2012 Sixth IEEE/ACM International Symposium on Networks-on-Chip

Reservation-based Network-on-Chip timing models for large-scale architectural simulation

Javier Navaridas*§, Behram Khan*, Salman Khan*, Paolo Faraboschi†, Mikel Luján*
* School of Computer Science, The University of Manchester, UK
† Intelligent Infrastructure Lab, Hewlett Packard
§Corresponding author: *javier.navaridas@manchester.ac.uk*

***Abstract*—Architectural simulation is an essential tool when it comes to evaluating the design of future many-core chips. However, reproducing all the components of such complex systems precisely would require unreasonable amounts of computing power. Hence, a trade off between accuracy and compute time is needed. For this reason most state-of-the-art tools do not have accurate models for the networks-on-chip, and rely on timing models that permit fast-simulation. Generally, these models are very simplistic and disregard contention for the use of network resources. As the number of nodes in the network-on-chip grows, fluctuations with contention and other parameters can considerably affect the accuracy of such models. In this paper we present and evaluate a collection of timing models based on a reservation scheme which consider the contention for the use of network resources. These models provide results quickly while being more accurate than simple no-contention approaches.**

I. Introduction

The relentless improvement of electronic miniaturization has provided the possibility of integrating several processing cores into a single chip. Most modern general purpose processors have several processing cores and, indeed, we can find processors with over 10 cores offered by several companies, such as the 10-core Intel Xeon processors [34], the 12-core AMD Opteron processors [15] or the 16-core Sparc processors [36], [38]. Even more processing cores are provided by the 48-core Larrabee processor from Intel [35] or the 64-core TILE64 processor offered by Tilera [9]. In fact, the design and development of new processor architectures able to integrate over *one thousand* cores in a single chip is a current *hot topic* [13]. Indeed, some authors speculate that such technologies may become a reality within this decade [21].

Several international projects are pursuing this objective, although with different perspectives and objectives. The ATAC system, for instance, explores the viability of using a broadcast optical medium as the communication infrastructure within a 1000-core chip [21]. In contrast, Rigel [20] has been devised as a programmable accelerator comparable to current GPUs because of its single-process multiple-data (SPMD) execution model. The main reason that such architectures have become so attractive for the scientific community is improved power and thermal characteristics by means of per-core frequency and voltage regulation (or even shutting off idle cores). They are also more resilient to failures due to their greater redundancy.

When designing new chip architectures it is essential to select appropriate evaluation methodologies. For example in the first phases of the design it is preferable to explore as much of the design space as possible. Thus, fast evaluation methodologies such as functional simulation or analytical modelling are favoured in these phases even when they offer limited accuracy. As the final design approaches, we need to assess chip functionality and performance through more detailed simulations–a practice that can be seen as *virtual prototyping*. The high complexity of these models demands large amounts of computing power to carry out simulation.

Our research group is currently working in the *first design phases* of a future 1000-core architecture: TERAFLUX [31], [37]. Given that simulation speed is a valuable characteristic for us now, we have selected the COTSon simulator [3] as it provides adequate accuracy while being lightweight enough. To speed-up execution COTSon processes a block of events at a time and offers other facilities such as statistical sampling. When modelling a large chip we need to provide timing models that do not slow-down the simulation while still being representative of the execution. How to strike such a balance is the key insight provided by this paper. Our focus is on how we can produce timing models for NoCs that we can use when considering the architectural design of large many-core chips without slowing down the simulation.

In this paper we propose a collection of models which improve accuracy, comply with COTSon restrictions and are lightweight enough to perform fast simulation. These models rely on the idea of reserving resources for the period of time that they are in use, allowing contention to be modelled. We perform an exhaustive evaluation using workloads of different nature: synthetic traffic from independent sources, traces from cache coherence, transactional memory and message passing applications and cache-coherency like synthetic traffic to simulate a 1024-core NoC. The wide variety of evaluation scenarios provide insights into the strengths and weaknesses of the different models.

II. Simulation of Future Many-Core System

As device scaling continues to track Moore's law and with the end of corresponding performance improvements in out-of-order processors, multicore systems have become the norm. If current trends continue, a chip with over 1000 cores may be available as early as 2016. Given manycores' inherent complexity, simulation is essential for exploring the design space of a future architecture. Moreover, while simulators

978-0-7695-4677-3/12 $26.00 © 2012 IEEE
DOI 10.1109/NOCS.2012.18

have focused on microarchitecture in the past, high level architecture (modelling the on chip network, memory hierarchy and coherence, etc.) is becoming increasingly more important. Simulating a processor involves making decisions on trade-offs between simulation speed and accuracy. At one end of the spectrum, purely functional simulation provides little insight into system performance, but allows for fast simulation. At the other end, a cycle-accurate full-system simulation gives reliable estimates of performance, but at the cost of very long running simulations. There exists a range of options between these extremes, some of which are explored in this paper.

