Rafiev A, Xia F, Iliasov A, Romanovsky A, Yakovlev A.

Selective Abstraction for Estimating Extra-Functional Properties in Networks-on-Chips using ArchOn Framework.

***In: 17th International Conference on Application of Concurrency to System Design (ACSD)*. 2017, Zaragoza, Spain: IEEE.**

DOI link to article:

https://doi.org/10.1109/ACSD.2017.13

Date deposited:

14/11/2017

Selective Abstraction for Estimating Extra-Functional Properties in Networks-on-Chips using ArchOn Framework

A. Rafiev, F. Xia, A. Iliasov, A. Romanovsky, A. Yakovlev – Newcastle University, UK
{ashur.rafiev, fei.xia, alexei.iliasov, alexander.romanovsky, alex.yakovlev}@ncl.ac.uk

***Abstract*—The analysis for extra-functional properties like power and performance takes a critical role in the system design workflow. Hardware-software co-simulation is one of the commonly used ways to perform this type of analysis. However, with the modern development of many-core systems the problem of scalability is becoming a bottleneck for all analysis techniques including simulation, especially when a simple extrapolation from the single core results is unacceptable. This paper presents a framework aimed at the extra-functional analysis during the rapid prototyping stages of system design. The tool is based on stochastic modelling and simulation of cross-layer system representations. The concept of selective abstraction is applied to ensure a sufficient level of accuracy where it is needed, while reducing the complexity of the parts that are of less importance. A set of Networks-on-Chip topologies has been analysed and presented as a use case example.**

I. Introduction

Simulation is an important method in the processes of system design and analysis. Many methods and frameworks concentrate on studying specific aspects of systems, such as Network-on-Chip (NoC) connectivity, on their own [8]. Other efforts have been made to encompass multiple issues and different parts of systems within the same design and analysis framework, e.g. hardware-software (HW-SW) co-design and co-simulation [3].

One of the most important issues simulation tools face is system complexity. The trade-off between accuracy and applicability has always been a crucial decision for designers to make at every stage of their work [10]. This is especially true for the initial stages of system design, when the global decisions on the overall system architecture must be made. This phase usually relies on rapid prototyping, which does not allow large investments in computation time.

Detail models (e.g. SPICE and up to RTL level) are accurate, but cannot be used for whole system studies when system size go beyond a small number of gates. And even models sitting at higher levels of abstraction, for instance ISA-level, cycle-accurate, etc., even though can usually be generated automatically from implementation specs, do not scale well to many cores. More abstract models (e.g. time-average or stochastic) make use of approximation and lose precision and accuracy, but can be small and fast and achieve better scaling. Designing a good abstraction, however, can be challenging.

This paper describes a flexible modelling framework, ArchOn, which may be used to target the study of a specific aspect of a system as well as used to study systems in their entirety. Based on the representation of resources and their dependencies, ArchOn is also flexible in the sense that it may be used at any level of abstraction. In addition, an ArchOn study may include parts of systems represented in detail (e.g. logic gates and software instructions) and other parts represented by highly abstract models (e.g. entire computers and application software).

ArchOn supports the study of extra-functional properties, such as power, performance, and reliability via deterministic or stochastic simulations. The framework includes front-end tools for quickly specifying new (non-existent) HW and SW, from ISA-agnostic models to ISA-specific.

These features are demonstrated using an NoC case study. A number of connectivity configurations have been simulated for performance and power and compared with each other, as well as a bus interconnect. The example systems are simulated with a convolution filter benchmark software model.

The paper is organised as follows. Section II overviews the related research and state-of-the-art tools. Section III describes the developed framework and underlying formalism. Section IV presents the case study with details on modelling and simulation results. Section V concludes the paper.

II. Related Work

Computation system simulation is a well-researched area with a large number of available methods and tools. For instance Transaction Level Modelling (TLM) is a well established paradigm which seeks to divide the overall problem into communication and computation. Helped by the widely-used language SystemC [4] allows models to be developed at different levels of abstraction, and simulations to be run at various levels to trade accuracy for cost and speed. In general, widely-used tools such as gem5 [1] tend to focus on a specific level of abstraction for each use case, for instance cycle-accuracy, instruction-set accuracy, etc. Mixing levels of abstraction in the same analysis is usually not directly supported, and require significant user effort.

