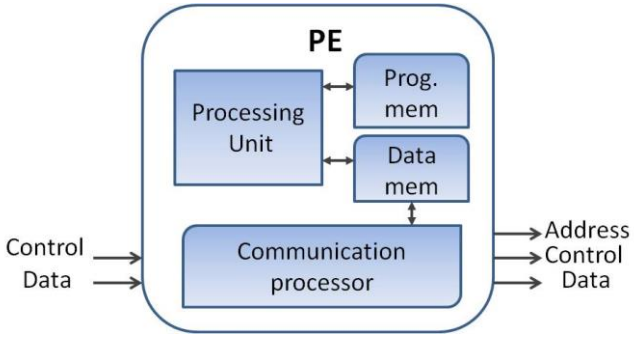

Figure 2-2: Processing element as modeled in trace driven workload simulator

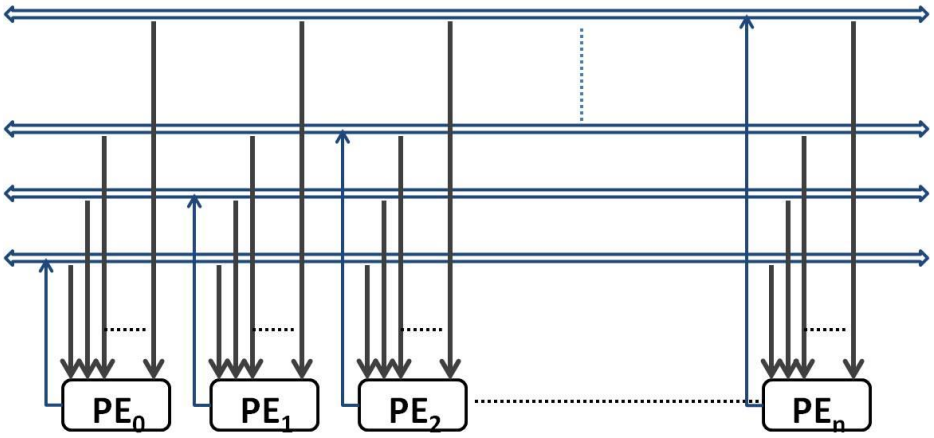

Figure 2-3: Full cross-bar Interconnect as modeled in trace driven workload simulator

2.2 Trace driven workload simulation kernel

The *trace-driven workload simulator kernel* consists of two major components:

1. *Thread workload simulator*

Thread workload simulator is responsible for updating the *local processor clock* while translating the thread branch bit-stream into PTG-edge sequence and accumulating the edge cycles. Once activated, the thread workload simulator continues to increment its local processor clock until the generated PTG-edge sequence reaches a node that requires synchronization and a strict time-ordered simulation by the *global scheduler*. Note that mutually exclusive resources (buses and data buffers) also maintain their own clocks to indicate their occupancy status during simulation.

2. *Global scheduler*.

   The global scheduler activates the thread trace simulator with the earliest local clock by maintaining a queue of active threads.

Figure 2-4 shows the working principles of trace driven workload simulation kernel. The *global scheduler* maintains a queue of active threads. The processor with the earliest clock is at the top of the queue. The thread with earliest clock is dispatched to the *trace workload simulator* where the local processor clock of the thread is updated while accumulating edge cycles until a synchronization node is reached, at which point control will be returned to the *global scheduler* where the thread is inserted into the processor queue with the appropriate thread status. The threads are sorted based on earliest time and incase of tie based on higher priority.

When a thread waiting for bus access is on top of the queue, the global scheduler performs one of the following

1. If the bus clock is earlier or same as the subject thread`s local clock, bus access is granted since this indicates that the bus is available. The local clock of the bus, as well as the thread, is updated accordingly. The updated value depends on the length of bus-workload and the modeled bus set-up time, bus latency, bridge characteristics etc. The thread is then inserted back into the processor queue.
2. If the bus clock is later than the subject thread`s local clock, the bus arbitration is lost since this indicates that the bus is occupied. The local clock of the thread is updated by the difference of the two local clocks i.e. the value *(busclock - threadclock)* and the thread is inserted back in the queue with a delayed status.

Figure 2-5 demonstrates the simulation flow of how a bus pending thread acquires the bus. The bus model and simulation flow will be discussed in further detail in the next section.

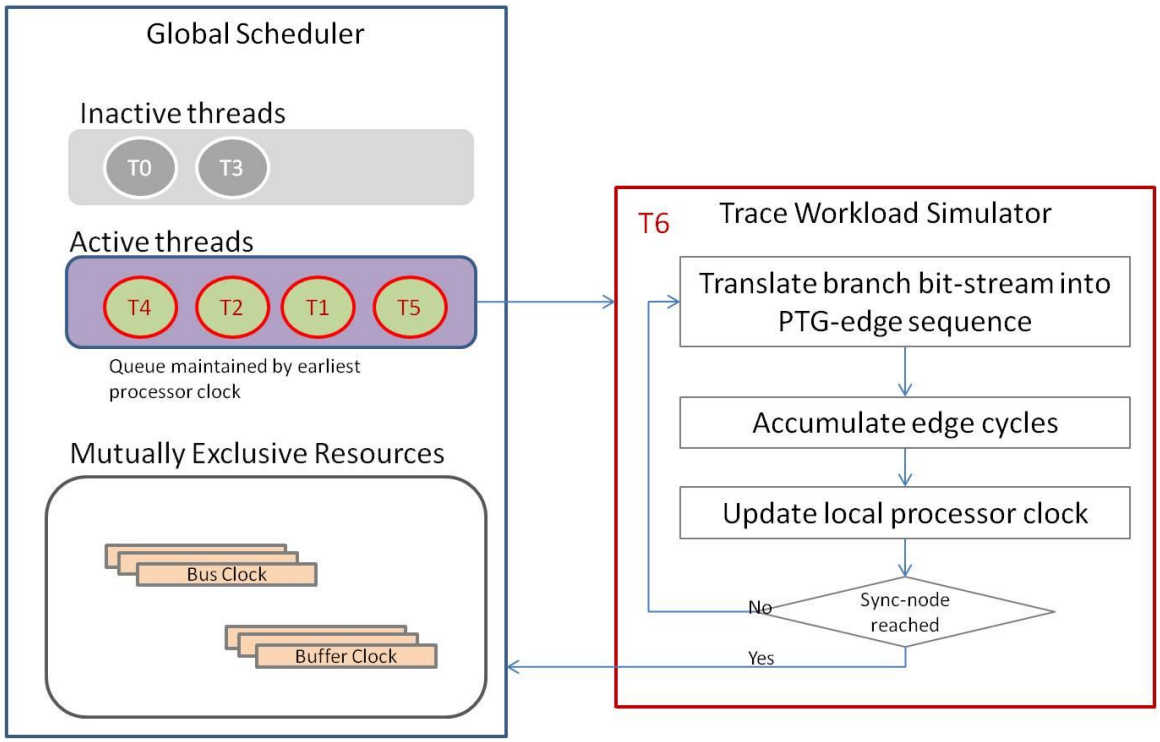

Figure 2-4: Overview of simulation kernel

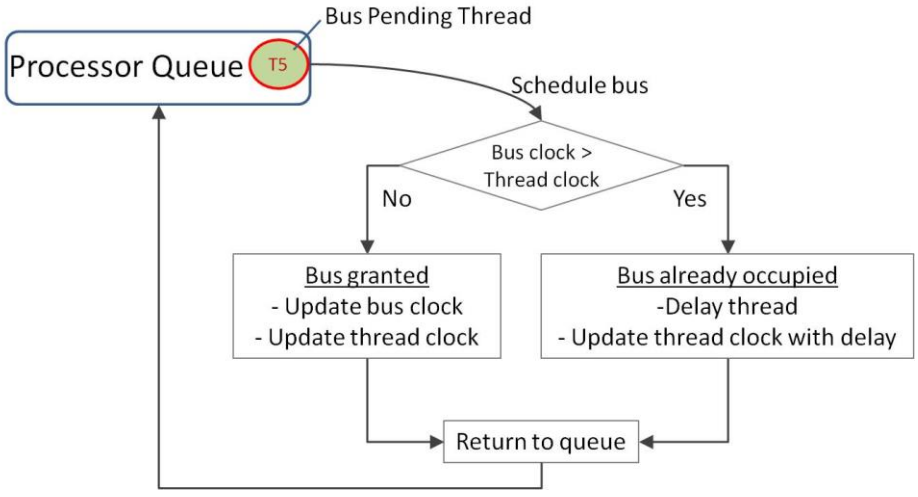

Figure 2-5: Simulation flow of bus access

2.3 Bus model

There are two types of bus models discussed in this section. Simulation based bus model is implemented with-in the trace simulator such that it relies on arbitration simulation to resolve bus requests whenever a bus request is issued on any of the competing bus-masters. On the other hand, the proposed, statistical based estimation model ignores bus arbitration (assuming full cross-bar) for a specific time window and invokes the statistical estimation model after the window has passed. The statistical model calculates the expected bus stall for each bus master over the period of the time window. This stall is eventually added to the processor clock to reflect the incurred bus stall.

2.3.1 Simulation-based bus arbitration model

Simulation based bus modeling in trace simulator is driven by the global scheduler inside the trace simulator kernel that dispatches the 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. Figure 2-6 shows an example system with three processors sharing a single bus. P0, P1 and P2 show the trace workload on Processors P0, P1 and P2 respectively. BUS shows the workload on the shared bus at any given time. A downward arrow with a "O" symbol indicates a bus request that wins bus arbitration while a down arrow with a "X" symbol on it indicates a blocked bus request. A horizontal dotted arrow indicates bus stall due to arbitration loss. The vertical dotted lines indicate points in time when any bus request is granted. On top of each bus grant, sorted priority list is shown to indicate how priorities are sorted. For this example a fixed priority protocol is assumed such that P0 has the highest priority while P2 has the lowest priority. Every time bus is available, the processor with the highest priority wins arbitration and acquires the bus. At grant 1, the processor queue is sorted by bus-request arrival time, therefore P2 wins bus arbitration. At grant 3, the processor priority is sorted based on processor priority. Note that when bus requests on two processors are stalled and waiting for bus-access, the higher priority processor takes precedence over the lower priority processor.

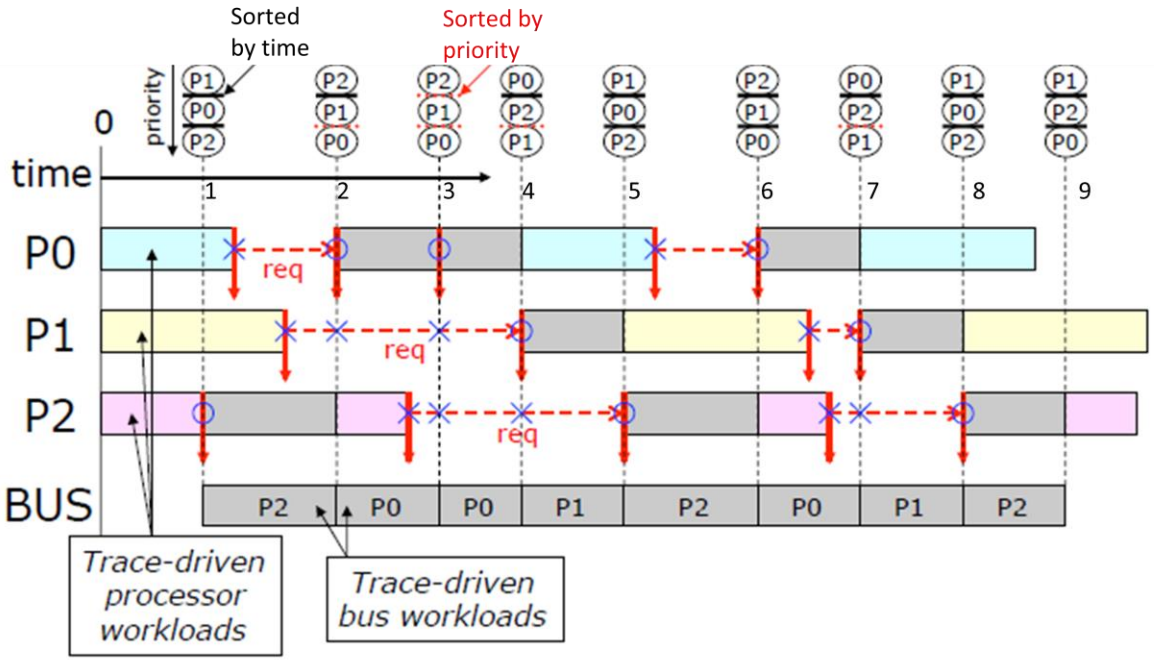

Figure 2-6: Simulation-based arbitration simulation overview [40]

In this model, multiple bus requests are arbitrated by cycle-accurate bus simulation that leads to a huge computational load on the entire trace simulator. This is due to the bus arbitration simulation and frequent manipulations on the processor queue.

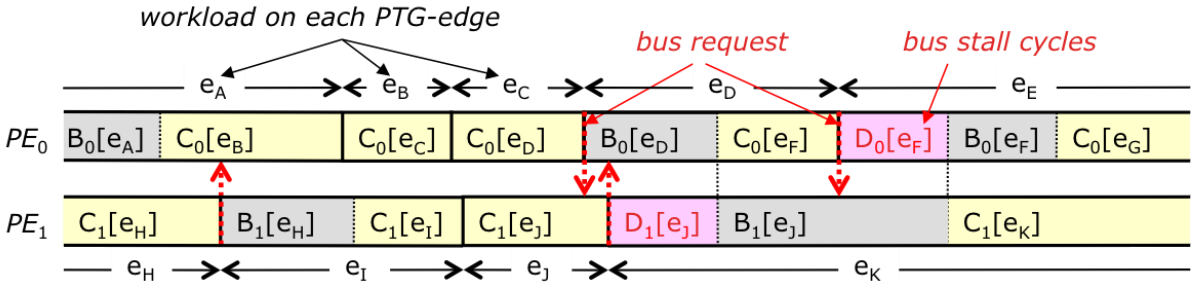

Figure 2-7: Working principles of the simulation based estimation

Working principles of the simulation based cycle count estimation is illustrated in Figure 2-7. PE_0 and PE_1 represent two processing elements and the figure shows corresponding workloads on each Processing Element (PE). On the trace simulator, the workload corresponding to each PTG-edge e_n at processor PE_i $(i = 0,1, \dots)$ 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.

1. Computation workload $C_i[e_n]$ denotes the number of execution cycles on the instruction stream contained in PTG-edge e_n.
2. 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.
3. When a PTG-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.

2.3.2 Statistical based bus model

The objective of the new method for bus stall cycle prediction is to avoid the time-consuming bus arbitration simulation by introducing statistical techniques in the bus model. Figure 2-8 illustrates the bus statistics used in stall cycle prediction that are collected during normal trace simulation.

1. At processor PE_i $(i = 0,1, \dots)$, 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.
2. Statistics of $C_i[e_n]$ and $B_i[e_n]$ are collected at PE_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) where histogram h_{Li} for L_i are collected.
    - Histogram h_{Bi} on all bus workloads B_i are collected.
    - N_i : total number of bus workloads within the bus prediction interval T

3. Normal workload simulation and statistics recording continues until the simulation clock elapses for bus prediction interval T cycles.

4. Statistics (N_i, h_{Li}, h_{Bi}) collected during bus prediction interval T at each processor PE_i is used to compute the expected bus stall cycles per request $E[D_i]$.

5. 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.

Figure 2-9 illustrates the bus stall prediction flow using bus workload statistics (N_i, h_{Li}, h_{Bi}) collected during trace simulation.

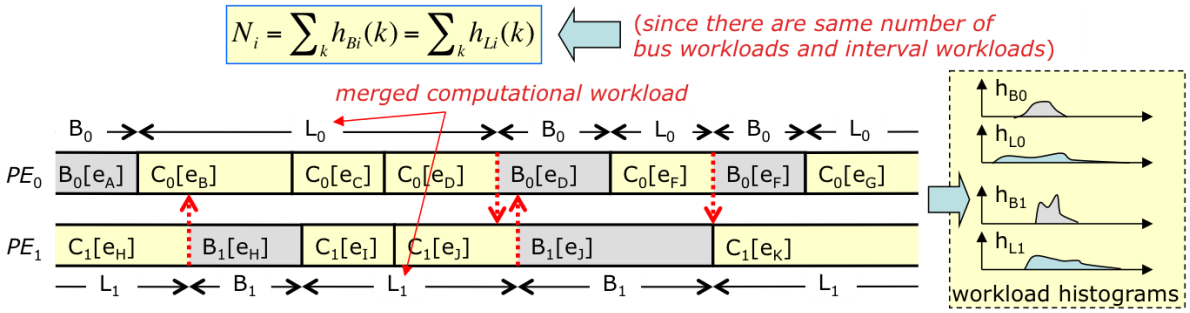

Figure 2-8: Bus statistics collected during trace simulation

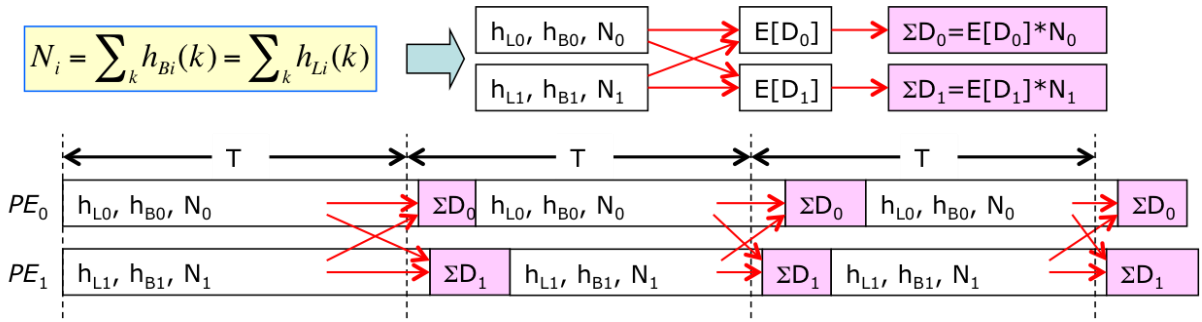

Figure 2-9: Bus stall prediction flow

3. SINGLE BLOCKING MODEL

This chapter presents a statistical model that can be used for bus stall estimation in the fashion described in the previous chapter. Developing the mathematical model consists of multiple stages such that a basic behavior is modeled first and more complicated behaviors are incrementally built on-top of the basic model. First we develop a basic model that only captures *single blocking* behavior.

3.1 Single blocking behavior

We define *single blocking* behavior as the assumption that a bus request on a Processing Element (PE) does not get blocked or lose bus arbitration more than once. This assumption dictates that a bus request on a PE will not be blocked for more than one bus workload on any other PE. Hence the maximum possible bus stall for a given bus request will not exceed one bus workload on the blocking PE as shown in Figure 3-1 i.e. when a bus request r_i is generated on a PE_i and it loses arbitration to another PE_j either due to r_i arriving later than r_j or due to PE_i having a lower processor priority, then the single blocking behavior assumes that request r_i will be granted immediately after PE_j releases the bus.

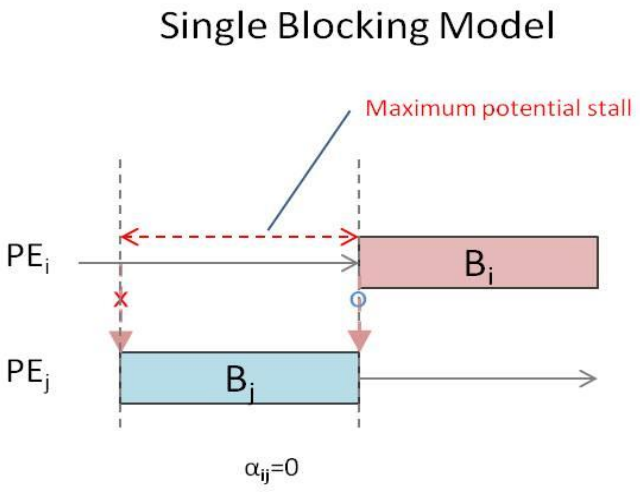

Figure 3-1: SBM maximum potential stall

Such a behavior is realistic if

1. There are only two PE competing for the bus
2. PE do not generate burst traffic i.e. there is at least one cycle gap between the end of a bus workload and generation of another bus request on a given PE. Figure 3-2 shows such a system where the two assumptions stated above result in a system that exhibits the single blocking behavior.

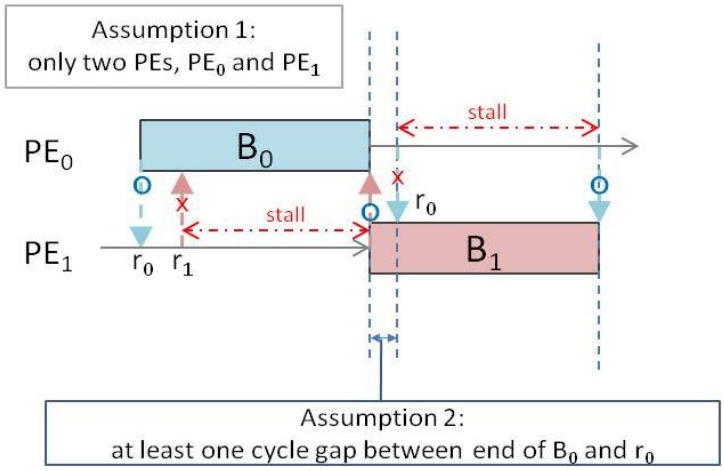

Figure 3-2: Typical SBM system

3.2 Single blocking model

Here, to model the single blocking behavior, we introduce a simplified statistical bus model called *single blocking model* with the following assumptions:

1. Random bus request generation process: bus request is generated randomly with probability λ_i.
2. Binary value α_{ij} describes the *fixed* priority relationship between processors PE_i and PE_j.

$$\alpha_{ij} = \begin{cases} 0 & \left(priority(PE_i) < priority\left(PE_j\right)\right) \\ 1 & \left(priority(PE_i) > priority\left(PE_j\right)\right) \end{cases} \quad \text{eq(3.1)}$$

3. Single blocking behavior constraint1: bus request on any PE can be blocked by, maximum, one bus workload at a time.
4. Single blocking behavior constraint2: PE do not generate burst traffic.

3.2.1 Request Probability and Average Interval Workload

On Single Blocking Model (SBM), a request is generated with probability λ_i at the *next cycle*, guaranteeing at least one cycle interval between successive bus workloads. Under this assumption, the probability that interval L_i equals n (cycles) decays with increasing n and is given as:

$$\Pr(L_i = n) = \lambda_i(1-\lambda_i)^{n-1} \quad (n \geq 1) \quad \text{eq(3.2)}$$

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

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

$$\therefore \lambda_i = \frac{1}{E[L_i]} \quad \text{eq(3.3)}$$

Figure 3-3 shows the decaying probability of $\Pr(L_i = n)$ for an assumed value of λ_i=0.25.

