NoC Performance Model for Efficient Network Latency Estimation

Oumaima Matoussi

NoC Performance Model for Efficient Network Latency Estimation

Oumaima Matoussi <Oumaima.MATOUSSI@cea.fr>
Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France

***Abstract*—We propose a flexible light-weight and parametric NoC model designed for fast performance estimation at early design stages. Our NoC model combines the benefits of both analytical and simulation-based NoC models. Our NoC features an abstract router model whose buffers are updated at runtime with information about the actual traffic. This traffic information is fed to a closed-form expression that computes packet latency and that accounts for network contention at a router basis. We evaluated our hybrid NoC model in terms of estimation accuracy and simulation speed. We compared the simulation results to the ones obtained with a cycle accurate NoC simulator called Garnet. Our NoC model achieves less than 17% error in average network latency estimation and attains up to 14× speedup for a 8×8 mesh.**

***Index Terms*—NoC modeling and simulation, network contention, network latency estimation, on-chip interconnect**

I. Introduction

As modern MPSoCs continue to scale toward hundreds of processing cores on a single chip (e.g. TILE-Gx, Kalray MPPA-256 [1]), the on-chip communication mechanism has, arguably, become one of the main contributors to the overall system performance and cost. The significant role played by the NoC in a complex MPSoC combined with the tight requirements for performance, cost and time to market, fueled the need for fast, complexity-effective and reliable NoC models. However, providing helpful insights on NoC performance characteristics, bottlenecks and impact on the full system performance for each feasible NoC design candidate should not be time consuming nor should it delay the design process. These NoC models are expected to provide fast and accurate evaluations of design candidates at early design stages with rapid sweeps of the network design space.

To address this speed-accuracy trade-off in NoC modeling, various NoC models at different abstraction levels were proposed. Analytical approaches, for instance, use closed-form formulae that depend on the system parameters to provide a safe bound on network performance. This bound is usually pessimistic (worst case latency) and do not account for unregulated traffic generation. Complementary to analytical methods, simulation-based approaches attempt to model all key elements of a NoC (i.e. detailed router microarchitecture). This level of detail enables comprehensive performance evaluation, which leads to accurate NoC performance estimates. However, for large scale systems with hundreds of cores, full system simulation (FSS) with such accurate NoC models becomes unaffordable, time-wise, especially at early design stages.

In order to provide fast, scalable and reliable performance evaluations, we propose a NoC model that exploits the advantages of both analytical and simulation-based NoC modeling approaches. At the very heart of the proposed hybrid NoC model is an abstract router represented by a set of FIFO buffers. This high-level router model that overlooks the architectural details for the benefit of simulation speed, establishes a closed-form expression that estimates the packet network latency. Unlike standard analytical approaches, the devised latency formula is governed by the state of each router (i.e. buffer load), which is in turn dictated by the dynamic (i.e. non-deterministic) behavior of any given workload during simulation. The advantage of our hybrid NoC model is twofold; on the one hand, it inherits the dynamic aspects of simulation-based models and is able to reflect the dynamic interactions between packets and their competition for network resources at runtime. Therefore, a realistic overview of the system behavior is somewhat maintained. On the other hand, the NoC model abstracts away router details and revolves around a simple formula for network latency, which proves to be beneficial to the simulation speed. This addresses the scalability challenge in large-scale NoC simulations. The simplicity and tractability of the model facilitates its integration in a FSS environment without compromising the overall simulation speed.

To evaluate the efficiency of our hybrid NoC model, we compared it, in standalone mode, to a cycle-accurate NoC simulator called Garnet [2]. This comparison was performed in terms of simulation speed and average network latency using different synthetic traffic patterns. We also integrated our NoC model in a fast FSS environment called VPSim [3] supporting a 32-core architecture configuration with a 6×6 mesh and a 3-level cache hierarchy and we evaluated its impact on the overall simulation speed.

II. NoC Background

In this section, we outline basic NoC concepts and review related work on NoC modeling.

A. NoC Basics

A typical NoC is characterized mainly by its topology (e.g. mesh, torus, etc.), routing algorithm (e.g. XY, west-first, etc.) and flow control (e.g. packet-level, Virtual Channel VC-level, etc.) [4]. The topology is defined by the connection patterns of router nodes through channels. A packet can thus cross the network from its source node to its destination node by making several *hops* across routers and channels. Routing determines the actual path that a packet should take through the network to reach its destination. Finally, flow control is in charge of

allocating internal resources of the router (e.g. buffers) and NoC channels to packets as they advance through their route.

