Comparative Performance Evaluation of Latency and Link Dynamic Power Consumption Modelling Algorithms in Wormhole Switching Networks On Chip

James Harbin[1,], Leandro Soares Indrusiak[1]*

Real-Time Systems Group, Department of Computer Science, University of York, UK

Abstract

The simulation of interconnect architectures can be a time-consuming part of the design flow of on-chip multiprocessors. Accurate simulation of state-of-the art network-on-chip interconnects can take several hours for realistic application examples, and this process must be repeated for each design iteration because the interactions between design choices can greatly affect the overall throughput and latency performance of the system. This paper presents a series of network-on-chip transaction-level model (TLM) algorithms that provide a highly abstracted view of the process of data transmission in priority preemptive and non-preemptive networks-on-chip, which permit a major reduction in simulation event count. These simulation models are tested using two realistic application case studies and with synthetic traffic. Results presented demonstrate that these lightweight TLM simula-

*Corresponding author: Telephone +44 (0)1904 325550, Fax +44 (0)1904 325599

Email addresses: james.harbin@york.ac.uk (James Harbin), leandro.indrusiak@york.ac.uk (Leandro Soares Indrusiak)

tion models can produce latency figures accurate to within mere flits for the majority of flows, and more than 93% accurate link dynamic power consumption modelling, while simulating 2.5 to 3 orders of magnitude faster when compared to a cycle-accurate model of the same interconnect.
Keywords: network on chip, transaction level modelling, TLM, NoC modelling, simulation models, dynamic power consumption

1. Introduction

As the number of cores upon on-chip multiprocessors and system-on-chip (SoC) devices has increased, inter-core communication has become a critical design issue. The design architecture of the NoC (network-on-chip) is a vital factor in performance tuning, given the large influence it has upon communication latency and power consumption. As a result of the highly dynamic nature of application traffic and the potential for interactions between traffic during transmission, most design flows use simulation rather than static analysis to evaluate the power and latency performance delivered by a candidate NoC architecture. NoC interconnect simulation (as distinct from the full system simulation including execution of code upon processing elements) has been identified as an important research issue [1]. The design space of viable NoCs for multicore or SoC problems spans a wide range of candidate architectures and topologies, and is further expanded by the possible variability in application task mapping decisions. Particularly during the early stages of the design process, it is important to accelerate NoC simulation

with as little impact upon accuracy as possible, allowing the design space to be explored rapidly. Therefore, methodologies other than cycle-accurate simulation are promising as candidates to rapidly explore the NoC design space.

This paper specifies and evaluates a family of NoC simulation models which are both fast and accurate in comparison to cycle-accurate references. Two NoC architectures are considered which can be accurately described using transaction-level modelling (TLM). The models assume delay-sensitive applications that have certain timing constraints, and therefore the application model includes priorities used in arbitration decisions. Since these application models typically require one flow to be prioritised over another, priority preemptive NoCs following the example of QNoC [2] are the first architecture assumed. However, given that priority preemptive NoC architectures have higher silicon area requirements for implementation, a non-preemptive architecture is also considered and evaluated.

The definition of TLM assumed in this work is that of Cai and Gajski [3], in which components are either transaction initiators, targets or interconnects. The relationship of our models to the TLM definitions specified by Cai and Gajski is considered in Section 5. Compared to cycle-accurate models, the proposed TLM algorithms are simplified to reduce the frequency of simulation events. Events are generated only upon flow admission, flow removal or when simulation state must be updated to ensure consistency. Removing the necessity to model low-level details such as the progress of

every data flit through arbiters, routers and other simulator-level elements permits the reduction of simulation event count by orders of magnitude.

A fine-grained cycle-accurate model can offer precise simulation of the NoC internals, including the occupation and free status of particular buffers. However, in order to improve execution time performance and reduce simulation algorithm complexity, the buffer occupation of intermediate routers is not considered within the TLM models described in this paper. Even under the design structure of a transaction level model, several design choices are possible regarding the abstraction levels chosen, with resulting implications for timing performance and accuracy. In our earlier work [4], [5], [6] the entire route was be treated as a single unified abstraction when making contention decisions. Although this modelling approach is simple and its execution timing performance favourable, heavy contention requires a more fine-grained approach to improve timing accuracy.

The major novelty in this paper is the presentation of the TLM model TLM-NPD, which provides a finer locking granularity at the level of individual links and the ability to model flow behaviour in single-cycle increments in case of contention. The further intent of this paper is to comparatively assess and evaluate the simulation latency accuracy and execution time performance of our family of transaction level models TLM-PRE [4], [5], TLM-NP [6] and TLM-NPD, compared to reference cycle-accurate implementations. These evaluations are performed with test cases incorporating two application models and with synthetic traffic.

The paper is structured as follows. Section 2 surveys the literature on TLM for NoCs, comparing and contrasting the approaches presented with the present work. Section 3 motivates the work by describing the difficulties in accurate latency prediction, particularly in non-preemptive NoCs. Section 4 describes the NoC scenarios, specifying the synthetic and application traffic models used in the evaluation results. Section 5 specifies in detail the family of TLM models evaluated in the work, typically via pseudocode implementations. Section 6 evaluates the accuracy and execution time performance of the implementation under a variety of traffic models, and provides a discussion of the comparative merits of the various models in view of the results. Finally, Section 7 details potential extensions to the current work, and Section 8 concludes the paper.

2. Literature Review

The goal of transaction level modelling is to improve simulation speed by the abstraction away of low level events such as individual flit transmissions, in favour of boundary events. TLM is frequently associated with SystemC [7] although the methodology is suitably generic to be applied to other languages and simulation frameworks. The TLM 2.0 [8] framework models a VLSI system such as a NoC or SoC as groups of transaction initiators or targets (communicating nodes) and interconnects which transfer transactions from initiators to targets. In [9], SystemC TLM models are used for NoC simulation by treating the transmission across multiple arbiters as a single

transaction. However, since blocking delays upon the path are only estimated statistically, accuracy may be compromised in complex application models.

Schirner and Dömer [10] investigate the tradeoff between TLM accuracy and simulation speed, finding that TLM may potentially be four orders of magnitude faster than cycle-accurate models. However, the work introduces simplifications that reduce accuracy, with a potential average inaccuracy of 35% reported in timing user transactions for their most abstract TLM model. TLM models have been applied to the individual processing elements, and can retain accuracy if using a granularity larger than individual instructions (assisted by an earlier cycle-accurate static analysis phase). The approach presented in this paper is distinct from this earlier work, as our work involves the application of TLM to the NoC and not code execution on the PEs.

Bus TLM modelling is considered in Result-Oriented Modelling (ROM) [11] which optimistically predicts transaction delays, and retroactively corrects in the case of contention. ROM can provide error-free timing prediction, however, frequent cascading corrections produce a reduction in simulation event speed and require an increase in modelling complexity. Considering TLM models for on-chip interconnects, timing points can be identified from the protocol specifications of the bus [12]. Simulation speed improvements of up to two orders of magnitude can be obtained while retaining accuracy in comparison to a cycle-accurate model. This approach relies on accurate identification of timing points from the bus protocol specification, which is difficult as interconnects become more complex. In contrast, our approach

discovers preemption points for simulation dynamically during simulation, from the arrival of contending traffic from the application model.

In [13], a speed-up of 50 times for the TLM models predicting NoC interconnect latency compared to the cycle-accurate reference is shown, with accuracy of 99.9%. This is obtained by using local time references for individual tasks communicating over the NoC, and only synchronising when tasks are common initiators or targets of a single transaction. Modifications to the simulation kernel to use lightweight schedulers [14] with a common time reference were shown to produce a 38% speed up. However, this approach requires simulation kernel modifications, which our approach does not require. In another approach [15], simulation parallelisation has been explored to take advantage of multiple CPU cores on the host simulation machine. By effectively dividing the independent tasks, a speedup almost linearly proportional to the number of cores can be demonstrated.

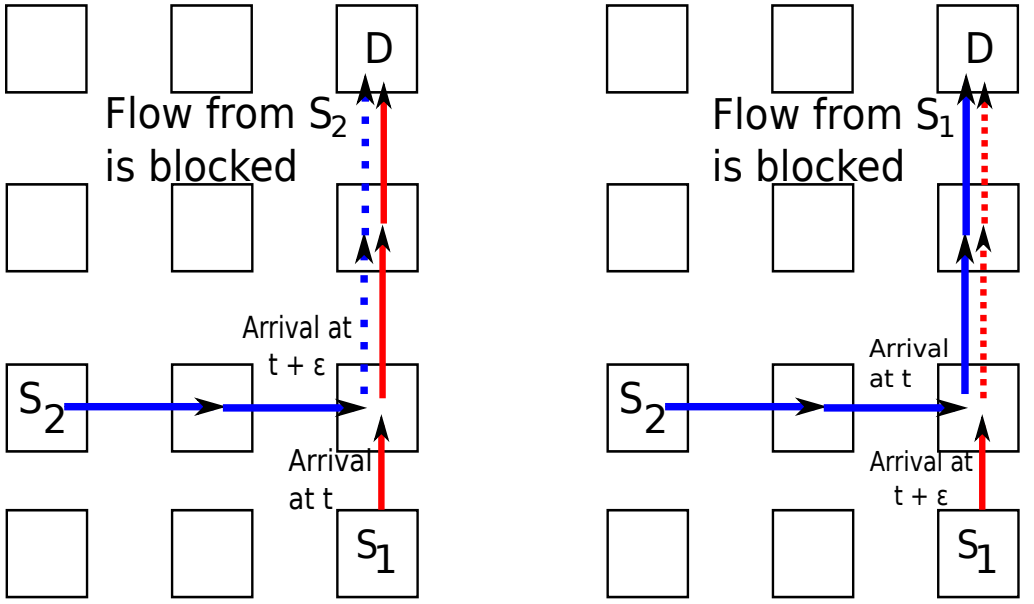
Existing work has covered simulation of wormhole NoCs [16], which reduce the total number of simulation events by simulating only packet headers and trailers. Our current approach requires the simulation of the progression of all flits since it is necessary to register their power consumption impact. Previous works by the current authors introduced a family of fast TLM algorithms which are further clarified in Section 5 and evaluated with new results. A TLM algorithm for priority preemptive NoCs [4] is referred to as TLM-PRE within this paper. This model was studied and evaluated for its power consumption accuracy in preemptive NoCs [5]. A fast non-

preemptive TLM model was applied to application task mapping in [6]. This non-preemptive model is referred to as TLM-NP in this paper. A more advanced non-preemptive NoC model was presented in [17], although the model presented in this current paper as TLM-NPD incorporates signficant alteration to its internal model of how flows advance through the network, in order to improve latency prediction performance.