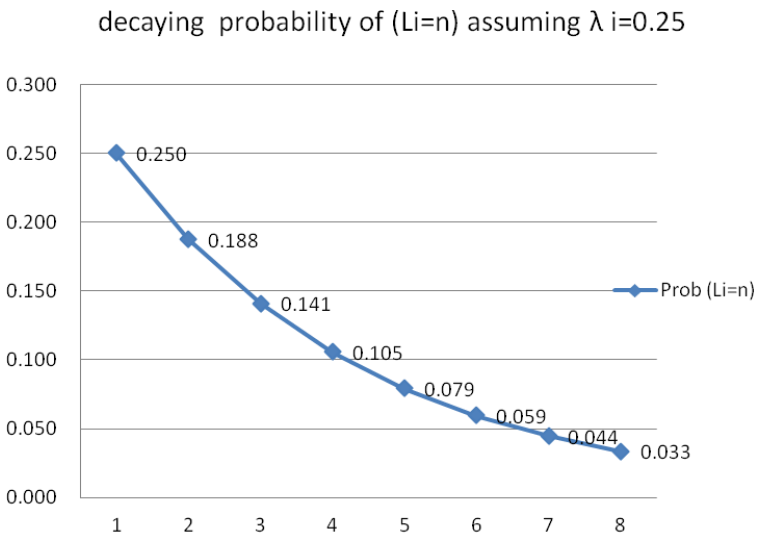

Figure 3-3: Probability of Pr(L_i=n)

3.2.2 Probability Mass Function of Bus Workloads

Probability mass function of bus workloads B_i is derived from histogram h_{Bi} as follows:

$$E[B_i] = \sum_{k} f_{Bi}(k) \cdot k$$

$$f_{Bi}(k) = \Pr(B_i = k) = \frac{h_{Bi}(k)}{\sum_m h_{Bi}(m)} = \frac{h_{Bi}(k)}{N_i} \quad \text{eq(3.4)}$$

Here, $h_{Bi}(k)$ is the number of bus workloads at processor PE_i whose length is k, and $f_{Bi}(k)$ is the probability that $B_i = k$.

3.2.3 Blocking Condition of Bus Requests

Below terminologies will be used 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

$R'_{ij} = \Pr(blk_{ij})$: probability that request r_i is blocked by bus event b_j

$b_j(k)$: 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

$R'_{ij}(k) = \Pr(blk_{ij}(k))$: probability that request r_i is blocked by bus event $b_j(k)$

$t_{ij}(k)$: time difference between the first cycle of $b_j(k)$ and request r_i

Request blocking condition of request r_i by bus event $b_j(k)$ is given as follows:

$$\alpha_{ij} \le t_{ij}(k) \le k - 1$$

Here, $1 \le t_{ij}(k) \le k - 1$ applies on either priority cases since $t_{ij}(k) \ge 1$ indicates that bus event $b_j(k)$ occurred at least one cycle prior to request r_i. When $\alpha_{ij} = 0$ $\left(priority(PE_i) < priority(PE_j)\right)$, $t_{ij}(k) = 0$, indicating that $b_j(k)$ and r_i occur on the same cycle, becomes the blocking condition since r_j (request of $b_j(k)$) has higher priority than r_i. When $\alpha_{ij} = 1$ $\left(priority(PE_i) > priority(PE_j)\right)$, on the other hand, $t_{ij}(k) = 0$ is *not* the blocking condition since r_i will not be blocked by $b_j($. Figure 3-4 shows the corresponding stall incurred for the cases $\alpha_{ij} = 0$ and $\alpha_{ij} = 1$ for $t_{ij} = 0$ and $t_{ij} > 0$

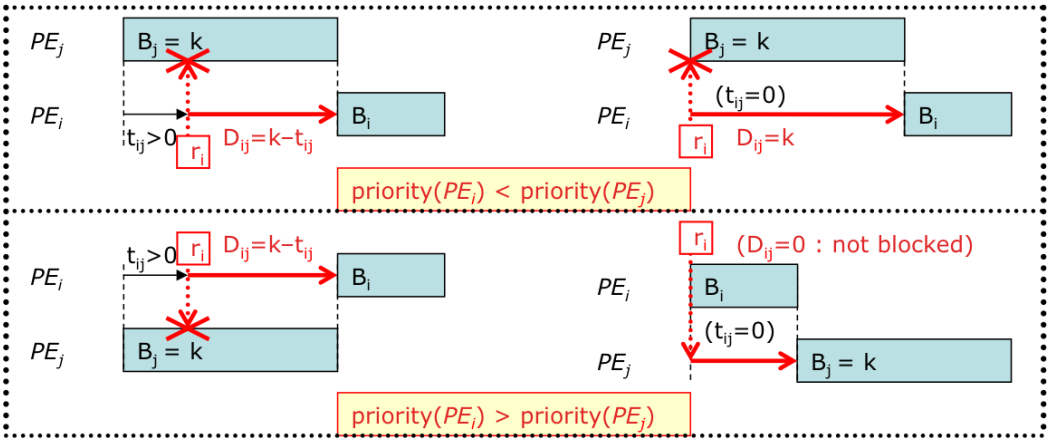

Figure 3-4: Stall incurred for various cases

3.2.4 Bus Stall Cycle Expectation Equations

First, bus event b_j that potentially blocks the request r_i is further categorized into two subclasses:

Consecutive bus event bb_{ij}: bus event b_j at processor PE_j follows immediately after (with no interval) a bus event b_i at processor PE_i. Let S_{ij} be the probability of event bb_{ij}:

$$S_{ij} = \Pr(bb_{ij})$$

Non-consecutive bus event $\overline{bb_{ij}}$: bus event b_j at processor PE_j follows a bus event b_i at processor PE_i after one or more cycles. Clearly, the following holds:

$$\overline{S_{ij}} = \Pr(\overline{bb_{ij}}) = 1 - \Pr(bb_{ij}) = 1 - S_{ij}$$

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 $\overline{bb_{ij}}$.

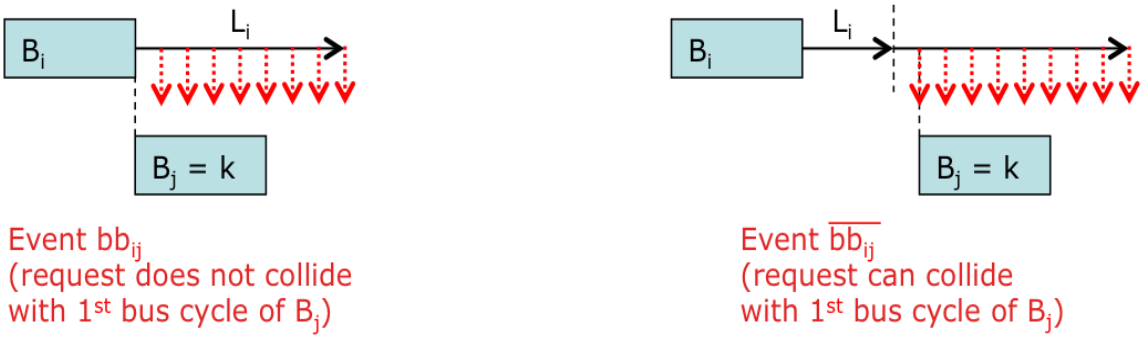

Figure 3-5: Event bb_{ij} and $\overline{bb_{ij}}$

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 $f_{Bj}(k)$, then $S_{ij}(k)$ and $\overline{S_{ij}}(k)$ are given as

$$\begin{cases} S_{ij}(k) & = & S_{ij} \cdot f_{Bj}(k) \\ \overline{S_{ij}}(k) & = & \overline{S_{ij}} \cdot f_{Bj}(k) = \left(1 - S_{ij}\right) \cdot f_{Bj}(k) \end{cases}$$

3.2.4.1 Conditional blocking probability and stall

Figure 3-6 illustrates the blocking probabilities and bus stall delays on event $bb_{ij}(k)$ with specific values of $t_{ij}(k)$. Here, the probability of $t_{ij}(k) = n$ $(n = 1,2, \dots, k-1)$ on event $bb_{ij}(k)$ is given as

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

which is derived from eq(3.2), and bus stall length is given as

$$D_{ij}\left(k | t_{ij}(k) = n, bb_{ij}(k)\right) = k - n$$

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

$$\begin{aligned} R'_{ij}\left(k | bb_{ij}(k)\right) &= \sum_{n=1}^{k-1} \lambda_i (1 - \lambda_i)^{n-1} &&= 1 - (1 - \lambda_i)^{k-1} \\ E\left[D'_{ij}(k) | bb_{ij}(k)\right] &= \sum_{n=1}^{k-1} \lambda_i (1 - \lambda_i)^{n-1} (k - n) &&= \frac{k\lambda_i - 1 + (1 - \lambda_i)^k}{\lambda_i} \end{aligned} \quad \text{eq(3.5)}$$

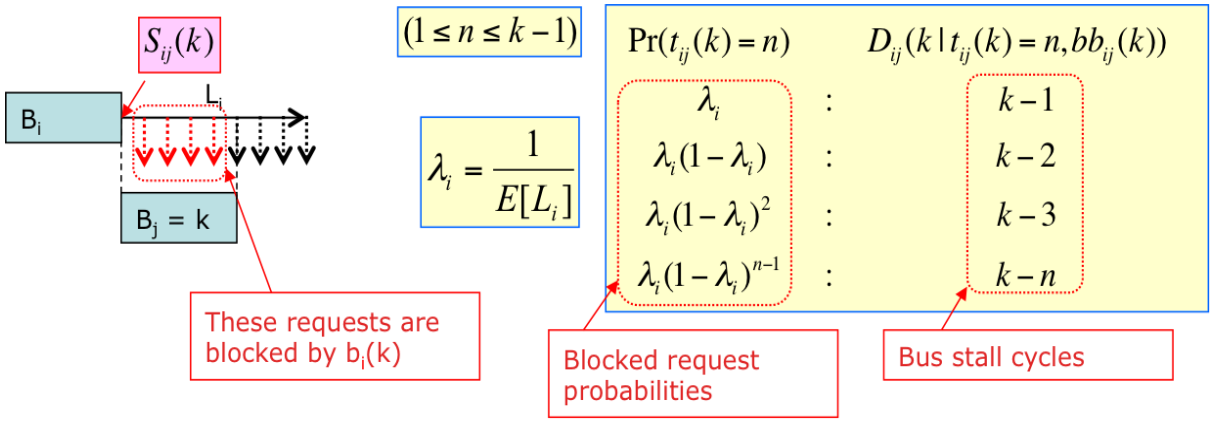

Figure 3-6: Conditional bus stall

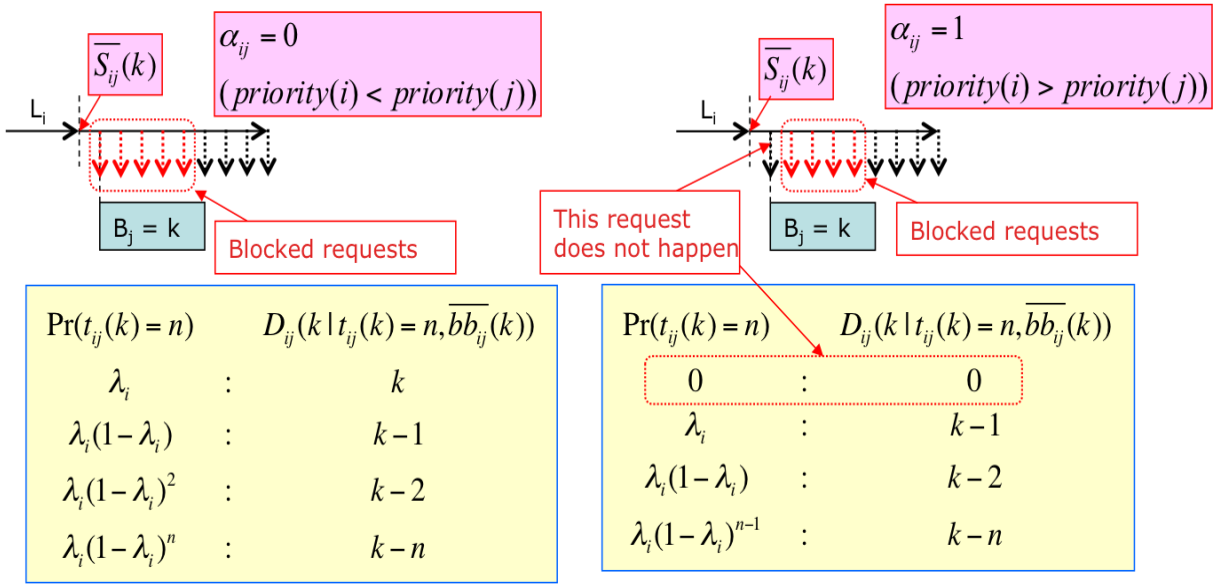

Figure 3-7: Conditional bus stall

Figure 3-7 illustrates the blocking probabilities and bus stall delays on event $\overline{bb_{ij}}(k)$ with specific values of $t_{ij}(k)$. Here, the situation changes for different priority relation α_{ij}. When $t_{ij}(k) = 0$, according to eq(5), the request blocking condition does not hold for $\alpha_{ij} = 1$ but holds for $\alpha_{ij} = 0$. Thus the probability of $t_{ij}(k) = n$ and its corresponding bus stall delay are:

$$\Pr\left(t_{ij}(k) = n | \overline{bb_{ij}}(k)\right) = \begin{cases} \lambda_i(1-\lambda_i)^n & (\alpha_{ij} = 0) \\ \lambda_i(1-\lambda_i)^{n-1} & (\alpha_{ij} = 1) \end{cases}$$

$$D_{ij}\left(k | t_{ij}(k) = n, \overline{bb_{ij}}(k)\right) = k - n$$

where the effective ranges of n on the above expressions are $\alpha_{ij} \le n \le k-1$. Then the conditional blocking probability and conditional bus stall delay on event $\overline{bb_{ij}}(k)$ are derived individually for higher priority and lower priority blocking PE.

3.2.4.1.1 Higher priority blocking PE case ($\alpha_{ij} = 0$)

$$\begin{aligned} R'_{ij}\left(k|\overline{bb_{ij}}(k)\right) &= \sum_{n=0}^{k-1} \lambda_i (1-\lambda_i)^n &&= 1-(1-\lambda_i)^k \\ E\left[D'_{ij}(k)|\overline{bb_{ij}}(k)\right] &= \sum_{n=0}^{k-1} \lambda_i (1-\lambda_i)^n (k-n) &&= \frac{(k+1)\lambda_i - 1 + (1-\lambda_i)^{k+1}}{\lambda_i} \end{aligned} \quad \text{eq(3.6)}$$

3.2.4.1.2 Lower priority blocking PE case ($\alpha_{ij} = 1$)

$$\begin{aligned} R'_{ij}\left(k|\overline{bb_{ij}}(k)\right) &= \sum_{n=1}^{k-1} \lambda_i (1-\lambda_i)^{n-1} &&= 1-(1-\lambda_i)^{k-1} \\ E\left[D'_{ij}(k)|\overline{bb_{ij}}(k)\right] &= \sum_{n=1}^{k-1} \lambda_i (1-\lambda_i)^{n-1} (k-n) &&= \frac{k\lambda_i - 1 + (1-\lambda_i)^k}{\lambda_i} \end{aligned} \quad \text{eq(3.7)}$$

3.2.4.2 Over all blocking probability and stall

Next, the overall blocking probability $R'_{ij}(k) = \Pr\left(blk_{ij}(k)\right)$ and the overall bus stall delay expectation $E\left[D'_{ij}(k)\right]$ on all bus events $blk_{ij}(k)$ are derived as follows:

$$\begin{aligned} R'_{ij}(k) &= R'_{ij}\left(k|bb_{ij}(k)\right)\Pr\left(bb_{ij}(k)\right) + R'_{ij}\left(k|\overline{bb_{ij}}(k)\right)\Pr\left(\overline{bb_{ij}}(k)\right) \\ E\left[D'_{ij}(k)\right] &= E\left[D'_{ij}(k)|bb_{ij}(k)\right]\Pr\left(bb_{ij}(k)\right) + E\left[D'_{ij}(k)|\overline{bb_{ij}}(k)\right]\Pr\left(\overline{bb_{ij}}(k)\right) \end{aligned} \quad \text{eq(3.8)}$$

Moreover the terms $R_{ij}(k)$ and $E\left[D_{ij}(k)\right]$ are introduced such that,

$$\begin{aligned} R_{ij}(k) &= Q_{ij} R'_{ij}(k) \\ E\left[D_{ij}(k)\right] &= Q_{ij} E\left[D'_{ij}(k)\right] \end{aligned}$$

Where, $Q_{ij} = \frac{N_j}{N_i}$ eq(3.9)

Here, $R'_{ij}(k)$ is the probability that $blk_{ij}(k)$ occurs among N_j bus events on processor PE_j (the probability that each bus event b_j blocks a request r_i), and $E\left[D'_{ij}(k)\right]$ is the bus stall delay expectation against those N_j bus events. On the other hand, $R_{ij}(k)$ is the probability that $blk_{ij}(k)$ occurs among N_i bus events on processor PE_i (the probability

that each request r_i is blocked by a bus event b_j), and $E[D_{ij}(k)]$ is the bus stall delay expectation against those N_i bus events. Since $R_{ij}(k)N_i = R'_{ij}(k)N_j$, the factor Q_{ij} is introduced to convert the two probabilities observed on processors PE_j and PE_i.

3.2.4.2.1 Higher priority blocking PE case ($\alpha_{ij} = 0$)

Substituting from eq(3.5) and eq (3.6) into eq (3.8), the overall blocking probability and overall bus stall delay expectation for the $\alpha_{ij} = 0$ case is derived as,

$$
\begin{aligned}
R_{ij}(k) &= Q_{ij}\left((1-(1-\lambda_i)^{k-1})\cdot S_{ij}(k) + (1-(1-\lambda_i)^{k})\cdot \overline{S_{ij}}(k)\right) \\
&= Q_{ij}\left((1-(1-\lambda_i)^{k-1})\cdot S_{ij}\cdot f_{Bj}(k) + (1-(1-\lambda_i)^{k})\cdot (1-S_{ij})\cdot f_{Bj}(k)\right) \\
&= Q_{ij}f_{Bj}(k)\left((1-(1-\lambda_i)^{k-1})S_{ij} + (1-(1-\lambda_i)^{k})(1-S_{ij})\right) \\
&= Q_{ij}f_{Bj}(k)\left(S_{ij} - (1-\lambda_i)^{k-1}S_{ij} + (1-S_{ij}) - (1-\lambda_i)^{k}(1-S_{ij})\right) \\
&= Q_{ij}f_{Bj}(k)\left(1-(1-\lambda_i)^{k-1}\left(S_{ij} + (1-\lambda_i)(1-S_{ij})\right)\right) \\
&= Q_{ij}f_{Bj}(k)\left(1-U_{ij}(1-\lambda_i)^{k-1}\right)
\end{aligned}
$$

where

$$U_{ij} = S_{ij} + (1-\lambda_i)(1-S_{ij}) = 1-\lambda_i(1-S_{ij}) \quad (3.10)$$

And,

$$
\begin{aligned}
E[D_{ij}(k)] &= Q_{ij}\left(\frac{k\lambda_i - 1 + (1-\lambda_i)^{k}}{\lambda_i}\cdot S_{ij}(k) + \frac{(k+1)\lambda_i - 1 + (1-\lambda_i)^{k+1}}{\lambda_i}\cdot \overline{S_{ij}}(k)\right) \\
&= \frac{Q_{ij}f_{Bj}(k)}{\lambda_i}\left((k\lambda_i - 1 + (1-\lambda_i)^{k})S_{ij} + ((k+1)\lambda_i - 1 + (1-\lambda_i)^{k+1})(1-S_{ij})\right) \\
&= \frac{Q_{ij}f_{Bj}(k)}{\lambda_i}\Big((k\lambda_i - 1)S_{ij} + (1-\lambda_i)^{k}S_{ij} + ((k+1)\lambda_i - 1)(1-S_{ij}) \\
&\qquad + (1-\lambda_i)^{k+1}(1-S_{ij})\Big) \\
&= \frac{Q_{ij}f_{Bj}(k)}{\lambda_i}\Big((k\lambda_i - 1)S_{ij} + (k\lambda_i - (1-\lambda_i))(1-S_{ij}) \\
&\qquad + (1-\lambda_i)^{k}\left(S_{ij} + (1-\lambda_i)(1-S_{ij})\right)\Big)
\end{aligned}
$$

$$= \frac{Q_{ij} f_{Bj}(k)}{\lambda_i} \left(k\lambda_i \left(S_{ij} + 1 - S_{ij} \right) - \left(S_{ij} + (1-\lambda_i)\left(1 - S_{ij}\right) \right) + (1-\lambda_i)^k U_{ij} \right)$$

$$= \frac{Q_{ij} f_{Bj}(k)}{\lambda_i} \left(k\lambda_i - U_{ij}(1 - (1-\lambda_i)^k) \right)$$

3.2.4.2.2 Lower priority blocking PE case ($\alpha_{ij} = 1$)

Substituting from eq(3.5) and eq (3.7) into eq (3.8), the overall blocking probability and overall bus stall delay expectation for the $\boldsymbol{\alpha_{ij}} = 1$ case is derived as,

$$R_{ij}(k) = Q_{ij} \left((1 - (1-\lambda_i)^{k-1}) \cdot S_{ij}(k) + (1 - (1-\lambda_i)^{k-1}) \cdot \overline{S_{\iota J}}(k) \right)$$

$$= Q_{ij} \left((1 - (1-\lambda_i)^{k-1}) \cdot S_{ij} \cdot f_{Bj}(k) + (1 - (1-\lambda_i)^{k-1}) \cdot \left(1 - S_{ij}\right) \cdot f_{Bj}(k) \right)$$

$$= Q_{ij} f_{Bj}(k)(1 - (1-\lambda_i)^{k-1})$$

$$E\left[D_{ij}(k)\right] = Q_{ij} \left(\frac{k\lambda_i - 1 + (1-\lambda_i)^k}{\lambda_i} \cdot S_{ij}(k) + \frac{k\lambda_i - 1 + (1-\lambda_i)^k}{\lambda_i} \cdot \overline{S_{\iota J}}(k) \right)$$

$$= \frac{Q_{ij} f_{Bj}(k)}{\lambda_i} \left(k\lambda_i - (1 - (1-\lambda_i)^k) \right)$$

3.2.4.2.3 Overall blocking probability and stall expressions

Therefore, the expressions for overall blocking probability and stall are,

$$R_{ij}(k) = \begin{cases} Q_{ij} f_{Bj}(k)\left(1 - U_{ij}(1-\lambda_i)^{k-1}\right) & (\alpha_{ij} = 0) \\ Q_{ij} f_{Bj}(k)(1 - (1-\lambda_i)^{k-1}) & (\alpha_{ij} = 1) \end{cases} \quad \text{eq(3.11)}$$

$$E\left[D_{ij}(k)\right] = \begin{cases} \frac{Q_{ij} f_{Bj}(k)}{\lambda_i} \left(k\lambda_i - U_{ij}(1 - (1-\lambda_i)^k) \right) & (\alpha_{ij} = 0) \\ \frac{Q_{ij} f_{Bj}(k)}{\lambda_i} \left(k\lambda_i - (1 - (1-\lambda_i)^k) \right) & (\alpha_{ij} = 1) \end{cases} \quad \text{eq(3.12)}$$

3.2.4.3 Overall blocking probability and stall on all bus events

Finally, the overall blocking probability $R_{ij} = \Pr\left(blk_{ij}\right)$ and the overall bus stall delay expectation $E\left[D_{ij}\right]$ on all bus events b_j (with arbitrary length) are:

$$R_{ij} = \Pr\left(blk_{ij}\right) = \sum_k \Pr\left(blk_{ij}(k)\right) = \sum_k R_{ij}(k)$$

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

$$R_{ij} = \begin{cases} Q_{ij}(1 - U_{ij}\, y_{ij}) & (\alpha_{ij} = 0) \\ Q_{ij}(1 - y_{ij}) & (\alpha_{ij} = 1) \end{cases} \quad \text{eq(3.14)}$$

$$E[D_{ij}] = \begin{cases} Q_{ij}\left(E[B_j] - U_{ij}\dfrac{1 - v_{ij}}{\lambda_i}\right) & (\alpha_{ij} = 0) \\ Q_{ij}\left(E[B_j] - \dfrac{1 - v_{ij}}{\lambda_i}\right) & (\alpha_{ij} = 1) \end{cases} \quad \text{eq(3.15)}$$

Where, y_{ij} is called *request inactivation probability*.

$$y_{ij} = \sum_k f_{Bj}(k)(1 - \lambda_i)^{k-1} \quad \text{eq(3.16)}$$

$$v_{ij} = \sum_k f_{Bj}(k)(1 - \lambda_i)^k = (1 - \lambda_i) y_{ij} \quad \text{eq(3.17)}$$

Note that except for $U_{ij} = 1 - \lambda_i(1 - S_{ij})$ which is derived next, all variables on the right hand side of the equations eq(3.14) and eq(3.15) are derived *directly* from the bus workload statistics (N_i, h_{Li}, h_{Bi}) and (N_j, h_{Lj}, h_{Bj}).

