Fast and Accurate Transaction-Level Model of a Wormhole Network-on-Chip with Priority Preemptive Virtual Channel Arbitration

Leandro Soares Indrusiak, Osmar Marchi dos Santos
Real-Time Systems Group, Department of Computer Science
University of York
York, United Kingdom

***Abstract*—Simulation is a bottleneck in the design flow of on-chip multiprocessors. This paper addresses that problem by reducing the simulation time of complex on-chip interconnects through transaction-level modelling (TLM). A particular on-chip interconnect architecture was chosen, namely a wormhole network-on-chip with priority preemptive virtual channel arbitration, because its mechanisms can be modelled at transaction level in such a way that accurate figures for communication latency can be obtained with less simulation time than a cycle-accurate model. The proposed model produced latency figures with more than 90% accuracy and simulated more than 1000 times faster than a cycle-accurate model.**

Keywords-system specification; transaction-level modeling; network-on-chip; on-chip multiprocessing; simulation.

I. INTRODUCTION

As the number of cores grows, the design of on-chip multiprocessors has become increasingly communication centric. The choice and customisation of the on-chip interconnect architecture is a critical element of performance tuning, as communication latency becomes a key issue. Despite of some progress in static analysis such as in [1], most design flows use simulation to evaluate the latency overhead imposed by on-chip interconnects. In the case of complex interconnects like Networks-on-Chip (NoCs), cycle-accurate simulation of a few seconds of the system's execution can take several hours or days [2]. Simulation has been identified many times before as a bottleneck on the embedded systems design flow, and many alternative techniques have been proposed such as emulation, rapid prototyping and abstract models. This has also happened in the domain of NoCs, but the alternatives are either too difficult to implement and maintain [2], or have to sacrifice accuracy in order to be faster. In this paper, we attempt to build a NoC simulation model that is both fast and accurate with regard to its cycle-accurate counterpart. We do that by restricting the NoC design space to a class of NoC interconnects that can be accurately described at higher levels of abstraction. The proposed model follows the transaction-level modelling approach, which is reviewed in Section II, followed by a discussion about the trade-off between accuracy and simulation speed. Section III presents the region of the NoC design space that we concentrate in this paper, and justifies the constraining of design space exploration in favour of predictability. The proposed TLM model and its functionality are detailed in Section IV, followed by extensive experimental results in Section V and our conclusions.

II. RELATED WORK

In brief, TLM attempts to speed-up simulation by abstracting away low-level events occurring during communication, focusing instead on large-granularity data transfers. It has mainly been applied in designs created with SystemC, but its methodology is generic enough to be used within other simulation frameworks and languages. In TLM 2.0, a transaction models the transfer of a payload between components of the system. Components are either transaction *initiators*, transaction *targets* or *interconnects*, which modify and forward transactions from initiators to targets [3].

The trade-off between accuracy and simulation speed in TLM was very well characterised by Schirner and Dömer [4]. While TLM models can simulate up to four orders of magnitude faster than their cycle-accurate counterparts, this comes at the price of low accuracy. Because TLM models are based on a simplified structure of the system, they have a larger granularity of data and arbitration handling. As a consequence, such models cannot model effects that happen at a finer granularity, resulting in loss of accuracy. However, the notion that TLM models can be either fast or accurate has been recently challenged by a number of works. TLM models of processing elements (PE) can increase simulation speed by dealing with a granularity which is larger than individual instructions. Accuracy of such models is kept high by the use of timing annotations extracted from code profiling, which even allows the modelling of effects such as pipelining, caching [5] and interrupts [6].

Accurate TLM models for on-chip interconnects have also been proposed. In [7], bus protocol specifications are used to identify a reduced set of timing points when models of a particular on-chip bus architecture should be simulated without loss of accuracy. The authors reported an improvement of two orders of magnitude in simulation speed without loss of accuracy, in comparison with a cycle-accurate model. The drawback is that this technique depends on the ability to identify a set of timing points which are small enough to significantly reduce the number of simulation events while

978-3-9810801-7-9/DATE11/©2011 EDAA

covering all possible protocol state transitions, which will not be straightforward in complex on-chip interconnects.