NoC performance is usually measured by a metric called packet latency, which is the time required by a packet to reach its destination. The latency of packet pkt_i from its source to its destination is expressed as shown in Equation 1, where:

Equation 1 Packet Network Latency

$$Lat_{pkt_i} = \underbrace{Plat_{pkt_i} \times nbr_{hops} + (L-1) \times FT}_{(a)} + \underbrace{Wait_{pkt_i}}_{(b)}$$

- $Plat_{pkt_i}$ is the physical delay, caused by the network physical aspects such as switch and link delays.
- nbr_{hops} is the number of routers traversed by the packet and determined by the routing algorithm.
- FT is the transmission time of one flit.
- L is the packet size in flits.
- $Wait_{pkt_i}$ is the contention delay.

This latency formula combines a hop-count delay (term 1-(a)) and a contention delay (term 1-(b)). The hop-count delay depends on the intra-router delay (i.e. routing delay + propagation delay through the switch), the inter-router delay (i.e. link propagation delay) and the number of hops taken [4]. The contention delay is defined in terms of:

- link contention, which occurs when packets' paths share one or more links (i.e. packets compete for the same channels),
- buffer congestion happens when limited-size router buffers reach their full capacity.

In this work, we only consider network latency defined by Equation 1, which means that we compute both physical and contention delays but we abstract away the waiting time in injection queues (a.k.a. source queuing time).

B. Related Work

Network modeling is addressed with a wide range of solutions, from simulation approaches to analytical methods. There is a variety of network simulators such as Garnet [2], Noxim [5], BookSim [6], that work at flit-level granularity and provide cycle-accurate packet latency results. These simulators, whether written in SystemC (Noxim) or C++ (BookSim, Garnet), share common grounds in implementing detailed router micro-architecture. However, because of their complexity and highly detailed models, they are not an adequate fit for a large-scale FSS environment running real applications and operating systems. For instance, Gem5 [7] supports Garnet [2] as an accurate NoC simulator, which favors precision to simulation speed (less than 1 MIPS [8]).

Analytical methods (e.g. [9], [10], [11]) aim at finding an upper bound on NoC performance using closed form expressions that depend on the system parameters [12]. Both real time-based analysis used in [10] and network calculus-based analysis employed in [11] compute an upper bound on the traversal time of a packet. Assumptions like a maximum packet injection rate to saturate the network or/and buffers filled to their capacity are usually made to capture the worst-case scenario. For systems that do not require a deadline constraint, *queuing theory* is mostly used (e.g. [9], [13]). Analytical methods are unable to account for non-deterministic traffic generation by the components and thus do not take into consideration delays due to dynamic memory behavior for example. They usually handle well a regulated flow assumption (i.e. a pre-defined delay between two nodes). However, their precision could drop dramatically in the presence of unregulated flows.

An interesting research direction that aims at inspecting a new way of NoC performance estimation relies on combining the benefits of NoC analytical models and NoC simulators (e.g. [14], [15]). In [14], a packet latency estimator called FIST is proposed. The main idea behind FIST is to model each router as a set of load-delay curves. There are two variants of FIST: a static one that relies on offline training to statically obtain these curves, and a dynamic one that adds online training to enhance packet latency estimation. Both static generation of load-delay curves and dynamic training rely on the existence of a cycle accurate NoC simulator. Moreover, the static curves are obtained for specific traffic patterns (e.g. random uniform traffic, neighbor traffic). Also, it is not possible to capture dynamic phenomena like resource contention and buffer congestion by FIST.

Simplified NoC models could also be extracted from existing networks, as presented in [15]. Similar to [14], an accurate NoC simulator is used as a reference to guide an abstract NoC model. A time-annotated trace and network parameters are generated using this reference network and are compared to a trace of network events generated by a FSS environment with zero latency assumption. Successive iterations might be needed following this comparison to refine the abstract NoC model until it converges to the requirements of the design. However, this simple NoC model does not consider the load at a link-basis. As a result, it does not capture network contention.

Following the steps of [14] and [15], we aim at combining the advantages of simulation and analytical approaches in a *hybrid* NoC model, in an effort to reach acceptable accuracy with little impact on simulation speed and complexity. Our hybrid NoC model is able to capture the complex interactions between packets and the NoC's architectural components at runtime by abstractly modeling contention at a link-level as well as buffer congestion. Moreover, unlike [14] and [15], our NoC model is self-contained in the sense that it does not rely on a third-party NoC simulator for result adjustments.