3.2.5 Consecutive Bus Event Probability

Probability $S_{ij}(k) = \Pr\left(bb_{ij}(k)\right)$ of consecutive bus event $bb_{ij}(k)$ (bus event $b_j(k)$ immediately follows bus event b_i) observed at PE_j, is modeled by considering the below two cases:

- When $blk_{ji}(k)$ occurs (bus event $b_i(k)$ blocks a request r_j)
- When request r_j occurs immediately after bus event $b_i(k)$

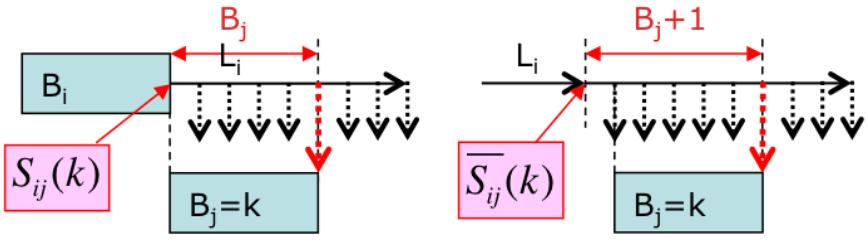

Figure 3-8: Consecutive bus-event cases

Here, event $bb_{ij}(k)$ is equivalent to an event where bus event $b_i(k)$ *with 1 extended cycle* blocks a request r_j (such bus event has $k+1$ cycle duration but occurs with probability of $f_{Bi}(k)$). Therefore, $S_{ij}(k) = \Pr\left(bb_{ij}(k)\right)$ is derived by modifying $R_{ji}(k) = \Pr\left(blk_{ji}(k)\right)$ in eq(3.11) to take account for the extended bus cycle:

$$S_{ij}(k) = \Pr\left(bb_{ij}(k)\right) = \begin{cases} Q_{ji} f_{Bi}(k)\left(1 - U_{ji}\left(1-\lambda_j\right)^k\right) & (\alpha_{ji} = 0) \\ Q_{ji} f_{Bi}(k)\left(1 - \left(1-\lambda_j\right)^k\right) & (\alpha_{ji} = 1) \end{cases}$$

$$S_{ij} = \Pr\left(bb_{ij}\right) = \begin{cases} Q_{ji}\left(1 - U_{ji} v_{ji}\right) & (\alpha_{ji} = 0) \\ Q_{ji}\left(1 - v_{ji}\right) & (\alpha_{ji} = 1) \end{cases} \quad \text{eq (3.18)}$$

Next we define the consecutive bus event Probability C_{ij} such that it is the consecutive bus event probability (b_i is immediately followed by b_j) observed at PE_i, as opposed to S_{ij} which is the same consecutive bus event probability observed at PE_j. Naturally, the expression for C_{ij} becomes

$$C_{ij} = \Pr\left(bb_{ij}\right) = \begin{cases} \left(1 - U_{ji} v_{ji}\right) & (\alpha_{ji} = 0) \\ \left(1 - v_{ji}\right) & (\alpha_{ji} = 1) \end{cases} \quad \text{eq(3.19)}$$

3.2.6 Simplified expressions for R_{ij} and $E[D_{ij}]$

The blocking probability and bus stall delay expectation can now be fully derived as below:

$$R_{ij} = \begin{cases} Q_{ij}\left(1 - v_{ij}\right) - \left(1 - v_{ji}\right)\lambda_i y_{ij} & (\alpha_{ij} = 0) \\ Q_{ij}\left(1 - y_{ij}\right) & (\alpha_{ij} = 1) \end{cases} \quad \text{eq(3.20)}$$

$$E\left[D_{ij}\right] = \begin{cases} Q_{ij}\left(E\left[B_j\right] - \dfrac{1-\lambda_i}{\lambda_i}\left(1 - v_{ij}\right)\right) - \left(1 - v_{ji}\right)\left(1 - v_{ij}\right) & \left(\alpha_{ij} = 0\right) \\ Q_{ij}\left(E\left[B_j\right] - \dfrac{1 - v_{ij}}{\lambda_i}\right) & \left(\alpha_{ij} = 1\right) \end{cases} \quad \text{eq(3.21)}$$

Let us now discuss the bus event count ratio $Q_{ij} = \frac{N_j}{N_i}$ defined on eq(2.11). 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 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

$$G_i = E[L_i] + E[B_i] + E[D_i] \quad \text{eq(3.23)}$$

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:

$$E[D_i] = \sum_{i \neq j} E\left[D_{ij}\right] \quad \text{eq(3.24)}$$

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

$$Q_{ij} = \frac{E[L_i] + E[B_i] + E[D_i]}{E\left[L_j\right] + E\left[B_j\right] + E\left[D_j\right]} \quad \text{eq(3.25)}$$

Here, $E[D_i]$ and $E\left[D_j\right]$ are the actually predicte bus stall delays that will be calculated by iterative method.

3.3 Bus Stall Delay Calculation Flow

Details of the bus stall calculation flow is described below. Here, let $PEset$ be the set of processor indices.

1. Coefficient calculations: from the new sets of bus workload statistics $(\mathrm{N_i}, \mathrm{h_{Li}}, \mathrm{h_{Bi}})$ obtained during the bus prediction period T, interval workload expectation $\mathrm{E[L_i]}$, request probability λ_i, bus workload expectation $\mathrm{E[B_i]}$. Also calculated are the following coefficients for $\forall \mathrm{i}, \mathrm{j} \in \mathrm{PEset}, \mathrm{i} \neq \mathrm{j}$

   a) $y_{ij} = \sum_k f_{Bj}(k) \cdot (1 - \lambda_i)^{k-1}$

   b) $v_{ij} = (1 - \lambda_i) y_{ij}$

   c) $DQ_{ij} = \begin{cases} E\left[B_j\right] - \dfrac{1 - \lambda_i}{\lambda_i}\left(1 - v_{ij}\right) & (\alpha_{ji} = 0) \\ E\left[B_j\right] - \dfrac{1 - v_{ij}}{\lambda_i} & (\alpha_{ji} = 1) \end{cases}$

   d) $Doffset_{ij} = \begin{cases} -\left(1 - v_{ji}\right)\left(1 - v_{ij}\right) & (\alpha_{ji} = 0) \\ 0 & (\alpha_{ji} = 1) \end{cases}$

e) $Qmax_{ij} = \begin{cases} \dfrac{1 + (1 - v_{ji})\lambda_i y_{ij}}{1 - v_{ij}} & (\alpha_{ji} = 0) \\ \dfrac{1}{1 - y_{ij}} & (\alpha_{ji} = 1) \end{cases}$

2. Initial bus stall delay values: $\forall i \in PEset, E[D_i] = 0$
3. Iterative refinement: below calculations are repeated until all bus stall delays $E[D_i]$ converges.

   a) $Q_{ij} = \dfrac{E[L_i] + E[B_i] + E[D_i]}{E[L_j] + E[B_j] + E[D_j]}$

   b) $E[D_{ij}] = \min(Q_{ij}, Qmax_{ij}) \cdot DQ_{ij} + Doffset_{ij}$

   c) $E[D_i] = \sum_{i \neq j} E[D_{ij}]$

   d) GOTO a) until convergence

Here, DQ_{ij} corresponds to the coefficient terms on Q_{ij} in eq(3.21), and $Doffset_{ij}$ is the constant term in eq(3.21). Also, the term $\min(Q_{ij}, Qmax_{ij})$ guarantees that $R_{ij} \leq 1$ in eq(3.20).

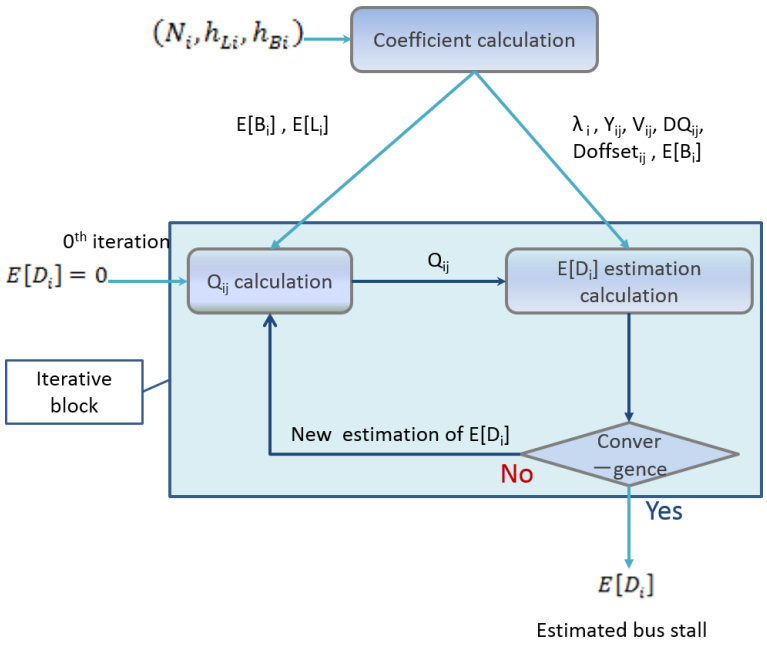

Figure 3-9: Bus stall calculation flow

4. BURST BLOCKING MODEL

Some real applications generate burst traffic and this needs to be modeled for an accurate performance estimation.

4.1 Burst Blocking behavior

First we define *burst blocking behavior* which dictates that a bus request on a Processing Element (PE) may be blocked for more than one consecutive bus workloads on another high priority PE. Hence the maximum possible bus stall for a given bus request may be a concatenation of multiple bus workloads on a single blocking PE as shown in Figure 4-1.

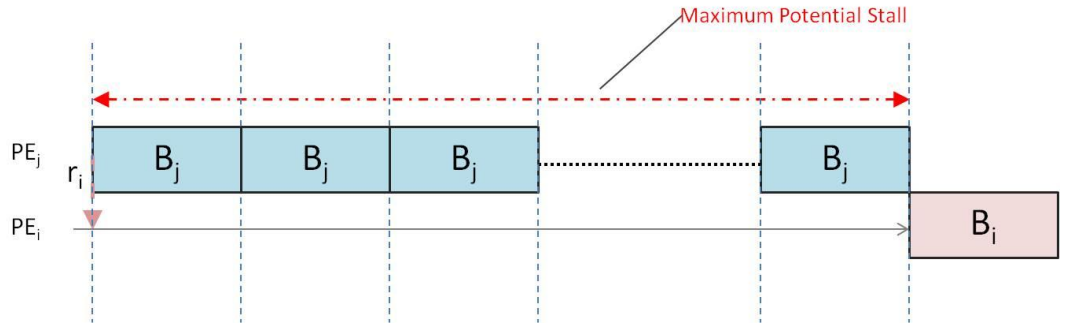

Figure 4-1: Potential stall for BBM

Such a behavior is realistic if,

1. There are only two PE in a system.
2. All processing elements can generate burst traffic i.e. a PE can issue a bus request zero cycles after the end of its last bus workload. This event is called a *zero interval* event.

The *zero interval* event results in different outcomes depending on the probability relation of the two PEs. First a zero interval bus request on a high priority PE can again block an already blocked Lower priority PE. While on a lower priority PE, the occurrence of a zero interval bus request does not block a bus request on the higher priority PE as it loses arbitration due to priority. Figure 4-2 highlights these two possibilities.

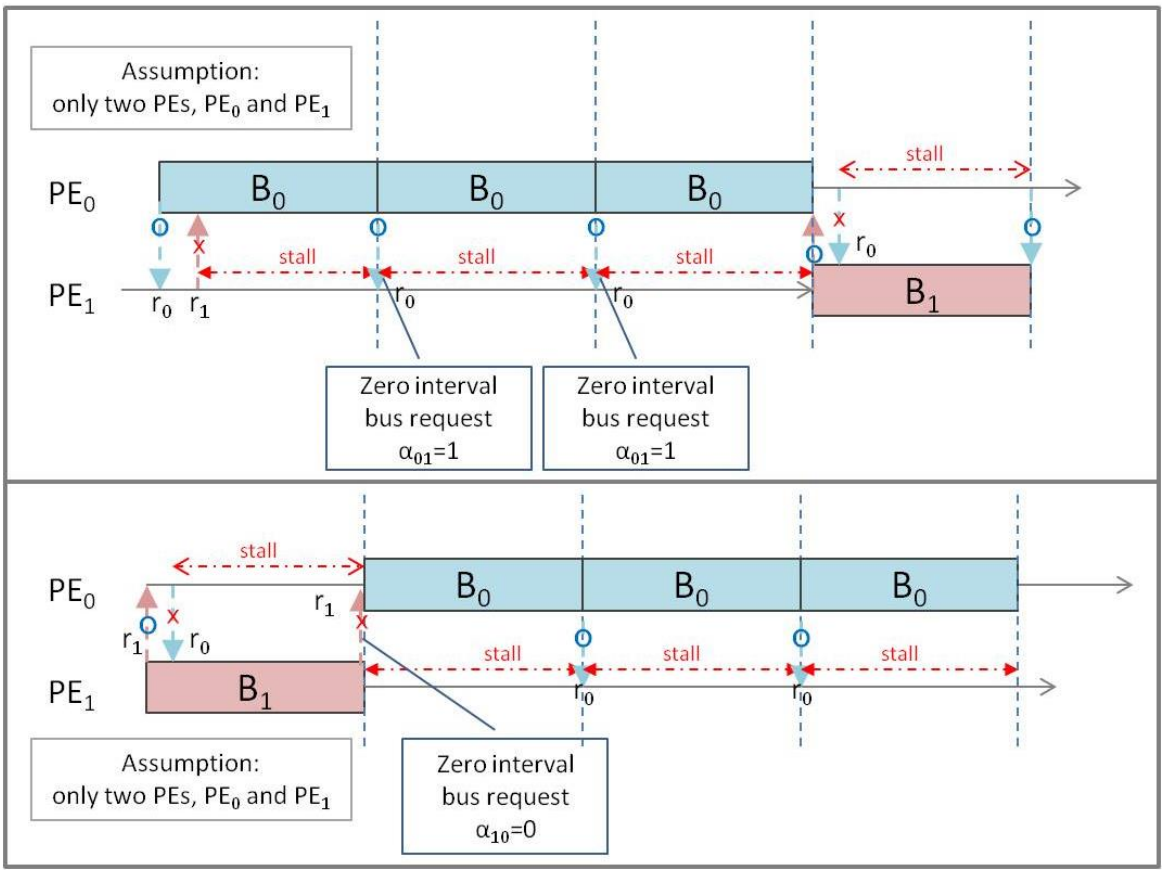

Figure 4-2: Typical BBM system

To model burst traffic, we first start with the same two processor case and introduce the *zero interval probability* μ_i to augment the non-zero interval request model.

- After the termination of a bus workload, a request is *immediately* generated with probability μ_i, in which case the interval workload becomes *zero cycles*.
- With a probability of $1 - \mu_i$, non-zero interval request process (with probability λ_i) starts.

4.2 Zero interval probability

Note that, since the zero interval workload can be recorded from the workload statistics, a separate probability μ_i is used. Zero interval probability μ_i is obtained from the collected statistics during the bus prediction interval (N_i, h_{Li}, h_{Bi}) as follows:

$$\mu_i = \Pr(L_i = 0) = \frac{h_{Li}(0)}{\sum_m h_{Li}(m)} = \frac{h_{Li}(0)}{N_i} \quad \text{eq(4.1)}$$

Then the probability of $L_i = n$ is:

$$\Pr(L_i = n) = \begin{cases} \mu_i & (n = 0) \\ (1-\mu_i)\lambda_i(1-\lambda_i)^{n-1} & (n \geq 1) \end{cases} \quad \text{eq(4.2)}$$

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

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

$$\therefore \lambda_i = \frac{1-\mu_i}{E[L_i]} \quad \text{eq(4.3)}$$

4.3 Merged bus workload

On the processor with higher priority, the occurrence of a zero interval workload in effect merges the two successive bus workloads, that is, for the processor with lower priority, it sees a *merged* bus workload that potentially blocks its request. Note that zero interval workloads can in fact occur consecutively as well. Let $z_j(m)$ be an event where m consecutive zero interval workloads occur on PE_j. Probability of $z_j(m)$ is given as:

$$\Pr(z_j(m)) = (1-\mu_j)\mu_j^m \quad \text{eq(4.4)}$$

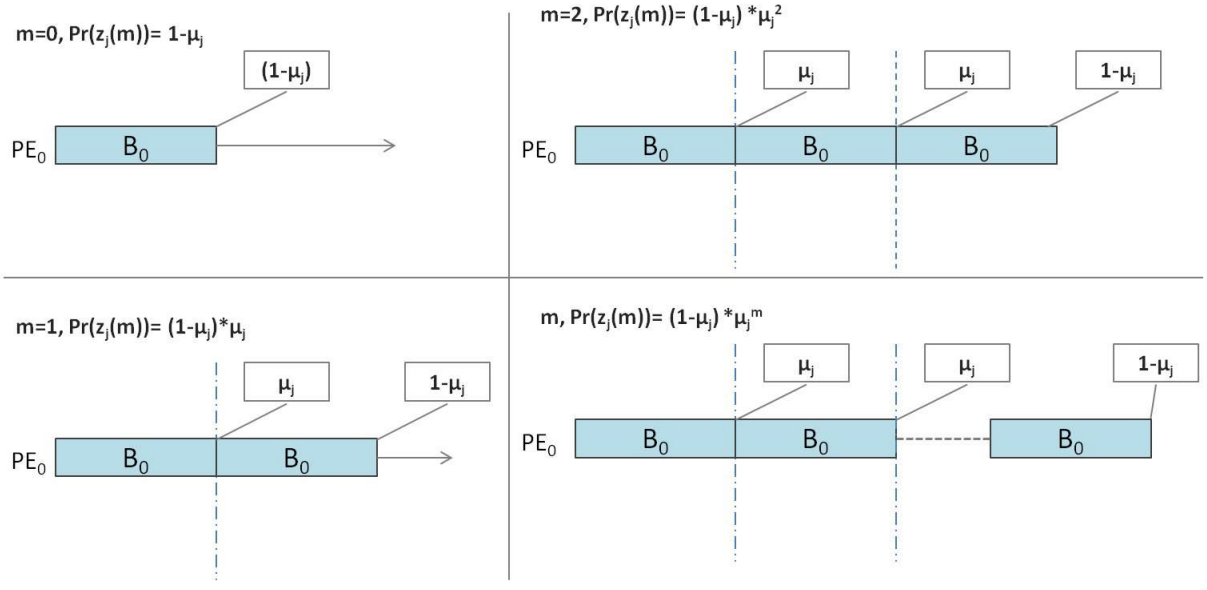

Figure 4-3: Workload merging and corresponding probabilities

4.3.1 Probability mass function of merged bus workload

Assuming that bus workloads and interval workloads are totally uncorrelated, let $f_{Bj}(m,k)$ be the probability mass function of the merged bus workloads at PE_j by m consecutive zero interval workloads ($m \geq 1$), which can be derived recursively by the convolution of probability mass functions where $f_{Bj}(0,k) = f_{Bj}(k)$:

$$f_{Bj}(m,k) = \begin{cases} f_{Bj}(k) & (m=0) \\ \sum_n f_{Bj}(m-1,n) f_{Bj}(k-n) & (m \geq 1) \end{cases} \quad \text{eq(4.5)}$$

4.3.2 Overall probability distribution and expectation of merged bus workload

Let's symbolize the overall probability distribution of merged bus workloads as $f_{Bj}^*(k)$ and expectation of the merged bus workloads as $E[B_j^*]$. Note that $f_{Bj}(m,k)$ is equivalent to the probability distribution of adding $m+1$ randomly chosen bus workloads that have the same probability distribution $f_{Bj}(k)$. Among N_j bus workloads, $(1-\mu_j)N_j$ bus workloads are distributed according to $f_{Bj}(0,k)$, $(1-\mu_j)\mu_j N_j$ bus workloads are distributed according to $f_{Bj}(1,k)$, $(1-\mu_j)\mu_j^2 N_j$ bus workloads are distributed according to $f_{Bj}(2,k)$, and so on. Therefore, the overall probability distribution of the merged bus workloads becomes:

$$f_{Bj}^*(k) = \sum_{m=0}^{\infty} (1-\mu_j)\mu_j^m f_{Bj}(m,k) \quad \text{eq(4.6)}$$

Also, the expectation of the merged bus workloads is derived (see AppendixA for detailed derivation) to be:

$$E[B_j^*] = \sum_k f_{Bj}^*(k) \cdot k = \frac{E[B_j]}{1-\mu_j} \quad \text{eq(4.7)}$$

Here, this extended bus workload distribution applies only to the case $\alpha_{ij} = 0$:

$$\begin{cases} S_{ij}(k) = S_{ij} \cdot f_{Bj}^*(k), & \overline{S_{ij}}(k) = (1-S_{ij}) \cdot f_{Bj}^*(k) \quad (\alpha_{ij}=0) \\ S_{ij}(k) = S_{ij} \cdot f_{Bj}(k), & \overline{S_{ij}}(k) = (1-S_{ij}) \cdot f_{Bj}(k) \quad (\alpha_{ij}=1) \end{cases}$$

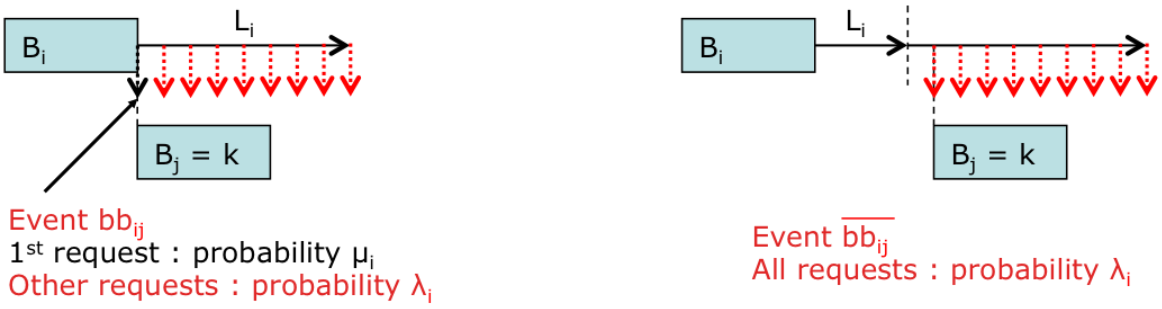

Figure 4-4: Request probabilities on event $bb_{ij}(k)$ and $\overline{bb_{ij}}(k)$

Figure 4-4 illustrates event $bb_{ij}(k)$ and $\overline{bb_{ij}}(k)$ with zero interval requests. Zero interval requests can happen only on $bb_{ij}(k)$, if $\alpha_{ij} = 0$. For $\alpha_{ij} = 1$ the following bus event $b_j(k)$ will be blocked ($bb_{ij}(k)$ cannot occur). Moreover, zero interval requests do not occur on $\overline{bb_{ij}}(k)$ as naturally dictated by the $\overline{bb_{ij}}(k)$ event.