Fast and accurate TLM models of busses are also discussed in [8], where the concept of Result-Oriented Modelling (ROM) is introduced. ROM optimistically predicts the delay of a particular transaction and retroactively corrects it in case it detects that the delay was affected by the outcome of other transactions. It presents convincing results without any loss of accuracy, as demonstrated through two case studies with CAN and AMBA AHB busses. However, its improvement on simulation speed is inversely proportional to the frequency of corrections needed by the optimistic predictions, and such corrections are likely to occur very often in interconnects with complex contention patterns.

Specific approaches to TLM of NoC interconnects are also available in the literature. In [9], a speed-up of 50x with 99.999% accuracy have been shown by using a timed TLM approach which reduces the overhead of the simulation kernel by using local time references for each individual task that communicates over the NoC. The local clocks of different tasks are only synchronised when those tasks are initiator or target of the same transaction. In [10] a modest speed-up of 38% has been reported by using lightweight schedulers that handle the time reference for a group of tasks. Unlike those approaches, in this paper we do not change TLM simulation semantics or simulation kernel implementation, but rather constrain our approach to use interconnect architectures whose behaviour can be better captured by standard TLM semantics.

Existing work on optimising simulation of wormhole networks has also been taken into account [11] [12], where a reduction of simulation events is achieved by simulating only packet headers and trailers. This work can be considered as an improvement of the work presented in [12].

III. INTERCONNECT ARCHITECTURE

The design space of NoC architectures is very large, as many of its components can be parameterized to better meet design goals: routers, arbiters, buffers, flow controllers, among others. However, the experience acquired through the development of many commercial and research-oriented NoCs allowed the identification of a few mechanisms that are adequate for a wide variety of NoC configurations, and so they were adopted widely. For example, sophisticated routing algorithms do not significantly reduce communication latency when compared with simple deterministic routers such as XY[13], so most NoC designers avoid the costs in area and power consumption by adopting the simpler solution. Wormhole switching is another example of a mechanism that was widely adopted in NoCs because it does not require large capacity buffers (which in turn means lower power and area overhead of the routers, a top priority among NoC designers).

In this paper, we also adopt such widely used architectural patterns and consider NoCs with mesh topology, wormhole switching and XY deterministic routing. However, such architectures are particularly vulnerable to network contention and its effects can only be accurately predicted by using cycle-accurate models. Taking as an example the situation when two packet headers arrive at a NoC router within one cycle of each other, this minimal time difference could determine which of the packets would be granted access to a mutually exclusive resource (e.g. an output port) while the other would have to wait, significantly affecting the latency of both of them. Such scenario can obviously not be modeled with a time granularity that is larger than one cycle, such as in TLM, therefore a TLM model of such system will certainly present low accuracy figures for latency. To solve this issue, we further constraint our design space and focus on architectural constructs whose behaviour can be accurately modelled at transaction level. While this decision is not based on a functional requirement of a particular design, it has the potential to speed up design space exploration by reducing the time to evaluate each alternative solution within that space. As a consequence, more alternatives can be analysed and designers are more likely to find solutions that fulfil functional requirements.

In this paper, we focus on one particular architectural construct that can be accurately described using TLM, namely a flow controller based on priority preemptive virtual channels. By assigning priorities to packets, and by allowing high priority packets to preempt the transmission of low priority ones, network contention scenarios become more predictable and do not require cycle-accurate models to be analysed. Fig. 1 shows the internal structure of a NoC router using such architecture, which is similar to QNoC [14] and HERMES [15]. In each input port, a different FIFO buffer stores flits of packets arriving through different virtual channels (one for each priority level). The router assigns an output port for each incoming packet according to their destination. A credit-based approach [16] guarantees that data is only forwarded from a router to the next when there's enough buffer space to hold it.