3. Problem Description

It is likely that cycle-accurate simulation of NoC interconnect data transmission will prove prohibitive for future realistic application cases, particularly when evaluating a wide design space. In our earlier work on assessing and improving NoC simulation algorithm execution speed, cycle-accurate simulation of 2 seconds of execution of a target application required approximately 10 minutes [5]. Since the cycle-accurate framework operates at flit granularity, simulator events are required every time a flit advances through an architectural entity such as an arbiter or buffer. This not only scales in proportion to the amount of data transmitted, but leads to wasteful overheads in simulator state management and event scheduling which are unrelated to the internal state of the NoC models.

A key challenge is to improve simulation speed by reducing overheads, reducing the number of events that have to be scheduled and processed. By contrast, in our TLM models events are scheduled upon flow admission or upon the estimated time of removal of the flow, leading to event counts grow-



(a) Timing advantage for flow from S_1

(b) Timing advantage for flow from S_2

Figure 1: Small packet arrival timing offsets can determine which flow receives arbitration at contention points, potentially influencing flow latency, particularly in non-preemptive NoCs (based upon [17])

ing with the contention between the flows. However, it is important in this process to retain the accuracy of the system under network contention. Figure 1 illustrates a non-preemptive NoC in which two packet headers arrive at a common router with a small timing offset ϵ. Even a relatively small timing offset of a few cycles could determine which packet would be granted access to a contended resource such as an arbiter output port. This would result in the packet not receiving arbitration suffering blocking, and needing to wait until the completion of its interferer in order to progress. Correct prediction of the arrival timings of packet headers at contention points is therefore im-

portant in developing an accurate TLM simulation model. This is especially critical in a NoC without priority preemption, in which large-scale latency can be influenced by small-scale timing inaccuracies. By contrast, priority preemption in the NoC reduces the impact of timing prediction errors upon higher priority packets, since higher priority packets can always preempt a lower priority interferer in the case they require access to a contended resource. As long as the TLM model correctly respects the priority ordering in its arbitration decisions, the latency outcome will be close to correct.

However, in a non-preemptive NoC interconnect, the absence of preemption means that latencies are harder to predict. The most sophisticated TLM model presented in this paper (TLM-NPD) compensates for this by tracking the positions of individual flows as they advance and allowing the earliest flow to arrive at the contending arbiter to receive arbitration, with priority used as a tie-break in the case of simultaneous arrival. The structures of the relevant preemptive and non-preemptive NoC models are presented in Section 4.2.

Another factor that may be affected by inability to predict the precise contention patterns is the dynamic power consumption. In this paper, dynamic power consumption is evaluated using a simple model considering link wires as capacitors, and transmission of successive flits as constituting bit transitions that charge and discharge the link capacitances. This power model is based upon [18] and its implementation in our model specified more fully in [5]. Inaccuracies in modelling the outcome of contention may influence

power consumption accuracy, since bit transitions will occur in a different order. The precise degree of inaccuracy in both power and latency results will be quantified by the experiments performed in this work.

4. Scenario Description

This section presents various assumptions used in the evaluation of the transaction level models, together with detailing the NoC interconnect structures and traffic models that are used for experimental evaluation.

4.1. Assumptions

The following assumptions regarding the NoC interconnect and its structure and routing behaviour have been made throughout the work presented:

Regular grid topology A regular grid topology, with homogeneous links

XY routing To provide predictability in the routing structure

Wormhole switching In order to reduce intermediate buffering requirements, by allowing partially transmitted packets to remain in the NoC

Power model Dynamic power consumption is approximated by the number of bit transitions upon the NoC links. Although other factors are important in NoC power consumption, the long length and therefore high capacitance of NoC links results in link switching activity becoming a significant proportion (30% or more [19] [20]) of total NoC dynamic power consumption [18] [21] [22].

No deadlock is possible It is not possible for the system to be deadlocked, as long as packets are not injected into the system by the application model beyond the rate at which they can be serviced. This is assured since there is a unique priority index for each flow in the system, and it is possible to establish a total ordering over priorities, preventing circular waiting and leading to the impossibility of deadlock.

4.2. NoC Arbiter Models

The present work focuses upon two particular architectural constructs that can be accurately described using TLM; a NoC architecture incorporating priority preemption, and a non-preemptive NoC architecture. A description of these architectures follows below.

4.2.1. Priority Preemptive Arbiter

In the priority preemptive case it is assumed that the NoC uses a virtual channel architecture similar to QNoC [2]. In each input port, a different FIFO buffer stores the flits of packets arriving in different virtual channels (one VC is statically mapped globally for each priority level). The router assigns an output port for each incoming packet according to their destination, which can be determined simply using XY routing. Credit-based flow control [23] guarantees that data is only forwarded from one router to the next when there is sufficient buffer space to hold it. When the requisite buffer space is available at the recipient, flits for the highest priority virtual channel requesting arbitration are allowed to use it, ensuring that the highest priority

flows are always preferred and therefore that their blocking times are the lowest.

4.2.2. Non-Preemptive Arbiter

The second architectural construct is the non-preemptive NoC, based upon HERMES [24]. The major disadvantage of a priority preemptive NoC arbiter is its requirement for virtual channels and their associated buffering, in order to store flits from different priorities independently. The silicon area requirements due to increased buffering in preemptive NoCs can be excessive [25]. Therefore, this non-preemptive NoC arbiter is potentially a more viable solution for practical synthesis. A key difference from the architecture in HERMES is that although the system is non-preemptive, priorities are used as a tiebreak during arbitration when multiple flows arrive and contend for a busy output port upon the same arbiter.

4.3. Application Traffic Models

The primary challenge that makes static analysis of NoC interconnect throughout and latency performance infeasible is that such results can be heavily influenced by the traffic patterns employed. Synthetic test traffic generated according to statistical workload distributions may fail to capture realistic dependencies that exist between tasks, therefore failing to account for sequential interactions, such as one task transmitting packets in response to a request from an earlier task. On the other hand, timing or mapping characteristics of real application models may be unusually favourable to

one particular architecture. Therefore, three traffic models are considered in this paper, to incorporate the best features of both realistic scenarios and synthetic traffic as a system testing tool.

4.3.1. Autonomous Vehicle Application

The autonomous vehicle application [26] consists of 38 communicating tasks representing the multimedia processing (for video camera analysis, navigation and communication processing) of an autonomous vehicle. The AV application is used with a static task mapping as employed in our previous work [5], intended to manually balance the load within the NoC.

The original AV application model assumed the presence of release jitter, as a fixed percentage of the flow transmission period, typically 10%. Given that the communication latency of flows during transmission is short relative to the periods between transmission of the same flow, release time jitter was disabled for the simulations performed in this paper. This results in the simultaneous admission of bursts of flows to the network, increasing contention as in the situation described in Figure 1.

4.3.2. H264 Decoder

This test application consists of an implementation of the H264 decoder (`h264dl_mesh_4x4.rtp`) from version 1.1 of the MCSL benchmark suite [27], providing 51 tasks that model the distributed decoding of a multimedia process. The tasks are organised as a branching tree structure, and the decoding process is periodically triggered from the timing of a single clock pulse rep-

resenting the arrival of data into the system. This represents the decoding of multimedia data via a distributed SoC. Although the original benchmark structure did not include priorities, flow priorities have been assigned, in order of the flow identifiers provided in the H264 benchmark definition. The only alteration that has been introduced is the use of a modified task mapping in order to produce additional contention, and to ensure that no source-destination pairs are mapped onto the same cores.

4.3.3. Synthetic Traffic

The synthetic traffic generator injects tasks during system execution. A fixed number of tasks are generated within the system every *task generation interval*, and assigned a fixed number of peers. Task peers, priorities and message sizes are chosen randomly, given an initial seed value that allows compatibility to be ensured between experimental and reference cycle-accurate models. Newly created tasks are dynamically mapped to the network processing elements according to a simple load minimisation mapping that seeks to balance the task loading across individual NoC cores. The advantage of this traffic model is that its random selections generate significant contention between flows of widely varying priorities and lengths. This serves to test the latency accuracy of the different TLM models with increasing levels of load as the network is populated during execution, with some packets experiencing multiple blockings during advancement to their destination.

5. Models and Implementation

5.1. Overview

This section defines the transaction level (TLM) and cycle-accurate reference models that are investigated and contrasted within the paper. In this section the fundamental design decisions and structure of each model are explained and justified, together with general algorithm descriptions that specify their underlying logic. Details of the reference implementation of the particular model within the simulation framework are also provided. For clarity the five models are referred to as CA-PRE, CA-NP, TLM-PRE [4] [5], TLM-NP [6] and TLM-NPD. CA-NP and CA-PRE are the reference cycle-accurate implementations, while TLM-PRE, TLM-NP and TLM-NPD are the experimental TLM models evaluated here. Table 1 summarises the key characteristics of these models. Criteria used for classifying the models include their priority preemption behaviour (priority preemptive or non-preemptive), their cycle-accuracy (whether they function as a reference or an experimental test model) and, for the TLM models, their granularity of modelling. The granularity of modelling refers to whether the models store and use individual link state in their NoC modelling decisions, or whether the complete flow and its associated source-destination route is the fundamental abstraction.

The relationship of the models to Cai and Gajski's TLM classification [3] is also defined in Table 1. The reference models CA-PRE and CA-NP are examples of Cai and Gajski's classification as a time-accurate communication

model, in that they provide a cycle-accurate model of communication but an abstraction of computation on the processing elements. The TLM-PRE, TLM-NP and TLM-NPD models are examples of bus-transaction models, in that they also approximate communication timings using the mechanisms defined in Sections 5.4, 5.5 and 5.6.

REFERENCE IMPLEMENTATIONS				
Model Name	Preemptive?	Cycle-accurate	Granularity	Cai-Gajski [3]
CA-PRE	✓	✓	Flit-level	Time-accurate
CA-NP	✗	✓	Flit-level	Time-accurate
TLM EXPERIMENTAL IMPLEMENTATIONS				
Model Name	Preemptive?	Cycle-accurate	Granularity	Cai-Gajski [3]
TLM-PRE	✓	✗	Flow-based	Bus-transaction
TLM-NP	✗	✗	Flow-based	Bus-transaction
TLM-NPD	✗	✗	Link-based	Bus-transaction

Table 1: The characteristics of the NoC models

Throughout the paper, the concept of *flows* is employed. A flow is defined as a sequence of associated flits from the same source and destination which travel together to their destination. Flows have an associated route, which represents the links that they traverse on the route to their destination. Packets therefore represent the simplest example of a flow (being the complete sequence of flits including a header and all transmitted data), although a flow can also represent a partial packet which has not completely entered its final flit into the NoC. All the TLM models presented track the state of flows using the concept of *flit position*, which represents the progress

of the header flit of the packet along its route from the source node to its destination. Flit position is required in the tracking of power consumption as well as latency, since it provides information on the position of individual flits relative to the links upon the route it traverses. Consider the progress of a single packet under transmission from source to destination via intermediate routers in an otherwise idle NoC, as shown in Figure 2.