4.4 Bus stall cycle expectation

Here, the probability of $t_{ij}(k) = n$ $(n = 0,1,2, \dots, k-1)$ on event $bb_{ij}(k)$ and $\overline{bb_{ij}}(k)$ are given as:

$$\Pr\left(t_{ij}(k) = n | bb_{ij}(k)\right) = \begin{cases} \mu_i & (n = 0) \\ (1-\mu_i)\lambda_i(1-\lambda_i)^{n-1} & (n \geq 1) \end{cases} \quad (\alpha_{ij} = 0)$$

$$\begin{cases} 0 & (n = 0) \\ \lambda_i(1-\lambda_i)^{n-1} & (n \geq 1) \end{cases} \quad (\alpha_{ij} = 1)$$

$$\Pr\left(t_{ij}(k) = n | \overline{bb_{ij}}(k)\right) = \begin{cases} \lambda_i(1-\lambda_i)^{n} & (\alpha_{ij} = 0) \\ \lambda_i(1-\lambda_i)^{n-1} & (\alpha_{ij} = 1) \end{cases}$$

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

4.4.1 Higher priority blocking PE case $\alpha_{ij} = 0$

First the expressions for the $\alpha_{ij} = 0$ case are derived using eq3.8 as derived for the SBM and using new expressions for $R'_{ij}\left(k | bb_{ij}(k)\right)$, $R'_{ij}\left(k | \overline{bb_{ij}}(k)\right)$, $E\left[D'_{ij}(k) | bb_{ij}(k)\right]$ and $E\left[D'_{ij}(k) | \overline{bb_{ij}}(k)\right]$. These new expressions are based on the $\Pr\left(t_{ij}(k)\right)$ expressions derived above which reflect the affect of burst blocking behavior.

4.4.1.1 Blocking probability

$$R'_{ij}\left(k|bb_{ij}(k)\right) = \mu_i + (1-\mu_i)(1-(1-\lambda_i)^{k-1}) = 1-(1-\mu_i)(1-\lambda_i)^{k-1}$$

$$R'_{ij}\left(k|\overline{bb_{ij}}(k)\right) = 1-(1-\lambda_i)^{k}$$

Substituting above values of $R'_{ij}\left(k|bb_{ij}(k)\right)$ and $R'_{ij}\left(k|\overline{bb_{ij}}(k)\right)$ in eq(3.8)

$$R'_{ij}(k) = (1-(1-\mu_i)(1-\lambda_i)^{k-1})\cdot S_{ij}\cdot f^*_{Bj}(k) + (1-(1-\lambda_i)^{k})\cdot\left(1-S_{ij}\right)\cdot f^*_{Bj}(k)$$

$$= f^*_{Bj}(k)\left((1-(1-\mu_i)(1-\lambda_i)^{k-1})S_{ij} + (1-(1-\lambda_i)^{k})\left(1-S_{ij}\right)\right)$$

$$= f^*_{Bj}(k)\left(1-(1-\mu_i)(1-\lambda_i)^{k-1}S_{ij} - (1-\lambda_i)^{k}\left(1-S_{ij}\right)\right)$$

$$= f^*_{Bj}(k)\left(1-\left((1-\mu_i)S_{ij} + (1-\lambda_i)\left(1-S_{ij}\right)\right)(1-\lambda_i)^{k-1}\right)$$

Therefore,

$$R'_{ij}(k) = f^*_{Bj}(k)\left(1-U^*_{ij}(1-\lambda_i)^{k-1}\right)$$

Where, $U^*_{ij} = (1-\mu_i)S_{ij} + (1-\lambda_i)\left(1-S_{ij}\right)$

4.4.1.2 Bus stall

$$E\left[D'_{ij}(k)|bb_{ij}(k)\right] = \mu_i k + (1-\mu_i)\frac{k\lambda_i - 1 + (1-\lambda_i)^k}{\lambda_i}$$

$$= \frac{\mu_i k\lambda_i + (1-\mu_i)(k\lambda_i - 1) + (1-\mu_i)(1-\lambda_i)^k}{\lambda_i}$$

$$= \frac{k\lambda_i - (1-\mu_i) + (1-\mu_i)(1-\lambda_i)^k}{\lambda_i}$$

$$= \frac{k\lambda_i - (1-\mu_i)(1-(1-\lambda_i)^k)}{\lambda_i}$$

$$E\left[D'_{ij}(k)|\overline{bb_{ij}}(k)\right] = \frac{(k+1)\lambda_i - 1 + (1-\lambda_i)^{k+1}}{\lambda_i}$$

Substituting above values of $E\left[D'_{ij}(k)|bb_{ij}(k)\right]$ and$E\left[D'_{ij}(k)|bb_{ij}(k)\right]$ in eq3.8,

$$E\left[D'_{ij}(k)\right] = \frac{k\lambda_i - (1-\mu_i)(1-(1-\lambda_i)^k)}{\lambda_i}S_{ij}f^*_{Bj}(k)$$

$$+ \frac{(k+1)\lambda_i - 1 + (1-\lambda_i)^{k+1}}{\lambda_i}\left(1-S_{ij}\right)f^*_{Bj}(k)$$

$$= \frac{f^*_{Bj}(k)}{\lambda_i}\Big(\Big(k\lambda_i - (1-\mu_i)(1-(1-\lambda_i)^k)\Big)S_{ij} + ((k+1)\lambda_i - 1 + (1-\lambda_i)^{k+1})\big(1-S_{ij}\big)\Big)$$

$$= \frac{f^*_{Bj}(k)}{\lambda_i}\Big((k\lambda_i - (1-\mu_i) + (1-\mu_i)(1-\lambda_i)^k)S_{ij} + ((k\lambda_i - (1-\lambda_i) + (1-\lambda_i)^{k+1})\big(1-S_{ij}\big)\Big)$$

$$= \frac{f^*_{Bj}(k)}{\lambda_i}\Big(k\lambda_i - (1-\mu_i)S_{ij} - (1-\lambda_i)\big(1-S_{ij}\big) + (1-\lambda_i)^k\Big((1-\mu_i)S_{ij} + (1-\lambda_i)\big(1-S_{ij}\big)\Big)\Big)$$

$$= \frac{f^*_{Bj}(k)}{\lambda_i}\Big(k\lambda_i - \Big((1-\mu_i)S_{ij} + (1-\lambda_i)\big(1-S_{ij}\big)\Big)\big(1-(1-\lambda_i)^k\big)\Big)$$

Therefore,

$$E\big[D'_{ij}(k)\big] = \frac{f^*_{Bj}(k)}{\lambda_i}\Big(k\lambda_i - U^*_{ij}(1-(1-\lambda_i)^k)\Big)$$

4.4.2 Lower priority blocking PE case $\alpha_{ij} = 1$

For the lower priority PE case i.e. when $\alpha_{ij} = 1$ the occurrence of a zero interval bus request does not block the higher priority PE, hence the same expressions for $E\big[D'_{ij}(k)\big]$ and $R'_{ij}(k)$ can be used as derived in eq(3.5) and eq(3.7)

4.4.3 Bus event count ratio Q_{ij}

When $\alpha_{ij} = 0$, since PE_i will only see $(1-\mu_j)N_j$ merged bus workloads, the bus event count ratio Q_{ij} must reflect this and is rewritten as

$$Q^*_{ij} = Q_{ij}\big(1-\mu_j\big) \quad (\alpha_{ij} = 0)$$

4.4.4 Overall blocking probability and stall on all bus events

Overall blocking probability is derived using the new expressions for $\alpha_{ij} = 0$ case

$$R_{ij}(k) = \begin{cases} Q^*_{ij} f^*_{Bj}(k)\big(1-U^*_{ij}(1-\lambda_i)^{k-1}\big) & (\alpha_{ij}=0) \\ Q_{ij} f_{Bj}(k)(1-(1-\lambda_i)^{k-1}) & (\alpha_{ij}=1) \end{cases}$$

$$E\left[D_{ij}(k)\right]=\begin{cases}\dfrac{Q_{ij}^{*}f_{Bj}^{*}(k)}{\lambda_i}\left(k\lambda_i-U_{ij}^{*}(1-(1-\lambda_i)^k)\right) & (\alpha_{ij}=0)\\ \dfrac{Q_{ij}f_{Bj}(k)}{\lambda_i}\left(k\lambda_i-(1-(1-\lambda_i)^k)\right) & (\alpha_{ij}=1)\end{cases}$$

Finally, the overall blocking probability R_{ij} and the overall bus stall delay expectation $E\left[D_{ij}\right]$ on all bus events b_j (with arbitrary length) are:

$$R_{ij}=\sum_k R_{ij}(k)$$

$$R_{ij}=\begin{cases}Q_{ij}^{*}\left(1-U_{ij}^{*}Y_{ij}\right) & (\alpha_{ij}=0)\\ Q_{ij}\left(1-y_{ij}\right) & (\alpha_{ij}=1)\end{cases} \quad \text{eq(4.8)}$$

And

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

$$E\left[D_{ij}\right]=\begin{cases}Q_{ij}\left(E\left[B_j\right]-U_{ij}^{*}\dfrac{1-\mu_j}{1-\mu_i}E[L_i]\left(1-V_{ij}\right)\right) & (\alpha_{ij}=0)\\ Q_{ij}\left(E\left[B_j\right]-E[L_i]\dfrac{1-v_{ij}}{1-\mu_i}\right) & (\alpha_{ij}=1)\end{cases} \quad \text{eq(4.9)}$$

where

$$Y_{ij}=\sum_k f_{Bj}^{*}(k)(1-\lambda_i)^{k-1}$$

$$V_{ij}=\sum_k f_{Bj}^{*}(k)(1-\lambda_i)^{k}=(1-\lambda_i)Y_{ij}$$

4.5 Request inactivation probability

The request inactivation probability, Y_{ij} is in fact a power series of $\mu_j(1-\lambda_i)y_{ij}$ (see AppendixB for detailed derivation):

$$Y_{ij}=\sum_k f_{Bj}^{*}(k)(1-\lambda_i)^{k-1}=\sum_k\left(\sum_{m=0}^{\infty}\left(1-\mu_j\right)\mu_j^m f_{Bj}(m,k)\right)(1-\lambda_i)^{k-1}$$

$$=\left(1-\mu_j\right)\sum_{m=0}^{\infty}\mu_j^m\left(\sum_k f_{Bj}(m,k)(1-\lambda_i)^{k-1}\right)=\left(1-\mu_j\right)\sum_{m=0}^{\infty}\mu_j^m y_{ij}(m)$$

$$= \frac{(1-\mu_j)y_{ij}}{1-\mu_j(1-\lambda_i)y_{ij}}$$

Similarly,

$$V_{ij} = (1-\lambda_i)Y_{ij} = \frac{(1-\lambda_i)(1-\mu_j)y_{ij}}{1-\mu_j(1-\lambda_i)y_{ij}}$$

4.6 Consecutive Bus Event Probability

Probability $S_{ij}(k) = \Pr\left(bb_{ij}(k)\right)$ of consecutive bus event $bb_{ij}(k)$ (bus event $b_j(k)$ immediately follows bus event b_i) observed at PE_j, is updated accordingly reflecting the new value of R_{ij} for $\alpha_{ij} = 0$ case:

$$S_{ij} = \Pr(bb_{ij}) = \begin{cases} Q_{ji}^*(1-U_{ji}^*V_{ji}) & (\alpha_{ji}=0) \\ Q_{ji}(1-v_{ji}) & (\alpha_{ji}=1) \end{cases} \quad \text{eq(4.10)}$$

5. MULTI BLOCKING MODEL

In real applications with multiple bus masters, a bus request might incur more stall than modeled by the SBM and BBM. First we define *multi blocking behavior* which dictates that a bus request on a Processing Element (PE) may be blocked for more than one consecutive bus workloads on multiple other PEs. Hence the maximum possible bus stall for a given bus request may be a concatenation of multiple bus workloads on multiple high priority PEs as shown in Figure 5-1.

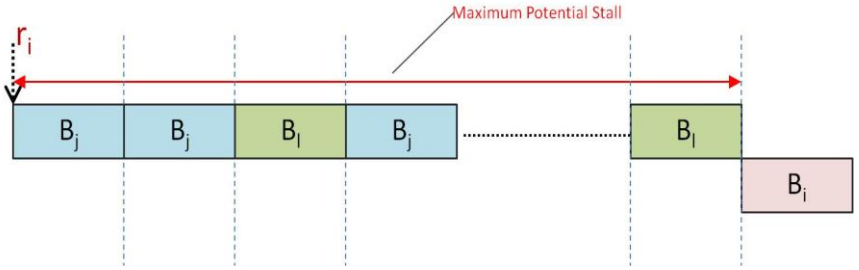

Figure 5-1: Potential stall for MBM

The Multi Blocking Model (MBM) captures multi blocking behavior. A single bus request on PE_i maybe 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 5-2 shows a typical three PE system where the lowest priority PE can be blocked by burst traffic on all high priority PE (event bb_{jj}), as well as the back to back occurrences of bus events on both higher priority PE (event bb_{jl}), where PE_j and PE_l are both higher priority PE.

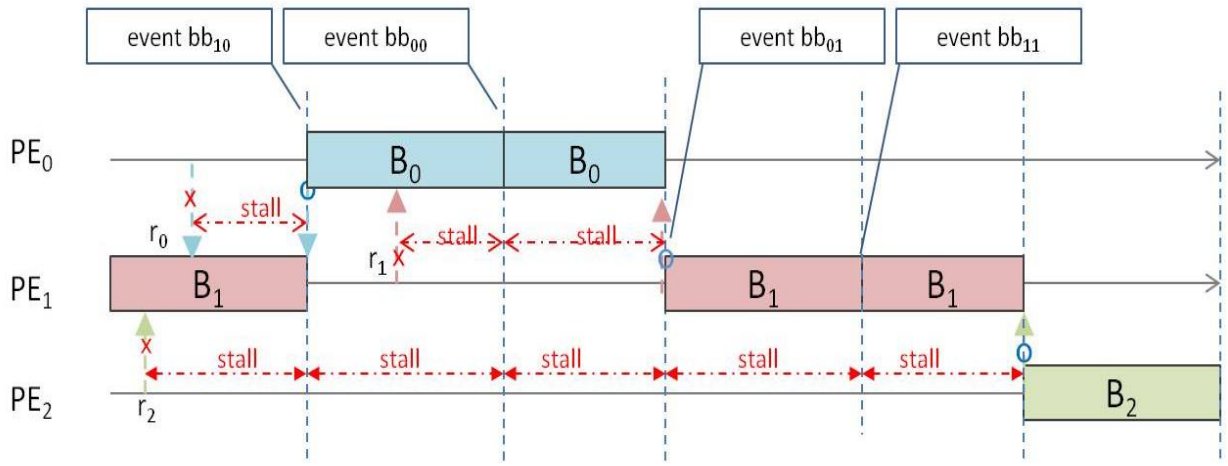

Figure 5-2: Typical MBM system

First we define some basic terms.

5.1 Key Terms

5.1.1 Effective bus workload

When a bus request on PE_i is blocked due to a bus workload on PE_j (such that priority of PE_j is higher than priority of PE_i), the occurrence of a zero interval workload on PE_j or a bus workload on PE_k (such that priority of PE_k is also higher than priority of PE_i) can block PE_i continuously for multiple bus workloads. This kind of blocking can happen in different combinations on all high priority PEs. Effectively, From the perspective of PE_i, 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 PE_j will be different for each PE_i as opposed to the *merged bus workload* defined for BBM which is the same for any lower priority PE. Hence it is denoted as B_{ij}i.e. the effective bus workload on PE_j from the perspective of PE_i. 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 r_i does not occur within the duration of the effective bus workload B_{ij} on PE_i is called effective *request inactivation probability* Y_{ij}.

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 Definitions

As noted above, the effective bus workload on each PE_j will be different for each observer PE_i. Lets first derive the equations for PE_i such that there are two higher priority PEs PE_j and PE_l.

Below defines some terms that will be used in the derivation.

C_{jl} : Probability of event bb_{lj}, i.e. b_l immediately follows b_j (observed at PE_j)

$1 - (C_{jj} + C_{jl})$: probability that neither b_j or b_l immediately follows b_j.

$f_{jl}(m,k)$: "scaled" probability mass function of $m+1$ merged bus events (b_j or b_l) starting with b_j and terminating with b_l

$f_{jl}^*(k)$: Probability mass function of *all* merged bus workloads (b_j or b_l) starting with b_j and terminating with b_l .

$F_{jl}(m)$: Scaling factor of $f_{jl}(m,k)$

$E[B_{jl}(m)]$: Expectation of $m+1$ merged bus events (b_j or b_l) starting with b_j and terminating with b_l , such that PE_i is lower priority than PE_j and PE_l.

$E[B_{jl}^*]$: Expectation of *all* merged bus workloads (b_j or b_l) starting with b_j and terminating with b_l , such that PE_i is lower priority than PE_j and PE_l.

Y_{ij}: *Request inactivation probability* of request r_i withprobability λ_i on the *effective* bus workload B_{ij}

Y_{ijl}: Partial *Request inactivation probability* of request r_i with probability λ_i on the merged bus workload starting with b_j and terminating with b_l.

5.2.2 Expression for $\mathbf{f_{jl}(m,k)}$

Assuming two higher priority PEs PE_0 and PE_1, first expressions for $f_{jl}(m,k)$ are derived.

Since:

$$f_{B0}(k) \text{ is divided as: } \begin{cases} \big(1-(C_{00}+C_{01})\big)f_{B0}(k) & \text{(no merging with following } b_0 \text{ or } b_1) \\ C_{00}f_{B0}(k) & \text{(merging with following } b_0) \\ C_{01}f_{B0}(k) & \text{(merging with following } b_1) \end{cases}$$

$$f_{B1}(k) \text{ is divided as: } \begin{cases} \big(1-(C_{10}+C_{11})\big)f_{B1}(k) & \text{(no merging with following } b_0 \text{ or } b_1) \\ C_{10}f_{B1}(k) & \text{(merging with following } b_0) \\ C_{11}f_{B1}(k) & \text{(merging with following } b_1) \end{cases}$$

Therefore:

$$\begin{bmatrix} f_{00}(0,k) & f_{10}(0,k) \\ f_{01}(0,k) & f_{11}(0,k) \end{bmatrix} = \begin{bmatrix} f_{B0}(k) & 0 \\ 0 & f_{B1}(k) \end{bmatrix}$$

$$\begin{bmatrix} f_{00}(1,k) & f_{10}(1,k) \\ f_{01}(1,k) & f_{11}(1,k) \end{bmatrix} = \sum_n \begin{bmatrix} f_{B0}(n) & 0 \\ 0 & f_{B1}(n) \end{bmatrix} [C]_2 \begin{bmatrix} f_{00}(0,k-n) & f_{10}(0,k-n) \\ f_{01}(0,k-n) & f_{11}(0,k-n) \end{bmatrix}$$

$$\begin{bmatrix} f_{00}(m,k) & f_{10}(m,k) \\ f_{01}(m,k) & f_{11}(m,k) \end{bmatrix}$$

$$= \sum_n \begin{bmatrix} f_{B0}(n) & 0 \\ 0 & f_{B1}(n) \end{bmatrix} [C]_2 \begin{bmatrix} f_{00}(m-1,k-n) & f_{10}(m-1,k-n) \\ f_{01}(m-1,k-n) & f_{11}(m-1,k-n) \end{bmatrix} \text{ eq(5.1)}$$

where

$$[C]_2 = \begin{bmatrix} C_{00} & C_{10} \\ C_{01} & C_{11} \end{bmatrix}$$

5.2.3 Probability mass function of all merged bus workloads

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

$$\begin{bmatrix} f^*_{00}(k) & f^*_{10}(k) \\ f^*_{01}(k) & f^*_{11}(k) \end{bmatrix} = [H]_2 \sum_{m=0}^{\infty} \begin{bmatrix} f_{00}(m,k) & f_{10}(m,k) \\ f_{01}(m,k) & f_{11}(m,k) \end{bmatrix} \quad eq(5.2)$$

where

$$[H]_2 = \begin{bmatrix} 1-(C_{00}+C_{01}) & 0 \\ 0 & 1-(C_{10}+C_{11}) \end{bmatrix}$$

5.2.4 Scaling factor $\mathbf{F_{jl}(m)}$

The scaling factor of $f_{jl}(m,k)$ represented by $F_{jl}(m)$ is derived as:

$$\begin{bmatrix} F_{00}(m) & F_{10}(m) \\ F_{01}(m) & F_{11}(m) \end{bmatrix} = \sum_k \begin{bmatrix} f_{00}(m,k) & f_{10}(m,k) \\ f_{01}(m,k) & f_{11}(m,k) \end{bmatrix}$$

$$= \sum_k \sum_n \begin{bmatrix} f_{B0}(n) & 0 \\ 0 & f_{B1}(n) \end{bmatrix} [C]_2 \begin{bmatrix} f_{00}(m-1,k-n) & f_{10}(m-1,k-n) \\ f_{01}(m-1,k-n) & f_{11}(m-1,k-n) \end{bmatrix}$$

$$= \sum_n \begin{bmatrix} f_{B0}(n) & 0 \\ 0 & f_{B1}(n) \end{bmatrix} [C]_2 \sum_k \begin{bmatrix} f_{00}(m-1,k-n) & f_{10}(m-1,k-n) \\ f_{01}(m-1,k-n) & f_{11}(m-1,k-n) \end{bmatrix}$$

$$= [C]_2 \begin{bmatrix} F_{00}(m-1) & F_{01}(m-1) \\ F_{10}(m-1) & F_{11}(m-1) \end{bmatrix} = ([C]_2)^m \begin{bmatrix} F_{00}(0) & F_{10}(0) \\ F_{01}(0) & F_{11}(0) \end{bmatrix} = ([C]_2)^m$$

Where,

$$\begin{bmatrix} F_{00}(0) & F_{10}(0) \\ F_{01}(0) & F_{11}(0) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Hence:

$$\begin{bmatrix} F_{00}(m) & F_{10}(m) \\ F_{01}(m) & F_{11}(m) \end{bmatrix} = ([C]_2)^m \quad eq(5.3)$$

5.2.5 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}.