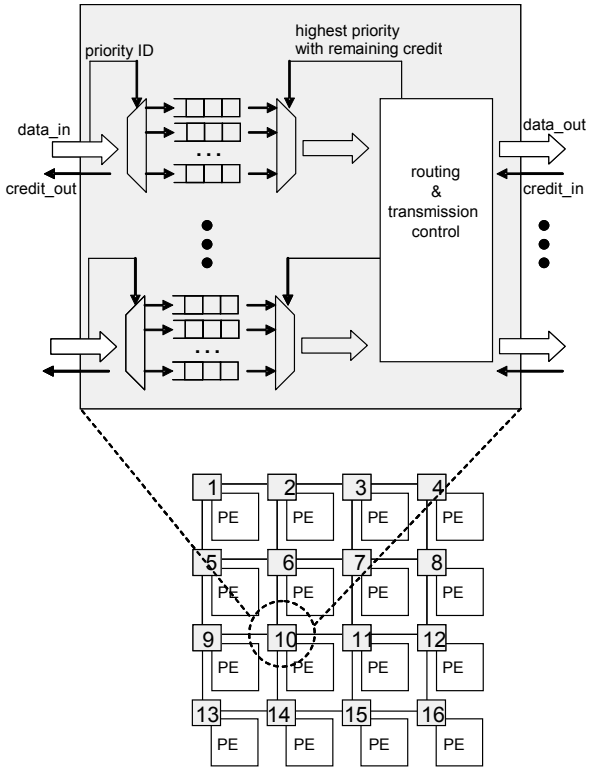

Figure 1. NoC architecture with detail of the router structure

At any time, a flit of a given packet will be sent through its respective output port if it has the highest priority among the packets being sent out through that port, and if it has credits (that is, buffer space on the respective buffer of the neighbouring node connected to that output port). If the highest priority packet can't send data because it is blocked elsewhere in the network, the next highest priority packet can access the output link. The identified architecture is therefore able to provide guaranteed throughput (GT) to traffic of higher priority, and also provides means to calculate upper latency bounds to best-effort (BE) traffic, as the priority ordering clearly shows when a packet will be blocked. Its approach to guarantee throughput is more efficient than time-division multiplexing (TDM), as used in many NoC architectures such as [17] and [18], where GT traffic gets a pre-assigned time-slot to use resources. Priority preemptive arbitration does not unnecessarily reserves resources, so low priority traffic can always use the NoC if there are no requests from GT flows.

IV. TRANSACTION-LEVEL MODEL OF THE NETWORK-ON-CHIP ARCHITECTURE

Unlike regular NoC simulation models which are composites of routers, buffers, arbiters and links, the proposed TLM model is a single interconnect component connected to all processing cores, which are both transaction initiators and targets (Fig. 2). Following TLM principles, a NoC communication flow is the main element of a transaction. Therefore, a transaction is initiated for every packet header received by the NoC and is alive until that packet is delivered to the target. Initiators create new transactions by calling the interconnect's non-blocking interface, passing as arguments the packet itself, its priority and the target's address. Likewise, the NoC calls the target's non-blocking interface when a packet is completely delivered and stored at its network interface.

While structurally simple, our TLM model must be able to accurately estimate the lifetime of each transaction, which in turn denotes the communication latency of each packet. The core of our approach is the notion of interference suffered by a given communication flow. If a flow has the whole network to itself, estimating its latency is trivial, as it becomes a function of the number of communication hops and the packet's flit count (the packet's so-called *basic latency*). However, the latency estimation becomes harder when multiple flows compete for the same NoC resources. When using priority pre-emption, it is possible to deterministically decide which flow is the first to acquire a shared resource, making it possible to quantify the amount of interference that each flow suffers from other flows of higher priority (i.e. the time it has to wait for them to release the shared resources). To do that, our model uses the internal representations and algorithms described below.

Each communication flow is characterised by a tuple $flow_i$(*src, dst,* t_r*, priority, payload*) where *src* and *dst* are the address of the packet's source and destination over the NoC (e.g. 1 to 16 in Figure 1); t_r is the time the packet header was received at the NoC router attached to *src*; *priority* is an integer number which is unique to each flow, denoting which flow has higher priority to acquire shared resources; and *payload* is an integer number denoting the number of flits of the packet.