A number of simulation tools can be used to model processor and system architectures. Simplescalar [4] has been popular amongst the processor architecture community. It provides detailed processor and memory models, but is not a full-system simulator. Simics [23] performs full-system simulation and has extensions such as Gems [24] and Simflex [17] which add considerable sophistication to the timing models. Graphite [26] is a Pin-based simulator that concentrates on running speed by parallelising execution and by using probabilistic models instead of cycle-accurate models. COTSon [3] is a full-system simulator that leverages AMD's SimNow [8] and scales up to simulating 1000 cores [22]. To improve simulation speed it does not allow callbacks. Therefore, we can not simulate the NoC and stall an event execution until the corresponding packets are delivered. Instead, we must calculate the latency for an event as soon as it is encountered. This allows for simplicity and speed in the simulation infrastructure, but means that care must be taken in implementing timing modules that accurately reflect device behaviour.

Network on Chip simulation is also supported by a number of tools. Hornet [19] is a cycle accurate NoC simulator that can be driven by traces, by a built-in MIPS simulator or by native applications instrumented by Pin [5]. Garnet [2] also provides a cycle accurate model, and can interface with Gems to model a full system, and with ORION [39] to provide power estimates. Noxim [29] is a more limited tool that models only 2D mesh interconnects and is driven through synthetic patterns rather than application-based traffic. SICOSYS [32] has very detailed models of several router architectures, which allows obtaining very accurate performance measurements, close to those obtained with a hardware-level simulator, but at a fraction of the required computing resources. It can interface with RSIM [28] and SimOS [33] to perform full-system simulation. Topaz [1] is a recently released extension of SICOSYS capable of interfacing with Gems to perform full-system simulation of CMPs. The Gem5 simulator [12], which merges the well-known M5 [11] and Gems simulators, has support for most state-of-the-art processor technologies while providing flexible models for the memory subsystem. Regarding the NoC, it allows two simulation models: a fast no-contention model and a detailed packet-level simulation. It does not provide an intermediate solution as those discussed in this paper.

To our knowledge, the only alternative implementation of fast timing modules akin to those proposed here is FIST [30]. Instead of modelling contention, FIST uses load-latency curves obtained from training simulations to estimate packet latency. It has, however, several limitations: the load-latency curves need to be obtained specifically for each traffic pattern, so if an application has a mixture of traffic patterns, or a patternless traffic it cannot be modelled properly. Also it requires tracking the load handled by the NoC. As instantaneous load tracking would be prohibitive in terms of synchronisation, a sweet-spot would need to be found for how often we calculate/estimate network load. Again this means reaching a trade-off between accuracy and speed. At any rate using FIST would involve executing a stand-alone network simulator to train the FIST estimator, which has a definite impact on the overall time required to perform simulation.

III. Fast Timing Modules

This Section is devoted to discussing the different timing models considered in this paper. We will start with some simplistic models that have been used by the community, but that in our opinion will not be appropriate once the number of on-chip cores goes over a few tens. Then we will present the reservation-based models we are proposing. Finally, we will consider some illustrative statistical models. In all cases we will discuss the strengths and limitations of the models.

A. Simple No Contention Models

The simplest models and, as discussed before, the most prominent in the literature, do not consider network contention. For instance, there are some models which provide a constant latency for all network accesses regardless of the packet size, the number of hops and so on (e.g. *vanilla* Simics [23]). It is clear that disregarding any knowledge about the network makes this model very inaccurate; it can be used to test functionality, but should not be used to evaluate the performance of a large system. In our experiments this model has been denoted as '*fixed*' and considers all communications to require 16 cycles to complete.

An improvement of this model would consider the *distance* and the *packet size* to model the latency, but again without considering any contention in the NoC. This model is still very poor as the NoC is a common resource that has to be shared by all the cores and, hence, it is not likely that network packets travel through the NoC without encountering any contention. In the discussions below we will denote this model '*no contention*'. As we are considering cut-trough switching this module will return a latency of $distance + \#flits - 1$.

B. Reservation-based Models

We propose several models that consider contention for the use of network resources. The basic element in these models is a 'resource', which in general represents a communication channel. To use one of these resources it is necessary to *reserve* it for a given period of time, forcing other accesses to the same resource to wait until it is freed. Given the no-callback limitation of COTSon, when a packet wants to reserve a resource which is in use, it then will reserve the first available

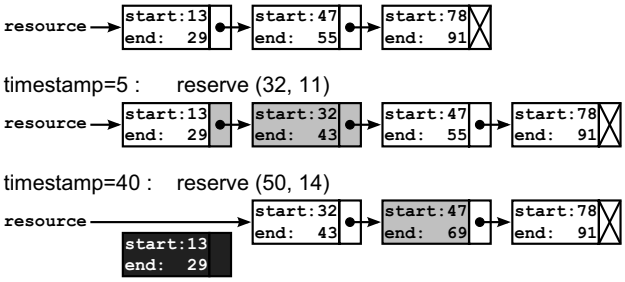

Fig. 1. Example of the reservation data structure. The resource starts with three reservations (top). A new reservation which requires adding a new element–in grey (middle). A new reservation which requires modifying an existing element–in grey–and permits removing an old element–in dark (bottom).

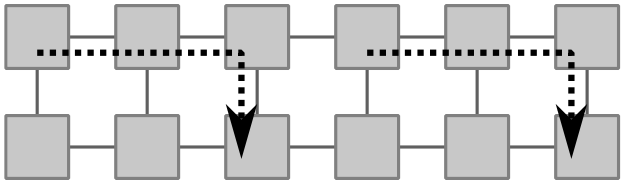

Fig. 2. Example of two packets which do not compete in a NoC but do in the 'direction con' model.

period. Note that as COTSon has to provide the time for executing a complete operation, more than one packet may be needed to transmit trough the NoC, and therefore, we may need to make reservations for the future.