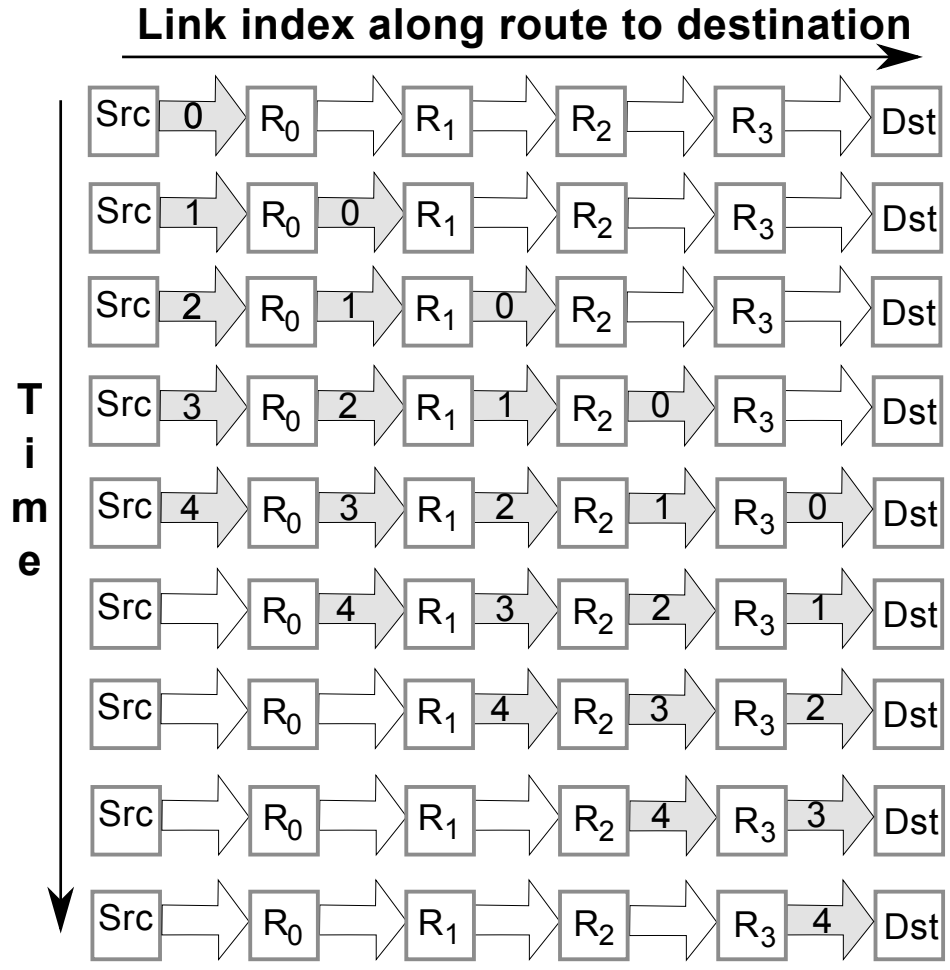


Figure 2: Structure of the algorithm and flow progression to the destination (based upon [5])

Potential values for flit position at a given time are in the range $0 \leq F_p < N + H - 1$, in which the hop count of the route is given by H and N refers to the total number of flits (including header flits) that the packet contains. A growing phase occurs when $F_p < H - 1$, in which data flits are in progress to the destination, but some links further along the route can be idle as data flits have not yet reached them. Following completion of the growing phase, every link on the route has transmitted a header flit for this flow, and the wormhole process can continue to operate normally. The flow finally experiences a shrinking phase in which $F_p \geq N$. In this phase no additional data flits are being injected, but the final flits are still in progress and have not yet reached their destination. During transmission in the example in Figure 2 this flow goes through flit positions from 0 to 8 at successive time intervals.

5.2. CA-PRE - Preemptive Cycle-Accurate

The first model considered is a preemptive cycle-accurate model, referred to as CA-PRE (Figure 3a). The figure shows the implementation of a test NoC within the Ptolemy II simulation environment [28].

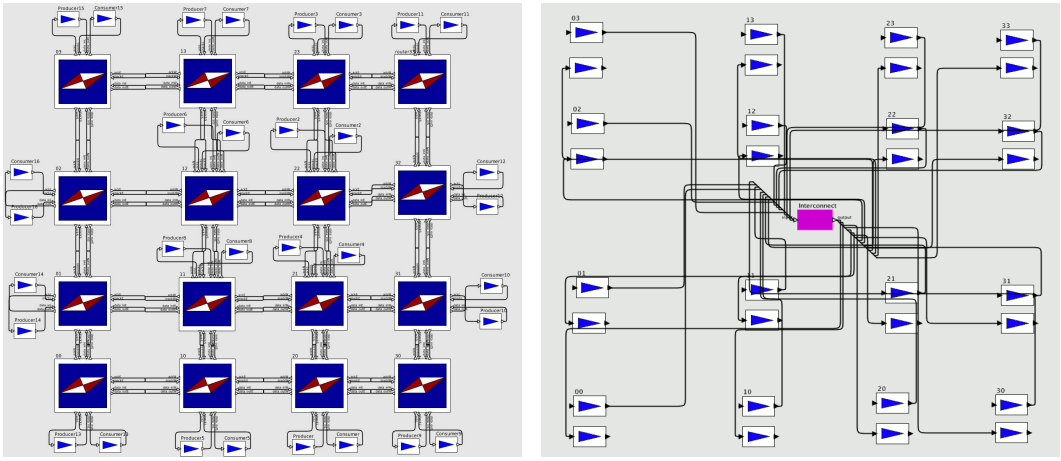
The CA-PRE model is structured to represent NoC components such as arbiters, buffers and their associated ports as entities within the simulator. On each transmission of a flit from one network component to another, simulator events have to be scheduled in order to process the arriving flit. For example, flits arriving at an arbiter go through a simulated arbitration process (using the multi-port mechanism of the Ptolemy II simulator to model

virtual channels) which processes the virtual channels in priority order, and ensures that the highest priority requesting arbitration is sent out upon the relevant output port. Priority sorting ensures that each output port is only used once for transmission per cycle, and port multiplexing state is tracked to model the arbitration delays. Credit based flow control is used to control the propagation of data through the network, allowing additional flits to enter input buffers as they empty.

Delays are defined and timing information is tracked such that the propagation of flits through the system is cycle accurate, but due to the requirement for simulation events to model the propagation of each flit through multiple simulation entities such as arbiters and buffers, time consumed in simulation scales in proportion to the amount of data sent. This model is therefore used as a reference implementation to validate the execution time performance and accuracy of the TLM-PRE TLM model for preemptive NoCs. Although the current implementation uses the Ptolemy II simulation environment, the model structure is sufficiently generic to be applied to any simulator that has objects with state, and port connections between them that trigger events upon message arrival.

5.3. CA-NP - Non-preemptive Cycle-Accurate

The second model considered is a non-preemptive cycle-accurate model, referred to as CA-NP. The fundamental structure of the CA-NP model at the simulator design level is identical to CA-PRE. The key distinction exists



(a) CA-PRE cycle-accurate simulation model (peer to peer NoC router topology)

(b) TLM-PRE TLM simulation model (entire NoC as single central entity)

Figure 3: The implementations of the simulation models for CA-PRE and TLM-PRE

within the arbitration code, in that their ports do not model the independent buffering provided by virtual channels. In the alternative non-preemptive design employed instead, the first flit to use a particular output port locks that input-output port combination until the transmissions have finished. All other requesting flows awaiting transmission on that output port of the arbiter are blocked until completion of the in-progress flow, and therefore cannot access the output port requested regardless of their priority level. However, the fact that the input ports are processed in priority order ensures that when two flows contend simultaneously for an output port, the priority levels are used as a tiebreak to determine which flow receives arbitration.

5.4. TLM-PRE - TLM Preemptive Single-Flow Activity

The TLM-PRE TLM preemptive model presented in this section is described more fully in [4], [5] and [29]. The goal of the algorithm is to simulate a priority preemptive NoC with a greatly reduced frequency of events. Events are only processed upon flows entering or exiting the interconnect, or at anticipated completion times for existing flows, in which simulation state must be updated to ensure consistency. In the implementation of this and the other TLM algorithms, an abstraction of the entire NoC is represented as a single simulation entity (depicted centrally in Figure 3b). All abstractions of processing elements are directly connected to this Interconnect entity.

The TLM-PRE algorithm is presented in Listing 1. The algorithm operates upon the set of currently active flows, processing them in priority order. The algorithm uses the concept of interference sets in defining contention between flows:

Definition 5.1. *The* interference set *of $flow_i$ is composed of flows of higher priority than $flow_i$ with routes sharing at least one link with the route of $flow_i$. Flows are considered to be in the interference set regardless of whether flits from an interfering flow request arbitration on those shared links simultaneously with $flow_i$.*

For a particular $flow_i$, the algorithm tracks its current activation status $active_i$, and its remaining flits for transmission $flitstosend_i$. During the update event, the progress of active flows is updated using the last activation

time t_{ai}. Completed flows are removed from the flow table. Activation of any flows in the interference set of $flow_i$ inactivates $flow_i$, preventing it from transmitting. Flows without any active members of their interference set are activated. Iteration over the flow set proceeds in priority order from the highest to the lowest, ensuring that the activation decisions respect flow priority. A further update event is scheduled at the expected completion time of $flow_i$. Therefore, simulation events only occur when flows enter or leave the NoC, or to update state at an expected flow completion time.

The *trackPower* function (Listing 2) is responsible for dynamic power consumption modelling for a particular flow. It allows dynamic power consumption of multiple flits to be tracked in a single simulator event, over a time window in which it was known that the flow had exclusive access to particular links. The *trackPower* function is invoked in two circumstances during the *update* function; when $flow_i$ completes, or when another higher priority flow inactivates $flow_i$.