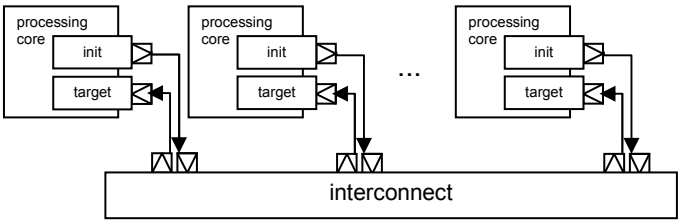

Figure 2. Structure of the proposed TLM model of a NoC

The TLM model maintains a list *flowlist* of all flows, sorted by their priority. Each entry includes the parameter tuple $flow_i$ of the respective flow and additional information required by the proposed algorithm to calculate the completion time of each transaction: $route_i$ is the path through which the flow's packet is routed across the NoC from src_i to dst_i according to the NoC routing algorithm; $interference_i$ is the set of flows $\{flow_a, flow_b, ..., flow_n\}$ whose route shares at least one link with $route_i$ and whose priority is higher than $priority_i$; $active_i$ denotes whether $flow_i$ is active, as opposed to blocked by any active $flow_n \in interference_i$; t_{ai} is the time $flow_i$ last became active; $flitstosend_i$ is the number of flits yet to be delivered at *dst*. Listing 1 describes the behaviour of the TLM model when a new transaction is initiated, showing in lines 5-7 the process of identifying interference lists for each flow based on their priorities and route intersections.

```
01  sendPacket(flow_i , currenttime) {
02        t_ai = currenttime; flitstosend_i = payload_i; active_i = false;
03        route_i = routing(src_i, dst_i);
04        for each flow_n in flowlist {
05           if (route_i ∩ route_n ≠ ∅) {
06             if (priority_i < priority_n) {add flow_n to interference_i;}
07             else {add flow_i to interference_n;}
08           }
09        }
10        add flow_i to flowlist;
11        requestUpdate(currenttime) ;
12  }
```

Listing 1. Pseudocode for initiating a new transaction

When new transactions are initiated or terminate, *flowlist* must be updated. Updates are used to discriminate which flows in *flowlist* are active, and to schedule additional updates to the time instant when active flows are likely to terminate. Such update events must be explicitly requested. For example, an update is requested at the exact time that a transaction is added (line 11 of Listing 1). For clarity, the pseudocode shows an update request with syntax that resembles a *wait-for-delay*. Its implementation, however, is similar to the approach presented in [6] and follows a *wait-for-event* semantics.

Active flows are all those that have no active flows on their interference sets, so the typical case is that several flows can be active at the same time on a NoC, as long as their routes do not intersect. When flows are added or terminated, interference sets may change and therefore a new update is required. Listing 2 shows the proposed update algorithm. Every time an update is triggered, the TLM model iterates over *flowlist* and updates the status of active (lines 7-8) and inactive (lines 12-13) flows according to changes on their interference set (i.e. a flow

becomes inactive because a new flow is added to its interference set, or becomes active because all active flows in its interference set terminated or became inactive). As *flowlist* is sorted by flow priorities, we can guarantee that when the algorithm iterates over a given entry of *flowlist*, all possible changes to the interference set of its flow have already been committed (because a flow only suffers interference from higher priority flows). When a flow becomes active, another update event is requested for the time it is likely to terminate (according to the *basic latency* of its remaining flits, line 14). If the flow is forced into inactivity before that time, that update event is then cancelled (a new one will be scheduled when the flow is activated the next time). Additionally, updates must check if each active flow has terminated (by checking whether it was active for long enough to send all remaining flits, lines 4-6), at which point its entry is removed from *flowlist*, its transaction is concluded and its corresponding packet is sent out to the transaction target.

The proposed algorithm can substantially reduce simulation time because it simulates the system only at the time instants when packets enter or exit the NoC, rather than every clock cycle or flit transmission. Therefore, potential speed up is directly proportional to the length of the packets and the hop count of their paths across the NoC.

```
01  update(currenttime) {
02    for each flow_i in flowlist {
03      if (active_i) {
04        flitstosend_i = flitstosend_i – sentFlits(currenttime- t_ai);
05        t_ai = currenttime;
06        if (flitstosend_i ==0) { remove flow_i from flowlist;}
07        for each flow_n in interference_i {
08          if(active_n) {active_i = false;}
09        }
10      }
11      else {
12        if ( !active_n for all flow_n in interference_i ){
13          active_i = true;
14          requestUpdate(currenttime + basiclat(flitstosend_i));
15        }
16      }
17    }
18  }
```

Listing 2. Pseudocode for updating list of communication flows

However, the proposed approach assumes route intersection as the one and only prerequisite for interference between flows (Listing 1, line 5). This is not necessarily the case, because in wormhole switching packets will gradually occupy their route from source to destination ("growing" phase), and the other way around when they finish transmission ("shrinking" phase). Furthermore, if the flit count of a packet is less than the hop count of its route, it will never fully occupy the route. In all those cases, our TLM model could assume interferences which would not occur in reality (e.g. flow A on its growing phase shares the last link of its route with the first link of the route of flow B, which is on its shrinking phase already). As a consequence, obtained latency figures may be higher than reality, and the percent difference will grow with the increase of the ratio between route hop count and packet flit count.