One method of reducing the simulation burden is to make use of stochastic methods. This allows system behaviour to be viewed as probabilistic, avoiding having to compute deterministic behaviours in detail. Formalisms for modelling stochastic behaviour exist and many of them also have good tool support [11], but these also tend not to directly support the mixing of abstraction levels in the same analysis, requiring

the user to invest considerable effort in the modelling process outside the formalisms and tools.

If a targeted type of system is known, one can often find a plethora of tools for modelling functional and extra-functional properties. For instance, in the area of NoC, a large number of simulation tools have been developed, often based on SystemC TLM methods [13], [7], [6], [5], [8].

III. ArchOn Framework

In this work, we attempt to address the simulation problem with a general method that can analyse both software and hardware without targeting any specific architecture. The method will also provide direct support within itself for mixing different levels of abstraction in the same analysis, and support the general representation of components at all levels of detail, from logic gates up to entire computers on the hardware side and from instructions up to entire applications to executions of multiple parallel applications on the software side.

The framework is based on the concepts of resource-driven modelling [9] and selective abstraction [10], briefly described in this section.

A. Resource Graphs

The central subject of our method is the study of a computational platform comprising a number of diverse resources and the way resources may be handled in order to realise a computation. A resource is in this case an indivisible element required by the system in order to change its state, and it is defined by its function and availability in relation to this transition. With the word "resources" we make the point that we do not exclude computation, communication, or other facilities, e.g. energy and time.

In [9] we proposed to represent a system with a relation graph, consisting of a set of vertices and a set of edges. Each vertex represents a single resource and each edge represents a dependency between two resources. *Platform architecture* is formally defined as a labelled directed graph of all platform resources, and all possible (allowed) dependencies between them. During the lifetime of an architecture instance we can observe the switching of resources, dependencies, and labelling. A *configuration* is understood to be an instantaneous sub-graph, representing a current allocation of the resources and active dependencies between them. *Resource graph evolution* is a top level transition system that works on resource dependency graphs. This can be considered as an FSM where graphs represent states. Each resources is allowed to have an internal state, which changes according to its function and the state of adjacent resource nodes.

Example 1. Let's consider Euclid's algorithm for computing the greatest common divisor (GCD) of two numbers (a and b): *if* $(a > b)$*, then* $a := a - b$*; if* $(a < b)$*, then* $b := b - a$*; repeat this until* $(a = b)$*, which will be the result*. Its implementation in ArchOn is shown in Figure 1. In this case, the resources are concrete hardware units: registers `reg_a`, `reg_b`, and two ALUs: `cmp` and `sub`. Resource states represent data, and the resource dependencies represent data transfer between the nodes. A register is an identity function that copies its pre-set state. Comparator `cmp` compares two inputs x, y and stores the result in its state, encoded *eq*, *lt*, or *gt* for "equal to", "less than", and "greater than" respectively. Subtraction `sub` is a stateless combinational logic element: the result is propagated to the output (post-set) node. The system starts in the configuration G_0 and then propagates into either G_1 or G_2 depending on the state of `cmp`, then comes back to G_0. The cycle continues until the state of `cmp` becomes *eq*.

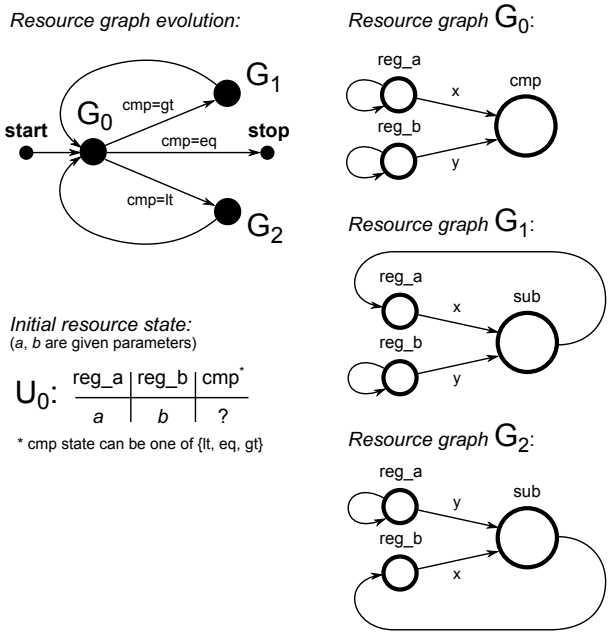

Figure 1. Simulating Euclid's algorithm for GCD(a, b). In this example, resources are hardware units with data dependencies between them. Allowed graph configurations G_0, G_1, and G_2 are shown on the right. Allowed transitions between these configurations are called *Resource graph evolution*.