We have implemented a sorted linked list in which every element in the list represents a period in which the resource is reserved (Fig 1). The only required operation allowed is to reserve the resource for a given period of time; given the timestamp the reservation should begin and the duration (in cycles). This operation searchs for a free period of time that can accommodate the required reservation and will return the timestamp when it ends. Therefore when a resource is reserved, the delay can be calculated instantaneously after reserving, just by subtracting the end of the granted period and the current timestamp. As a secondary effect, every time this operation is invoked, it will remove all *outdated* reservations from the list (those which have already finished). This helps to keep the list in a manageable size independently of the simulation length. To further reduce the number of elements in the list, a new element will be added to the list only if extending an existing one (i.e. increasing the timestamp it ends at) is not possible.

Based on this data structure we have implemented four different timing modules. The first two models are aware of the network topology – for the purpose of this paper we will consider a mesh – the other two are topology agnostic.

The first model has been denoted '*direction con*'. It considers each row and each column of the NoC as a shared resource in each direction. A core trying to inject a packet will reserve the row and the column as dictated by XY routing. First it will reserve the row for the number of hops required in the X dimension starting in the current moment, the reservation of the column will start after the end of the previous one and will last for the number of hops in the Y dimension. To the final latency obtained by reserving the column we will add the packet length. The main limitation of this model is that it only allows one packet travelling in each row/column and direction of the NoC, while an actual NoC may allow several packets travelling in the same row, provided they are not competing for resources. For example the two packets in Fig. 2 do not compete in a real NoC, but require one waiting for the other in this model as both of them are using the same row. As we will see later, this will be the reason for this model reaching congestion before the modelled NoC.

The second model, '*path con*', considers each link of the topology as a resource. A core has to reserve all the links towards the destination for a period equal to the packet length. Resembling the way that packets travel through a network, the end of a reservation will affect the starting timestamp of all subsequent reservations. As we are modelling a virtual cut-through network, a link can be reserved once the header has arrived to a router, in other words we can start reserving $\#flits - 1$ cycles before the end of the reservation of the previous link. For simplicity we have considered that the packet always follow a XY path, but any routing algorithm could be easily implemented. In principle this is the model whose behaviour most closely mimics the actual network as it emulates packets movement. However as router arbitration is not modelled its behaviour can differ from the actual network. Also it is the most complex of the models as it requires modelling lots of the components of a network (all the links).

The topology-agnostic models consider the network as a collection of channels or *pipes* without any particular arrangement. When a core wants to send a packet it randomly selects one of the pipes and reserves it for the time required to perform the communication ($\#flits + distance - 1$). This simplifies simulation while still considering contention for the use of resources. We have implemented two different versions of this model. In the first one, '*pipes*', all the cores share all the pipes, so the contention in any part of the network will affect all the cores equally. The second one, '*pipes dist*' is a distributed implementation in which the system is divided in groups of cores which share a collection of pipes. This way contention in an area of the network does not affect other areas. This model simplifies distributed simulation as there is no need for a shared data structure.

C. Statistical Models

We will close our study with two timing modules that do not consider contention directly but assume that travelling packets will suffer some extra delay due to other in-transit packets. The first model, '*load estimation*', estimates the current network load and approximates the latency either as *non-congested* in which latency is barely affected, or as *congested* in which latency is greatly increased. The approximation uses a exponential distribution to select the latency, based on the estimated load and the distance the packet has to travel. In this

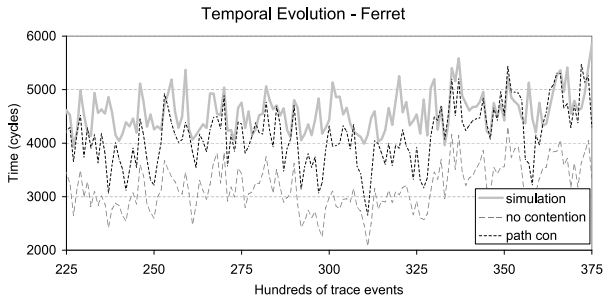

Fig. 3. Extract of the temporal evolution of ferret.

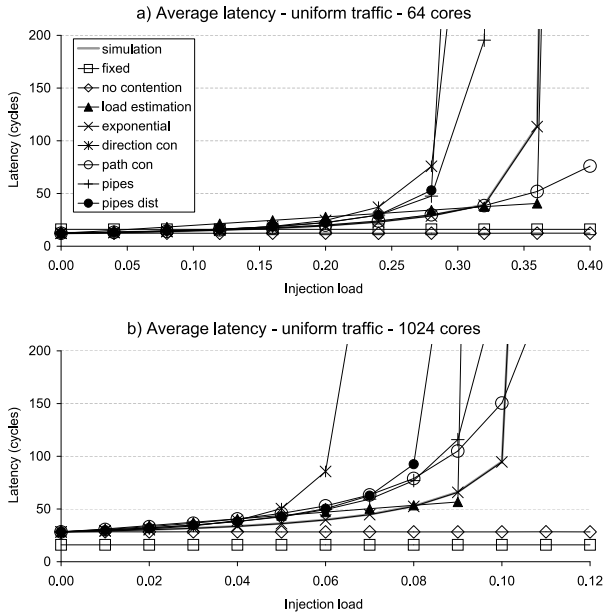

Fig. 4. Average latency. Random uniform traffic. a) 64 cores. b) 1024 cores.

paper, our estimator for the load is calculated as the number of injected packets divided by the elapsed time. Note that this model follows the same idea as FIST [30] but in a simplified way.