V. Experimental Results

To evaluate the simulation speed-up and the accuracy of the proposed TLM model, we performed a number of experiments, aiming to compare it to a cycle-accurate (CA) model of the same NoC architecture. For the sake of fairness, the same simulation framework is used in both cases, namely Ptolemy II. Following TLM principles, transaction initiators and targets are mostly the same, and include abstract models of application tasks, operating system, processing cores and NoC interfaces. The only difference appears at the network interface modules, which transmit packets flit-by-flit in the CA model, while in TLM they pass a reference to the whole packet when transactions are initiated. The chosen configuration for both NoC models has a 4x4 mesh topology, router input ports with eight virtual channels and two-position buffers each, XY routing, operating at 100 MHz. For the first set of experiments, we used as a testbench the application presented in [1], which models the video processing, navigation and stability control subsystems of an autonomous vehicle (AV). The application comprehends 33 tasks and 38 inter-task fixed-priority communication flows. Each flow has fixed-size payloads, the smallest with 7kbits and the largest 525kbits. Flows are initiated sporadically but respect a minimum inter-release time (which may or may not suffer jitter). After using a static mapping heuristic to assign tasks to each of the 16 cores of the chosen platform, we simulated the execution of the application and obtained latency figures for each communication flow.

To evaluate the simulation speed-up obtained by the proposed TLM model, we obtained the time required by the chosen simulation host (an Intel Core Duo at 3.02 GHz) to run both CA and TLM models for distinct target execution times (i.e. simulated periods of the vehicle application execution): 1s, 2s, 4s, 8s, 20s, 200s, 800s and 1400s. The figures for the last three target times were obtained only for the TLM model, as the CA model would take several days to simulate them on the chosen host. The results, depicted in Fig. 3, show that after simulation initialisation (which took about 7 seconds on the chosen host), the TLM model presents a speed-up of more than three orders of magnitude. In terms of accuracy, the worst case latency results have a maximum percent difference of 6.25% when using a NoC with flit size of 64 bits, 3.23% for flit size of 32 bits and 1.64% for 16 bits (Table I). Such figures are in line with the expectations that the accuracy loss of the proposed approach is less significant when simulating packets with a larger number of flits for the same route (i.e. for the same payload, packets with half the flit size require almost two times the number of flits). Accuracy figures for average and best case latency results were higher, as interference plays a lesser role.

application time (s)	simulation time (s)	
	cycle-accurate	TLM
1	1465.18	7.67
2	2895.69	7.89
4	5978.32	7.98
8	11806.91	8.28
20	29927.24	9.77
200		26.55
800		86.29
1400		169.06

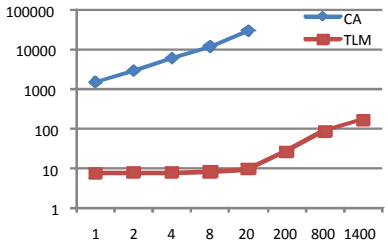

Figure 3. Simulation time comparison (for distinct target times)