III. Proposed NoC Model

Most NoC structural and functional features (e.g. switching scheme, arbitration, virtual channel allocation, etc.) have an impact on the timing behavior. Taking into account all these details leads to accurate simulation results at the price of simulation speed [2]. To abstract some implementation details for the sake of simulation speed, we propose a simple router model represented by a set of FIFO buffers associated with the router's output channels as sketched in Fig. 1-(a). At simulation time, these buffers help keep track of the packets traversing a given router and most importantly the ones competing for a router's output ports. For example, pkt_i and pkt_j in Fig. 1-(a) share the same output buffer OB_E (E for east) of router r_x.

An alternative is to consider contention at input buffers instead, which is thus equivalent to say that pkt_i and pkt_j compete for the West input buffer IB_W of the downstream router connected to r_x. Only one type of buffers (input or output) needs to be modeled. This is explained by the fact that packets that enter a router through the same input port have undoubtedly arrived through the same output port of the previous router, where their interference was already taken into account. Therefore, we only consider output buffers in our NoC modeling approach.

Thus, we are able to model resource contention effects on packet latency while omitting intricate NoC details. We elaborate next packet tracing and contention estimation.

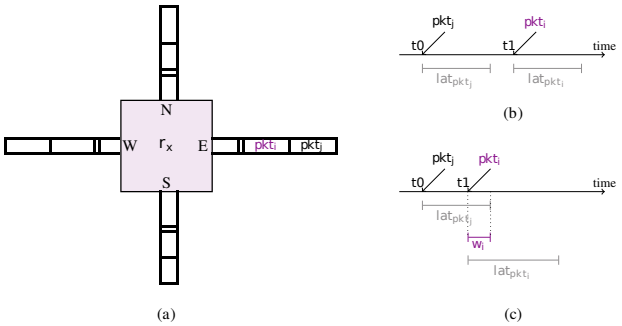


Fig. 1: Router model and contention

A. Overview of the Modeling Approach

Contention modeling implies that the latency computation of a given packet depends on the latency of other packets having a route interfering with the one of the packet in question. Our hybrid NoC model captures this dynamic packet behavior. Algorithm 1 illustrates how packet information is traced at runtime. Each time a packet enters the NoC during simulation, we compute its path (the XY routing algorithm is used), Algorithm 1-line 2. Then, we update the output buffers that are on the packet's path with the packet's id (line 3). We store packet information, namely its path and id, in a trace (line 4). Packets that share the same output buffer are susceptible of causing contention delay. However, in addition to NoC resource-sharing there is another element that determines whether contention is bound to happen. This deciding element is packet arrival time to the output buffer. For example, if (i) a packet pkt_i requires OB_E of r_x at time $t1$ and (ii) another packet pkt_j occupied that same buffer at time $t0$ ($t0 < t1$) but, (iii) if by time $t1$, pkt_j has already left r_x then there is no contention caused by those packets (Fig 1-(b)).

This timing behavior is tricky to model without simulating the NoC at a cycle-accurate level, which is not the purpose of our NoC model. To alleviate the complexity of these time constraints and still provide an acceptable approximation of the contention delay, we propose to evaluate resource contention for a predefined period of time that we coined *contention interval* (CI). We also assume a uniform inter-arrival distribution of packets inside the chosen interval.

We use a *quantized* tracing approach, which means that information about packets are saved only over the span of a CI. In this light, a trace is a snapshot of the communication system state for a given CI. The begining of the first interval is marked with the arrival time of the first packet to the NoC (Algorithm 1-line 5). The begining of a new interval is marked by the arrival time of a packet whose timestamp surpasses the upper bound of the previous interval (lines 11 and 15). Packet information collected in a trace over CI_i are used in CI_{i+1} to compute packet latency (line 12, packet latency computation is explained in Algorithm 2 hereinafter), then discarded (lines 13 and 14) to make room for new incoming information. Accordingly, this simulation model is able to accomodate the variations in NoC behavior in response to a traffic pattern occuring during a CI. Packets are stored in the trace as they enter the NoC. So, they are ordered over a CI based on their injection time, which is crucial when replaying the trace for latency computation (Algorithm 2-line 1). For each packet in the trace, its latency evaluation is realized by examining each router, more precisely each output buffer, encountered on its path (line 2). The position of a packet in an output buffer is an essential factor in contention delay estimation. The first packet in the FIFO does not suffer from contention delay (lines 3