Finally there exists the possibility to estimating latencies from a real simulation of the network. Although very complicated models can be extracted we think that estimations from the average latency in simulation does suffice for the purpose of this paper. We will use an exponential distribution in which the λ parameter depends on the average delay measured during an actual simulation and the distance to be traversed, so that it provides the same average latency as simulation. We denoted this model '*exponential*'.

IV. Experimental Work

In this section we show how the different timing models for the NoC behave under different operating conditions and compare them with INSEE, a lightweight time-accurate network simulator [27]. For the purpose of this paper we considered minimal NoCs: simple mesh topologies using XY routing and a single virtual channel. First we will use synthetic traffic from independent traffic sources which allows us to easily vary the pressure exerted on the communication infrastructure. Next we will test them using traces from applications. This will allow us to assess their accuracy for a range of applications of interest. The use of traces of applications simplifies comparing the timing modules because network utilization will remain the same for each workload. If full-system simulation would have been used, each timing module could have had a different set of messages, as the performance of the NoC may affect the overall traffic; e.g. in the case of two memory accesses to the same memory address, one for reading and the other for writing, the order in which they arrive to the cache will affect the subsequent communications: if the read arrives first it may require an extra invalidation packet once the write is executed.

For the sake of completeness we use traces from applications of a diverse nature: directory-based shared memory, transactional memory and message passing. The former two are of interest in our research as they are the 'hot' application models to be run in the manycore systems we investigate. The latter, used in parallel and distributed systems, has a two-fold purpose. On one hand, it provides specific traffic characteristics that are not covered by the previous two models. On the other hand, it allows us to assess whether the proposed models may be used in other design domains such as, for instance, by the cluster computing community. Finally we will use synthetically generated coherency-like traffic to evaluate systems composed of 1024 cores. This will provide some insights on the scalability of the evaluated timing models.

The employed methodology is as follows: the results obtained by the different modules are contrasted with those from the time-accurate network simulator. We will consider the following three figures of merit:

1) *Simulated execution time of the application*. This provides a first approximation to the accuracy of the different models.
2) *Similarity score* metric. A more profound assessment of accuracy. We measured the simulated time to execute every 100 trace events and compute the average difference with the actual simulator. A lower value means that the evolution of the application is closer to simulation, i.e. more accurate.
3) Actual *running time*. This figure gives an insight into simulation speed. To provide fair speed estimates we developed the timing models as stand-alone tools.

Fig. 3 illustrates the similarity score metric. It shows the evolution of ferret with the simulator, the 'path con' model and the 'no contention' model. The average of the differences between the simulator and a model in each of these points is its similarity score. For instance, we can see how the evolution of the 'path con' model is always closer to the evolution of the simulator than the 'no contention' model. Therefore it will have a lower similarity score, meaning that it is more accurate.

A. Random Uniform Traffic

The use of independent traffic sources is a typical evaluation tool which allows to extract some *raw* characteristics of

a network [16]. In this study we use it with a different perspective, though. Since independent traffic sources allow to adjust network load at will, we can use them to show the behaviour of each model with a wide range of communication needs. We model the traffic sources as independent entities injecting packets randomly along time following an exponential distribution. Spatially, packets are distributed uniformly through the network.

The average latencies reported by each model are shown in Fig. 4. In the plot we can see that when the network load is low all the models offer latency figures very similar to those of the real simulation. As network load increases all the models follow a trend similar to the real simulation. The only exceptions are the two simple models: 'fixed' and 'no contention' as they are unaware of the network load. The main difference, though, is when the different models start to behave as saturated, i.e. having very high latency figures. For example the 'direction con' model reaches saturation very early when compared with the real simulation. This is because modelling each row/column as a single shared resource is an extremely restrictive model (see again Fig. 2). Anyhow the network of a many-core system is not likely to suffer from persisting states of saturation when running shared memory or transactional memory applications as the threads requesting the use of the network will commonly stall until the reception of an *ack* packet indicating that the operation has succeeded. In general we can state that the contention-aware timing modules produce latency figures that resemble the shape of those of a NoC. Notice that the 'exponential' model practically overlaps with the simulator in this experiment. This is because this model uses the simulation average latency to generate a latency distribution with the same value. However, we will see later that with the real applications it can not capture application dynamics properly, throwing worse results than the reservation-based models.

B. Directory-based Cache Coherency

Undoubtedly, the directory-based cache-coherent shared memory application model is the most important one to bear in mind for studying many-core architectures. To generate the traces we used the COTSon framework [3], extended to implement directory based cache coherence protocol. Table I presents the main parameters of the simulated architecture. The network simulation is driven through a trace generated from COTSon. This trace logs all coherence and data messages that enter the network. We utilized the Parsec benchmarks [10] with 'simsmall' input data run with 32 cores (arranged in a 8×4 mesh), the maximum available in our configuration. As discussed in [7] the spatial patterns of the application composing this benchmark suite do not present any noticeable 'hot spot'. We will see later that this may happen with the transactional memory applications.