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

5.2.5.1 Expectation of m + 1 merged bus events

Expectation of $m+1$ merged bus events is calculated as:

$$\begin{bmatrix} E[B_{00}(m)] & E[B_{10}(m)] \\ E[B_{01}(m)] & E[B_{11}(m)] \end{bmatrix} = \sum_k k \begin{bmatrix} f_{00}(m,k) & f_{10}(m,k) \\ f_{01}(m,k) & f_{11}(m,k) \end{bmatrix}$$

$$= [C]_2 \begin{bmatrix} E[B_{00}(m-1)] & E[B_{10}(m-1)] \\ E[B_{01}(m-1)] & E[B_{11}(m-1)] \end{bmatrix} + \begin{bmatrix} E[B_0] & 0 \\ 0 & E[B_1] \end{bmatrix} ([C]_2)^m$$

See AppendixC

5.2.5.2 Expectation of all merged bus workloads

Expectation of *all* merged bus workloads is derived while using eq(5.2) as,

$$\begin{bmatrix} E[B_{00}^*] & E[B_{10}^*] \\ E[B_{01}^*] & E[B_{11}^*] \end{bmatrix} = \sum_k k \begin{bmatrix} f_{00}^*(k) & f_{10}^*(k) \\ f_{01}^*(k) & f_{11}^*(k) \end{bmatrix} = H_2 \sum_k k \sum_{m=0}^{\infty} \begin{bmatrix} f_{00}(m,k) & f_{10}(m,k) \\ f_{01}(m,k) & f_{11}(m,k) \end{bmatrix}$$

$$= H_2 \sum_{m=0}^{\infty} \sum_k k \begin{bmatrix} f_{00}(m,k) & f_{10}(m,k) \\ f_{01}(m,k) & f_{11}(m,k) \end{bmatrix} = [H]_2 \sum_{m=0}^{\infty} \begin{bmatrix} E[B_{00}(m)] & E[B_{10}(m)] \\ E[B_{01}(m)] & E[B_{11}(m)] \end{bmatrix}$$

$$= [H]_2 \sum_{m=0}^{\infty} [C]_2 \begin{bmatrix} E[B_{00}(m-1)] & E[B_{10}(m-1)] \\ E[B_{01}(m-1)] & E[B_{11}(m-1)] \end{bmatrix}$$

$$+ \begin{bmatrix} E[B_0] & 0 \\ 0 & E[B_1] \end{bmatrix} ([C]_2)^m$$

$$= [H]_2 (I - C_2)^{-1} \begin{bmatrix} E[B_0] & 0 \\ 0 & E[B_1] \end{bmatrix} (I - [C]_2)^{-1}$$

See AppendixD.

$$\begin{bmatrix} E[B_{00}^*] & E[B_{10}^*] \\ E[B_{01}^*] & E[B_{11}^*] \end{bmatrix} = [H]_2 (I - C_2)^{-1} \begin{bmatrix} E[B_0] & 0 \\ 0 & E[B_1] \end{bmatrix} (I - [C]_2)^{-1}$$

Where 5.2.7 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 $E[B_{jl}^*]$ for all $l < i$,

$E[B_{ij}] = E[B_{jj}^*] + E[B_{jl}^*]$ therefore,

$$[E[B_{20}] \quad E[B_{21}]] = [1 \quad 1][H]_2 (I - C_2)^{-1} \begin{bmatrix} E[B_0] & 0 \\ 0 & E[B_1] \end{bmatrix} (I - [C]_2)^{-1}$$

$$= [1 \quad 1] \begin{bmatrix} E[B_0] & 0 \\ 0 & E[B_1] \end{bmatrix} (I - [C]_2)^{-1}$$

Note that in the above calculation infinite bus event count 'm' is assumed although the actual number of bus events is not infinite. The actual number of bus events on each PE_j observed during the bus predication interval is given as N_j. However we need to take into account the fact that these bus event counts resulted while the bus stall delays were ignored, and therefore this will lead to inaccuracy if used directly. As shown in *section 3.2.6* the actual value of bus workloads N_j and $N_i$are incorporated in the calculation of

$E[D_{ij}]$ by using the ratio $Q_{ij} = \frac{N_j}{N_i}$ which is rewritten as $Q_{ij} = \frac{G_i}{G_j}$. The *average bus access interval* G_i incorporates the effect of bus stalls and is calculated as

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

Moreover as shown in 5.2.7, 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_j nor b_l immediately follows b_j calculated as $1 - (C_{jj} + C_{jl})$ 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 6.6.2.

5.2.5.3 Calculation of $[\mathbf{1} \quad \mathbf{1}][\mathbf{H}]_\mathbf{2}(\mathbf{I} - [\mathbf{C}]_\mathbf{2})^{-\mathbf{1}}$

Let $a = 1 - C_{00}$, $b = -C_{10}$, $c = -C_{01}$, $d = 1 - C_{11}$ then $(I - [C]_2)^{-1} = \frac{1}{ad-bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$, $[H]_2 = \begin{bmatrix} a+c & 0 \\ 0 & b+d \end{bmatrix}$

Then the product$[1 \quad 1][H]_2(I - [C]_2)^{-1}$ becomes:

$$= [1 \quad 1]\frac{1}{ad-bc}\begin{bmatrix} d(a+c) & -b(a+c) \\ -c(b+d) & a(b+d) \end{bmatrix}$$

$$= \frac{1}{ad-bc}[(a+c)d - (b+d)c \quad -(a+c)b + (b+d)a]$$

$$= \frac{1}{ad-bc}[ad - bc \quad ad - bc] = [1 \quad 1]$$

5.2.6 Effective request inactivation probability

First the partial *Request inactivation probability* is calculated as,

$$\begin{bmatrix} Y_{200} & Y_{210} \\ Y_{201} & Y_{211} \end{bmatrix} = \sum_k \begin{bmatrix} f_{00}^*(k) & f_{10}^*(k) \\ f_{01}^*(k) & f_{11}^*(k) \end{bmatrix} (1 - \lambda_2)^{k-1}$$

$$= [H]_2 \sum_k \sum_{m=0}^{\infty} \begin{bmatrix} f_{00}(m,k) & f_{10}(m,k) \\ f_{01}(m,k) & f_{11}(m,k) \end{bmatrix} (1 - \lambda_2)^{k-1} = [H]_2 \sum_{m=0}^{\infty} \begin{bmatrix} Y_{200}(m) & Y_{210}(m) \\ Y_{201}(m) & Y_{211}(m) \end{bmatrix}$$

Where,

$$\begin{bmatrix} Y_{200}(m) & Y_{210}(m) \\ Y_{201}(m) & Y_{211}(m) \end{bmatrix} = \sum_k \begin{bmatrix} f_{00}(m,k) & f_{10}(m,k) \\ f_{01}(m,k) & f_{11}(m,k) \end{bmatrix} (1 - \lambda_2)^{k-1}$$

$$= ([V]_2[C]_2)^m \begin{bmatrix} Y_{200}(0) & Y_{210}(0) \\ Y_{201}(0) & Y_{211}(0) \end{bmatrix} = ([V]_2[C]_2)^m \begin{bmatrix} y_{20} & 0 \\ 0 & y_{21} \end{bmatrix}$$

Such that, $[V]_2 = \begin{bmatrix} v_{20} & 0 \\ 0 & v_{21} \end{bmatrix}$,

Hence:

$$\begin{bmatrix} Y_{200} & Y_{210} \\ Y_{201} & Y_{211} \end{bmatrix} = [H]_2 \sum_{m=0}^{\infty} ([V]_2[C]_2)^m \begin{bmatrix} y_{20} & 0 \\ 0 & y_{21} \end{bmatrix} = [H]_2 (I - [V]_2[C]_2)^{-1} \begin{bmatrix} y_{20} & 0 \\ 0 & y_{21} \end{bmatrix}$$

(see AappendixE for detailed derivation)

Where 5.2.7 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,

$Y_{ij} = Y_{ijj} + Y_{ijl}$ therefore,

$$[Y_{20} \quad Y_{21}] = [1 \quad 1][H]_2 (I - [V]_2[C]_2)^{-1} \begin{bmatrix} y_{20} & 0 \\ 0 & y_{21} \end{bmatrix}$$

5.2.7 $\lim_{\mathbf{m} \to \infty} ([\mathbf{C}]_2)^{\mathbf{m}} = \mathbf{0}$, $\lim_{\mathbf{m} \to \infty} ([\mathbf{V}]_2[\mathbf{C}]_2)^{\mathbf{m}} = \mathbf{0}$

Here we show that:

$$\lim_{\mathrm{m} \to \infty} ([\mathrm{C}]_2)^{\mathrm{m}} = 0, \lim_{\mathrm{m} \to \infty} ([\mathrm{V}]_2[\mathrm{C}]_2)^{\mathrm{m}} = 0$$

Recall that,

C_{jl} : Probability of event bb_{lj}, i.e. b_l immediately follows b_j (observed at PE_j)

$1 - (C_{jj} + C_{jl})$: probability that neither b_j or b_l immediately follows b_j.

And $[C]_2 = \begin{bmatrix} C_{00} & C_{10} \\ C_{01} & C_{11} \end{bmatrix}$

And $[V]_2 = \begin{bmatrix} v_{20} & 0 \\ 0 & v_{21} \end{bmatrix}$

First of all, we know that practically speaking all probabilities $C_{jl} < 1$, moreover, $(C_{jj} + C_{jl}) < 1$, assuming that the probability that neither b_j or b_l immediately follows b_j is greater than zero.

Let $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ represent $[C]_2$

Then, $\begin{bmatrix} a & b \\ c & d \end{bmatrix}^2 = \begin{bmatrix} aa + bc & ab + bd \\ ac + cd & bc + dd \end{bmatrix}$

Since $(aa + bc) < (a + c) < 1$,$(ac + cd) < (a + c) < 1$

and $(ab + bd) < (b + d) < 1$, $(cb + dd) < (b + d) < 1$

So we establish that all values of the resulting matrix are < 1.

Moreover, as value of m increases for $\begin{bmatrix} \mathrm{a} & \mathrm{b} \\ \mathrm{c} & \mathrm{d} \end{bmatrix}^{\mathrm{m}}$, 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 $\sum_{m=0}^{\infty}([C]_2)^m$ 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 $0 < C_{jl} < 1$, $0 < C_{jj} < 1$ and $1 - \left(C_{jj} + C_{jl}\right) > 0$; Perron-Frobenius Theorem [41] can also be used to deduce that $\sum_{m=0}^{\infty}([C]_2)^m$ converges to 0 i.e.

$$\sum_{m=0}^{\infty}([C]_2)^m = (I - [C]_2)^{-1}$$

5.2.8 Consecutive bus event

The consecutive bus event probability i.e. $\Pr\left(bb_{ij}\right)$ is calculated on two observers, PE_i and PE_j. C_{ij} : probability of event bb_{ij} observed at PE_i 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 μ_i and a higher priority request does not arrive, i.e. event bb_{il} 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_{il} does not happen for all $(l < j)$. (1) can be easily modeled as a blk_{ji} where bus event $b_i(k)$ has 1 extended cycle.

Therefore, for $\alpha_{ji} = 1$ (1) can be calculated as,

$$\sum_{n=0}^{k} \lambda_i(1-\lambda_i)^n = 1 - (1-\lambda_i)^k$$

And for $\alpha_{ji} = 0$ (1) becomes,

$$\left(1 - U_{ji}^*(1-\lambda_j)^k\right)$$

Therefore, overall C_{ij} can be calculated by summing over all k,

$$C_{ij} = \begin{cases} \left(1 - U_{ji}^* v_{ji}^*\right)\left(1 - \sum_{l=0}^{i-1} C_{il}\right) & (\alpha_{ji} = 0) \\ \left(1 - v_{ji}\right)\left(1 - \sum_{l=0}^{i-1} C_{il}\right) & (\alpha_{ji} = 1) \end{cases}$$

where $U_{ji}^* = (1-\mu_j)S_{ji} - (1-\lambda_j)(1-S_{ji})$. Note that the merged inactivation probability v_{ji}^* is used since the occurrence of bb_{ii} means that bb_{ij} could still occur as elaborated in Figure 5-3.

S_{ij} is observed at PE_j as opposed to C_{ij} which is observed at PE_i. To reflect this difference the bus event count ratio $Q_{ji} = \frac{N_i}{N_j}$ is introduced.

$$S_{ij} = \begin{cases} Q_{ji}\left(1 - U_{ji}^{*} v_{ji}^{*}\right)\left(1 - \sum_{l=0}^{i-1} C_{il}\right) & (\alpha_{ji} = 0) \\ Q_{ji}\left(1 - v_{ji}\right)\left(1 - \sum_{l=0}^{i-1} C_{il}\right) & (\alpha_{ji} = 1) \end{cases}$$

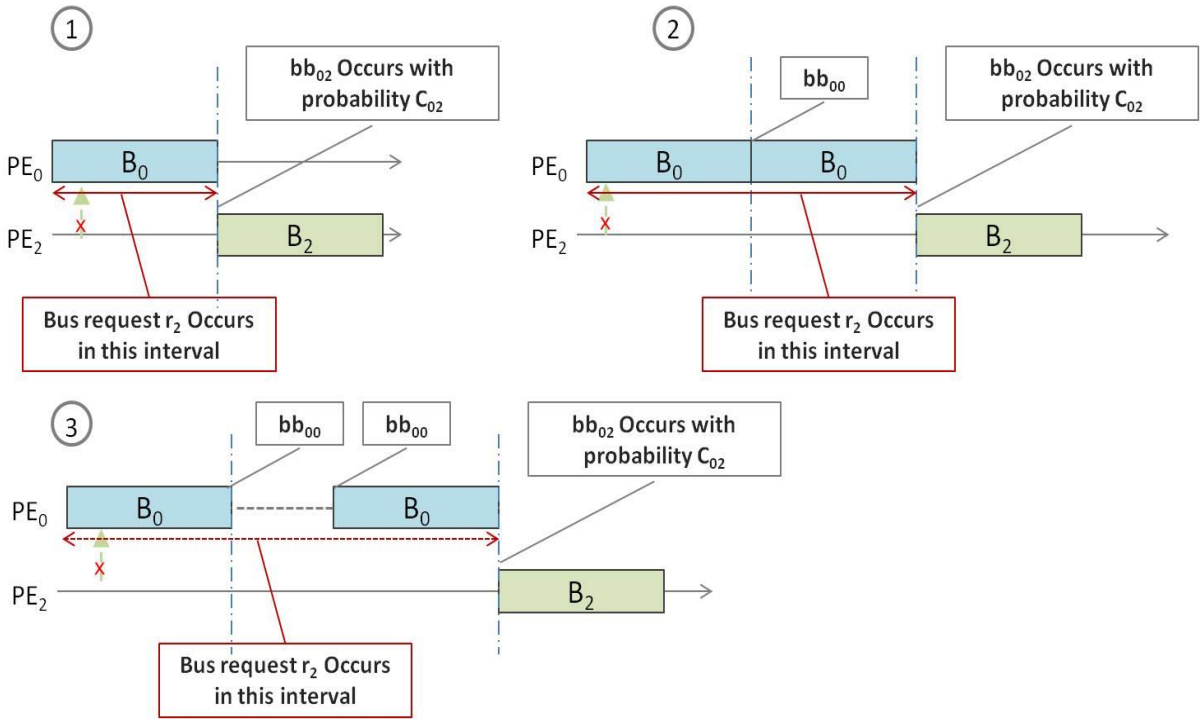

Figure 5-3: Occurrence of event bbji

5.2.9 Expressions for n-PE system

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,

$$\begin{bmatrix} E[B_{i0}] & E[B_{i1}] \cdots E[B_{in}] \end{bmatrix} = [1 \quad 1 \cdots 1]([B]_n)(I - [C]_n)^{-1}$$

Such that,

$$[B]_n = \begin{bmatrix} E[B_0] & 0 & . & . & 0 \\ 0 & E[B_1] & 0 & . & 0 \\ . & 0 & . & . & 0 \\ . & . & . & . & 0 \\ 0 & 0 & . & . & E[B_n] \end{bmatrix}$$

$$[C]_n = \begin{bmatrix} C_{00} & C_{10} & . & . & C_{n0} \\ C_{01} & C_{11} & . & . & C_{n1} \\ . & . & . & . & . \\ . & . & . & . & . \\ C_{0n} & . & . & . & C_{nn} \end{bmatrix}$$

And effective request inactivation probability,

$$[Y_{i0} \quad Y_{i1} \cdots Y_{in}] = [H]_n (I - [V]_n [C]_n)^{-1} ([Y]_n)$$

Such that,

$$[V]_n = \begin{bmatrix} v_{i0} & 0 & . & . & 0 \\ 0 & v_{in} & . & . & 0 \\ . & . & . & . & . \\ . & . & . & . & . \\ 0 & . & . & . & v_{in} \end{bmatrix}, [Y]_n = \begin{bmatrix} y_{i0} & 0 & . & . & 0 \\ 0 & y_{in} & . & . & 0 \\ . & . & . & . & . \\ . & . & . & . & . \\ 0 & . & . & . & y_{in} \end{bmatrix}$$

$$[H]_n = \begin{bmatrix} \left(1 - \sum_{l \le n} C_{0l}\right) & 0 & . & . & 0 \\ 0 & & . & . & 0 \\ . & & . & . & . \\ . & & . & . & . \\ 0 & & . & . & \left(1 - \sum_{l \le n} C_{nl}\right) \end{bmatrix}$$

5.2.10 Bus stall expectation

The bus stall expectation is given as,

$$E[D_{ij}] = \begin{cases} Q_{ij}^* \left(E[B_{ij}] - U_{ij}^* \frac{1}{\lambda_i} (1 - V_{ij}) \right) & (\alpha_{ij} = 0) \\ Q_{ij} \left(E[B_j] - \frac{1}{\lambda_i} (1 - v_{ij}) \right) & (\alpha_{ij} = 1) \end{cases}$$

when $\alpha_{ij} = 0$, PE_i will only see the *effective bus workloads*, i.e. the bus workloads that do not follow immediately after another higher priority bus workload. The bus event count ratio Q_{ij} in this case is rewritten as:

$$Q_{ij}^{*} = Q_{ij}\left(1 - \sum_{l<i} S_{lj}\right) \quad (\alpha_{ij} = 0)$$

5.3 Calculation flow

Detail of the bus stall calculation flow is described below. Here, let PEset be the set of processor indices.

4. Coefficient calculations: from the new sets of bus workload statistics (N_i, h_{Li}, h_{Bi}) obtained during the bus prediction period T, interval workload expectation $E[L_i]$, request probability λ_i, zero interval probability μ_i and bus workload expectation $E[B_i]$ are calculated.
5. Initial bus stall delay values: $\forall i \in PEset, E[D_i] = 0$
6. Variables that do not contain the iterative value Q_{ij} are calculated.
7. 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. Variables that depend on the value Q_{ij} are re-calculated each iteration.

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

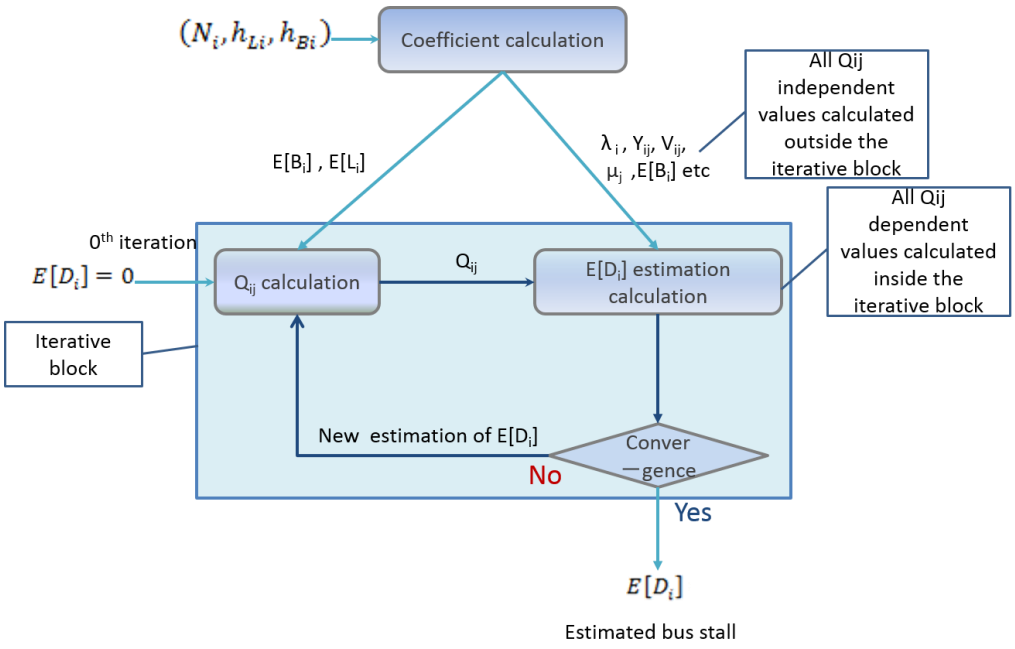

Figure 5-4: Bus stall calculation flow

6. EXPERIMENTS AND RESULTS

This chapter gives a detailed account of the experiment performed to verify and evaluate the proposed method of performance estimation and the accuracy of developed analytical model. Through experiments the accuracy and speed up of our technique as compared to the simulation based technique is demonstrated. A comparison of the simulation flow for both techniques has been reported in chapter 2. First we verify our model by using recorded traffic statistics from a synthetic traffic generator as input to our model. Synthetic traffic generators is a good tool to verify different aspects of our model as we can generate different kinds of traffic as per our requirement. Second, we use Recorded traffic patterns from real benchmark applications as reported by Liu, Xu, et al [] to verify the applicability of our technique to real applications.

6.1 Synthetic traffic generator

We use a synthetic traffic generator application 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

(e) Probability of a zero interval event

(f) Total simulation time.

Note that the prediction model is oblivious to the contents of the traffic characteristics and treats the traffic generator as any other application.

Typically bus request rates in multimedia applications are around 12% (at the lower end of this range), traffic ranges of 17-20% can be thought of as intensive bus traffic where as 50% would be regarded as extremely intensive bus traffic. The synthetic traffic generator gives us the capability to control traffic patterns and test the developed models for various kinds of traffic with various characteristics.

6.2 Recorded traffic patterns from real benchmark applications

Recorded traffic patterns from real benchmark applications as presented by Liu et. al. [42] were used to verify the applicability of our technique to real applications. A recorded traffic pattern is generated during cycle-accurate simulations for an application model. It contains accurate computation and communication traces, where all the task execution and packet generation events are recorded. The recorded traffic patterns are reusable on different communication architecture and configurations. The recorded traffic patterns keep the packet dependencies instead of exact timings, so the temporal relations can be reconstructed correctly, while applying to different configurations and architectures. Traffic patterns are recorded for various architectures with different number of PE. The applications and their traffic characteristics have been reported below.

1. Benchmark1: ROBOT
   The first benchmark used is Newton-Euler dynamic control calculation for 6-degrees-of-freedom Stanford manipulator. It consists of 88 tasks and 131 communication links. Detailed and accurate trace of task execution and communication is recorded during cycle-accurate simulation as Recorded Traffic Pattern (RTP). Our model uses the RTP statistics to predict contention stall as explained before. It consists of 88 tasks and 131 communication links while the observed value of zero-interval probability for the four PE version of it are "μ" is 0.268, 0.202, 0.30 and 0.545 on PE0, PE1, PE2 and PE3 respectively.
2. Benchmark2: SPARSE matrix solver
   The Second benchmark used is "Random sparse matrix solver for electronic circuit simulations". It consists of 96 tasks and 67 communication links. The observed value of zero-interval probability for the four PE version of it are "μ" is 0.175, 0.113, .133 and .226 on PE0, PE1, PE2 and PE3 respectively.
3. Benchmark3: FFT-1024-complex
   The third benchmark application used is a "Fast Fourier Transform" with 1024 inputs of complex numbers. It consists of 16384 tasks and 25600 communication links. The observed value of zero-interval probability for the four PE version of it are "μ" is 0.175, 0.113, .133 and .226 on PE0, PE1, PE2 and PE3 respectively. This benchmark application has a bigger number of tasks and communication links as compared to the previous applications.

4. Benchmark4: FPPPP

   The fourth benchmark application used is “SPEC95 Fpppp” which is a chemical program performing multi-electron integral derivatives. It consists of 334 tasks and 1145 communication links. The observed value of zero-interval probability for the four PE version of it are “μ” is 0.7, 0.733, 0.76 and .712 on PE0, PE1, PE2 and PE3 respectively. As evident from the number of communication links compared to tasks and the zero-interval probabilities on each PE, the traffic is very bursty.

   Table1 shows a traffic comparison of all four benchmark.

Table 1: Traffic details of benchmark applications

Benchmark	Tasks	Communication links	Probability μ
ROBOT	88	131	0.268,0.202,0.3,0.545
SPARSE	96	67	0.175,0.113,0.133,0.226
FPPPP	334	1145	0.7, 0.733, 0.76, 0.712
FFT_1024complex	16384	25600	0.175, 0.113, 0.133, 0.226

A Recorded Traffic Pattern (RTP) is given by a representation of the recorded behaviors of the set of tasks scheduled and executed on a PE. Each task has execution time t, and a unique sequence number for scheduling it on the PE. The execution condition of a task is given by its input set of information including the set of incoming edges and the data on every incoming edge as obtained from the corresponding predecessor task. The result of the task execution is given by the output of a task, including the set of outgoing edges, the generated data and the destination task. The data size determines the number of packets that may be delivered through the inter-connect. Since the absolute timing between task executions and communications is dependent on architecture, the RTP keeps data dependencies and temporal relations using relative timing instead of absolute timing. The relative timing will be generated according to the real dependencies in the applications and is guaranteed to be the correct semantics.