TABLE I. FLOW LATENCY RESULTS FOR AUTONOMOUS VEHICLE

Flow	Communication Latency (ms)								
	CA			*TLM*			*% diff*		
	flit size (bits)			*flit size (bits)*			*flit size (bits)*		
	16	*32*	*64*	*16*	*32*	*64*	*16*	*32*	*64*
1	0.491	0.251	0.131	0.492	0.252	0.132	0.20	0.40	0.76
2	0.488	0.248	0.128	0.489	0.249	0.129	0.20	0.40	0.78
3	0.970	0.490	0.250	0.981	0.501	0.261	1.13	2.22	4.31
4	1.452	0.732	0.372	1.476	0.756	0.396	**1.64**	**3.23**	**6.25**
5	1.514	0.762	0.386	1.528	0.776	0.400	0.92	1.82	3.56
6	0.492	0.252	0.132	0.492	0.252	0.132	0.00	0.00	0.00
7	0.971	0.491	0.251	0.981	0.496	0.261	1.02	1.01	3.91
8	0.491	0.251	0.131	0.492	0.252	0.132	0.20	0.40	0.76
9	9.608	4.808	2.408	9.609	4.809	2.409	0.01	0.02	0.04
10	9.608	4.808	2.408	9.609	4.809	2.409	0.01	0.02	0.04
11	9.608	4.808	2.408	9.609	4.809	2.409	0.01	0.02	0.04
12	10.638	5.326	2.670	10.654	5.342	2.686	0.15	0.30	0.60
13	9.608	4.808	2.408	9.609	4.809	2.409	0.01	0.02	0.04
14	9.608	4.808	2.408	9.609	4.809	2.409	0.01	0.02	0.04
15	9.608	4.808	2.408	9.609	4.809	2.409	0.01	0.02	0.04
16	9.608	4.808	2.408	9.609	4.809	2.409	0.01	0.02	0.04
17	9.608	4.808	2.408	9.609	4.809	2.409	0.01	0.02	0.04
18	9.608	4.808	2.408	9.609	4.809	2.409	0.01	0.02	0.04
19	1.928	0.968	0.488	1.929	0.969	0.489	0.05	0.10	0.20
20	3.850	1.930	0.970	3.858	1.938	0.978	0.21	0.41	0.82
21	1.041	0.529	0.273	1.042	0.53	0.274	0.10	0.19	0.37
22	2.059	1.035	0.523	2.060	1.036	0.524	0.05	0.10	0.19
23	2.059	1.035	0.523	2.060	1.036	0.524	0.05	0.10	0.19
24	32.779	16.395	8.203	32.780	16.396	8.204	0.00	0.01	0.01
25	32.776	16.392	8.200	32.777	16.393	8.201	0.00	0.01	0.01
26	8.200	4.104	2.056	8.201	4.105	2.057	0.01	0.02	0.05
27	2.061	1.037	0.525	2.075	1.051	0.539	0.68	1.34	2.63
28	1.032	0.520	0.264	1.033	0.521	0.265	0.10	0.19	0.38
29	1.038	0.526	0.270	1.039	0.527	0.271	0.10	0.19	0.37
30	2.056	1.032	0.520	2.057	1.033	0.521	0.05	0.10	0.19
31	1.999	1.007	0.511	2.029	1.037	0.541	1.49	2.94	5.70
32	2.056	1.032	0.520	2.057	1.033	0.521	0.05	0.10	0.19
33	8.206	4.110	2.062	8.207	4.111	2.063	0.01	0.02	0.05
34	2.540	1.276	0.644	2.558	1.294	0.662	0.71	1.40	2.76
35	2.059	1.035	0.523	2.060	1.036	0.524	0.05	0.10	0.19
36	1.035	0.523	0.267	1.036	0.524	0.268	0.10	0.19	0.37
37	1.032	0.520	0.264	1.033	0.521	0.265	0.10	0.19	0.38
38	2.058	1.034	0.522	2.072	1.048	0.536	0.68	1.34	2.65

In the second set of experiments, our goal was to evaluate the scalability of our approach in terms of size and complexity of the application using the NoC as communication media. For that, we created a set of random flows that would periodically send a random amount of data to a random destination over the NoC. We then simulated, in turn, 20, 40, 60, 80 and 100 of those flows with both CA and TLM NoC models, and measured the host simulation time for each case. The results depicted in Fig. 4 show that the scalability of our approach is similar to the CA model, but again with a speed-up that exceeds three orders of magnitude. The percent difference between worst case, average and best case latencies of both models was below 1%.

application flows	simulation time (s) cycle-accurate	TLM
20	17105.42	5.47
40	31490.13	19.17
60	55561.87	45.78
80	70231.76	83.06
100	86626.07	105.28

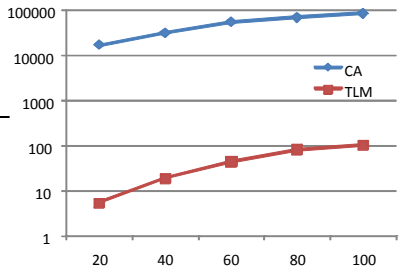

Figure 4. Simulation time comparison (for distinct application flow counts)