Fig. 5 shows the results obtained by each of the timing modules. We can see how the four reservation-based models offer noticeably more accurate results, both in terms of execution time and similarity score than the other models. Also looking at the time to execute the different modules we can see that we can achieve noticeable acceleration with respect to the simulation, in par with the simpler models.

Feature	Description
L1 Cache	32-KB, 64 byte cache line, 4-way associative, write-through, 1 cycle latency
L2 Cache	512-KB, 64-byte cache line, 8-way associative, write-back, 16 cycle latency
Network	2D mesh topology, 16 cycles link latency
Main Memory	150 cycles latency
Directory	Full-bit vector sharer list; directory cache 10 cycle latency

TABLE I
MEMORY SYSTEM CONFIGURATION

Benchmark	Parameters
Vacation	-n2 -q90 -u98 -r8192 -t4096 -c32
Kmeans	-m40 -n40 -t0.05 -i random2048-d16-c16.txt -p32
Genome	-g128 -s32 -n8192 -t32
Intruder	-a10 -l4 -n2038 -s1 -p32

TABLE II
STAMP BENCHMARK SUITE PARAMETERS

C. Transactional Memory

Transactional memory is a novel memory model devised to simplify the development of shared memory applications and, especially, thread synchronisation [18]. This scheme provides support for transparent *atomic* execution of instructions in a shared memory system. Since this application model is gaining interest in the many-core community and it is an essential characteristic of our design, our study includes some examples of transactional memory applications. We generated 32-core traces of some of the STAMP benchmarks [14] following a procedure similar to the one used for Parsec. The parameters for these benchmarks are listed in Table II.

The traffic generated by this kind of applications has similar nature to cache-coherency's. However there is a noticeable difference: if the region of the memory which has to be accessed transactionally is small it will be likely located in a single core's cache. This core will then become a 'hot stop' as all the transactions will require traffic to and from this node. For this reason, the spatial patterns of these applications are more likely to have an unbalanced use of the network.

The results for the transactional memory experiments are plotted in Fig. 6. In the plot we can appreciate that in general, for this kind of applications, the differences in simulation time are minute. This is because these applications generate very low traffic into the network and therefore there is not much contention. However the similarity score shows that the reservation-based models are more accurate than the other ones. Again, the computation times required by all the timing modules remain similar and noticeably faster than simulation.

D. Message Passing

To conclude with the trace-based experimentation, we compare the different models using the *message passing* version of the NAS Benchmarks [6]. The main difference of this application model when compared with the previous two

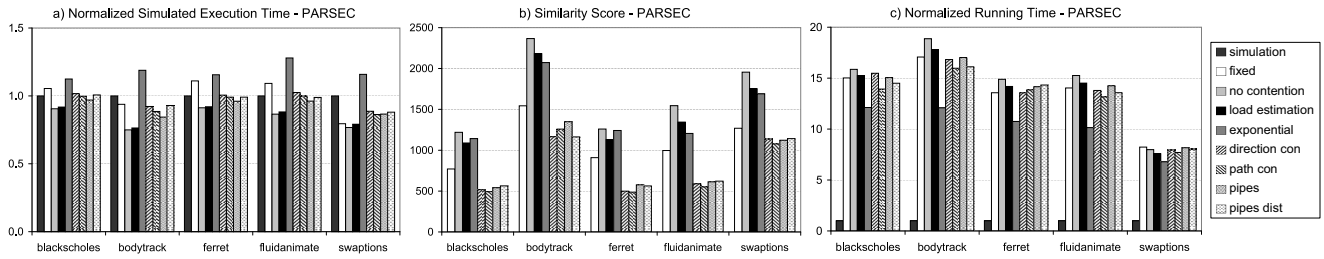

Fig. 5. PARSEC benchmarks. 32 cores. a) Normalized simulated execution time. b) Similarity score. c) Required computation time.

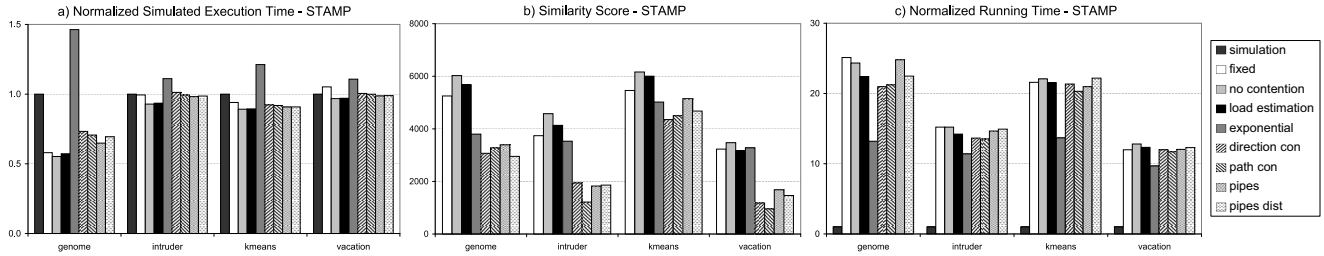

Fig. 6. STAMP benchmarks. 32 cores. a) Normalized simulated execution time. b) Similarity score. c) Required computation time.

is that communications are determined explicitly within the applications. This does not prevent tasks from injecting as much traffic as they want into the network. For this reason, network can suffer long-lasting periods of saturation. The traces were obtained from a real cluster of PCs interconnected by means of a Myrinet interconnect. The class A of BT, CG, IS, LU, MG and SP benchmarks were run using a customised version of MPI logging to capture the traces [25]. We use the largest trace we were able to generate with our set-up, 64 tasks. Hence, a 8×8 mesh was used in this experiment.