6.3 Architecture configuration

The MPSoC architecture model is specified in a file. The architecture file contains various configurable parameters which can be used to control Processor type, Processor frequency, thread to processor mapping, bus topology and architecture and bus mapping. For experiments we used two architectures i.e. four PE system connected via a shared bus and 8-PE system connected via a shared bus as shown in Figure 6-1 and Figure 6-2.

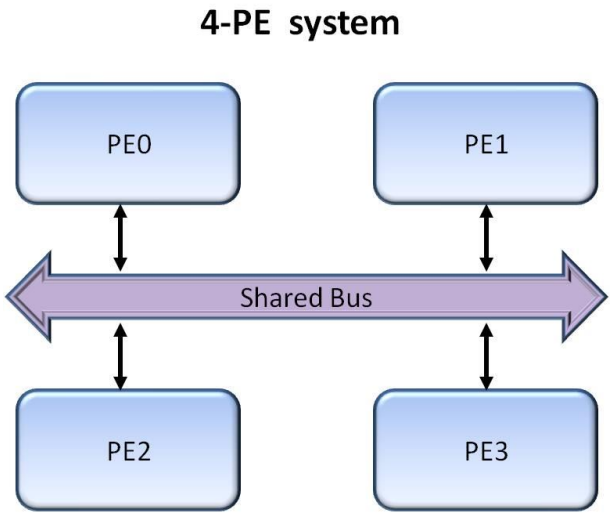

Figure 6-1: 4PE architecture

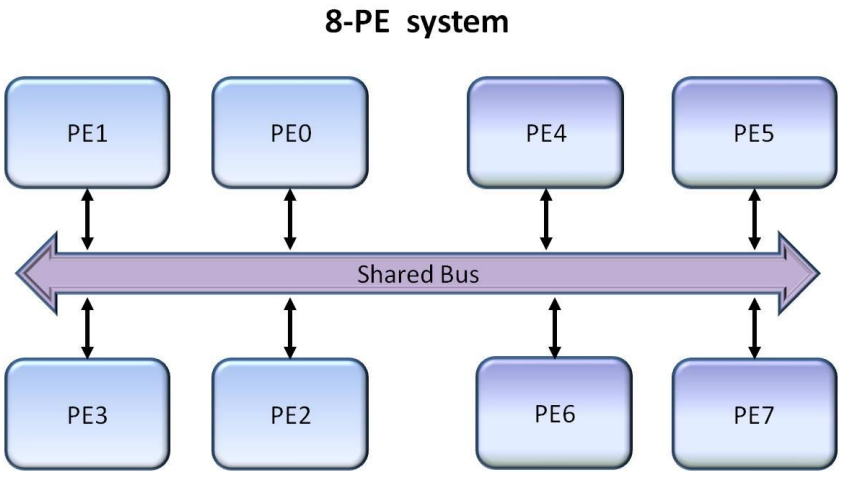

Figure 6-2: 8PE architecture

6.4 Single Blocking Model

First the accuracy of SBM is evaluated using synthetic traffic generator. We report results for a 2-PE system with a shared bus such that $\mu = 0$ for all PE. This system is in accordance with the assumptions for single blocking model. The bus behavior can be modeled by SBM with very small error. Synthetic traffic with different parameters is generated in multiple experiments such that there is no burst traffic. The traffic patterns are varied such that the length of bus-workload ranges from 10%-50% of total execution time. Estimation errors are calculated using the formula:

$$error\% = \left(\frac{estimated\ cycles\ by\ simulation - \ estimated\ cycles\ by\ prediction}{estimated\ cycles\ by\ simulation}\right) \times 100$$

As expected the estimation errors for 2-PE system such that $\mu = 0$ for all PE is well under 1%. Secondly we repeat the experiments such that burst traffic is generated with a probability μ for various values of μ. The cycle estimation error increases drastically, as the value of μ increases, using the SBM model.

6.5 Burst Blocking Model

After evaluating SBM using synthetic traffic, The burst blocking model is used for cycle estimation with the same traffic patterns where The traffic patterns are varied such that the length of bus-workload ranges from 10%-50% of total execution time and burst traffic is generated with a probability μ for various values of μ. The prediction results using Single Blocking Model (SBM) and Burst Blocking Model (BBM) are compared in Figure 6-3. As the value of μ increases, prediction error increases drastically for the SBM model however the BBM is able to capture the burst behavior of the traffic and stays well under 1%.

In further experiments, real benchmark application traffic is used to evaluate BBM cycle estimation. The Recorded Traffic Patterns (RTP) of the real benchmark applications are fed to the workload simulator, while bus stall cycle estimation is performed using the BBM. The first benchmark used is Newton-Euler dynamic control calculation for 6-degrees-of-freedom Stanford manipulator. Figure 6-4 shows a comparison of simulated and predicted cycle-counts on each PE. As seen in the figure prediction results are very

accurate with an average estimation error of .002% which is almost negligible. The Second benchmark used is “Random sparse matrix solver for electronic circuit simulations”. Figure 6-5 shows a comparison of simulated and predicted cycle-counts on each PE. Prediction results are very accurate with an average estimation error of .015%. Hence the quality of estimation for these applications while using BBM is very good. The third benchmark application used is a “Fast Fourier Transform” with 1024 inputs of complex numbers. This benchmark application has a bigger number of tasks and communication links as compared to the previous applications tested. It is expected that the multi-blocking behavior would be more observable.

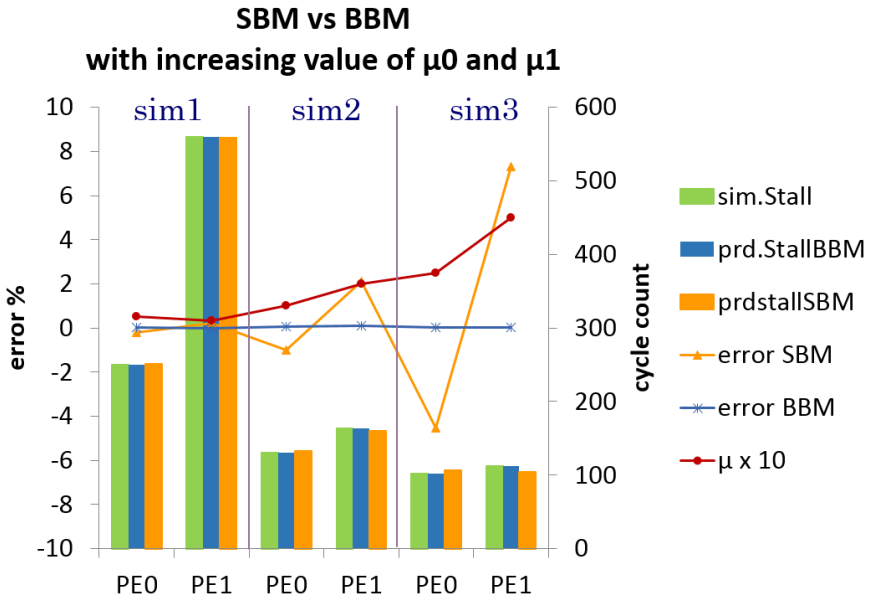

Figure 6-3: Cycle estimation error on SBM and BBM

Figure 6-6 shows a comparison of simulated and predicted cycle-counts on each PE. As we expect, the stall predicted for high priority PE is higher than the simulated stall, similarly stall predicted for lower priority PE is lower than the simulated stall. This behavior is as expected since the multi-blocking behavior in reality means that high priority PE wins arbitration more often and the lower priority PE loses arbitration more often than as captured by the BBM model. The fourth benchmark application used is “SPEC95 Fpppp” which is a chemical program performing multi-electron integral derivatives. As evident from the number of communication links compared to tasks and

the zero-interval probabilities on each PE, the traffic is very bursty. Figure 6-7 shows a comparison of simulated and predicted cycle-counts on each PE. As we can see the prediction result is much more accurate compared to the FFT_1024complex. Figure 6-8 shows cycle-count estimation error using the proposed estimation technique for 4 benchmark applications. Average estimation error on all PEs is reported. The FFT solver application shows the highest average error of about 7%. The reason for high error in the FFT solver is that it is a communication intensive application. Which means all masters are trying to request the bus much more frequently. Hence the probability of multi-blocking behavior is high. Since our proposed model does not capture the multi-blocking behavior yet, we observe relatively higher estimation error. From these results we can conclude that the burst blocking model is reasonably accurate for 4PE architectures on the given benchmark applications. The FFT benchmark application shows the most error and must be examined closely in further experiments.

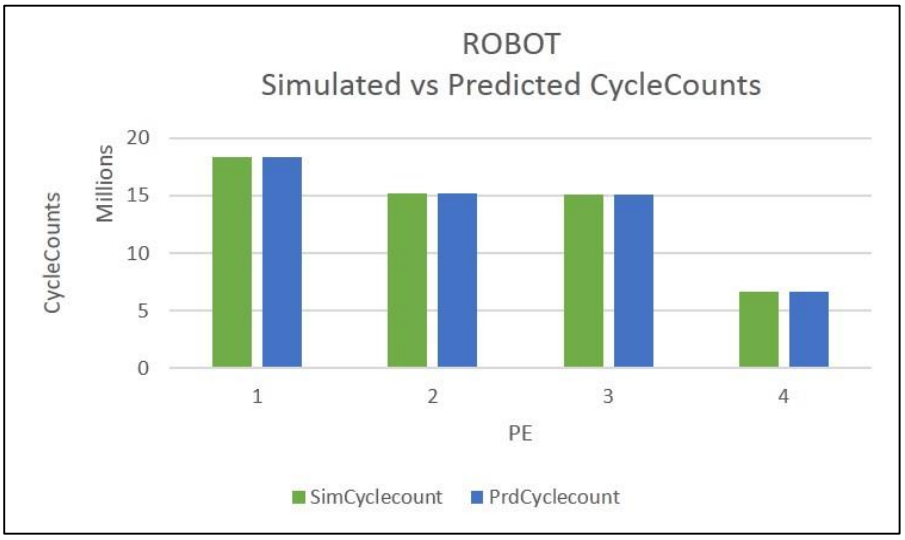

Figure 6-4: Estimated cycles for ROBOT benchmark

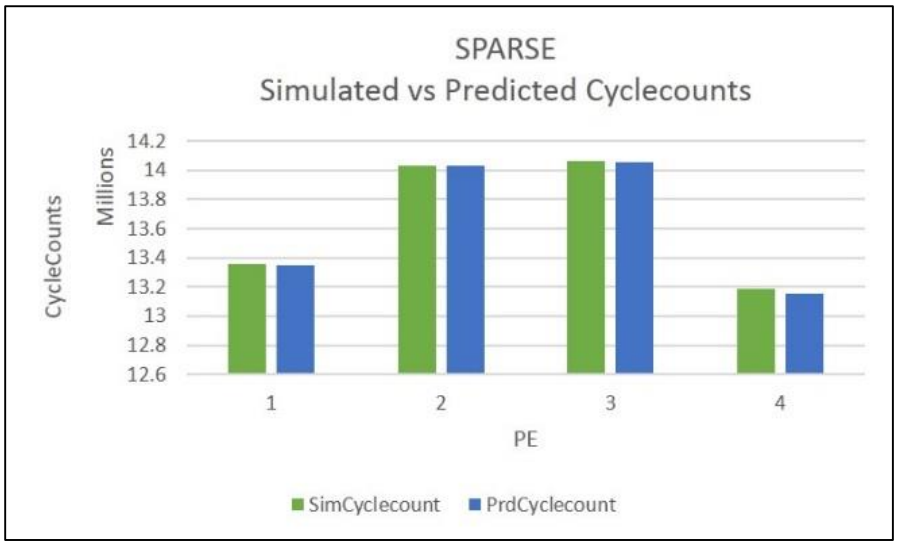

Figure 6-5: Estimated cycles for SPARSE benchmark

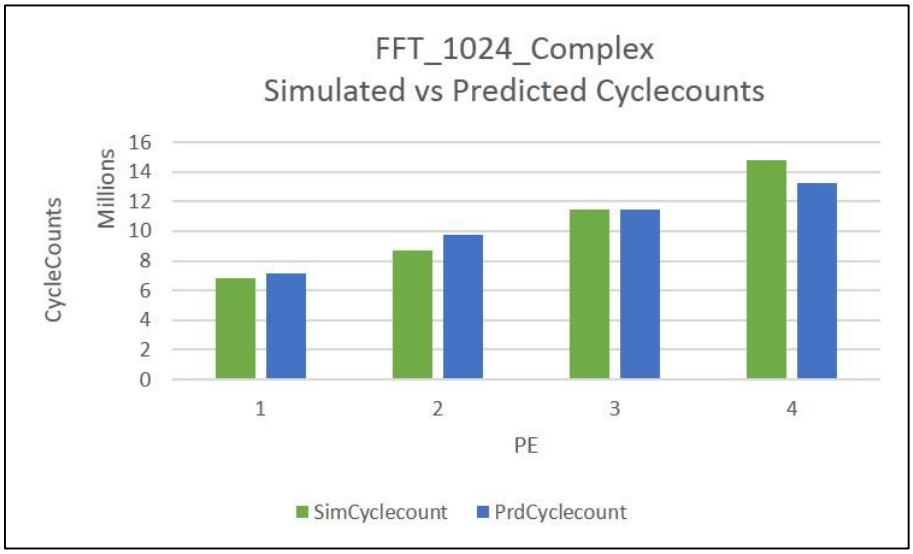

Figure 6-6: Estimated cycles for FFT benchmark

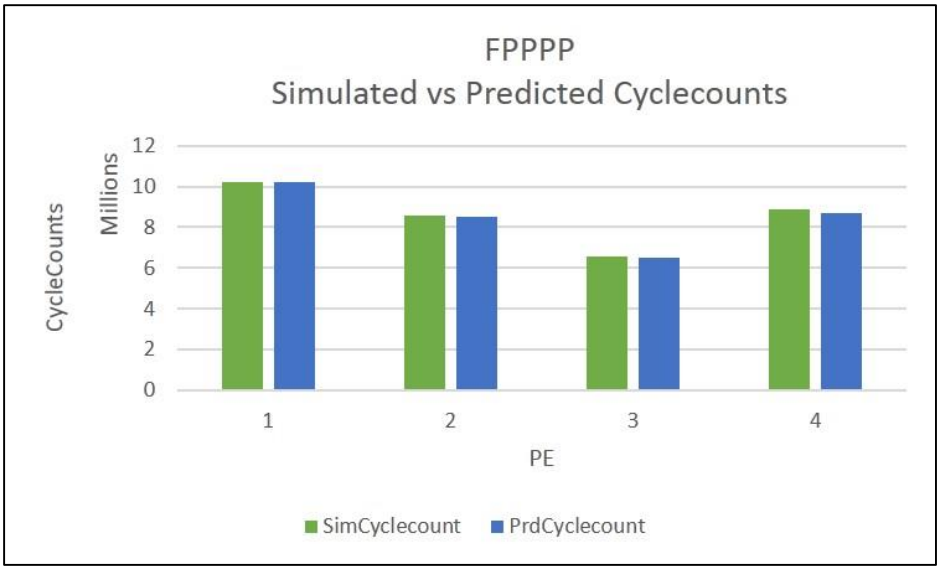

Figure 6-7: Estimated cycles for FPPPP benchmark

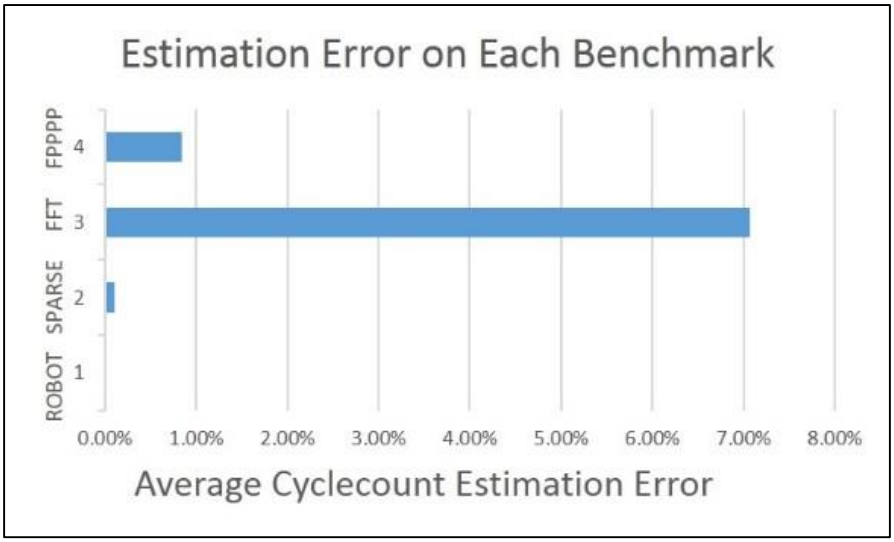

Figure 6-8: Estimation error comparison

6.6 Multi-Blocking Model

After evaluating both SBM and BBM, further experiments are performed to evaluate accuracy of estimation using MBM. For these experiments two different MPSoC architectures are used. A 4-PE system with a single shared bus and an 8-PE system with a single shared bus. For the 4-PE case, the “Fast Fourier Transform” and “SPEC95 Fpppp” benchmark applications are evaluated. While for the 8-PE system the *ROBOT*, *SPARSE*

matrix solver, *Fast Fourier Transform* and *SPEC95 Fpppp* are evaluated. Figure 6-9 – Figure 6-13 show the comparison of estimated cycles using the bus simulation method, calculated using prediction by BBM and by MBM methods. 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. 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 is observed. 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. Tabe2 shows a summarized comparison of the prediction errors for BBM and MBM.

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.

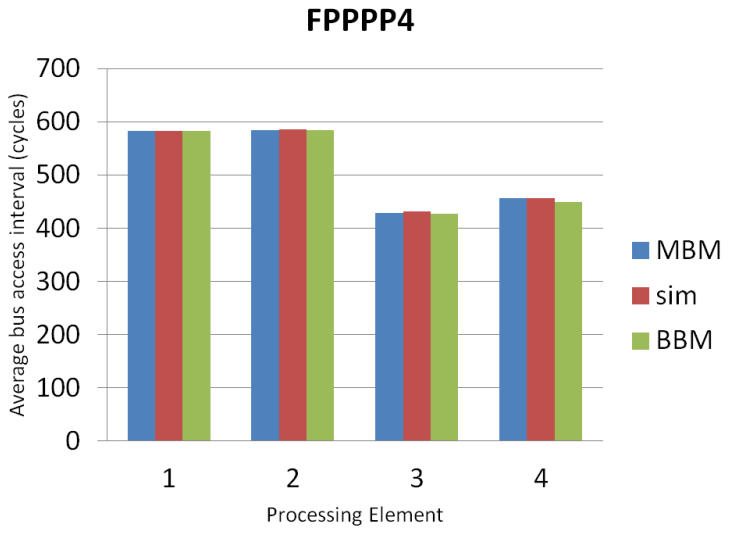

Figure 6-9: Estimated cycles for FPPPP benchmark (4PE)

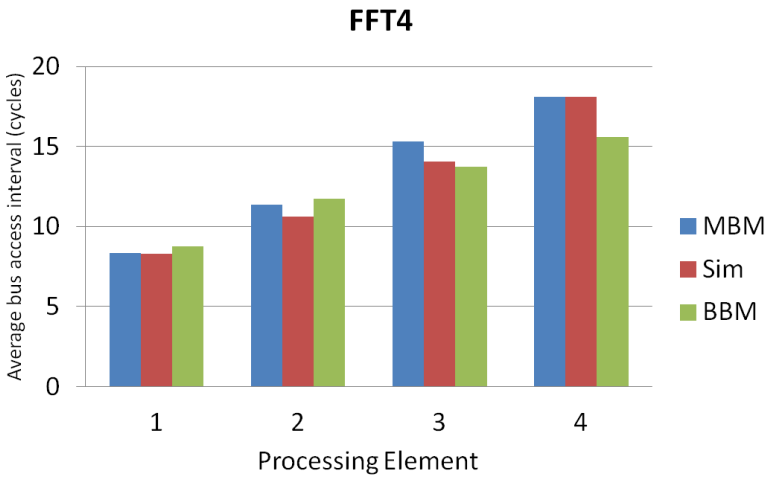

Figure 6-10: Estimated cycles for FFT benchmark (4PE)

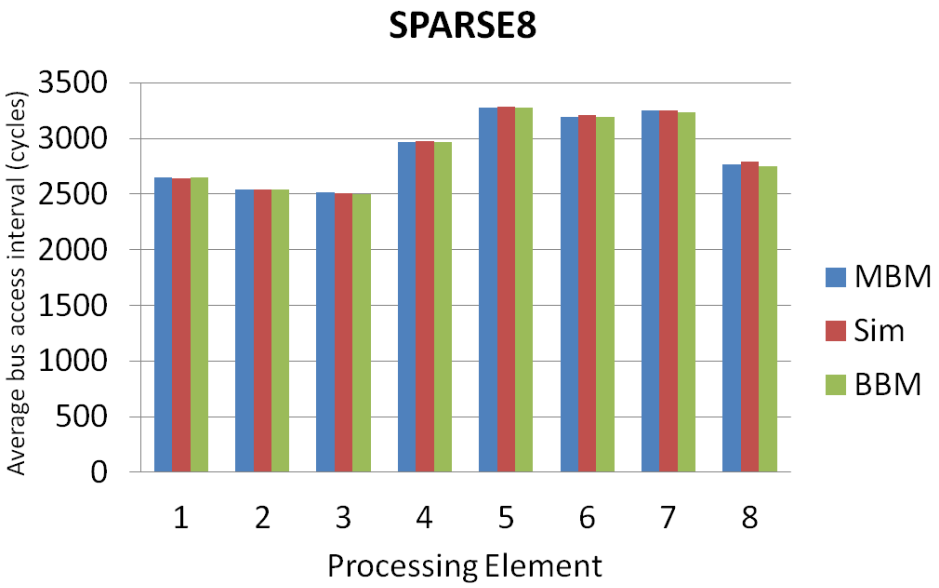

Figure 6-11: Estimated cycles for SPARSE benchmark (8PE)

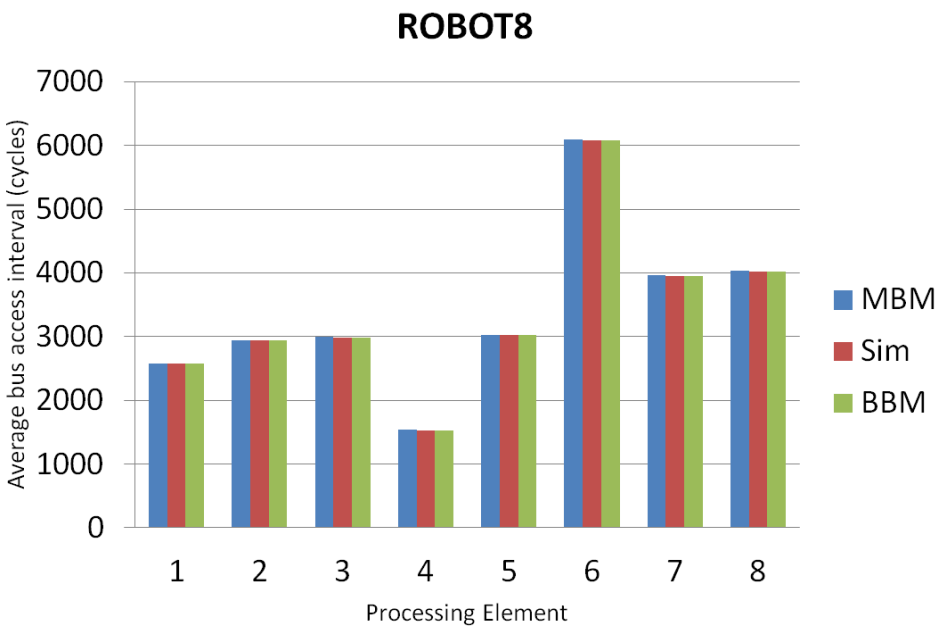

Figure 6-12: Estimated cycles for ROBOT benchmark (8PE)