This level of detail allows modelling for cycle-accurate simulations. The next section describes how levels of abstraction can be managed in ArchOn in order to deal with this complexity and achieve scalability.

B. Selective Abstraction and Incremental Modelling

Selective abstraction is the idea of allowing the concepts from different levels of abstraction to interact within the same model. It imposes extra challenges for the designer: 1) there is no straightforward way to connect these concepts, 2) the choice of the level of detail must be done individually per model element. These problems can be addressed systematically using Order Graphs [10], or the choice of abstraction can be driven by the workflow.

1) Order Graphs: An Order Graph describes the hierarchical system structure that is known during simulation time. Mixed abstraction models are obtained dynamically as so-called *cross-layer cuts* in this structure. The choice of the cut is driven by a metric, which attempts to minimise the model error via balancing the values of the estimated extra-functional property across the elements in the cut.

2) Incremental Modelling: An example of workflow-driven selective abstraction is the concept of working incrementally from the detailed models of small sub-systems to large compound models with selectively abstracted components. The

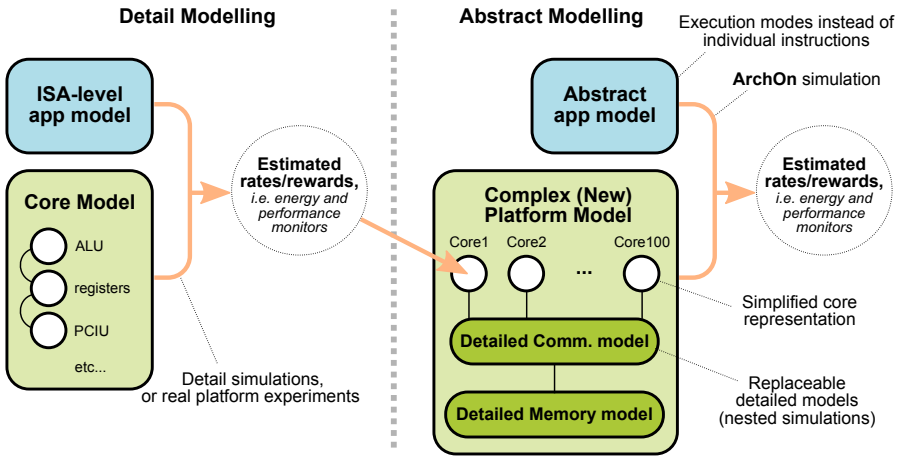

Figure 2. The concept of incremental abstraction in a modelling workflow.

Table I
SOME COMMANDS OF GRAPH ASSEMBLY LANGUAGE

command	description
a $\rightarrow$ b	set a dependency between resources a and b
a $\xrightarrow{x}$ b	set a labelled dependency between resources
a $\nrightarrow$ b	unset a dependency
$G = \emptyset$	clear all dependencies
go!	"execute": fire all resource state transitions
go to X	continue assembly from label X (jump)
if cond **go to** X	conditional jump

results of detailed simulations are used to build higher level models. This workflow is illustrated in Figure 2 in the context of HW-SW co-simulation.

Abstraction allows simulating hundreds of cores, however an abstract application model cannot execute individual instructions. Instead, it models different modes of operation: initialisation, processing, output, etc. Coupled with the simplified core models, it provides time-averaged resource access count (e.g., ALU instructions, memory accesses, etc.). Transitions between the modes can be deterministic or stochastic with defined probabilities. Another way to obtain an abstract application model is to use a stochastic instruction pool, described in Section IV-A.

Important parts of the system, like communication and memory, can be modelled in greater detail for better accuracy, or remain abstract for quick prototyping. ArchOn provides modularity, hence all component models can be developed independently and replaced when needed.

C. Tool description

The ArchOn framework is a set of tools and libraries written in Java. The core of the tool implements the infrastructure for resource graph representation and the simulation of resource graph evolutions.