Algorithm 1 Trace_Packets(packet, contention_interval)

```
1: if (packet is first to arrive to the NoC) then
2:   path=Compute_Route(packet.src,packet.dest)
3:   Update_Routers_Buffers(packet.id,buffers)
4:   Log_Packet_In_Trace(packet.id,path,trace)
     ▷ Set contention interval (CI) bounds
5:   interval_start=packet.timestamp
6:   interval_end=interval_start+contention_interval
7: else                                   ▷ packet is not the first in the NoC
8:   if ((packet.timestamp ≥ interval_start) & (packet.timestamp ≤ interval_end)) then
9:     Update_NoC(packet)                 ▷ equivalent to lines 2, 3 and 4
10:  else
11:    if (packet.timestamp>interval_end) then        ▷ a new CI begins
       ▷ compute latency of every packet in previous CI
12:      total_latency+=Compute_Latency(trace)
13:      Clear_Trace()
14:      Clear_Buffers()
         ▷ set new bounds for CI
15:      interval_start=packet.timestamp
16:      interval_end=interval_start+contention_interval
17:      Update_NoC(packet)               ▷ equivalent to lines 2, 3 and 4
```

Algorithm 2 Compute_Latency(trace)

```
1: for all packets pkt in trace do
2:   for all routers rt in (trace[pkt].path) do
3:     if (Position(pkt, rt.buffer)==first) then        ▷ packet is first in buffer
4:       wait_pkt=0                          ▷ packet waiting time in buffer is null
5:     else
6:       prev_pkt=Previous_Packet(pkt,rt.buffer)        ▷ find previous packet
7:       prev_rt=Previous_Router(rt,trace[pkt].path)    ▷ find previous router
8:       if (prev_pkt≠Previous_Packet(pkt,prev_rt.buffer) then
         ▷ Account for contention
9:         wait_pkt+=Queuing_Delay(wait_prev_pkt,Plat,CI,pkts_rt.buffer,VC)
         ▷ next router on previous packet's path
10:      next_rt_prev=Next_Router(rt,trace[prev_pkt].path)
         ▷ next router on current packet's path
11:      next_rt_curr=Next_Router(rt,trace[pkt].path)
12:      if (next_rt_curr≠next_rt_prev) then          ▷ possible HOL blocking
13:        if (Position(prev_pkt, next_rt_prev.buffer)≥BS*VC) then
           ▷ confirmed HOL blocking
14:          wait_pkt+=Congestion_Delay(prev_pkt,next_rt_prev,BS,VC)
```

and 4); for example the waiting time of pkt_j in Fig. 1 is zero. If a packet occupies any other position, then a contention delay might occur. We reiterate that only in case the arrival time of a packet to a router overlaps with the latency of the previous packet during a CI, that the contention delay is non-null (e.g. in Fig. 1-(c), the waiting time of pkt_i in OB_E of r_x is $w_i > 0$). Likewise, the position of a packet in a buffer is a deciding factor in congestion delay computation. If a packet's position in a buffer is greater than the predefined buffer size BS (e.g. $Position(pkt, buffer)$ in Algorithm 2-line 13) then this packet causes additional delay due to buffer congestion.

Both causes of contention, *link contention* and *buffer congestion* (described in Section II-A), are addressed by Algorithm 2 (line 9 for the queue waiting time and line 14 for the congestion delay, which will be elaborated with the help of Equations 2 and 3 respectively in Section III-B). Moreover, computing link contention delay of a packet at each crossed router without considering *packet serialization* phenomenon could lead to pessimistic latency estimates. Therefore, the NoC model takes also into consideration the pipelined behavior of a NoC, in that the interference between packets in their shared sub-paths is handled only once at their first convergence router (line 8).

These NoC phenomena, be it contention, congestion or serialization, require information not only about the currently treated router on a given path but also routers preceding or/and succeeding the current router. Knowledge about the state of packets preceding a given packet is also necessary. For this reason, we utilize functions like $Previous\_Router(rt, path)$, which returns the router preceding router rt in $path$ (e.g. Algorithm 2-line 7), or $Previous\_Packet(pkt, pos, buffer)$, which returns the packet preceding a given packet pkt by pos positions (default $pos = 1$) in $buffer$ (e.g. line 6).