The *trackPower* function operates as follows. Firstly, the *flowing time* can be computed by subtracting the current time from its last activation time. The number of flits transmitted over this flowing time is computed using the flow's flit transmission rate, which is by default assumed equal to the system clock speed. A range of flits between $startFlitPos$ and $endFlitPos$ is therefore determined. Iterating over both flit position and link index upon the route, the flits which crossed a link are looked up from the flow's associated data. Then links are notified of the presence of these flits using

Listing 1: Pseudocode for updating a list of flows with power tracking (from [5])

```
1 update(currentTime) {
2   for each flow_i in flowlist {
3     if (active_i) {
4       flitstosend_i = flitstosend_i - sentFlits(currentTime -
            t_ai);
5       t_ai = currentTime;
6       if (flitstosend_i == 0) {
7         remove flow_i from flowlist;
8         trackPower(flow_i);
9       }
10      for each flow_n in interference_i {
11        if (active_n) {
12          active_i = false;
13          trackPower(flow_i);
14        }
15      }
16    } else {
17      if (!active_n for all flow_n in interference_i) {
18        active_i = true;
19        requestUpdate(currentTime + basicLat(
              flitstosend_i));
20      }
21    }
22  }
23 }
```

Listing 2: Pseudocode for TLM dynamic power tracking (from [5])

```
1 trackPower(currentFlow_i) {
2   flowing_time = currentTime - t_ai;
3   endFlitPos = startFlitPos + flowing_time *
        flitTransmissionRate(currentFlow_i);
4   for (flit_index = startFlitPos to endFlitPos) {
5     for (link_index = 0 to hopLength(currentFlow_i)) {
6       flit_for_link = flit_index - link_index;
7       if (flit_for_link >= 0 and
8           flit_for_link < endFlitLimit(currentFlow_i)) {
9         link = getLink(currentFlow_i, link_index);
10        flit = getFlit(flit_for_link);
11        registerTransmission(link, flit);
12      }
13    }
14  }
15  setFlitPos(currentFlow_i, endFlitPos);
16 }
```

registerTransmission. This function handles dynamic power consumption tracking for a particular flit and link, by tracking and accumulating bit transitions between sequential flits using a particular link. Since the algorithm iterates over flit indices in its outer loop and along the route links in its inner loop, the power consumption impact of previous flits will have been registered in advance. It is therefore possible to extend the present work to apply more advanced power models which incorporate cross-coupling between adjacent links, e.g. [30] [31].

The example in Figure 2 shows a particular example of power tracking execution. Using a conventional cycle-accurate model in which every flit transmission is directly modelled using low-level simulation events, simulator events would be required upon every grey arrow. The TLM power tracking described is able to model every flit transmission depicted with a single simulation event, as long as the flow is not preempted during transmission. In the case of preemption, an additional simulation event would be required, which would handle modelling the completed flits up to the preemption point.

Consider Figure 2 to represent the algorithm's internal processing. Moving from left to right along the horizontal axis (modelled in the inner loop of Listing 2) represents advancing flits in single steps along their route to their destination, with each gray arrow representing a power registration event for a single flit. The rows represent sequential time-steps, which are handled in the outer loop of Listing 2. Therefore, the link from processing core **Src** to router $\mathbf{R_0}$ would receive in sequence flits $0, 1, 2, 3, 4$ for power tracking reg-

istration. No link activity is modelled upon the vacant links closest to the destination for flit positions 0 to 3, since the flow is still in its growing phase and its flits have not reached this point yet. Correspondingly, the shrinking phase occurs when the links closest to the source are idle.

5.5. TLM-NP - TLM Non-Preemptive Single-Flow Activity

In a preemptive NoC, it is always possible for a newly arriving higher priority flow to preempt another transmission in progress. The TLM simulation model presented in this section, referred to as TLM-NP [6], operates upon non-preemptive NoCs which do not permit such preemption. Upon flow admission, flows calculate their interference sets and activation status with any other flow, in order to determine which flows will potentially interfere.

The pseudocode for the TLM-NP algorithm is the same as the previously described TLM-PRE (Section 5.4), with a modification to the sorting order. Instead of sorting flows for processing in order of priority, flows are sorted on their distance to the next contention point (the link in the NoC at which the routes of the two flows intersect). Priority is used as a secondary sorting criterion if the distances to the next contention point are equal. For example, two simultaneously arriving flows which wish to use the same link are sorted such that the closest one to the contending router is processed first. Priority is used as a tiebreak if their arrival at the point of contention would be simultaneous. During the update event the flow processed first will reach the contention point, is allowed to claim it and is set to active. Once a flow has

passed through the contention point, then the model assumes that it cannot be preempted and will flow until its $flitsToSend$ is zero.

5.6. TLM-NPD - TLM Non-Preemptive Dynamic Link Claiming

When one or more flows which share a link in common are active in the network simultaneously, the TLM simulation algorithms presented in the previous sections have only allowed one of the them to be active simultaneously. This may result in the simulation producing overestimates of latency, since the model is too conservative in the temporal separation it provides. In the real hardware implementation of a non-preemptive wormhole switching NoC, both flows could advance through the arbiters along their routes in parallel, until the latest of the two reaches the arbiter at which they are blocked or contend for an output port. In the case of simultaneous arrival at an arbiter, flow priorities would determine which would receive arbitration.

The TLM-NPD model is introduced in order to compensate for this, by allowing individual links upon the route to be claimed dynamically. The TLM-NPD model is the most sophisticated presented within this work, since it models this situation by allowing multiple flows upon intersecting routes to proceed simultaneously, claiming the links upon their arbiters up until the point at which they will be blocked. This is therefore much more likely to accurately predict latency in the problem case defined in Figure 1.

An earlier form of this non-preemptive NoC model (TLM-NPD) was presented in [17], although the model presented here has been modified to im-

prove latency prediction performance under contention, by restricting flow advancement during the growing phase to a single flit at a time. By contrast, when a number of growing flows existed which attempt to request access to multiple links, the early version of the algorithm described in [17], would advance system time in a single operation, granting access to the contended links to whichever flow was selected. This selection criteria was based on a criterion of dominance, which considered flit positions to calculate distances to the first contention point, as well as priorities. The dominance ordering was computed up front and used as a sorting order. The necessity to advance system time in relatively course steps while assigning dominance in this form produced problems in accurate link claiming prediction in complex scenarios, when high contention lead to multiple blocking events occurring over the update interval. The dynamic flow advancement in single flit steps in the algorithm presented here as TLM-NPD solves this problem.

The increased reliance upon dynamic calculations means that the logic for flow admission using in TLM-NPD is simplified. Upon admitting a flow to the NoC, it is merely added to the flow table. Route intersection checking or interference set (Definition 5.1) calculation is no longer required during flow admission, since the dynamic flow move calculation during the update events replaces them. The simulation now uses a time window start *lastUpdateTime* which records the time over which the interval should be reconstructed. During flow admission, if no other flows are present in the system, *lastUpdateTime* is set to the new flow admission time.

Listing 3: Pseudocode for the TLM-NPD algorithm: update of a list of flows

```
1   update(currentTime) {
2     int arbiterPlusLinkPeriods = arbiterPeriods + 1;
3     double time = lastUpdateTime;
4
5     while (time <= currentTime) {
6       // Process flows sorted in priority order
7       foreach (flow_i sorted by priority) {
8         if (flow_i.isGrowing()) {
9           headLink = flow_i.headLinkAtFlitPosition();
10          if (headLink != null and !headLink.
                isClaimed())
11            tryGrowAdvance(flow_i, time, headLink);
12        } else tryFlowAdvance(flow_i, time);
13      }
14      time += minFlowPeriod();
15    }
16
17    lastUpdateTime = currentTime;
18
19    foreach (flow_i sorted by priority) {
20      if ((flow_i.flitPositionFinished())) {
21        removeFlow(flow_i, currentTime);
22      } else {
23        completionTime = flow_i.timeToCompletion(
              currentTime, arbiterPlusLinkPeriods);
24        scheduleUpdateAt(completionTime);
25      }
26    }
27 }
```

Listing 4: Pseudocode for the TLM-NPD TLM function tryGrowAdvance

```
1   tryGrowAdvance(flow_i, time, headLink) {
2     if (time >= (flow_i.lastMoveTime+minFlowPeriod()*
        arbiterPlusLinkPeriods)) {
3       // Claim it
4       headLink.claimLink(flow_i, time);
5       // Move forwards one, update move time, track
          power
6       moveForwardTrackPower(flow_i, 1, time,
          currentTime);
7   }
```

Listing 5: Pseudocode for the TLM-NPD TLM function tryFlowAdvance

```
1   tryFlowAdvance(flow_i, time) {
2     if (time >= (flow_i.lastMoveTime + flow_i.flowPeriod)
        ) {
3       // Move forwards one, update move time, track
          power
4       moveForwardTrackPower(flow_i, 1, time,
          currentTime);
5
6       if (flow_i.isShrinking()) {
7         // Flow tail links can be released
8         tailLink = flow_i.tailLink();
9         if (tailLink != null) {
10          tailLink.releaseLink(flow_i, time);
11        }
12      }
13    }
14  }
```

The pseudocode for *update* is presented in Listing 3. The first section of the algorithm operates differently to the other TLM algorithms, in that it is based upon advancing all flows simultaneously through the network in single-flit steps as blocking permits, rather than computing the maximum move limited by time and then advancing the flow in a single operation. This is necessary in order to accurately model the claiming and releasing of link locks when shorter flows are present in the network, since the release of the lock upon a link may allow another flow behind it to advance. The outer loop of the algorithm between lines 5 and 15 advances time forwards in increments of the minimum processing period of flows in the network (by default, this is equal to the system clock speed). The inner loop in lines 7 to 13 iterates over all active flows in priority order. This priority-ordered iteration ensures that the all the higher priority flows in a flow's interference set (Definition 5.1) have been considered to check if they are eligible to claim a link under contention, which respects the priority-favouring design of the real hardware system.

If the flow is in its growing phase (in which it has not yet acquired all the necessary links on the route to its destination), then the next link that will be required by the advancing head is tested to determine if it is free or claimed. If it is free, then *tryGrowAdvance* (Listing 4) is executed. This compares the iteration time with the time of the flow's next permitted move, in order to determine if it can proceed yet. If this condition is met, then the flow is permitted to advance forwards. Advancing a growing flow first

consists of claiming the relevant link at the head. This is followed by (in the function *moveForwardTrackPower*) incrementing the flow flit position, and tracking the power impact of registering the single flits in one move.

If the flow is not in its growing phase (that is, its header has claimed all the links to its destination) then it is immune to blocking given that the NoC design assumed is non-preemptive. Therefore, *tryFlowAdvance* (Listing 5) is called to determine whether the flow can move forward. *tryFlowAdvance* verifies that the time elapsed since the last move is equal to or greater than the flow processing period, and if so advances the flow. If the flow is in its shrinking phase, then it is also necessary to free the link at the tail every time it moves forwards.