Resource behaviours are implemented using Java classes. At the moment the ArchOn library already includes ISA-level resources for ARMv7-M and MC8051 instruction sets, as well as an abstract app-core scheduling model and the transaction-level NoC interconnect presented in this paper. The input for the simulator is given in a low-level domain-specific language

Algorithm 1 GCD(a, b) in graph assembly language

```
G0:
   reg_a → reg_a;   reg_b → reg_b
   reg_a -x-> cmp;   reg_b -y-> cmp
   go!   G = ∅
   if U [cmp] = "eq" stop
   if U [cmp] = "gt" go to G1
   if U [cmp] = "lt" go to G2
G1:
   sub → reg_a;   reg_b → reg_b
   reg_a -x-> sub;   reg_b -y-> sub
   go!   G = ∅
   go to G0
G2:
   reg_a → reg_a;   sub → reg_b
   reg_a -y-> sub;   reg_b -x-> sub
   go!   G = ∅
   go to G0
```

called *Graph Assembly Language* (GAL). Table I shows some commands for step-by-step graph configurations as well as explicit invocations of resource state transitions. This way the specification is more compact than the traditionally used adjacency matrices when applied to sparsely connected graphs with many vertices. GCD example from Figure 1 written in GAL is shown in Algorithm 1.

Editing large models directly in GAL is not very practical. Previously, for ISA-level simulations, we used automated conversion from ARM or MC8051 assembly. In this work, we designed a GUI front-end, which can be used for convenient editing of NoC meshes by "painting" the node types over a 2D mesh of an arbitrary size.

IV. USE CASE EXAMPLE

In order to demonstrate the scalability of HW-SW co-simulation in ArchOn, we selected NoC as the case study. In this use case we limit NoC to 4-way connected 2D meshes. And to show tool's flexibility and platform-independence, we also added an example system with blocking communication (bus interconnect).

In our earlier work on cycle-accurate simulations [9], we used convolution filter as an application benchmark. The results of that work has been used to characterise our new

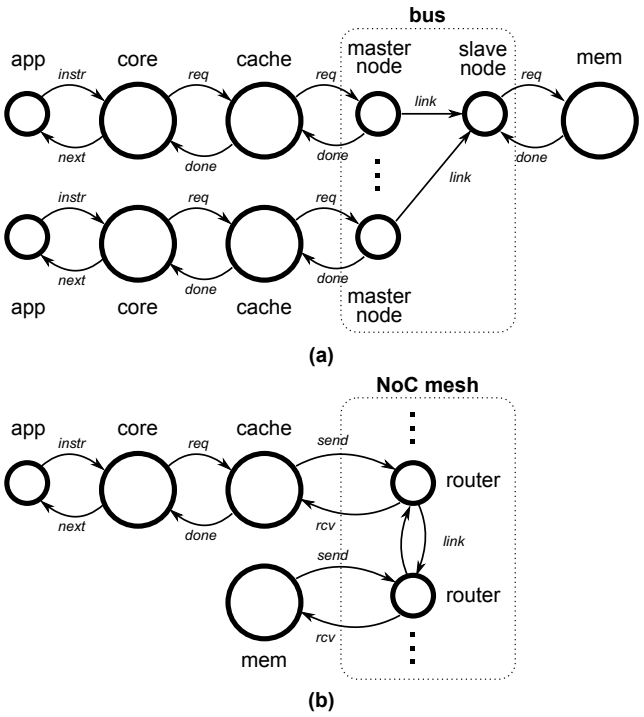

Figure 3. Mixed-abstraction platform model of a many-core system with (a) bus connectivity and (b) NoC.

Figure 4. Example NoC configurations.

high-level application models to support the incremental workflow, proposed in Section III-B. The benchmark is a typical application, not tailored for NoC: it needs access to a shared memory and does not employ peer-to-peer communication between cores. Private memory of the cores is functioning as an L1 cache.

We don't have an actual NoC platform to experiment with, hence the analysis of our simulation results is limited to scaling and precision, and does not evaluate the accuracy. This is a subject for future work.

A. High Level Application Model

The convolution filter application has a relatively uniform workload distribution: it performs similar processing for each image pixel. Modelling this application using modes has no real benefit, hence we choose a different approach.

In this model, application behaviour is defined as an unordered set of instructions. A random instruction is issued from this pool and passed to the core. Once the core completes the instruction, it sends a request for another. This approach allows time-averaged estimation of the extra-functional properties without actual data calculation. Another benefit of using this method is that the total number of instructions can be tuned to balance the precision versus simulation time. We classify all instructions into three types: computation (CPU), memory read and memory write. For the convolution filter benchmark, from the previous studies we know that the number of instructions are distributed as follows: $4.72 \cdot 10^6$ (79.12%) computation, $1.18 \cdot 10^6$ (19.78%) memory reads, and $0.07 \cdot 10^6$ (1.10%) memory writes.