The results for the different benchmarks are plotted in Fig. 7. We can see how, with this kind of applications, differences between models are more significant than with the previous two. This is because, as discussed, network saturation does appear and persist. In fact most of NAS benchmarks are composed of different phases in which computation and communication are alternated. This means that during computation phases, network is barely used but, in turn, during communication phases the network suffers from severe saturation.

We can see how, in general, all the models are very inaccurate as the simulation times differ greatly from the simulator. This is because, modelling the behaviour of a saturated network is nearly impossible without modelling the whole network. Some of the reservation-based models do a relatively good work with some of the benchmarks but fail with the others. For example, as stated before, 'direction con' is severely affected by restriction of only one packet per direction. In the case of BT and SP benchmarks, they use a near neighbour communication pattern which essentially allows 7 packets in each row or column whereas 'direction con' has to deliver the packets sequentially (Fig. 2).

If we look at the actual computation time, we can see that the high pressure exerted over the network by this kind of applications makes the reservation-based models slower than the other models. This is because the large amount of packets in the network is translated into lots of reservations in the data structures containing our resources. As the resources are implemented as sorted lists, each time a reservation is called, the whole structure has to be browsed which lowers performance. At any rate the reservation-based modules are still noticeably faster than the simulator in this context.

To close this subsection we want to remind the reader that this kind of application is not actually of interest in our research but were included to show the inadequacy of such models for saturated networks. For the rest of the paper, we do not consider the conclusions of this subsection, since this workload forces the network to operate in a state that is not likely to occur in our set-up with our target applications.

E. Directory-like traffic in a 1024-core chip

As stated before, our experimental set-up restricts the size of the applications we can use to evaluate our models. However, the ultimate aim of our research is to evaluate chips with up to 1024-cores. For this reason, following a methodology akin to the one presented in [21], we have generated synthetic directory-like traffic for a 1024-core system (32×32 mesh).

We present the results for these synthetic communications in Fig. 8. In general, all the reservation-based models but 'direction con' consistently provide better accuracy than the other ones. The reason for the low accuracy of 'direction con' is clear: allowing only one packet per each 32-core row (or column) is extremely restrictive, and results in overly pessimistic performance. Regarding the computation time, we can see how all models except 'path con' execute more than two orders of magnitude faster than the simulator, which means good scalability levels. In the case of 'path con', it requires using roughly 4,000 'resource' structures and use, on

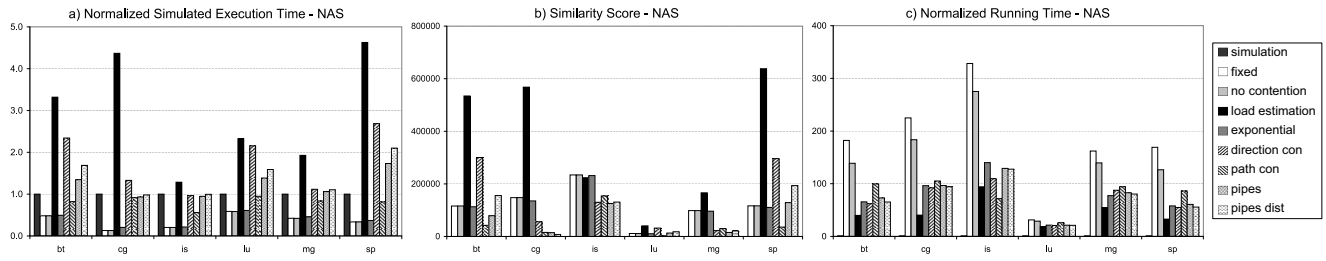

Fig. 7. NAS benchmarks. 64 cores. a) Normalized simulated execution time. b) Similarity score. c) Required computation time.

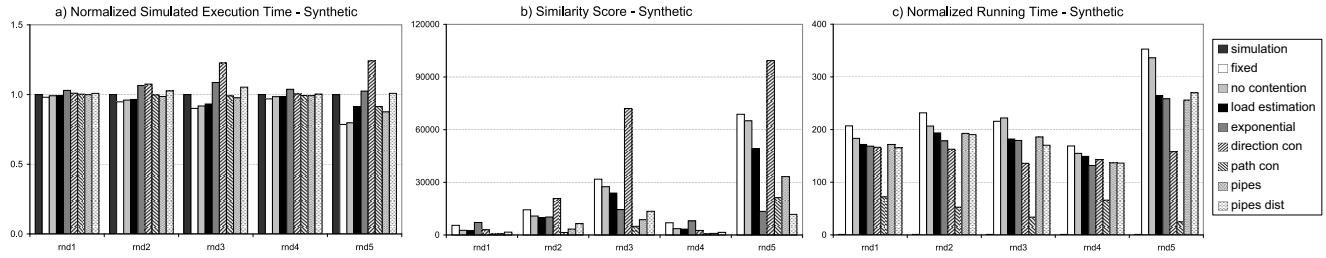

Fig. 8. Synthetic directory-like traffic. 1024 cores. a) Normalized simulated execution time. b) Similarity score. c) Required computation time.

average around 20 of them for each packet. This render this timing module noticeably slower than the others (but still one order of magnitude faster than actual simulations). This may not be acceptable for the simulation of large chips, as it may become an important simulation bottleneck.