The last step for *update* in lines 17 to 26 of Listing 3 is to register the time of this last update, and to work out which flows have completed and which need to be scheduled for further processing. Accordingly, the code iterates over all flows within the network, testing their flit positions to determine whether these flows have already completed. If they have completed, then they are removed from the network. If not, then their estimated completion time (assuming no further blocking events occur) is computed. An additional update is scheduled at this time in order to handle their remaining transmissions.

One particular issue involved that may produce inaccuracies in the TLM-NPD model is consideration of buffering in the network. In the reference cycle-accurate NoC model there will exist storage in the form of input buffers

attached to particular input ports (one buffer per virtual channel in the priority-preemptive NoC). This allows several flits to be stored within one arbiter input buffer, which can produce a bunching up of multiple flits at intermediate buffers. However, for modelling simplicity, the concept of flit position used for the TLM models assumes that all flits are arranged sequentially along their respective route, without bunching up within the buffers of intermediate routers. This therefore can lead to some incorrect predictions of flit position and the resulting preemption behaviour by the TLM models, as examined in the results.

6. Results

This section presents simulation results to evaluate the simulation execution time performance, latency and power consumption modelling accuracy of the various TLM models compared to the cycle-accurate reference. The default parameters used throughout the simulations are given in Table 2. If alternatives to these default parameters are used in any particular simulation run, it will be specified during the description of the particular experiment.

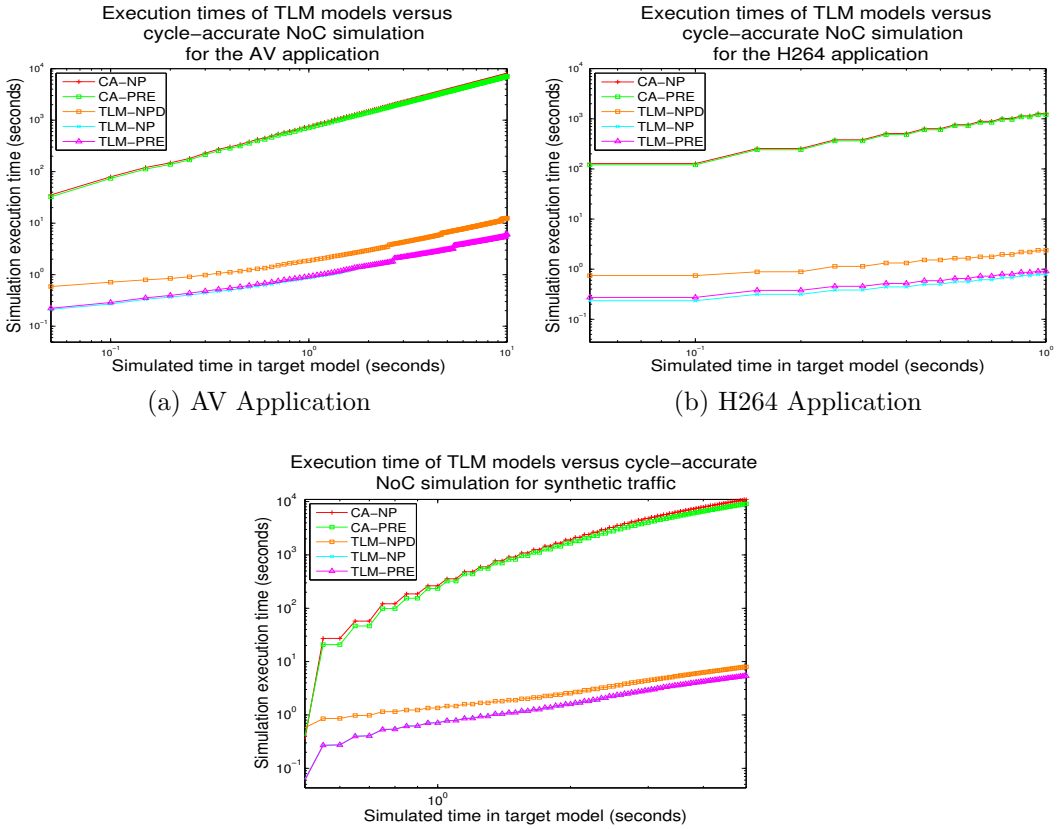
6.1. Simulation Execution Time Performance Results

A major issue for NoC simulations is improving their overall execution time while retaining accuracy, in order to allow realistic application cases to be simulated. This subsection considers the wall-clock execution times of the TLM simulation models defined in this paper. The models are compared

Parameter	Description	Value
L_{arb}	Arbitration latency (cycles)	3
NoC Dimensions	The structure of the XY grid for application cases	4x4
NoC Dimensions	The structure of the XY grid for synthetic traffic	6x6
D_{AV}	Simulated execution duration of AV application	10s
D_{H264}	Simulated execution duration of H264 application	1s
D_{SYNTH}	Simulated execution duration of synthetic traffic	5s
Flit width	The number of bits per flit	32
t_{gen}	Synthetic traffic task generation interval	0.5s
t_{msg}	Synthetic traffic message transmission interval	0.2s
P	Task peer count (fixed)	2
T_{MAX}	Maximum number of tasks in the network	72
M_{min}	Synthetic traffic minimum message size	20 flits
M_{max}	Synthetic traffic maximum message size	100 flits

Table 2: Parameters for the overall simulation methodology, and for synthetic traffic generation

against reference cycle-accurate models for the three application cases considered. Figure 4a illustrates the execution time performance of the TLM and reference models for the AV application, indicating that the simpler TLM-NP model is approximately 3.1 orders of magnitude faster than cycle-accurate. The TLM-NPD algorithm is approximately 2.7 orders of magnitude faster. This relative reduction in speed for TLM-NPD compared to TLM-NP occurs due to the requirement for the TLM-NPD algorithm to iterate in a nested loop forwards over the time window and also over all active flows, which is more time-consuming than the logic of TLM-NP. Considering the preemptive models for the AV application, the TLM-PRE algorithm has effectively equivalent execution time performance to TLM-NP, since the structures of



(a) AV Application

(b) H264 Application

(c) Synthetic Traffic

Figure 4: Performance of the transaction-level model simulations compared with a cycle-accurate reference model

both algorithms are very similar. Similarly, the preemptive cycle-accurate reference CA-PRE displays very similar execution time performance to CA-NP, since their simulation structure and event counts are very similar.

Figure 4b illustrates the execution time performance of the H264 application. The simulation duration is much shorter, and there is a smaller increase in execution time for the TLM models since the tree structure of this applica-

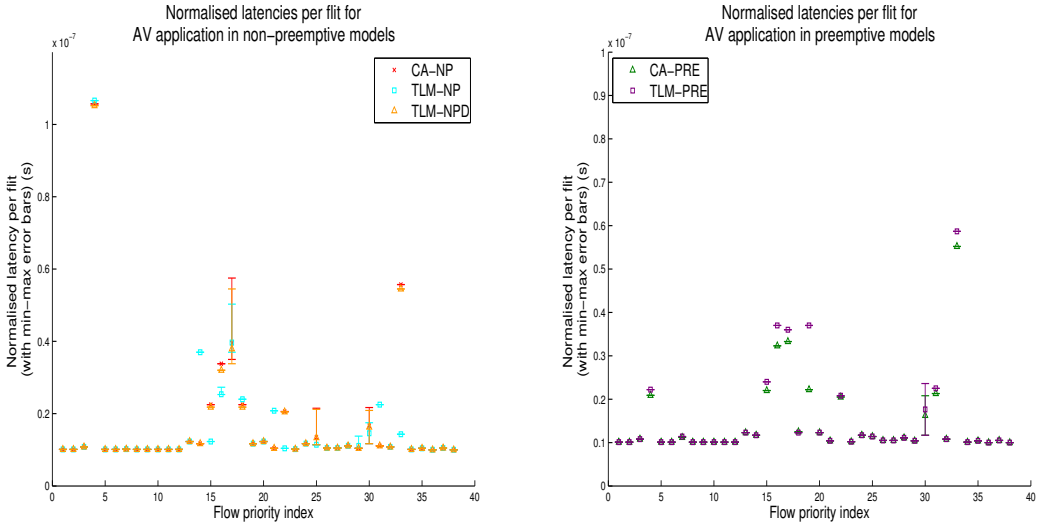
tion produces lower contention. The TLM-NP and TLM-PRE models exhibit 3.1 orders of magnitude execution time performance improvement over the CA-NP and CA-PRE cycle-accurate simulations. The TLM-NPD model delivers 2.7 orders of magnitude improvement. The vertical steps shown in the results arise due to the staggered release of packets from the central triggering clock of the H264 application, which produces variations in network loading at different sampling intervals.

Figure 4c illustrates the equivalent execution time performance for the synthetic traffic generator, starting with the injection of the first tasks into the system. Since tasks are generated dynamically for this simulation starting with an empty network, at the beginning of the simulation model-independent features such as simulation setup, data generation and packet generation are dominant over the modelling of simulation transmission events. Therefore, at the start the execution timing performance advantage of the TLM simulations over cycle-accurate is comparatively low. However, as additional tasks are generated as simulation execution progresses, the load placed upon the NoC and total flit transmission rate throughout the NoC increases. Since the cycle-accurate models require simulation events per every flit transmitted, the advantage of the TLM models becomes greater, producing approximately 3.3 and 3.1 orders of magnitude timing performance improvement for the TLM-NPD and TLM-NP models over cycle-accurate. As in the AV application case, TLM-PRE provides an equivalent execution time performance improvement to TLM-NP, due to the similar application structure.