B. Network Latency Estimation

The waiting time of a packet in a router's buffer is the sum of its $Queuing\_Delay()$ (line 9 in Algorithm 2) and $Congestion\_Delay()$ (line 14 in Algorithm 2). The former is computed according to Equation 2 and the latter according to Equation 3. To avoid any confusion, if an equation features more than one buffer, then the buffers are explicitly designated. By way of illustration, we express the waiting time of pkt_i in buffer $[r_x \rightarrow r_y]$, r_x is the router to which the buffer belongs and r_y is r_x's neighbor to which pkt_i is headed, by $wait_{pkt_i}[r_x \rightarrow r_y]$ instead of simply $wait_{pkt_i}$. The queuing

Equation 2 Queuing Delay of pkt_i in buffer $[r_m \rightarrow r_n]$

$$wait_{pkt_i} = wait_{pkt_j} + Plat_{pkt_j} - CI/pkts[r_m \rightarrow r_n],$$
with $pkt_j = Previous\_Packet(pkt_i, [r_m \rightarrow r_n])$.

delay of a given packet in a buffer, as presented by Equation 2, depends on the queuing delay and the physical latency of the previous packet in that same buffer, the duration of the CI, and the number of packets accumulated in that buffer ($pkts[buffer]$) during a CI, assuming a uniformly distributed packet arrival time within a CI. The example in Fig. 2 delineates the queuing delay of pkt_3 across the routers from its source (router r_1 where pkt_3 is initially injected by processing

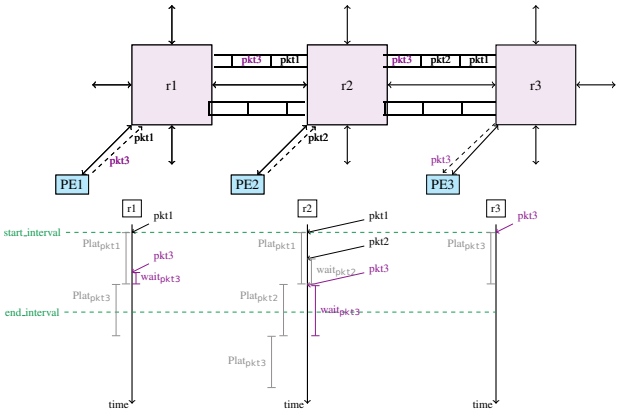


Fig. 2: Buffer waiting time

element PE_1) to its destination (r_3 from which pkt_3 is ejected to PE_3). In buffer $[r_1 \rightarrow r_2]$, pkt_3 is preceded by pkt_1, which is the head of the buffer. Applying the formula in Equation 2 results in: $wait_{pkt_3}[r_1 \rightarrow r_2] = 0 + Plat_{pkt_1}[r_1 \rightarrow r_2] - \frac{CI}{2}$, where $wait_{pkt_1}[r_1 \rightarrow r_2] = 0$ as pkt_1 is the first packet in the buffer. In router r_2, the delivery of pkt_3 to its destination might be further delayed by the arrival of pkt_2 to r_2. Three packets are now competing for output buffer $[r_2 \rightarrow r_3]$. Thus, $wait_{pkt_3}[r_2 \rightarrow r_3] = wait_{pkt_2}[r_2 \rightarrow r_3] + Plat_{pkt_2}[r_2 \rightarrow r_3] - \frac{CI}{3}$, where $wait_{pkt_2}[r_2 \rightarrow r_3]$ depends on the physical delay of pkt_1 in that same buffer. Finally, $wait_{pkt_3}[r_3 \rightarrow PE_3] = 0$ as pkt_3 is the only packet occupying the output buffer leading to PE_3. The overall latency of pkt_3 from source to destination, in the absence of congestion delay, is thus the sum of its physical latency and queuing delay across the different routers on its path: $Lat_{pkt_3} = Plat_{pkt_3}[r1 \rightarrow r2] + wait_{pkt_3}[r_1 \rightarrow r_2] + Plat_{pkt_3}[r_2 \rightarrow r_3] + wait_{pkt_3}[r_2 \rightarrow r_3] + Plat_{pkt_3}[r_3 \rightarrow PE_3] + wait_{pkt_3}[r_3 \rightarrow PE_3]$. Congestion delay might occur

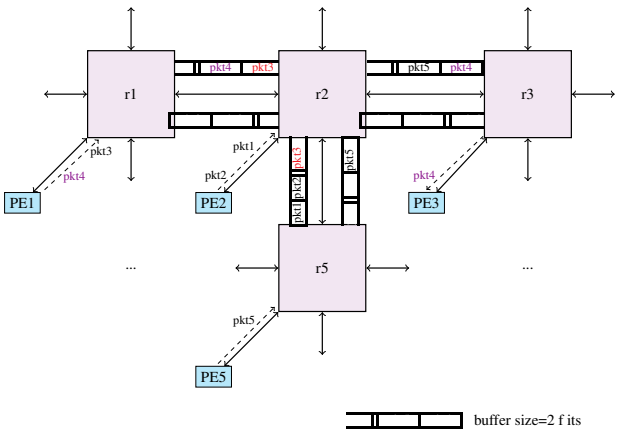


Fig. 3: Head of Line blocking example

Equation 3 HOL Blocking Delay of pkt_i in buffer $[r_m \rightarrow r_n]$

$$hol\_delay_{pkt_i} = wait_{pkt_k}[r_m \rightarrow r_p],$$
with $pkt_k = Previous\_Packet(pkt_j, BS - 1, [r_m \rightarrow r_p])$
and $pkt_j = Previous\_Packet(pkt_i, [r_l \rightarrow r_m])$.