F. Discussion

Looking at the results, we can see how 'path con' is consistently the most accurate. This is because it is the one having highest resemblance with the actual behaviour of the network. It models the topology and the way physical links are employed during normal operation, but avoids modelling all the complex logic within the router (requesting, VC management, arbitration, QoS, congestion control). However, we found that this model cannot scale well to our target of 1024 cores as it may slowdown simulation considerably.

In the case of 'direction con' we find out that the restriction of having a single packet per row or column is exceedingly restrictive, specially with high loads or large systems. For this reason we should discard this model. However, taking into account that it produces reasonably good results with the 32-node systems we may consider an intermediate solution which splits a large network into smaller 'direction con' networks which should provide better results for large systems while still being lightweight enough.

Another model which is worth mentioning is the 'pipes dist'. It has shown good accuracy and reasonable speed and scalability, but the more interesting property is that it can be used in distributed simulations without the need for a shared resource for simulating the network, which would simplify parallelising simulation as no synchronization would be needed among different parallel instances. These two characteristics make it a good candidate to scale up our simulator up to the 1000-core systems subject of our research.

In general, it is reasonable to state that the proposed models provide more accurate results than simpler no-contention models simply by considering packet interaction in a very simplified way. Moreover, these models do not only provide more accuracy when simulating the same amount of traffic but may help to detect when the network becomes a bottleneck. For instance, consider two alternate architectural designs with different communication needs, one of them being communication biased. If we evaluate these two designs using a no-contention model the communication-biased design would be in clear advantage as network contention and saturation do not affect its performance. The same evaluation with a contention-aware model would provide a more sensible evaluation.

Finally, although energy estimation is outside of the scope of this paper, it is fair to say that the proposed models should be more accurate than no-contention models because both are aware of the distance travelled by the packets but, in addition, the contention-aware models also provides an estimation of the buffering time employed by the packets, which has a definite impact on NoC consumption.

V. Conclusions

In this paper we have proposed and evaluated a collection of timing models for the NoC in the context of fast simulation of many-core systems. The main novelty of these models is that they implement a reservation scheme that allows modelling network contention without the need for tracking instantaneous network load, or implementing a complex and slow simulation requiring callbacks. Our study has found that the proposed models consistently show more accuracy than the no-contention model that we had previously implemented in our set-up and that is widely used by the community.

The next logic step is to implement the most promising of those modules in the COTSon simulator we use in our

daily research. We have found that a reservation model which resembles network topology seems to be the most accurate model but it may be excessively demanding to provide very fast simulation. A good alternative is a reservation-based model which provides a number of 'pipes' which have to be reserved to submit packets. This also has the advantage of supporting distributed simulation seamlessly, which will simplify simulating the 1000-core chip we are designing.

It is worth noticing that in this paper we have considered a simple NoC design (mesh topology, cut-through switching and deterministic XY routing). However, modelling other NoC designs based on the proposed 'reservation' structure is straightforward. It only requires to organize the pool of resources following the topology of choice, to reserve them as directed by the routing and to chain them as dictated by the switching technology. A proof of concept for different flavours of NoCs remains as future work.

To close the paper, authors want to emphasise that these models are intended to accelerate the simulation of many-core processors but should not replace a proper evaluation of the communication infrastructure when it comes to design such systems. Our experiments showed that, under scenarios of saturation, none of the models were able to emulate network behaviour in an appropriate way.

ACKNOWLEDGEMENTS

Dr. Navaridas holds a Newton International Fellowship. This research is supported by the TERAFLUX project funded by the EU FP7 with grant agreement number ICT-FP7-249013. Dr. Luján holds a Royal Society University Research Fellow.

REFERENCES

[1] P. Abad et al. Topaz: An open-source interconnection network simulator for chip multiprocessors and supercomputers. In *Intl Symposium on Networks-on-Chip*, NOCS '12, 2012.

[2] N. Agarwal, L.-S. Peh, and N. Jha. Garnet: A detailed interconnection network model inside a full-system simulation framework. Technical Report CE-P08-001, Princeton University, 2008.

[3] E. Argollo et al. Cotson: infrastructure for full system simulation. *SIGOPS Oper. Syst. Rev.*, 43:52–61, 2009.

[4] T. Austin, E. Larson, and D. Ernst. SimpleScalar: An infrastructure for computer system modeling. *Computer*, 35:59–67, 2002.

[5] M. Bach et al. Analyzing parallel programs with pin. *Computer*, 43(3):34 –41, march 2010.