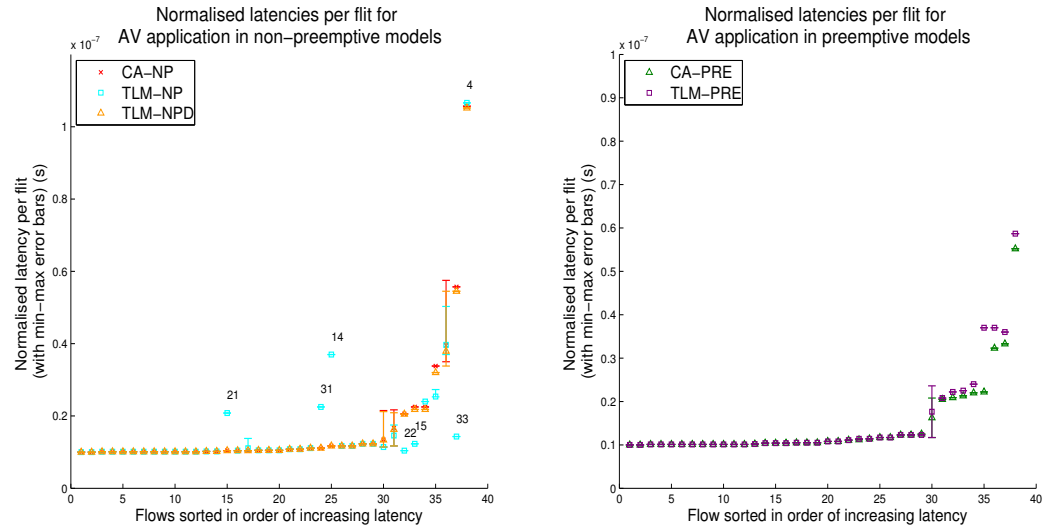
6.2. Latency Accuracy

The communication latencies produced during simulation by the TLM models will vary according to how their algorithms handle the contention between flows. This section presents and compares the normalised latencies delivered for preemptive and non-preemptive TLM simulation models, grouping packets according to their priorities. Normalised latency refers to the latency per flit, that is, the total communication latency divided by the number of flits transmitted. The total latency experienced by a packet during transmission depends on the contention experienced during transmission, which may change due to the activities of other flows at the transmission interval. Therefore, a maximum-minimum range is specified at each priority level. Priorities are defined so that the low priority index values represent the highest priorities (e.g. priority 1 is therefore the highest priority in the system). All packets transferred between a source-destination pair in the application model use the same priority.

The latencies for the autonomous vehicle application are considered in Figure 5a, covering results occurring over 10 seconds of simulation runtime. The results show that the accuracy of the TLM-NPD algorithm is overall very high, with the largest error in the maximum normalised latency under-estimation upon flow priority 17 corresponding to 15 flit-times (0.25 flit-times per message flit, with a message size of 60 flits). By contrast the TLM-NP model exhibits large latency errors in several flow priorities, particularly overestimates at priority 14 and a major underestimate for priority



(a) Priorities as in application



(b) Flows sorted in latency order

Figure 5: Latency results for the AV application under both preemptive and non-preemptive models

33. This occurs since the simpler TLM-NP algorithm in incapable of tracking precisely where contention occurs during the progression of a flow during its growing phase. In the preemptive case, the TLM-PRE algorithm is more accurate, since a preemptive architecture is inherently more predictable in its timing. The only flow priority for which the latency is particularly inaccurate is 19, in which there is an underestimate since the TLM-PRE algorithm cannot model accurately contention during the growing and shrinking phases of routes, conservatively assuming the route is active throughout. Figure 5b shows the flows from the same simulation rearranged in order of increasing mean normalised latency (in the cycle-accurate reference model), to demonstrate the general trend in TLM simulation accuracy for flows with increasing contention.

For the TLM-PRE algorithm for the AV application case, the normalised latencies are overall a closer match to the cycle-accurate case, since the scenario is overall more predictable. The largest latency error occurs for priority 19, in which the normalised latency is approximately 66% higher for the cycle-accurate reference.

For the H264 application (Figure 6) in the non-preemptive case, both TLM-NP and TLM-NPD correctly model overall structure of the latency 'ramp-up' effect with priority that exists for low and high priorities. This feature of the application arises as a result of the fan-out inherent in two places in the H264 application, in which one source task transmits simultaneously to multiple destination peers. Given the complex mapping and

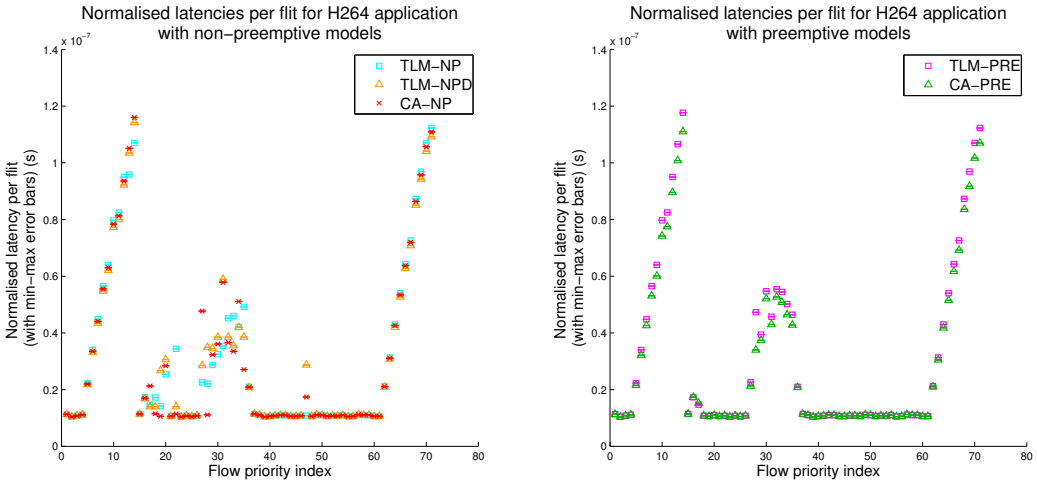


Figure 6: Latency results for the H264 application under both preemptive and non-preemptive models

simultaneous application release, the model is very sensitive to cascading latency errors, and for both TLM-NP and TLM-NPD there are two transmissions that are predicted incorrectly (priority 22 for TLM-NP and priority 28 for TLM-NPD). For the H264 preemptive case, the overall application structure is more predictable since the NoC is preemptive and there is less dependence on exact details of timing. This produces lower errors between TLM-PRE and the CA-PRE cycle-accurate reference.

For the synthetic traffic simulations, results are displayed using flow resorting (in order of increasing mean cycle-accurate latency) only. Since the flow priorities are randomly selected and there is no defined application structure to the network, flow patterns are difficult to process intelligibly without resorting. Figure 7 displays the resorted normalised latency of flows. For the TLM-NP model, mean normalised latency values are highlighted with black

markers in addition to the TLM-NP data series. It is clear that the TLM-NP algorithm provides a very poor estimate of latency in the non-preemptive case, in that there is no relation between the shape of the cycle-accurate curve and the TLM model. This occurs since the synthetic traffic produces much higher contention than the other application examples, given that the task peering relationships and mappings are randomly generated. Many flows are blocked two or more times during their growing phase, which given the wide variety of flow lengths and mappings can produce normalised latencies per flit up to five times higher, or twice as low as the cycle accurate (CA-NP) reference. By contrast, TLM-NPD provides much closer estimations of cycle-accurate latency, with an overall close correspondence between the mean values for TLM-NPD and the cycle-accurate reference. This is due to its ability to track the timings and advancements of flows and therefore anticipate which flow will receive arbitration in the event of a small timing offset.

In the CA-PRE and TLM-PRE model comparison, it is notable that in most cases the TLM model produces a latency above the reference cycle-accurate model. However, occasionally the minimum latency is below the minimum latency of the cycle-accurate reference. This can occur due the dependencies between flows, in which a latency prediction error for one flow in the TLM model can cause a dependent flow to miss contention that would have occurred in the cycle-accurate model.

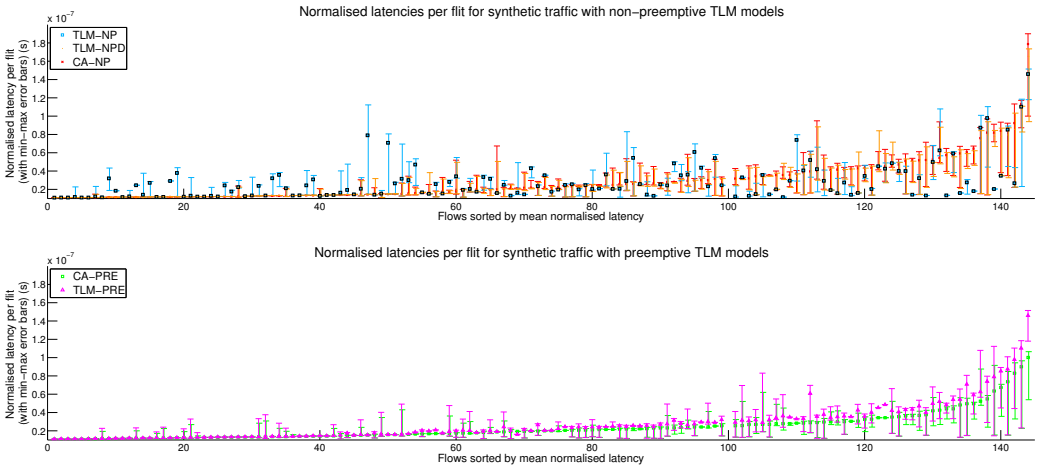


Figure 7: Latency results for synthetic traffic under both preemptive and non-preemptive models

6.3. Worst Case Latency Errors

As a summary, the maximum worst-case normalised latencies (per-flit latency) errors generated by the various TLM models are presented in Table 3. These errors are computed by comparing the peak normalised latency produced a TLM model against the latency produced by the relevant cycle-accurate model. The priority level which produced the latency error is also annotated.

It is notable that considering the non-preemptive cases, the TLM-NPD model is significantly more accurate than the TLM-NP model predictions in the AV application and synthetic traffic cases. Particularly in the synthetic traffic case, there are a large number of priority levels which produce a high relative error for TLM-NP (as depicted in Figure 7) in addition to the worst case value given in the table. In the H264 case in which there is a complex

mapping and the significant admission of packets to multiple destinations, the worst case normalised latencies of the two models are approximately equal.