The final set of experiments aimed at showing that the percent difference between CA and the proposed TLM model could be much higher than the values obtained by the experiments reported so far. By carefully selecting flow release times, routes and intersection patterns, we were able to identify cases where the percent difference would be arbitrarily large. For instance, we could create in the CA model an arbitrarily large high priority flow A on its growing phase that would nearly block another flow B on its shrinking phase. In the TLM model, A would block B for an arbitrarily long period of time (proportional to A's basic latency), because it has higher priority and they share a link, driving up the percent difference of B's latency. While this may seem as a shortcoming of the proposed approach, it actually mimics possible interference scenarios that would occur in a real system if flows are allowed some release jitter (which is often the case). Even a small delay on the release of a particular flow may cause another interference pattern to arise, which would be more likely to be captured by our approach than by a CA model.

Taking that into account, the proposed approach can be considered safe and conservative, as it is able to capture each and every interference pattern that appears when simulating the CA model, and it offers an additional margin of safety by also capturing interference patterns that are likely to appear if the release jitter of the flows follows a stochastic behaviour. To gather evidence on worst case latency increase due to jitter, we extended the autonomous vehicle application used in the first set of experiments to allow for release jitter on each of its 38 flows. During simulation, jitter was introduced before the release of each packet, assuming a random delay value between zero and 10% of the flow's minimum inter-release time. Fig. 5 shows the worst case latency results obtained from simulating both scenarios - with and without jitter- for a target time of 1000.0 seconds. It can be seen that for many flows (e.g. 24, 25, 27, 28, 29) the introduction of jitter clearly resulted in increased worst case latencies because new interference patterns arose. Interestingly, the worst case latency with jitter for some flows (such as 4 and 31) was actually lower than without jitter, because the particular flow release configuration that resulted in those higher numbers did not appear in the random jitter scenario (as the probability of randomly assigning zero jitter to all interfering flows is very low).

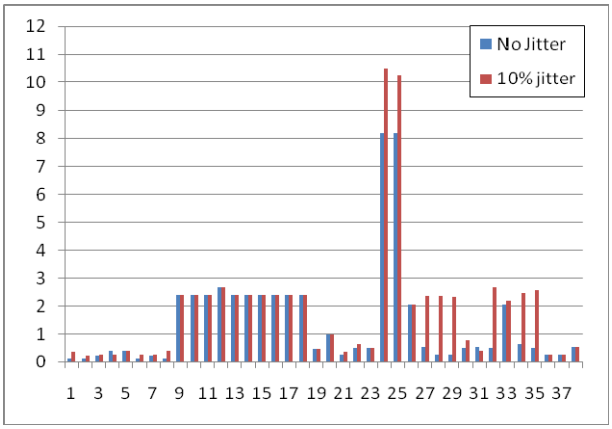

Figure 5. Worst case latency (in ms) for all 38 flows of AV application – TLM model, flit size = 64 bits (without jitter, and with jitter bounded to 10% of the flow's minimum inter-release time)

Based on that evidence, we argue that that the results obtained from the proposed TLM model are actually useful even when there is a possibility that they could report inflated latency resulting from interference patterns that do not appear on a cycle-accurate simulation. If jitter is taken into account, those interference patterns are likely to happen, and in that case the inflated worst case latency result would actually be accurate.

In any case, a fully accurate model could be still be developed, using a mixed-time approach to capture the cycle-accurate behaviour of the growing and shrinking phases of each flow, but the increased number of simulation events resulting from that would significantly reduce simulation speed. Therefore, we state that the modelling of the growing and shrinking phases of flows is not justified, because a slightly conservative model is a very low price to pay for the unprecedented speed-up that was obtained with the proposed TLM approach.

VI. CONCLUSIONS

This paper has proposed a novel TLM modelling approach for NoC interconnects, aiming to achieve fast and accurate simulation of on-chip communication. Unlike previous approaches, this work does not change TLM simulation semantics or kernel implementation. Instead, it identifies a NoC architecture whose resource arbitration mechanisms can be adequately modelled at time granularities that are larger than a clock cycle or a flit, namely a wormhole NoC with priority preemptive virtual channel arbitration. The proposed approach is therefore independent from modelling language and simulation kernel, and it can potentially deliver the same level of speed-up in all cases, because its is based on the reduction of the number of events that must be simulated, rather than on simulating the same set of events faster.