[6] D. Bailey et al. The NAS parallel benchmarks. *International Journal of High Performance Computing Applications*, 5(3):63–73, 1991.

[7] N. Barrow-Williams, C. Fensch, and S. Moore. A communication characterisation of Splash-2 and Parsec. In *IEEE Intl Symposium on Workload Characterization, IISWC 2009*, pages 86 –97, oct. 2009.

[8] R. Bedicheck. Simnow: Fast platform simulation purely in software. In *Hot Chips 16*, Aug. 2004.

[9] S. Bell et al. Tile64 - processor: A 64-core SoC with mesh interconnect. In *Solid-State Circuits Conference Digest of Technical Papers ISSCC 2008 IEEE International*, pages 88 –598, feb. 2008.

[10] C. Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[11] N. Binkert et al. The m5 simulator: Modeling networked systems. *Micro, IEEE*, 26(4):52 –60, july-aug. 2006.

[12] N. Binkert et al. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39:1–7, 2011.

[13] S. Borkar. Thousand core chips: a technology perspective. In *Design Automation Conference*, DAC '07, pages 746–749, 2007.

[14] C. Cao Minh et al. STAMP: Stanford transactional applications for multi-processing. In *IISWC '08: Proceedings of The IEEE Intl Symposium on Workload Characterization*, September 2008.

[15] P. Conway et al. Cache hierarchy and memory subsystem of the AMD Opteron processor. *Micro, IEEE*, 30(2):16 –29, 2010.

[16] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[17] N. Hardavellas et al. Simflex: a fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture. *SIGMETRICS Perform. Eval. Rev.*, 31:31–34, 2004.

[18] M. Herlihy and J. E. B. Moss. Transactional memory: architectural support for lock-free data structures. In *International Symposium on Computer Architecture*, ISCA '93, pages 289–300, 1993.

[19] M. Hsieh et al. SST: A scalable parallel framework for architecture-level performance, power, area and thermal simulation. *The Computer Journal*, 2011.

[20] J. H. Kelm et al. Rigel: an architecture and scalable programming interface for a 1000-core accelerator. In *International Symposium on Computer Architecture*, ISCA '09, pages 140–151, 2009.

[21] G. Kurian et al. ATAC: a 1000-core cache-coherent processor with on-chip optical network. In *Intl Conference on Parallel Architectures and Compilation Techniques*, PACT '10, pages 477–488, 2010.

[22] M. Lis et al. Scalable, accurate multicore simulation in the 1000-core era. In *Performance Analysis of Systems and Software (ISPASS), 2011 IEEE International Symposium on*, pages 175 –185, april 2011.

[23] P. S. MAGNUSSON. Simics : A full system simulation platform. *Computer*, 35(2):50–58, 2002.

[24] M. Martin et al. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *SIGARCH Comput. Archit. News*, 33, 2005.

[25] J. Miguel-Alonso, J. Navaridas, and F. J. Ridruejo. Interconnection network simulation using traces of MPI applications. *International Journal of Parallel Programming*, 37:153–174, 2009.

[26] J. E. Miller et al. Graphite: A distributed parallel simulator for multicores. In *Intl Symposium on High-Performance Computer Architecture*, 2010.

[27] J. Navaridas et al. Simulating and evaluating interconnection networks with INSEE. *Simulation Modelling Practice and Theory*, 19(1):494 – 515, 2011.

[28] V. Pai, P. Ranganathan, and S. Adve. RSIM: An execution-driven simulator for ilp-based shared-memory multiprocessors and uniprocessors, 1997.

[29] M. Palesi et al. Noxim simulator. http://noxim.sourceforge.net/.

[30] 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 *Intl Symposium on Networks-on-Chip*, NOCS '11, 2011.

[31] A. Portero, Z. Yu, and R. Giorgi. TERAFLUX: Exploiting tera-device computing challenges. *Procedia Computer Science*, 7, 2011.

[32] V. Puente, J. Gregorio, and R. Beivide. SICOSYS: An integrated framework for studying interconnection network performance in multiprocessor systems. In *Parallel, Distributed, and Network-Based Processing, Euromicro Conference on*, 2002.

[33] M. Rosenblum et al. Complete computer system simulation: the SimOS approach. *Parallel Distributed Technology: Systems Applications*, 3(4), 1995.

[34] S. Sawant et al. A 32nm Westmere-EX Xeon enterprise processor. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*, pages 74 –75, feb. 2011.

[35] L. Seiler et al. Larrabee: A many-core x86 architecture for visual computing. *Micro, IEEE*, 29(1):10 –21, 2009.

[36] J. Shin et al. A 40nm 16-core 128-thread SPARC SoC processor. In *Solid State Circuits Conference (A-SSCC), 2010 IEEE Asian*, nov. 2010.

[37] TERAFLUX Consortium. TERAFLUX: Exploiting dataflow parallelism in teradevice computing. http://www.teraflux.eu/, 2012.

[38] M. Tremblay and S. Chaudhry. A third-generation 65nm 16-core 32-thread plus 32-scout-thread CMT SPARC processor. In *Solid-State Circuits Conference, ISSCC 2008.*, feb. 2008.

[39] H.-S. Wang et al. Orion: a power-performance simulator for interconnection networks. In *International Symposium on Microarchitecture*, MICRO 35, 2002.