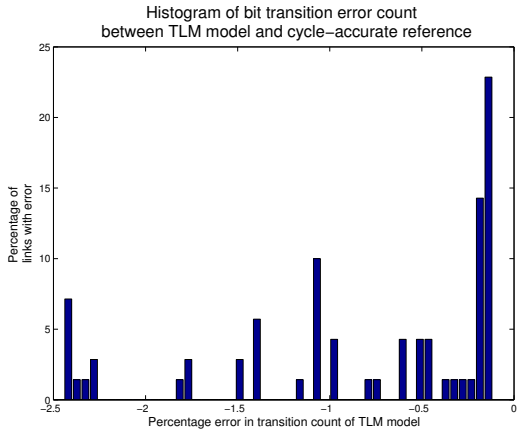
Application	TLM-NP	TLM-NPD	TLM-PRE
AVApp	-74.3% (priority 33)	-5.2% (priority 17)	+66% (priority 19)
H264	+218.0% (priority 22)	+218.0% (priority 28)	+39% (priority 28)
Synthetic traffic	+687% (priority 76)	+32% (priority 29)	+42% (priority 70)

Table 3: Maximum latency errors experienced by the TLM models

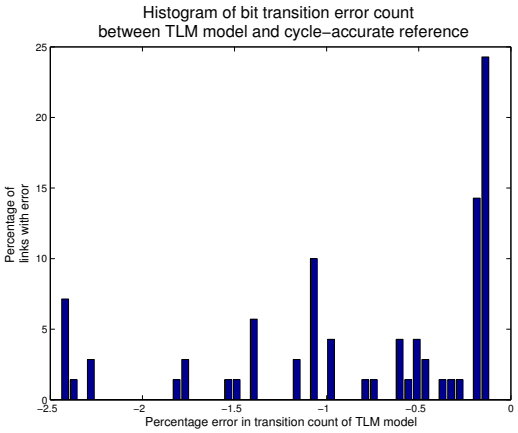
6.4. Link Transition Modelling Accuracy

This section considers the error in NoC link dynamic power consumption modelling for the various TLM models, compared to the cycle-accurate reference. The dynamic power consumption on the links is approximated using a model which counts the number of bit transitions upon the NoC links. Although this does not include power consumption effects obtained from switching logic or power coupling costs, due to the length of NoC links they comprise a significant individual source of NoC dynamic power consumption [18] [32] [21] [22] [19]. The results are presented as histograms indicating the proportion of links in the network which exhibited the indicated error, which allows the distribution of link errors to be examined.

Figure 8a demonstrates the link bit transition errors for the AV application for the TLM-NP model, indicating that the majority of links have dynamic power estimates within 0.1% accurate. The maximum inaccuracy

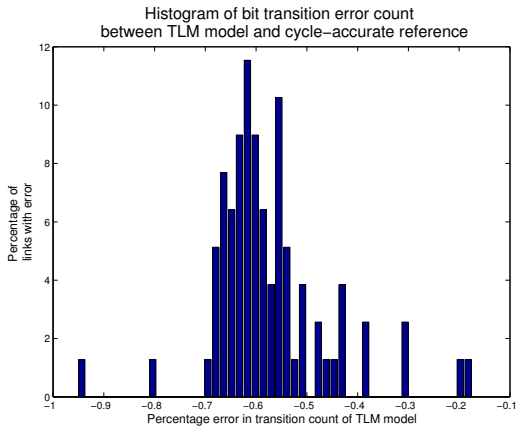


(a) TLM-NP model for AVApp

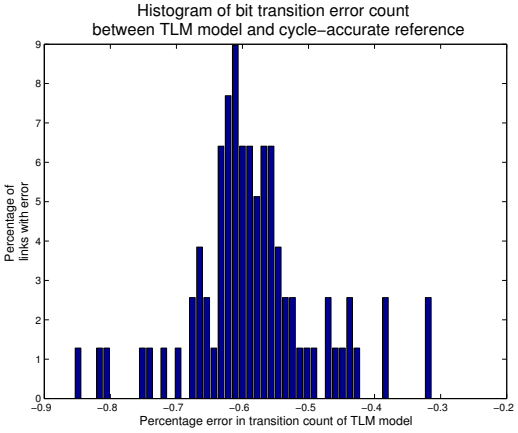


(b) TLM-PRE model for AVApp

Figure 8: Histogram of link bit transition estimation errors for TLM models relative to cycle-accurate reference in AV application



(a) TLM-NP model for H264



(b) TLM-PRE model for H264

Figure 9: Histogram of link bit transition estimation errors for the TLM models relative to cycle-accurate reference for H264 application

of individual links is in the small cluster around a 2.5% underestimate. These links are however carrying only a small amount of data, so this relative inaccuracy is less important for overall link power consumption. For the TLM-NPD

model, a power result histogram is not shown as the bit transition totals are identical to the cycle-accurate reference, indicating zero link dynamic power consumption error. The results for the TLM-PRE model in Figure 8b show similar bit transition patterns, with minor variations caused by the transition differences resulting from preemptions during flow transmission.

Figure 9a illustrates results for the H264 application using the TLM-NP model. The variance in link power errors is much smaller than for the AV application, with typical link power underestimates ranging between 0.5% to 0.7%. Similarly, for the TLM-NPD model, the power results were not shown as a result histogram since they are identical to the cycle-accurate reference. For the TLM-PRE model, shown in Figure 9b the results are overall similar, but there is a slightly higher peak around a 0.61% underestimate. There is more variation between TLM-PRE and TLM-NP for the H264 application than for the AV application, due to the increased contention in the application and mapping used. In particular, preemptions during transmission cause slightly increased bit transition errors for TLM-PRE in the 0.8% to 0.9% underestimate range.

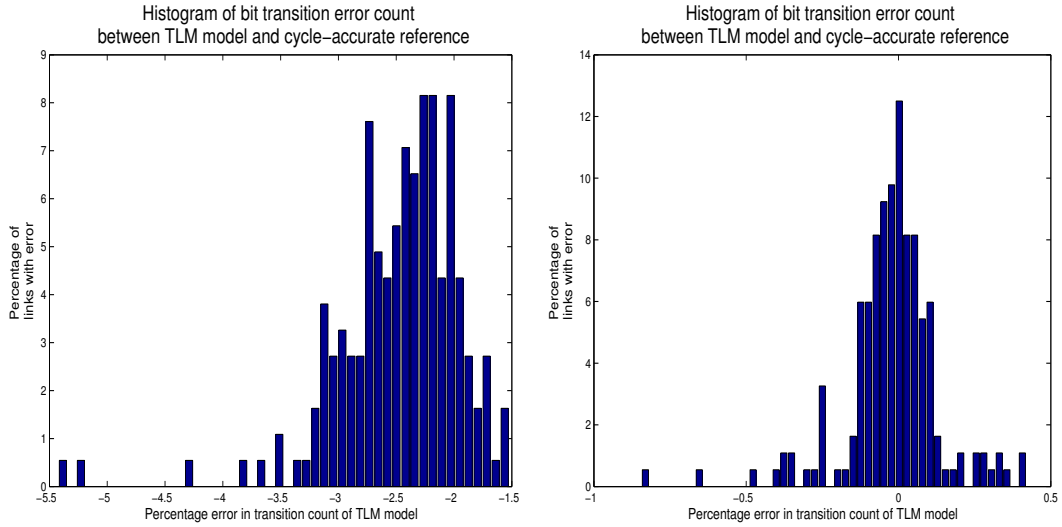
Figure 10 illustrates the equivalent results for synthetic traffic. The results for the TLM-NP model in Figure 10a illustrate an underestimate with a peak around 2.3%, although outliers illustrate a proportion of errors approximating 5.5% in a few cases. Given the wide range of flows and intersections produced by the increased contention, the variance of the errors produced is larger than in the application models. The results in Figure 10b show

that although the TLM-NPD model does produce some bit transition errors under the complex contention pattern of synthetic traffic, it is better at modelling arbitration decisions and therefore transitions correctly. The distribution of link bit transitions errors is centred around zero with a variance of approximately 0.2%. In the preemptive TLM-PRE model (Figure 10c), the underestimate of total transitions is around 2.2%. This is to be expected from the similar structure of the algorithm to TLM-NP, given that both TLM-PRE and TLM-NP operate on interference sets as defined in Section 5.4, and will therefore experience similar bit transition errors when flows are in their growing and shrinking phases. With TLM-PRE the errors are more closely clustered around the median value, although the worst case error for a particular link is slightly higher at 6.6%, representing additional errors from preemptions.

6.5. Discussion

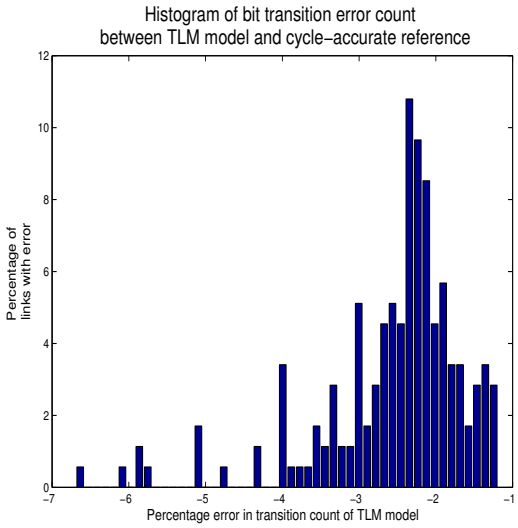
This section summarises the points arising from the results presented in the previous subsections, and considers the utility of the various TLM models in the analysis of NoC power and latency during the design flow stage. From the results presented above, the following considerations arise.

For the production of early power consumption estimates in non-preemptive NoCs in which dynamic power consumption needs to only be approximated roughly, the TLM-NP algorithm is suitable for approximating bit transitions. This is especially true in situations in which the average packet size is large



(a) TLM-NP model

(b) TLM-NPD model



(c) TLM-PRE model

Figure 10: Histogram of power consumption estimation errors of individual links for synthetic traffic relative to cycle-accurate reference

and blocking is infrequent, and therefore bit transition errors resulting from incorrect contention prediction are likely to be rare. If these conditions are not met, then the TLM-NPD algorithm is more suitable for link dynamic power consumption prediction.

If prediction of the latency of specific packets is not critical (because the network does not have hard real-time constraints) or the network utilisation/mappings are not expected to produce significant contention, than the TLM-NP algorithm can be useful to approximate overall network latencies. However, TLM-NP is not the most suitable in cases in which there is heavy contention (multiple blockings due to traffic intersecting simultaneously upon a route).

When traffic patterns interleaving long packets with smaller packet sizes are used, or higher accuracy in latency prediction for specific flows is required then the finer-grained locking algorithm TLM-NPD is important for accurately estimating latencies of individual flows. The TLM-NPD algorithm is capable of modelling the specific positions of flows and their contention timings, and has been demonstrated to produce close accuracy to the cycle-accurate case for the majority of flow priorities in the cases studied. Although the execution of TLM-NPD is about 2-3 times slower than TLM-NP, it is still sufficiently fast compared to cycle-accurate to be practical for real simulation workflows.

A preemptive NoC is generally more time-predictable than a non-preemptive NoC. Considering the three application case studies used, the less sophisti-

cated TLM-PRE algorithm is still capable of obtaining more accurate latency results under challenging contention patterns such as the H264 application. This TLM-PRE algorithm also has the lowest execution time of the three TLM models presented due to its simplicity.