due to a phenomenon called Head of Line (HoL) blocking. HoL blocking happens when packet pkt_i (e.g. pkt_4 in Fig. 3)

is blocked by packet pkt_j (e.g. pkt_3) such that (i) pkt_i and pkt_j share the same output buffer of a given router (e.g. $[r_1 \rightarrow r_2]$), (ii) pkt_j is positioned at the head of that buffer, (iii) both packets are heading to different output ports of the next router (e.g. pkt_3 is headed to $[r_2 \rightarrow r_5]$ and pkt_4 is headed to $[r_2 \rightarrow r_3]$) and (iv) the output buffer to which pkt_j is headed is congested (e.g. buffer $[r_2 \rightarrow r_5]$ is holding two packets pkt_1 and pkt_2, which is its full capacity). Although pkt_i and pkt_j are not competing for the same output port, pkt_i has to wait for pkt_j because the output buffer to which it's headed is at its maximum capacity. The waiting time of pkt_i due to HoL blocking (expressed by Equation 3) is actually the waiting time of pkt_j to be granted access to its destination buffer (i.e. time for a spot to be freed in the buffer). Applying Equation 3 to the example in Fig. 3 results in: HoL blocking delay of pkt_4 in buffer $[r_2 \rightarrow r_3]$ is the time taken by $pkt_2 = Previous\_Packet(pkt_3, BS-1, [r_2 \rightarrow r_5])$ to reach the head of buffer $[r_2 \rightarrow r_5]$, which corresponds to when a slot is made available for $pkt_3 = Previous\_Packet(pkt_4, [r_1 \rightarrow r_2])$ in buffer $[r_2 \rightarrow r_5]$.

Given that NoC performance is influenced by the number of its virtual channels, a high-level approximation of the impact of VCs on network latency is also supported by the NoC model. We chose not to include the VC parameter in the equations presented above for simplicity reasons. In Equations 2 and 3, we consider that each physical channel is represented by one FIFO buffer (i.e. 1 VC). In case of more than 1 VC, we associate as many FIFO buffers as specified VCs to each output port (i.e. a FIFO buffer per VC per output port). So Equations 2 and 3 can be applied the same way to these VC buffers. In this case, the number of packets is computed at a VC buffer level instead of the physical channel buffer.

IV. NoC Evaluation

We evaluate our hybrid NoC performance model with respect to its network latency estimation results and its simulation speed. Our NoC model could be used in standalone mode with traffic generators or execution traces or integrated in FSS.

A. NoC Model in Standalone Mode

Our NoC model is evaluated in standalone mode under various synthetic traffic patterns. The results generated by our NoC model are compared to the ones provided by Garnet [2], a cycle accurate NoC simulator used as a reference. To enable fast evaluation of different NoC parameters and facilitate design configurability, the NoC model offers different tunable parameters such as mesh size, router latency, link latency, buffer size, number of virtual channels and CI.

Fig. 4 showcases average network latency under uniform random traffic for different packet injection rates, while varying the mesh size. The number of VCs is fixed to 2 VCs per physical channel and the buffer size is fixed to one packet. Graphs (a)-(c) in Fig. 4 depict two curves, one for the proposed NoC model and the other one for Garnet. The similarity between the two curves in response to the different injection rates can be noticed in the different graphs. A maximum error of 12% in average network latency is observed. In addition to mesh size, we also evaluated the impact of the number of virtual channels on average latency. Fig. 5 shows the results obtained with three different VC configurations applied to a 4×4 mesh under uniform random traffic. Again, the curves obtained using the proposed NoC model are very close to the ones obtained using Garnet, reaching a maximum estimation error of 14%. In addition to the uniform random traffic pattern, other patterns were injected into the NoC to ensure that irrespective of the traffic the NoC behavior does not deviate from our reference, the Garnet model. The error in average network latency of a sample from the traffic patterns that we tested is reported in Table I. These results were generated with a 4×4 mesh and 2 VCs per physical channel. We observed that the proposed NoC model does not stray away from the reference model under different kinds of traffic. Moreover, our NoC model obtains these latency estimates faster than Garnet and we provide a simulation speed comparison for different mesh sizes in the bar graph in Fig. 6. A speedup of approximately 14× is reached with an 8×8 mesh under an injection rate of 0.8 flits/node/cycle.