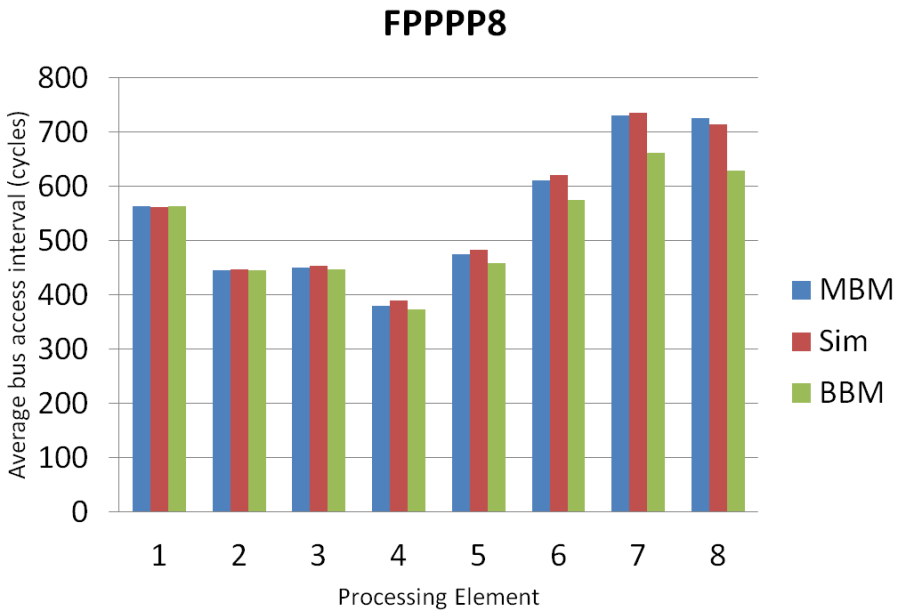

Figure 6-13: Estimated cycles for FPPPP benchmark (8PE)

Table 2: Comparison of prediction errors on BBM and MBM

		BBM Min error %	BBM Max error %	MBM Min error %	MBM Max error %
4PE	FPPPP	0.1	1.75	0.02	0.6
	FFT	2.2	13.91	0.6	8.8
8PE	SPARSE	0.075	1.5	0.003	0.945
	ROBOT	0.005	0.08	0.005	0.08
	FPPPP	0.28	11.8	0.098	2.7

6.6.1 The curious case of FFT benchmark

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%. 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. *Table2* 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 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 On the other hand, 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, $E[D_i]$ is calculated and total bus stall during the bus

prediction interval ($E[D_i] \cdot N_i$) is added to the processor's simulation clock, where N_i 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 $(N_i - 1)$ bus workloads on lowest priority PEs are wrongly predicted to incur stall due to highest priority PEs. The prediction model calculates the incurred bus stall assuming all bus requests must be serviced before the next window starts. This adds a huge stall to all bus requests however, in reality the remaining $(N_i - 1)$ requests will not be issued until the first bus request is serviced. In contrast to FFT8, the workload completion record of FPPPP8 benchmark, as shown in *Table3*, 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.

Table 3: Record of workload completion on each PE on FFT benchmark

Check Points	PE0	PE1	PE2	PE3	PE4	PE5	PE6	PE7
Start	0	0	0	0	0	0	0	0
CP 0	F	30237	13087	1264	7	0	0	0
CP1	F	F	21917	5343	530	0	0	0
CP2	F	F	F	20125	8122	827	0	0
CP3	F	F	F	F	25352	8902	1115	7
CP4	F	F	F	F	F	20744	7208	736
CP5	F	F	F	F	F	F	24013	9179
CP6	F	F	F	F	F	F	F	22664
CP7	F	F	F	F	F	F	F	F
Total BW	41220	41080	40960	40880	40960	40900	40860	40820

Table 4: Record of workload completion on each PE on FPPPP benchmark

Check Points	PE0	PE1	PE2	PE3	PE4	PE5	PE6	PE7
Start	0	0	0	0	0	0	0	0
CP0	635	716	F	759	699	500	393	425
CP1	668	831	F	F	748	566	444	477
CP2	686	F	F	F	763	572	489	503
CP3	813	F	F	F	F	732	583	568
CP4	913	F	F	F	F	F	636	679
CP5	F	F	F	F	F	F	681	926
CP6	F	F	F	F	F	F	F	727
CP7	F	F	F	F	F	F	F	F
Total BW	940	880	720	940	920	780	720	780

6.6.2 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 [43], 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 the rate with which $([C]_n)^m$ shrinks being very small. For PE_7 this probability is $1 - \sum_{x=0}^{6} C_{jx}$. The values of $\sum_{x=0}^{6} C_{jx}$ probabilities for our experiments have been reported in *Table 4*. For the FFT8 example, we noted that when $\sum_{x=0}^{6} C_{jx}$ becomes greater than 0.9 (i.e. $1 - \sum_{x=0}^{6} C_{jx}$ becomes less that 0.1) the traffic pattern starts to cause starvation i.e. as seen by lowest priority PE, b_j is immediately followed by a bus event on a higher priority PE at least 90% of the times.

Table 5: Sum of C_{jx} probabilities to identify starvation

Application	C_{0x}	C_{1x}	C_{2x}	C_{3x}	C_{4x}	C_{5x}	C_{6x}	starvation
FFT8	0.9999	0.9994	0.999	0.9978	0.9972	0.9944	0.9918	Yes
FFT8 2XBBW	0.9969	0.9919	0.9877	0.9776	0.9718	0.9506	0.933	Yes
FFT8 2XBBW 1/3XCmp	0.9041	0.8598	0.8413	0.7946	0.7881	0.7206	0.6947	No
FPPP8	0.8816	0.8965	0.884	0.8525	0.848	0.783	0.7144	No

6.6.2.1 Identifying bus starvation

Usually it is not straight forward 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 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.

6.6.2.2 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 6-14* 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

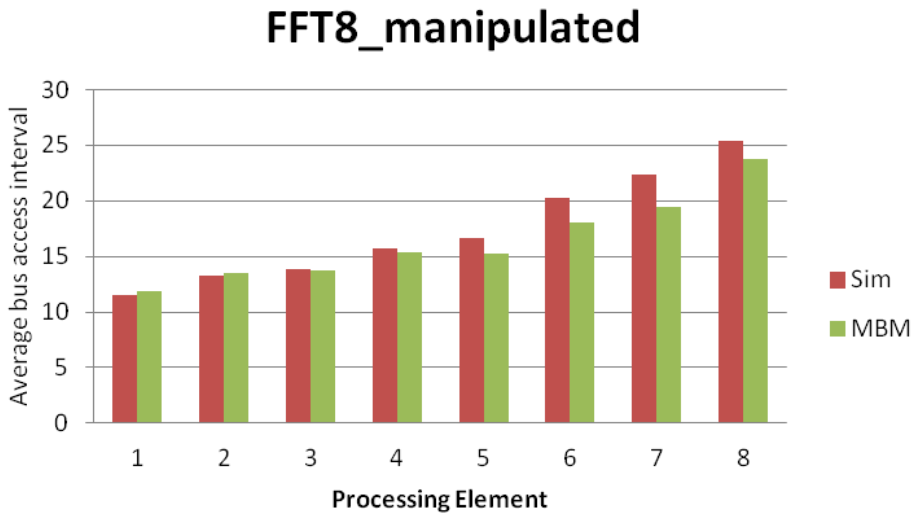

Figure 6-14: Estimated cycles for FFT manipulated (8PE)

Table 6: Record of workload completion on each PE on FFT manipulated

Check Points	PE0	PE1	PE2	PE3	PE4	PE5	PE6	PE7
Start	0	0	0	0	0	0	0	0
CP0	F	35467	32882	26828	21536	11074	3865	623
CP1	F	F	38428	31709	26061	14385	6036	1393
CP2	F	F	F	34147	28288	16468	7808	2412
CP3	F	F	F	F	35181	22515	13297	6322
CP4	F	F	F	F	F	27669	18295	10460
CP5	F	F	F	F	F	F	27669	9179
CP6	F	F	F	F	F	F	F	30493
CP7	F	F	F	F	F	F	F	F
Total BW	41220	41080	40960	40880	40960	40900	40860	40820

6.7 Simulation speed-up

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,

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

Where, t_{ba} 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. Overall

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

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,

$$T_{prd} = (\sum_{W} t_m)$$

Where, t_m is the time it takes to solve the mathematical equations and W is the number of prediction time-windows. The speed-up ratio then simply becomes,

$$Speedup = \frac{T_{sim}}{T_{prd}}$$

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_{prd} 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_{prd}

as the simulation data length increases. As a result the speedup ratio increases as "I" increases.

Figure 6-15 – Figure 6-18 report T_{sim} for each individual PE with an increasing value of *"I"* and a comparison between total T_{sim}, total T_{prd} and the resulting speedup ratio for the SPARSE, ROBOT, FPPP and FFT benchmarks on an 8PE system respectively. Experiments were performed for different values of I i.e. "I =1, 10, 100, 500 and 1000". 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_{prd} 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 shown in Figure 6-15 – Figure 6-18. 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.

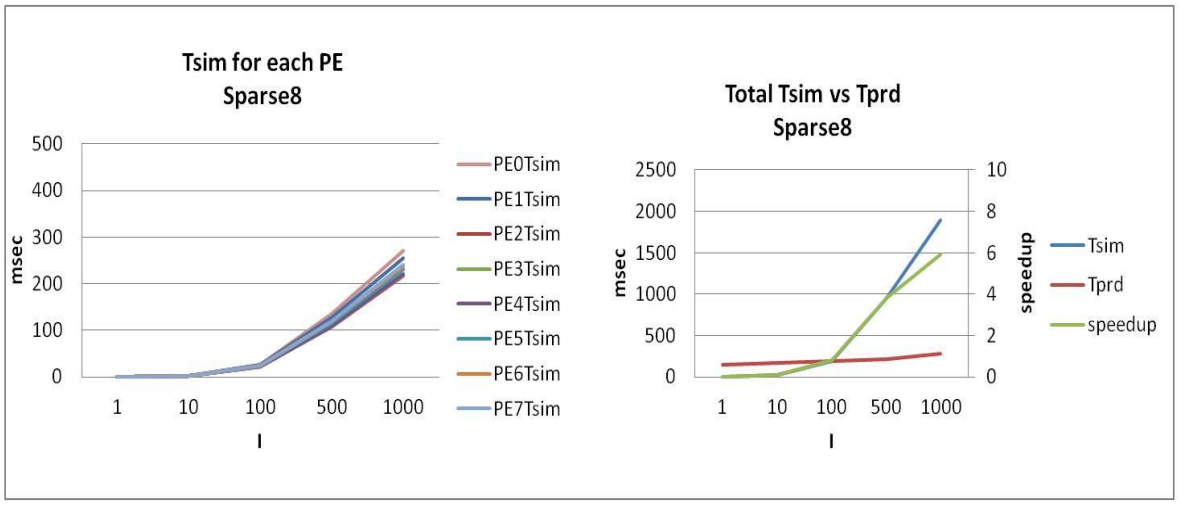

Figure 6-15: Tsim for each PE and Speed-up for SPARSE8 benchmark

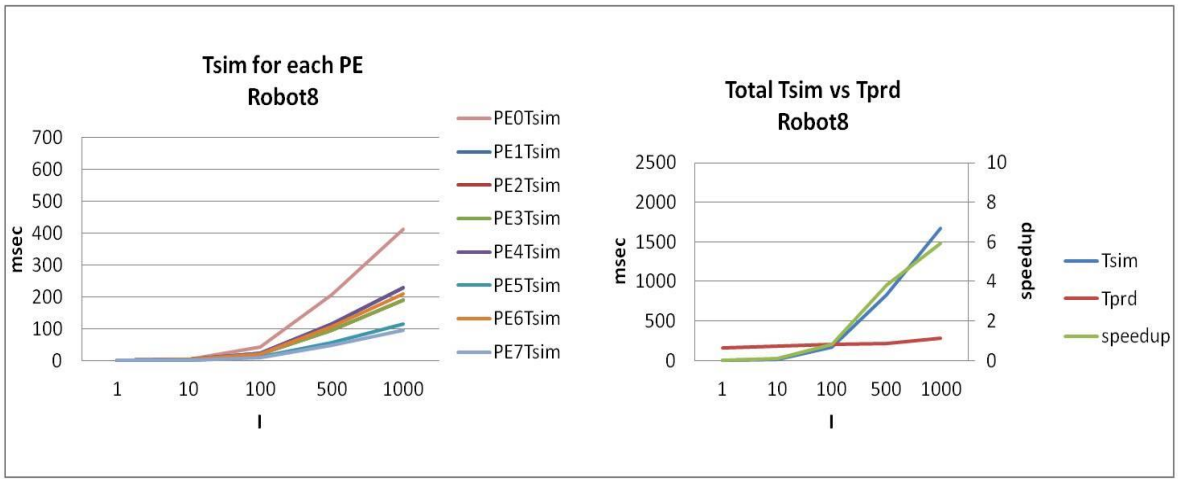

Figure 6-16: Tsim for each PE and Speed-up for ROBOT8 benchmark

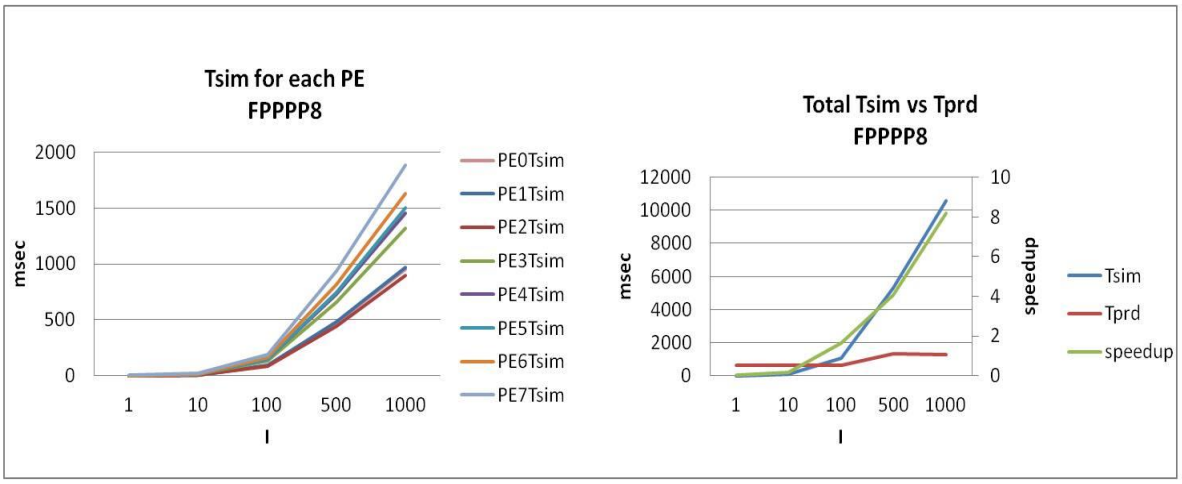

Figure 6-17: Tsim for each PE and Speed-up for FPPPP8 benchmark

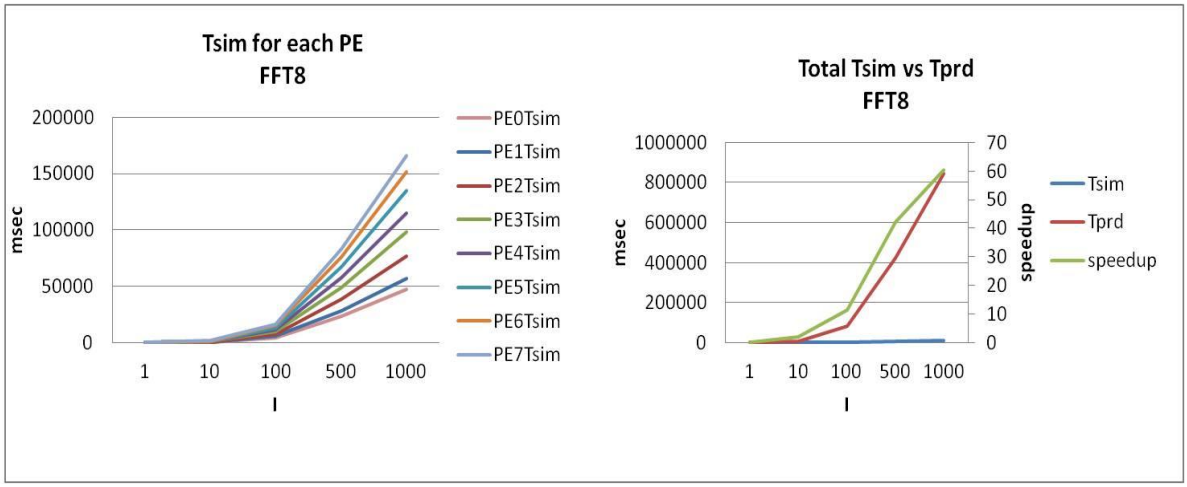

Figure 6-18: Tsim for each PE and Speed-up for FFT8 benchmark

6.8 Summary

Here a summary of the experiments and results is presented. The proposed model was evaluated for prediction accuracy and simulation speed-up using two kinds of traffic namely, synthetic traffic generated with varying traffic parameters and real benchmark`s recorded traffic patterns. The synthetic traffic can be generated using a synthetic traffic generator application which can generate varying traffic based on defined traffic parameters, defined in a text file. The prediction model has no knowledge of the traffic parameters and performs stall prediction for the generated application traffic as usual. Table 7 shows a summary of prediction accuracy results for the SBM and BBM models. Secondly, real benchmark`s Recorded Traffic Patterns (RTP) are used to evaluate the proposed model`s applicability to real applications. The RTP are read from a text file and stall prediction is performed accordingly. Simulations were performed for several benchmark applications for a few different MPSoC architectures. A summary of the prediction accuracy results for BBM and MBM are shown in Table 8. Moreover, using the proposed technique, simulation speed-up of up to 5x for low traffic applications and up to 10x was achieved for high traffic applications.

Table 7: Synthetic traffic prediction error

		SBM error%		BBM error%	
Simulation	μ	Min	Max	Min	Max
Sim1	0.03 ~ 0.05	0.19	0.21	0.001	0.005
Sim2	0.1 ~ 0.25	1.01	2.1	0.059	0.1
Sim3	0.25 ~ 0.5	4.54	7.3	0.04	0.02

Table 8: Real benchmarks prediction error

Architecture	Application	BBM error %		MBM error %	
		Min	Max	Min	Max
4PE	SPARSE	0.02	0.23	0.02	0.23
	ROBOT	0.002	0.0023	0.002	0.0023
	FPPPP	0.1	1.75	0.02	0.6
	FFT	2.2	13.91	0.6	8.8
8PE	SPARSE	0.075	1.5	0.003	0.945
	ROBOT	0.005	0.08	0.005	0.08
	FPPPP	0.28	11.8	0.098	2.7

7. CONCLUSION

In this chapter we will summarize the work presented in the previous chapters and outline the future directions for our work.

The growing complexity of MPSoC architectures fuels a need for better and efficient performance estimation techniques. Electronic System Level design tools are developed to enable fast and productive design cycles by providing software development platforms before detailed architectures is fixed and help in performance estimation at the system level, however it requires a considerable development effort. New techniques in Instruction Set Simulators try to improve ISS simulation times but simulation speeds become slower with increasing number of processor cores and resources found in modern MPSoC. Other techniques use statistical work-load models of target applications for estimating performance and driving optimization hence eliminating the need for complete OS, drivers and models etc. however, deriving a proper workload is highly application dependent. To tackle this issue, Trace Driven Workload Model based performance estimation technique has been developed. The Trace-driven workload model is very useful in fast performance estimation and shows a clear advantage over the ISS simulation based performance estimation methods. However, for bus traffic it has to resort to simulation techniques to resolve bus arbitration. In this research we address this issue by presenting a novel performance estimation technique for on-chip bus based architecture.

7.1 Contributions

In this research we present 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.

The following summarizes our research contributions.

i. Presenting a methodology to enable bus performance estimation much faster than simulation based performance estimation techniques while achieving much higher accuracy than static analysis based estimation techniques. Using this methodology

we enable bus-performance aware application development, application partitioning and task mapping. Using these techniques we aim to enable faster design cycles by providing performance estimation at system level covering larger design space faster while maintaining acceptable estimation accuracy.

ii. Developing a mathematical model for estimation of MPSoC bus stall using traffic statistics. Our model achieves higher accuracy by collecting traffic parameters from real application trace of computation-workloads as well as bus-workloads. Using real computation-workload statistics, we are able to model an accurate bus request arrival rate. Moreover, burst traffic is recorded separately and modeled as a separate parameter in the model to account for it as a special case in order to accurately model burst traffic. Over all three models are developed namely SBM(Single Blocking Model), BBM (Burst Blocking Model) and MBM (Multi-Blocking Model). While the SBM accurately models single blocking bus masters, that do not generate burst traffic, sharing a single bus. The BBM also models burst traffic while MBM models N-PE system with burst traffic and multiple interfering bus masters.

iii. Using developed mathematical model together with a trace-driven workload simulator enabling fast bus performance estimation in a trace-driven workload simulation environment. The performance estimation works such that workload statistics for computation as well as bus traffic on each processing element for a target application obtained from the trace driven workload model are used as input to the statistical model to estimate bus stall cycles. The estimated stall cycles are added to the cycle-estimation on each PE to achieve an over-all cycle-count on each Processing Element. The resulting estimates on performance are used to tweak the bus architecture or application or partitioning or task-mapping to meet desired constraints.

The proposed model is evaluated using four benchmark applications on two architectures. From experiments we can deduce that the simpler model, BBM is accurate enough for four bus masters sharing a single bus. For the bus architecture with eight bus masters on a single bus, BBM shows an increased estimation error while MBM achieves better accuracy. In conclusion, the proposed estimation model enables fast performance estimation of MPSoC bus architectures while maintaining a high level of accuracy.

7.2 Future work

Further development of our proposed technique has a couple of directions. First of all 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. Secondly, current work models a fixed priority based arbitration scheme however, in future work, 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.

8. BIBLIOGRAPHY

[1] Qualcomm technologies Inc, "snapdragon821 processor product-brief," 2016.

[2] Chipworks Inc., "Apple iPhone 6s Complementary Teardown Report with Additional Commentary," October 2015.

[3] Samsung electronics inc., "Exynos 8 Octa (8890), LTE-Advanced modem integrated one chip solution built on 14nm FinFET process," 2016.

[4] Media Tek Inc, "http://mediatek-helio.com/x20/".

[5] S. Shukla, C. Pixley and G. Smith, "The True State of the Art of ESL Design," *IEEE Design & Test of Computers*, vol. 23, no. 5, pp. 335-337, 2006.

[6] F. Ghenassia, Transaction-Level Modeling with SystemC, Springer US, 2005.

[7] M. Reshadi, P. Mishra and N. Dutt, "Instruction Set Compiled Simulation: A Technique for Fast and Flexible Instruction Set Simulation," in *Proc. DAC*, 2003, pp. 758 - 763.