7. Further Work

The TLM-NPD algorithm presented made use of fine-grained locking in non-preemptive NoCs. However, for preemptive NoCs as specified in Section 5, when several flows contend for access to a certain link the preemptive TLM model TLM-PRE only allows one of them to proceed through the network at any given time. Therefore, useful further work would be to add an additional TLM model of a preemptive NoC with fine-grained link claiming, effectively providing a high-performance simulation of the virtual channel mechanism. The algorithm involved would be considerably more complex, since it would not be possible to claim links throughout the lifetime of the flow in a preemptive NoC, since they could always be preempted by higher priority flows. Also, in a preemptive NoC, flows will not always travel from source to destination in a contiguous flit sequence; preemptions occurring in transit will break them into smaller chunks. Therefore, operations for flow splitting need to be defined to ensure similar accuracy to TLM-NPD.

8. Conclusion

This paper has presented a set of transaction-level models (TLM) for NoC interconnects, aimed at providing scalable models extensible to the massively parallel NoC architectures planned for future use. Simulations have been performed in a pair of realistic benchmark application models and with synthetic traffic in order to investigate their performance. The accuracy and execution time performance has been quantified to demonstrate that the TLM models presented provide an improved accuracy of approximately 93% in the prediction of link dynamic power consumption measured through bit transitions. The TLM simulations have exhibited execution time performance 2.5 to 3 orders of magnitude faster when compared to a cycle-accurate model of the same interconnect. In addition, the dynamic non-preemptive TLM-NPD simulation model has been shown capable of accurately predicting per-flit latencies even in synthetic traffic producing considerable contention. Although the TLM models typically predict flow latencies accurate to within mere flits of the cycle-accurate values, the worst-case latency error for any flow in the most sophisticated non-preemptive NoC TLM model (TLM-NPD) is 218%, as opposed to 66% in the preemptive NoC TLM model (TLM-PRE). This is due to the inherent time-predictability of preemptive NoCs which makes them more suitable for TLM modelling.

Acknowledgements

Financial support for this work was provided by the EPSRC, under projects 'LowPowNoC' (contract EP/J003662/1) and 'MCC' (EP/K011626/1).

References

[1] R. Marculescu, U. Y. Ogras, L.-S. Peh, N. E. Jerger, Y. Hoskote, Outstanding research problems in NoC design: System, microarchitecture, and circuit perspectives, IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems 28 (1) (2009) 3 –21. doi:10.1109/tcad.2008.2010691.

[2] E. Bolotin, I. Cidon, R. Ginosar, A. Kolodny, QNoC: QoS architecture and design process for network on chip, J. Syst. Archit. 50 (2-3) (2004) 105–128. doi:10.1016/j.sysarc.2003.07.004.

[3] L. Cai, D. Gajski, Transaction level modeling in system level design, Tech. rep., University of California, Irvine (2003).

[4] L. S. Indrusiak, O. M. dos Santos, Fast and accurate transaction-level model of a wormhole network-on-chip with priority preemptive virtual channel arbitration, in: DATE 2011: Design, Automation Test in Europe Conf., 2011, pp. 1–6.

[5] J. Harbin, L. S. Indrusiak, Fast transaction-level dynamic power consumption modelling in priority preemptive wormhole switching networks

on chip, in: SAMOS: Int. Conf. Embedded Computer Systems Architectures Modelling and Simulation, 2013.

[6] J. Harbin, L. Indrusiak, Dynamic task remapping for power and latency performance improvement in priority-based non-preemptive networks on chip, in: ReCoSoC 2013: 8th Int. Workshop Reconfigurable and Communication-Centric SoC, 2013, pp. 1–7. doi:10.1109/ReCoSoC.2013.6581526.

[7] A. Rose, S. Swan, J. Pierce, J. M. Fernandez, et al., Transaction level modeling in SystemC, Open SystemC Initiative 1 (1.297).

[8] L. Cai, D. Gajski, Transaction level modeling: an overview, in: CODES+ISSS 2003: 1st Int. Conf. Hardware/Software Codesign and System Synthesis, 2003, pp. 19 –24. doi:10.1109/codess.2003.1275250.

[9] A. Kohler, M. Radetzki, A systemc TLM2 model of communication in wormhole switched networks-on-chip, in: Specification Design Languages, 2009. FDL 2009. Forum on, 2009, pp. 1–4.

[10] G. Schirner, R. Dömer, Quantitative analysis of the speed/accuracy trade-off in transaction level modeling, ACM Trans. Embed. Comput. Syst. 8 (1) (2009) 4:1–4:29. doi:10.1145/1457246.1457250.

[11] G. Schirner, R. Domer, Result-Oriented Modeling: A novel technique for fast and accurate TLM, IEEE Trans. Computer-Aided

Design of Integrated Circuits Systems 26 (9) (2007) 1688–1699. doi:10.1109/TCAD.2007.895757.

[12] H. W. M. Van Moll, H. Corporaal, V. Reyes, M. Boonen, Fast and accurate protocol specific bus modeling using TLM 2.0, in: DATE 2009: Design Automation Test in Europe Conf., 2009, pp. 316–319. doi:10.1109/DATE.2009.5090680.

[13] E. Viaud, F. Pêcheux, A. Greiner, An efficient TLM/T modeling and simulation environment based on conservative parallel discrete event principles, in: DATE 2006: Design Automation Test in Europe Conf., 2006, pp. 94–99.
URL http://dl.acm.org/citation.cfm?id=1131481.1131514

[14] M. Hosseinabady, J. Nunez-Yanez, SystemC architectural transaction level modelling for large NoCs, in: FDL 2010: Forum on Spec. Design Lang., 2010, pp. 1–6. doi:10.1049/ic.2010.0143.

[15] M. Eggenberger, M. Radetzki, Scalable parallel simulation of networks on chip, in: Networks on Chip (NoCS), 2013 Seventh IEEE/ACM International Symposium on, 2013, pp. 1–8. doi:10.1109/NoCS.2013.6558402.

[16] L. Ost, F. Moraes, L. Möller, L. S. Indrusiak, M. Glesner, S. Määttä, J. Nurmi, A simplified executable model to evaluate latency and throughput of networks-on-chip, in: SBCCI 2008: Proc 21st Ann. Symp. Integrated Circuits and System Design, SBCCI '08, ACM, New York,

NY, USA, 2008, pp. 170–175. doi:10.1145/1404371.1404420.
URL http://doi.acm.org/10.1145/1404371.1404420

[17] J. Harbin, L. S. Indrusiak, Fine-grained link locking within power and latency transaction level modelling in wormhole switching non-preemptive networks on chip, in: Proceedings of Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms, PARMA-DITAM '14, ACM, New York, NY, USA, 2014, pp. 33:33–33:38. doi:10.1145/2556863.2556865.
URL http://doi.acm.org/10.1145/2556863.2556865

[18] W. Fornaciari, D. Sciuto, C. Silvano, Power estimation for architectural exploration of HW/SW communication on system-level buses, in: CODES '99: 7th Int. Workshop Hardware/Software Codesign, 1999, pp. 152 –156. doi:10.1109/hsc.1999.777411.

[19] K. Chatha, K. Srinivasan, Layout aware design of mesh based noc architectures, in: Hardware/Software Codesign and System Synthesis, 2006. CODES+ISSS '06. Proceedings of the 4th International Conference, 2006, pp. 136–141. doi:10.1145/1176254.1176288.

[20] K. Srinivasan, K. Chatha, G. Konjevod, Linear-programming-based techniques for synthesis of network-on-chip architectures, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 14 (4) (2006) 407–420. doi:10.1109/TVLSI.2006.871762.

[21] M. Palesi, G. Ascia, F. Fazzino, V. Catania, Data encoding schemes in networks on chip, Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 30 (5) (2011) 774 –786. doi:10.1109/tcad.2010.2098590.

[22] J. C. S. Palma, L. S. Indrusiak, F. G. Moraes, A. Garcia Ortiz, M. Glesner, R. A. L. Reis, Inserting data encoding techniques into noc-based systems, in: VLSI, 2007. ISVLSI '07. IEEE Computer Society Annual Symposium on, 2007, pp. 299 –304. doi:10.1109/isvlsi.2007.58.

[23] H. Kung, R. Morris, Credit-based flow control for atm networks, IEEE network 9 (2) (1995) 40–48.

[24] F. Moraes, N. Calazans, A. Mello, L. Möller, L. Ost, HERMES: an infrastructure for low area overhead packet-switching networks on chip, Integr. VLSI J. 38 (1) (2004) 69–93. doi:10.1016/j.vlsi.2004.03.003.

[25] A. Mello, L. Tedesco, N. Calazans, F. Moraes, Virtual channels in networks on chip: implementation and evaluation on Hermes NoC, in: Proceedings of the 18th annual symposium on Integrated circuits and system design, ACM, 2005, pp. 178–183.

[26] Z. Shi, A. Burns, L. S. Indrusiak, Schedulability analysis for real time on-chip communication with wormhole switching, International Journal of Embedded and Real-Time Communication Systems (IJERTCS) 1 (2) (2010) 1–22.

[27] W. Liu, J. Xu, X. Wu, Y. Ye, X. Wang, W. Zhang, M. Nikdast, Z. Wang, A NoC traffic suite based on real applications, in: ISVLSI 2011: IEEE Com. Soc. Annual Symp., 2011, pp. 66–71. doi:10.1109/ISVLSI.2011.49.

[28] C. Brooks, E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, H. Zheng, Heterogeneous concurrent modeling and design in Java (vol .2: Ptolemy II software architecture), Tech. Rep. UCB/EECS-2008-29, EECS Department, University of California, Berkeley (Apr. 2008).

[29] L. Indrusiak, J. Harbin, O. M. dos Santos, Fast simulation of networks-on-chip with priority-preemptive arbitration, ACM TODAES (Transactions on Design Automation of Electronic Systems) (2015) (accepted).

[30] N. Banerjee, P. Vellanki, K. S. Chatha, A power and performance model for network-on-chip architectures, in: DATE 2004: Design Automation Test in Europe Conf., Vol. 2, 2004, pp. 1250–1255. doi:10.1109/date.2004.1269067.

[31] J. A. Davis, J. D. Meindl, Compact distributed RLC interconnect models - Part II: Coupled line transient expressions and peak crosstalk in multilevel networks, IEEE Trans. Electron Devices 47 (11) (2000) 2078 –2087. doi:10.1109/16.877169.

[32] L. Ost, G. Guindani, F. Moraes, L. Indrusiak, S. Maatta, Exploring noc-based mpsoc design space with power estimation models, Design Test of Computers, IEEE 28 (2) (2011) 16–29. doi:10.1109/MDT.2010.116.