traffic pattern \ injection rate	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
uniform-random	2.6	0.5	2.56	2.66	2.5	4.75	5.6	0.8
bit-complement	1.61	4.77	10	14.82	7.1	2.92	-	-
transpose	1.25	4.19	5.63	17	0.58	-	-	-

TABLE I: Error(%) in average network latency w.r.t. Garnet

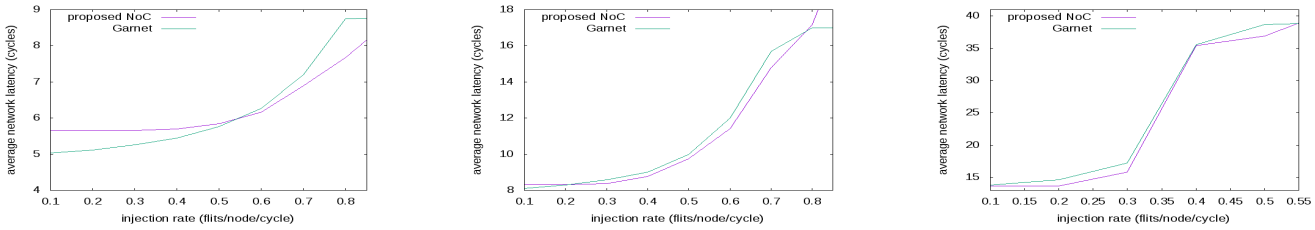
B. NoC Model in Full System Simulation

As a proof of concept, the NoC performance model was integrated in a FSS called VPSim [3]. VPSim is a SystemC/TLM-based virtual prototyping platform for fast software/hardware co-validation and design space exploration of complex architectures, *fast* being the operative word. So, integrating a NoC model in VPSim should not strip this simulator from one of its main features, which is simulation speed.

To make the integration of our NoC performance model in VPSim possible, a functional NoC model is needed alongside the performance model. The former ensures communication between the different components by forwarding transactions from initiators to targets by means of TLM sockets and transport functions. When a packet enters the NoC, key information such as the source-destination pair and the packets' timestamps is determined and fed to the timing model. Once network latency computation is completed, the packet is delivered to its destination while carrying timing information updated with the NoC latency. To evaluate the impact of integrating the NoC model in VPSim on the overall simulation speed, we used workloads from PARSEC and SPLASH-2 benchmark suites running on a Linux kernel. Table II illustrates the slowdown (measured in MIPS) observed for a sample of the executed workloads. This slowdown is obtained in comparison to VPSim with a simple hop-count NoC model. In these experiments, we simulated a 32-core architecture composed of two private

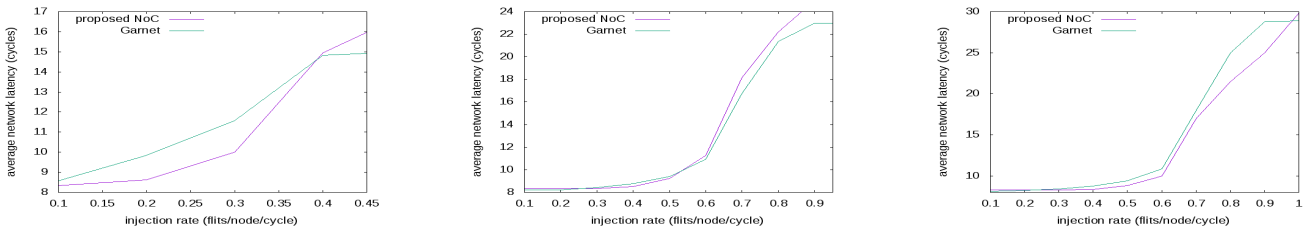
	swaptions	radiosity	barnes	fmm	blackscholes	water-spatial
slowdown	1.63	2.5	1.54	2.31	2	1.5

TABLE II: Slowdown in MIPS of VPSim with the NoC model



(a) 2*2 mesh (b) 4*4 mesh (c) 8*8 mesh

Fig. 4: Average network latency of different mesh sizes under uniform random traffic