Extensive experimentation has shown that the proposed TLM model consistently allowed for a simulation speed-up of more than three orders of magnitude. Such speed-up is not completely unusual for TLM models, but this is unprecedented when it comes to complex NoC-based multiprocessors. The accuracy of the model exceeds 90%, which is comparable to the figures obtained from TLM models of systems of much lower complexity, therefore showing that the proposed approach could handle well the additional complexity of NoC-based multiprocessors.

The paper also discussed the fact that the proposed model is conservative, in the sense that it takes into account all possible flow interference scenarios that appear on a CA model, as well as additional interference scenarios that are likely to appear if flow release jitter is taken into account. From a designer's perspective, this adds a safety margin that prevents undesirable timing hazards from going unnoticed.

REFERENCES

[1] Z. Shi, A. Burns, and L. S. Indrusiak, "Schedulability Analysis for Real Time On-Chip Communication with Wormhole Switching," *International Journal of Embedded and Real-Time Communication Systems*, vol. 1, no. 2, pp. 1 - 22, Jun. 2010.

[2] N. Genko et al., "A novel approach for network on chip emulation," in *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pp. 2365-2368 Vol. 3, 2005.

[3] J. Aynsley, "TLM-2.0 Language Reference Manual," OSCI, 2009.

[4] G. Schirner and R. Dömer, "Quantitative analysis of the speed/accuracy trade-off in transaction level modeling," *ACM Trans. Embed. Comput. Syst.*, vol. 8, no. 1, pp. 1-29, 2008.

[5] Kai-Li Lin, Chen-Kang Lo, and Ren-Song Tsay, "Source-level timing annotation for fast and accurate TLM computation model generation," in *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*, pp. 235-240, 2010.

[6] Ke Yu and N. Audsley, "A Mixed Timing System-Level Embedded Software Modelling and Simulation Approach," in *Embedded Software and Systems, 2009. ICESS '09. International Conference on*, pp. 193-200, 2009.

[7] H. van Moll, H. Corporaal, V. Reyes, and M. Boonen, "Fast and accurate protocol specific bus modeling using TLM 2.0," in *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09.*, pp. 316-319, 2009.

[8] G. Schirner and R. Dömer, "Result-Oriented Modeling—A Novel Technique for Fast and Accurate TLM," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, no. 9, pp. 1688-1699, 2007.

[9] E. Viaud, F. Pêcheux, and A. Greiner, "An efficient TLM/T modeling and simulation environment based on conservative parallel discrete event principles," in *Proceedings of the conference on Design, automation and test in Europe: Proceedings*, pp. 94-99, 2006.

[10] M. Hosseinabady and J. L. Nunez-Yanez, "SystemC Architectural Transaction Level Modelling for Large NoCs," in *ECSI Forum on Specification and Design Languages (FDL)*, 2010.

[11] P. K. McKinley and C. Trefftz, "MultiSim: A Simulation Tool for the Study of Large-Scale Multiprocessors," in *Proceedings of the International Workshop on Modeling, Analysis, and Simulation On Computer and Telecommunication Systems*, pp. 57–62, 1993.

[12] L. Ost et al., "A simplified executable model to evaluate latency and throughput of networks-on-chip," in *Proceedings of the 21st annual Symposium on Integrated Circuits and System Design*, 2008.

[13] A. Vieira de Mello, L. C. Ost, F. G. Moraes, and N. Calazans, *Evaluation of Routing Algorithms on Mesh Based NoCs*. Porto Alegre: Faculdade de Informatica - PUCRS, 2004.

[14] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "QNoC: QoS architecture and design process for network on chip," *Journal of Systems Architecture*, vol. 50, no. 2, pp. 105-128, Feb. 2004.

[15] A. Mello, L. Tedesco, N. Calazans, and 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*, pp. 178-183, 2005.

[16] T. Bjerregaard and J. Sparso, "Virtual channel designs for guaranteeing bandwidth in asynchronous network-on-chip," in *Norchip Conference, 2004. Proceedings*, pp. 269-272, 2004.

[17] K. Goossens, J. Dielissen, and A. Radulescu, "AEthereal network on chip: concepts, architectures, and implementations," *Design & Test of Computers, IEEE*, vol. 22, no. 5, pp. 414-421, 2005.

[18] M. Schoeberl, "A Time-Triggered Network-on-Chip," in *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, pp. 377-382, 2007.