[8] A. Nohl, G. Braun, O. Schliebusch and R. Leupers, "A Universal Technique for Fast and Flexible Instruction-set Architecture Simulation," in *Proc. DAC*, 2002, pp. 22-27.

[9] IBM, "IBM On-chip CoreConnect Bus Architecture," [Online]. Available: www.chips.ibm.com.

[10] ARM, "ARM AMBA Specification (rev2.0)," 2001. [Online]. Available: www.arm.com/products/system-ip/amba-specifications.php.

[11] SONICSINC, "Sonics Integration Architecture," [Online]. Available: http://sonicsinc.com/.

[12] STMicroelectronics, "STBus Communication System: Concepts and Definitions," May2003.

[13] A. Jantsch and H. Tenhunen, Networks on chip, Hingham, MA, USA: Kluwer Academic Publishers, 2003.

[14] L. Benini and G. DMicheli, "Networks on Chips: A New SoC Paradigm," *IEEE Computers*, pp. 70-78, Jan2002.

[15] T. Isshiki, D. Li, H. Kunieda, T. Isomura and K. Satou, "Trace-Driven Workload Simulation Method for Multiprocessor System-On-Chips," in *DAC*, San Francisco, California, USA, 2009.

[16] T. Sherwood, E. Perelman and B. Calder, "Basic block distribution analysis to find periodic behavior and simulation points in applications,"," *Parallel Architecture and Compilation Techniques*, p. pp. 3–14, 2001.

[17] P. V. Knudsen and J. Madsen, "Communication Estimation for Hardware/Software Codesign," *(CODES/CASHE '98) Proceedings of the Sixth International Workshop on Hardware/Software Codesign*, pp. 55-59, 1998.

[18] W. Wolf and T. Yen, "Communication synthesis for distributed embedded systems," in *ICCAD*, San Jose, CA, USA, Nov. 1995.

[19] J. Daveau, T. B. Ismail and A. A. Jerraya, "Synthesis of system-level communication by an allocation based approach," in *Int. Symp. System Level Synthesis*, Cannes, France, Sep. 1995.

[20] A. Marculescu and R. Nandi, "System level power/performance analysis for embedded systems design," in *DAC*, Las Vegas, NV, USA, Jun. 2001.

[21] M. Drinic, D. Kirovski, S. Meguerdichian and M. Potkonjak, "Latency-guided on-chip bus network design," in *ICCAD*, San Jose, California, USA, Nov, 2000.

[22] N. Thepayasuwan and A. Doboli, "Layout conscious bus architecture synthesis for deep submicron system on chip," in *DATE*, Paris, France, Feb. 2004.

[23] Y. S. Cho, E. J. Choi and K. R. Cho, "Modeling and analysis of the system bus latency on the SoC platform," in *SLIP '06 the international workshop on System-level interconnect prediction*, Munich, 2006.

[24] M. e. a. Loghi, "Analyzing On-Chip Communication in a MPSoC Environment," in *DATE*, Paris, France, Feb. 2004.

[25] L. Semeria and A. Ghosh, "Methodology for hardware/software co-verification in C/C++," in *ASP-DAC Asia and South Pacific Design Automation Conference*, Yokohama, 2000.

[26] P. Kalla, X. Hu and J. Henkel, "A flexible framework for communication evaluation in SoC design," in *ASP-DAC*, Shanghai, China, 2005.

[27] Caldari et al, "Transaction-Level Models for AMBA Bus Architecture Using SystemC 2.0," in *DATE*, Munich, Germany, 2003.

[28] O. Ogawa, S. Bayon de Noyer, P. Chauvet and K. Shinohara, "A practical approach for bus architecture

optimization at transaction level," in *Design, Automation and Test in Europe Conference and Exhibition*, Munich, Germany, 2003.

[29] M. Ariyamparambath, D. Bussaglia, B. Reinkemeier, D. Kogel and T. Kempf, "A highly efficient modeling style for heterogeneous bus architectures," *Proceedings of International Symposium on System-on-Chip.*, pp. 83-87, 2003.

[30] E. Viaud, F. Pecheux and A. Greiner, "An efficient TLM/T modeling and simulation environment based on conservative parallel discrete event principles," in *DATE Design Automation and Test in Europe*, Munich, Germany, 2006.

[31] Schirner, Gunar and R. Dömer, "Quantitative analysis of the speed/accuracy trade-off in transaction level modeling," *ACM Transactions on Embedded Computing Systems (TECS),* vol. 8, no. 1, p. article 4, Dec. 2008.

[32] G. Beltrame, D. Sciuto, C. Silvano, D. Lyonnard and C. Pilkington, "Exploiting TLM and Object Introspection for System-Level Simulation," in *DATE Design Automation & Test in Europe*, munich, 2006.

[33] D. Chiou, D. Sunwoo, J. Kim, N. Patil, W. Reinhart and D. Johnson, "FPGA-accelerated simulation technologies (fast): Fast, full-system, cycle-accurate simulations," *Proc. of International Symposium on Microarchitecture,* pp. 249-261, 2007.

[34] D. Papamichael, J. Hoe and O. Mutlu, "FIST: A fast, lightweight, FPGA-friendly packet latency estimator for NoC," in *Intl Symp on Networks on Chip*,, 2011.

[35] Z. Tan, A. Waterman, H. Cook, S. Bird, K. Asanovic and D. Patterson, "A case for FAME: FPGA architecture model execution,," in *Proc of Intl Symposium on Computer Architecture*, 2011.

[36] K. Lahiri, A. Raghunathan and S. Dey, "System-level performance analysis for designing system-on-chip communication architecture," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.,* vol. 20, pp. 768-783, Jun. 2001.

[37] S. Kim, C. Im and S. Ha, "Schedule-aware performance estimation of communication architecture for efficient design space exploration," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 13, no. 5, pp. 539-552, May. 2005.

[38] R. Kawahara et al. IBM Research Tokyo, "Coarse-grained simulation method for performance evaluation of ashared memory system," in *ASP-DAC*, Yokohama, Jan, 2011.

[39] M. Z. Urfianto, T. Isshiki, A. U. Khan, D. Li and H. Kunieda, "A Multiprocessor SoC Architecture with Efficient Communication Infrastructure and Advanced Compiler Support for Easy Application Development," *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences,* Vols. Vol.E91-A, no. 4, pp. 1185-1196, April, 2008.

[40] T. Isshiki, Trace-Driven MPSoC Simulation with Cache Modeling, 13th International Forum on Embedded MPSoC and Multi-Core, 2013.

[41] V. Richard S, Matrix Iterative Analysis. Vol.27, Berlin: Springer Science & Business Media, 2009.

[42] W. Liu, J. Xu, X. Wu, Y. Ye, X. Wang, W. Zhang, M. Nikdast and Z. Wang, "A NoC Traffic Suite Based on Real Applications," *IEEE Computer Society Annual Symposium on VLSI (ISVLSI),* July, 2011.

[43] Raynal and Michel, "Concurrent Programming: Algorithms, Principles, and Foundations.," *Springer Science & Business Media,* p. 10–11, 2013.

AppendixA

The expectation of the merged bus workload $E[B_j^*]$ is derived as follows:

$$E[B_j^*] = \sum_k f_{Bj}^*(k) \cdot k$$

From eq(4.6):

$$f_{Bj}^*(k) = \sum_{m=0}^{\infty} (1-\mu_j)\mu_j^m f_{Bj}(m,k)$$

therefore:

$$E[B_j^*] = \sum_k \left(\sum_{m=0}^{\infty} (1-\mu_j)\mu_j^m f_{Bj}(m,k) \right) \cdot k$$

$$= (1-\mu_j) \sum_{m=0}^{\infty} \mu_j^m \left(\sum_k f_{Bj}(m,k) \cdot k \right) = (1-\mu_j) \sum_{m=0}^{\infty} \mu_j^m E[B_j(m)]$$

where $E[B_j(m)]$ is the expectation of $m+1$ merged bus workloads, and is derived as follows:

$$E[B_j(m)] = \sum_k f_{Bj}(m,k) \cdot k$$

$$= \sum_k \left(\sum_n f_{Bj}(m-1,n) f_{Bj}(k-n) \right) \cdot k$$

$$= \sum_n f_{Bj}(m-1,n) \sum_k f_{Bj}(k-n) \cdot k$$

$$= \sum_n f_{Bj}(m-1,n) \sum_{k'} f_{Bj}(k')(k'+n) \quad (k' = k-n)$$

$$= \sum_n f_{Bj}(m-1,n) \left(\sum_{k'} f_{Bj}(k') \cdot k' + \sum_{k'} f_{Bj}(k') \cdot n \right)$$

$$= \sum_n f_{Bj}(m-1,n)\left(E[B_j] + n\right)$$

$$= E[B_j] \sum_n f_{Bj}(m-1,n) + \sum_n f_{Bj}(m-1,n) \cdot n$$

$$= E[B_j] + E[B_j(m-1)]$$

$$E\left[B_j(m)\right] - E\left[B_j(m-1)\right] = E\left[B_j\right]$$

Also since $E\left[B_j(0)\right] = E\left[B_j\right]$,

$$E\left[B_j(m)\right] = (m+1) \cdot E\left[B_j\right]$$

Using this value in the expression for expectation of the merged bus workload $E\left[B_j^*\right]$:

$$E\left[B_j^*\right] = \left(1-\mu_j\right) \sum_{m=0}^{\infty} \mu_j^m (m+1) \cdot E\left[B_j\right]$$

$$= \left(1-\mu_j\right) \cdot E\left[B_j\right] \sum_{m=0}^{\infty} \left(m\mu_j^m + \mu_j^m\right)$$

Since,

$$\sum_{m=0}^{\infty} \mu_j^m = \frac{1}{1-\mu_j}$$

$$\sum_{m=0}^{\infty} \mu_j^m m = \sum_{m=1}^{\infty} \mu_j^m m = \sum_{m=1}^{\infty} \mu_j^m + \sum_{m=2}^{\infty} \mu_j^m + \sum_{m=3}^{\infty} \mu_j^m + \cdots = \sum_{k=1}^{\infty} \sum_{m=k}^{\infty} \mu_j^m = \sum_{k=1}^{\infty} \frac{\mu_j^k}{1-\mu_j}$$

$$= \frac{\mu_j}{\left(1-\mu_j\right)^2}$$

$$\therefore E\left[B_j^*\right] = \left(1-\mu_j\right) \cdot E\left[B_j\right] \left(\frac{\mu_j}{\left(1-\mu_j\right)^2} + \frac{1}{1-\mu_j} \right) = \frac{E\left[B_j\right]}{1-\mu_j}$$

AppendixB

The request inactivation probability, Y_{ij}^{*} is in fact a power series of $\mu_j(1-\lambda_i)Y_{ij}$ and is derived as follows:

$$Y_{ij}^{*} = \sum_{k} f_{Bj}^{*}(k)(1-\lambda_i)^{k-1} = \sum_{k}\left(\sum_{m=0}^{\infty}(1-\mu_j)\mu_j^{m} f_{Bj}(m,k)\right)(1-\lambda_i)^{k-1}$$

$$= (1-\mu_j)\sum_{m=0}^{\infty}\mu_j^{m}\left(\sum_{k} f_{Bj}(m,k)(1-\lambda_i)^{k-1}\right) = (1-\mu_j)\sum_{m=0}^{\infty}\mu_j^{m} Y_{ij}(m)$$

$$Y_{ij}(m) = \sum_{k} f_{Bj}(m,k)(1-\lambda_i)^{k-1}$$

$$= \sum_{k}\left(\sum_{n} f_{Bj}(m-1,n) f_{Bj}(k-n)\right)(1-\lambda_i)^{k-1}$$

$$= \sum_{n} f_{Bj}(m-1,n)\sum_{k} f_{Bj}(k-n)(1-\lambda_i)^{k-1}$$

$$= \sum_{n} f_{Bj}(m-1,n)\sum_{k'} f_{Bj}(k')(1-\lambda_i)^{k'+n-1} \quad (k' = k-n)$$

$$= \sum_{n} f_{Bj}(m-1,n)(1-\lambda_i)^{n}\sum_{k'} f_{Bj}(k')(1-\lambda_i)^{k'-1}$$

$$= \sum_{n} f_{Bj}(m-1,n)(1-\lambda_i)^{n} Y_{ij}$$

$$= (1-\lambda_i)Y_{ij}\cdot\left(\sum_{n} f_{Bj}(m-1,n)(1-\lambda_i)^{n-1}\right)$$

$$= (1-\lambda_i)Y_{ij}\cdot Y_{ij}(m-1)$$

$$= \left((1-\lambda_i)Y_{ij}\right)^{m}\cdot Y_{ij}$$

$$\therefore Y_{ij}^{*} = (1-\mu_j)\sum_{m=0}^{\infty}\mu_j^{m}\left((1-\lambda_i)Y_{ij}\right)^{m}\cdot Y_{ij} = (1-\mu_j)Y_{ij}\sum_{m=0}^{\infty}\left(\mu_j(1-\lambda_i)Y_{ij}\right)^{m}$$

$$Y_{ij}^{*} = \frac{(1-\mu_j)Y_{ij}}{1-\mu_j(1-\lambda_i)Y_{ij}}$$

AppendixC

Expectation of $m+1$ merged bus events is calculated as:

$$\begin{bmatrix} E[B_{00}(m)] & E[B_{10}(m)] \\ E[B_{01}(m)] & E[B_{11}(m)] \end{bmatrix} = \sum_k k \begin{bmatrix} f_{00}(m,k) & f_{10}(m,k) \\ f_{01}(m,k) & f_{11}(m,k) \end{bmatrix}$$

$$= \sum_k k \sum_n \begin{bmatrix} f_{B0}(n) & 0 \\ 0 & f_{B1}(n) \end{bmatrix} C_2 \begin{bmatrix} f_{00}(m-1,k-n) & f_{10}(m-1,k-n) \\ f_{01}(m-1,k-n) & f_{11}(m-1,k-n) \end{bmatrix}$$

$$= \sum_n \begin{bmatrix} f_{B0}(n) & 0 \\ 0 & f_{B1}(n) \end{bmatrix} C_2 \sum_k k \begin{bmatrix} f_{00}(m-1,k-n) & f_{10}(m-1,k-n) \\ f_{01}(m-1,k-n) & f_{11}(m-1,k-n) \end{bmatrix}$$

$$= \sum_n \begin{bmatrix} f_{B0}(n) & 0 \\ 0 & f_{B1}(n) \end{bmatrix} C_2 \sum_{k'} (k'+n) \begin{bmatrix} f_{00}(m-1,k') & f_{10}(m-1,k') \\ f_{01}(m-1,k') & f_{11}(m-1,k') \end{bmatrix}$$

$$= \sum_n \begin{bmatrix} f_{B0}(n) & 0 \\ 0 & f_{B1}(n) \end{bmatrix} C_2 \sum_{k'} k' \begin{bmatrix} f_{00}(m-1,k') & f_{10}(m-1,k') \\ f_{01}(m-1,k') & f_{11}(m-1,k') \end{bmatrix}$$

$$+ \sum_n n \begin{bmatrix} f_{B0}(n) & 0 \\ 0 & f_{B1}(n) \end{bmatrix} C_2 \sum_{k'} \begin{bmatrix} f_{00}(m-1,k') & f_{10}(m-1,k') \\ f_{01}(m-1,k') & f_{11}(m-1,k') \end{bmatrix}$$

$$= C_2 \begin{bmatrix} E[B_{00}(m-1)] & E[B_{10}(m-1)] \\ E[B_{01}(m-1)] & E[B_{11}(m-1)] \end{bmatrix}$$

$$+ \begin{bmatrix} E[B_0] & 0 \\ 0 & E[B_1] \end{bmatrix} C_2 \begin{bmatrix} F_{00}(m-1) & F_{10}(m-1) \\ F_{01}(m-1) & F_{11}(m-1) \end{bmatrix}$$

$$\begin{bmatrix} E[B_{00}(m)] & E[B_{10}(m)] \\ E[B_{01}(m)] & E[B_{11}(m)] \end{bmatrix}$$

$$= C_2 \begin{bmatrix} E[B_{00}(m-1)] & E[B_{10}(m-1)] \\ E[B_{01}(m-1)] & E[B_{11}(m-1)] \end{bmatrix} + \begin{bmatrix} E[B_0] & 0 \\ 0 & E[B_1] \end{bmatrix} C_2^m$$

AppendixD

$$\sum_{m=0}^{\infty}\begin{bmatrix}E[B_{00}(m)] & E[B_{10}(m)]\\ E[B_{01}(m)] & E[B_{11}(m)]\end{bmatrix}$$

$$= \begin{bmatrix}E[B_{00}(0)] & E[B_{10}(0)]\\ E[B_{01}(0)] & E[B_{11}(0)]\end{bmatrix} + \sum_{m=1}^{\infty}\begin{bmatrix}E[B_{00}(m)] & E[B_{10}(m)]\\ E[B_{01}(m)] & E[B_{11}(m)]\end{bmatrix}$$

$$= \begin{bmatrix}E[B_0] & 0\\ 0 & E[B_1]\end{bmatrix} + \sum_{m=1}^{\infty}\left(C_2\begin{bmatrix}E[B_{00}(m-1)] & E[B_{10}(m-1)]\\ E[B_{01}(m-1)] & E[B_{11}(m-1)]\end{bmatrix} + \begin{bmatrix}E[B_0] & 0\\ 0 & E[B_1]\end{bmatrix}C_2^m\right)$$

$$= \sum_{m=1}^{\infty} C_2\begin{bmatrix}E[B_{00}(m-1)] & E[B_{10}(m-1)]\\ E[B_{01}(m-1)] & E[B_{11}(m-1)]\end{bmatrix} + \sum_{m=0}^{\infty}\begin{bmatrix}E[B_0] & 0\\ 0 & E[B_1]\end{bmatrix}C_2^m$$

$$= \sum_{m'=0}^{\infty} C_2\begin{bmatrix}E[B_{00}(m')] & E[B_{10}(m')]\\ E[B_{01}(m')] & E[B_{11}(m')]\end{bmatrix} + \begin{bmatrix}E[B_0] & 0\\ 0 & E[B_1]\end{bmatrix}(I-C_2)^{-1}$$

$$\sum_{m=0}^{\infty}(I-C_2)\begin{bmatrix}E[B_{00}(m)] & E[B_{10}(m)]\\ E[B_{01}(m)] & E[B_{11}(m)]\end{bmatrix} = \begin{bmatrix}E[B_0] & 0\\ 0 & E[B_1]\end{bmatrix}(I-C_2)^{-1}$$

$$\sum_{m=0}^{\infty}\begin{bmatrix}E[B_{00}(m)] & E[B_{10}(m)]\\ E[B_{01}(m)] & E[B_{11}(m)]\end{bmatrix} = (I-C_2)^{-1}\begin{bmatrix}E[B_0] & 0\\ 0 & E[B_1]\end{bmatrix}(I-C_2)^{-1}$$

AppendixE

$$\begin{bmatrix} Y_{200}^* & Y_{210}^* \\ Y_{201}^* & Y_{211}^* \end{bmatrix} = \sum_k \begin{bmatrix} f_{00}^*(k) & f_{10}^*(k) \\ f_{01}^*(k) & f_{11}^*(k) \end{bmatrix} (1-\lambda_2)^{k-1}$$

$$= H_2 \sum_k \sum_{m=0}^{\infty} \begin{bmatrix} f_{00}(m,k) & f_{10}(m,k) \\ f_{01}(m,k) & f_{11}(m,k) \end{bmatrix} (1-\lambda_2)^{k-1}$$

$$= H_2 \sum_{m=0}^{\infty} \sum_k \begin{bmatrix} f_{00}(m,k) & f_{10}(m,k) \\ f_{01}(m,k) & f_{11}(m,k) \end{bmatrix} (1-\lambda_2)^{k-1} = H_2 \sum_{m=0}^{\infty} \begin{bmatrix} Y_{200}(m) & Y_{210}(m) \\ Y_{201}(m) & Y_{211}(m) \end{bmatrix}$$

$$\begin{bmatrix} Y_{200}(m) & Y_{210}(m) \\ Y_{201}(m) & Y_{211}(m) \end{bmatrix} = \sum_k \begin{bmatrix} f_{00}(m,k) & f_{10}(m,k) \\ f_{01}(m,k) & f_{11}(m,k) \end{bmatrix} (1-\lambda_2)^{k-1}$$

$$= \sum_k \sum_n \begin{bmatrix} f_{B0}(n) & 0 \\ 0 & f_{B1}(n) \end{bmatrix} C_2 \begin{bmatrix} f_{00}(m-1,k-n) & f_{10}(m-1,k-n) \\ f_{01}(m-1,k-n) & f_{11}(m-1,k-n) \end{bmatrix} (1-\lambda_2)^{k-1}$$

$$= \sum_n \begin{bmatrix} f_{B0}(n) & 0 \\ 0 & f_{B1}(n) \end{bmatrix} C_2 \sum_k \begin{bmatrix} f_{00}(m-1,k-n) & f_{10}(m-1,k-n) \\ f_{01}(m-1,k-n) & f_{11}(m-1,k-n) \end{bmatrix} (1-\lambda_2)^{k-1}$$

$$= \sum_n \begin{bmatrix} f_{B0}(n) & 0 \\ 0 & f_{B1}(n) \end{bmatrix} C_2 \sum_{k'} \begin{bmatrix} f_{00}(m-1,k') & f_{10}(m-1,k') \\ f_{01}(m-1,k') & f_{11}(m-1,k') \end{bmatrix} (1-\lambda_2)^{k'+n-1}$$

$$= \sum_n \begin{bmatrix} f_{B0}(n) & 0 \\ 0 & f_{B1}(n) \end{bmatrix} (1-\lambda_2)^n C_2 \sum_{k'} \begin{bmatrix} f_{00}(m-1,k') & f_{10}(m-1,k') \\ f_{01}(m-1,k') & f_{11}(m-1,k') \end{bmatrix} (1-\lambda_2)^{k'-1}$$

$$= \begin{bmatrix} V_{20} & 0 \\ 0 & V_{21} \end{bmatrix} C_2 \begin{bmatrix} Y_{200}(m-1) & Y_{210}(m-1) \\ Y_{201}(m-1) & Y_{211}(m-1) \end{bmatrix}$$

$$\begin{bmatrix} Y_{200}(m) & Y_{210}(m) \\ Y_{201}(m) & Y_{211}(m) \end{bmatrix} = V_2 C_2 \begin{bmatrix} Y_{200}(m-1) & Y_{210}(m-1) \\ Y_{201}(m-1) & Y_{211}(m-1) \end{bmatrix}$$

where $V_2 = \begin{bmatrix} v_{20} & 0 \\ 0 & v_{21} \end{bmatrix}$

PUBLICATIONS

Journal Papers

1. Farhan Shafiq, Tsuyoshi Isshiki, Dongju Li, “An Accurate and Fast Trace-aware Performance Estimation Model For Prioritized MPSoC Bus With Multiple Interfering Bus-Masters”, *IPSJ Transactions on System LSI Design Methodology*, p. Vol.10 Feburary issue, pp 13-27, 2017.

2. Farhan Shafiq, Tsuyoshi Isshiki, Dongju. Li and Hiroaki Kunieda, "A Fast Trace Aware Statistical Based Prediction Model with Burst Traffic Modeling for Contention Stall in A Priority Based MPSoC Bus, *IPSJ Transactions on System LSI Design Methodology*, p. Vol.9 Feburary issue, pp 1–12, 2016.

Conference paper

3. Farhan Shafiq, Tsuyoshi Isshiki, Dongju Li and Hiroaki Kunieda, "A Fast and Highly Accurate Statistical Based Model for Performance Estimation of MPSoC on-Chip Bus," in *SASIMI*, Yilan, Taiwan, 2015.

.