(a) 1 VC (b) 3 VCs (c) 4 VCs

Fig. 5: Impact of VC variation on average network latency of a 4*4 mesh under uniform random traffic

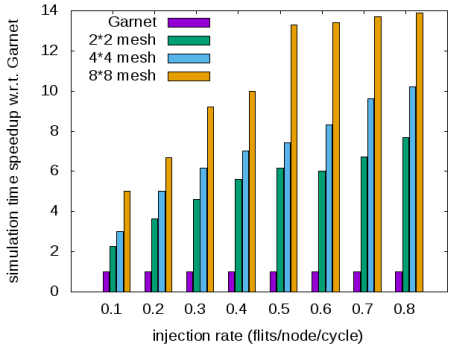


Fig. 6: Speedup of proposed NoC model w.r.t. Garnet

L1/L2 caches per core, a shared 32-bank L3, a physically distributed off-chip memory of 4 DDRs and a 6×6 NoC. These experiments give an overview of the possible simulation overhead due to the integration of our NoC model in VPSim. Since the real system is not available, analyzing the overall accuracy of VPSim augmented with the NoC performance model is not possible yet. Instead, we validated the accuracy of the NoC separately as reported in the previous section.

V. Conclusion

We proposed a NoC modeling approach that defines an appropriate abstraction level providing reasonable performance estimates with little impact on simulation speed. A router was modeled as a set of buffers, where packets are stored during simulation for a predefined time interval. This router model is coupled with an analytical formula that computes packet network latency based on the state of the router at runtime. Our hybrid NoC model is able to capture resource contention and buffer congestion. The experiments showed that our NoC model ensures representative results (less than 17% error in network latency estimation) with good simulation speed (14× speedup compared to Garnet) and is fit for FSS ($\leq$2.5× slowdown in overall simulation speed when paired with VPSim).

Acknowledgment

This work is partially funded by H2020 European Processor Initiative (grant agreement No 826647).

References

[1] B. D. de Dinechin and A. Graillat, "Network-on-chip service guarantees on the kalray mppa-256 bostan processor," in *AISTECS*, NY, USA, 2017.
[2] N. Agarwal, T. Krishna, L.-S. Peh, and N. Jha, "Garnet: A detailed on-chip network model inside a full-system simulator," 04 2009, pp. 33–42.
[3] A. Charif, G. Busnot, R. Mameesh, T. Sassolas, and N. Ventroux, "Fast virtual prototyping for embedded computing systems design and exploration," in *RAPIDO '19*, 2019.
[4] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, CA, USA, 2003.
[5] V. Catania, A. Mineo, S. Monteleone, and et al., "Noxim: An open, extensible and cycle-accurate network on chip simulator," in *ASAP*, 2015.
[6] Nan Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally, "A detailed and flexible cycle-accurate network-on-chip simulator," in *ISPASS*, 2013.
[7] N. Binkert, B. Beckmann, G. Black, and et al., "The gem5 simulator," *SIGARCH Comput. Archit. News*, Aug. 2011.
[8] S.-h. Kang, J. Kang, and S. Ha, "Fast parallel simulation of a manycore architecture with a flit-level on-chip network model," in *SAMOS*, 2018.
[9] U. Y. Ogras, P. Bogdan, and R. Marculescu, "An analytical approach for network-on-chip performance analysis," *CADICS*, 2010.
[10] L. Abdallah, M. Jan, J. Ermont, and C. Fraboul, "Wormhole networks properties and their use for optimizing worst case delay analysis of many-cores," in *SIES*, 2015.
[11] F. Giroudot and A. Mifdaoui, "Tightness and computation assessment of worst-case delay bounds in wormhole networks-on-chip," in *Real-Time Networks and Systems*, 2019.
[12] Z. Qian, P. Bogdan, C.-Y. Tsui, and R. Marculescu, "Performance evaluation of noc-based multicore systems: From traffic analysis to noc latency modeling," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 21, 2016.
[13] N. Nikitin and J. Cortadella, "A performance analytical model for network-on-chip with constant service time routers," in *ICCAD*, 2009.
[14] M. K. Papamichael, J. C. Hoe, and O. Mutlu, "Fist: A fast, lightweight, fpga-friendly packet latency estimator for noc modeling in full-system simulations," in *the Fifth ACM/IEEE International Symposium*, 2011.
[15] D. Lugones, D. Franco, D. Rexachs, J. C. Moure, E. Luque, E. Argollo, A. Falcon, D. Ortega, and P. Faraboschi, "High-speed network modeling for full system simulation," in *IISWC*, 2009.