B. Connectivity Model

Figure 3 shows the chain of dependencies across all resources, starting from the application. The resources are connected via *request*/*done* dependencies, which help track resource usage and timing. The direction of a dependency does not reflect the direction of the data transfer. For instance, the *link* arc in Figure 3(a) shows that the interaction is always initiated by a master node, however the data is exchanged in both directions.

The NoC platform model uses a similar dependency stack having the bus part replaced with router resources. Each router links with up to 4 adjacent routers and has *send*/*receive* dependencies with its core or memory node. In our study, we implemented X-first-then-Y routing strategy, however ArchOn framework is not generally restricted to that and allows arbitrary connectivity functions. Since it is impractical to edit NoCs manually using GAL, we developed an auxiliary graphical editor. In addition to core and memory nodes, it is possible to place router-only nodes (depicted as a dot) and breaks in the mesh (depicted using a cross mark).

Example topologies used for simulations are shown in Figure 4. Each topology can be scaled to an arbitrary $N \times N$ number of cores. Example (a) has only one memory node in the north-west corner of the mesh; example (b) consists of 4 memory nodes, one in each corner, regardless of the number of cores; example (c) has a single row of N memory nodes; example (d) has $4N$ memory nodes, one row on each side.

C. Model Characterisation

Table II shows the model characterisation parameters. Cortex-A7 power and performance models has been used for the core nodes. Computation latency, `cpuDelay`, has been calculated from 1000MHz CPU frequency and 0.787 average CPI experimentally measured from ARM performance counters using Odroid XU3 board [2]. Active power has been measured at 70mW, which for the same frequency gives the corresponding energy per instruction. Cache and memory characteristics have been taken from ARM white

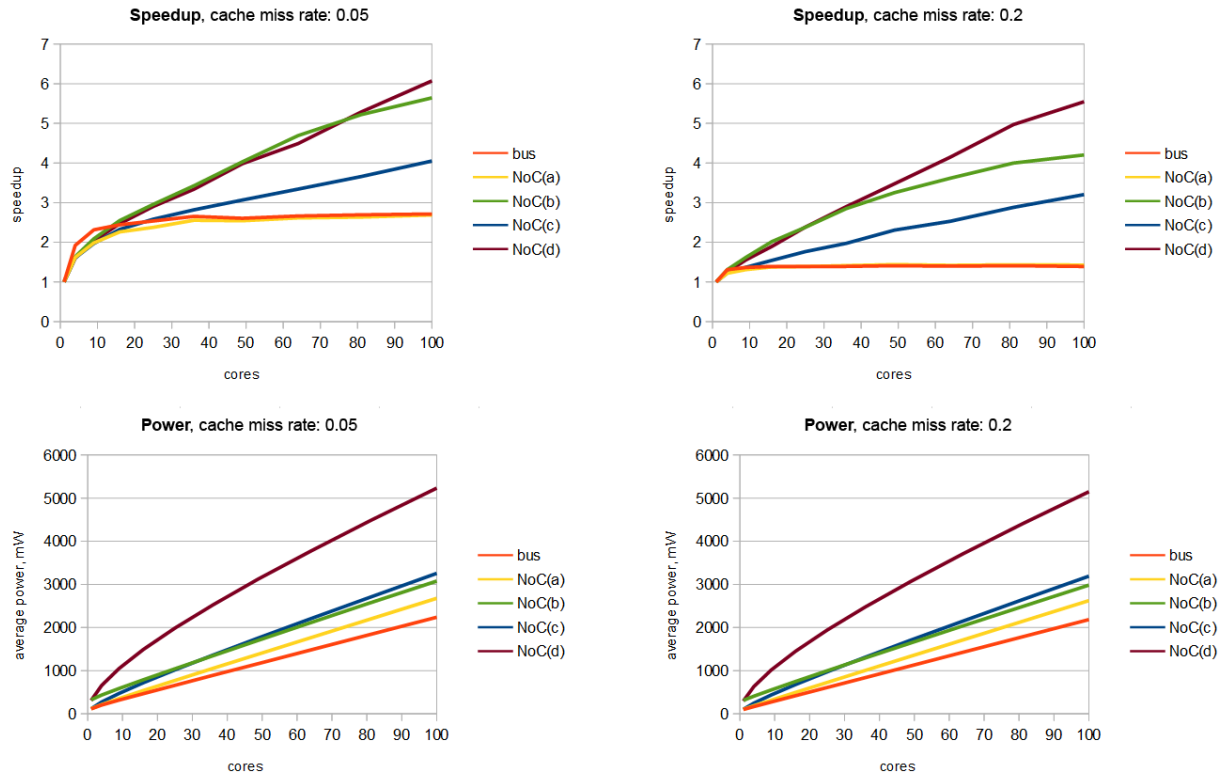

Figure 5. Power and performance results for different interconnect configurations.

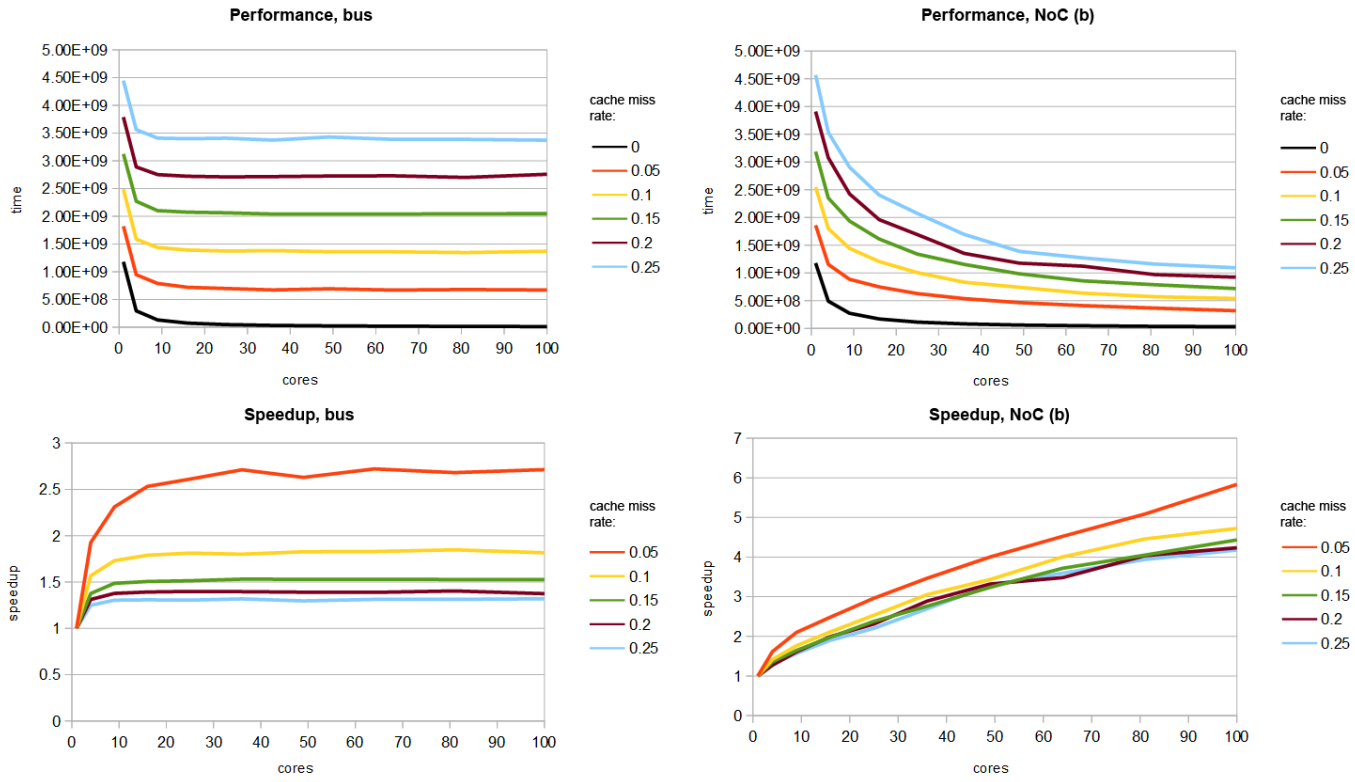

Figure 6. Performance results for different cache miss rates for bus interconnect (on the left) and NoC(b) interconnect (on the right).

Table II
RESOURCE CHARACTERISATION PARAMETERS

resource	parameter	value	unit
core (Cortex-A7)	`delayCpu`	1270	ps
	`energyCpu`	88.889	pJ
	`static`	19	mW
cache (L1 in Cortex-A7)	`delayHit`	4000	ps
	`energyHit`	35.000	pJ
	`static`	2	mW
memory (LPDDR3e)	`delayRead`	100000	ps
	`delayWrite`	100000	ps
	`energyRead`	39.750	pJ
	`energyWrite`	99.000	pJ
	`static`	60	mW
comm	`delayHop`	1333	ps
	`energyHop`	20.000	pJ
	`static`	4	mW

papers [12], but haven't been experimentally confirmed yet. Communication delay and energy are educated guesses, which is still acceptable for a relative comparison within our use case examples. For the purpose of cross-platform comparison, we use the same delay and energy values for a single bus master/slave resource node as well as for a single router resource node (named *comm* in the table).

D. Simulation results

Platforms shown in Figure 4 have been simulated for different numbers of cores and a range of cache miss rates. FIFO size of 64 was empirically found to be acceptable if the number of cores stays below 256. Beyond this, it often caused network stalls. Performance results in terms of the total execution time in ps, the speedup and the estimated power are shown in Figures 5 and 6. NoC(a) and the bus show similar performance characteristics due to the bottleneck in the single memory node, however NoC power consumption is higher due to area overheads. NoC(d) mesh is estimated to have the highest performance and power consumption. NoC(b) shows surprisingly good performance scalability, considering it has a constant number of memory nodes, which also results in a relatively low power consumption.

Table III shows the simulation time and precision for NoC(b) example with a 0.2 cache miss rate, averaged over 200 simulation runs on a 2.9GHz Intel i7-3520M. Precision is represented with square root of variance shown as a percentage of the mean value. Because the total workload is evenly distributed between the cores, the number of cores has a negligible impact on the simulation time, however larger systems suffer reduced precision. This still proves the scalability of the method, as even large systems can be simulated with acceptable precision in a matter of seconds. Small power variation can be explained with the fact that the cores are predominantly idle waiting for memory response and the memory is constantly busy. For example, for 100 core NoC(b), the total delay for the memory access is 287.8ns on average, which is 226.6 times slower than a single computation instruction.

Table III
SIMULATION TIME AND PRECISION

total workload	16 cores NoC(b)			256 cores NoC(b)		
	sim. time	perf. var.	pwr. var.	sim. time	perf. var.	pwr. var.
$64 \cdot 10^2$	0.010s	5.84%	0.20%	0.012s	6.65%	0.16%
$64 \cdot 10^3$	0.066s	2.81%	0.11%	0.068s	3.46%	0.09%
$64 \cdot 10^4$	0.583s	2.53%	0.09%	0.588s	3.34%	0.09%
$64 \cdot 10^5$	5.775s	2.54%	0.10%	5.758s	3.88%	0.10%

V. CONCLUSIONS

The ArchOn framework has been developed to support the holistic and specific study of systems through simulations. It allows the user to directly trade off accuracy for cost of analysis through selective abstraction. System analysis with mixed levels of abstraction in the same study is directly provided and the method is architecture-agnostic and abstraction level-agnostic with a high flexibility in modelling. Based on the concept of resources and their dependencies, ArchOn is especially useful for studying extra-functional properties such as performance and power, and yet it does not preclude functional modelling and analysis. The features of ArchOn has been demonstrated with the comparative study of multiple NoC topologies and a bus structure as a case study. The immediate future work includes the further development of the NoC models and comparing them with real system experiments and/or existing NoC simulators.

Acknowledgement This work is supported by EPSRC as a part of PRiME project EP/K034448/1.

REFERENCES

[1] The gem5 simulator system. http://www.m5sim.org.

[2] A. Aalsaud et al. Power–aware performance adaptation of concurrent applications in heterogeneous many-core systems. In *Proceedings to ISLPED*, pages 368–373. ACM, 2016.

[3] J. R. Andrews. Chapter 4 - hardware/software co-verification. In J. R. Andrews, editor, *Co-verification of Hardware and Software for {ARM} SoC Design*, pages 119 – 163. Newnes, Burlington, 2005.

[4] L. Cai and D. Gajski. Transaction level modeling: an overview. In *First IEEE/ACM/IFIP International Conference on Hardware/ Software Codesign and Systems Synthesis (IEEE Cat. No.03TH8721)*, pages 19–24, Oct 2003.

[5] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti. Noxim: An open, extensible and cycle-accurate network on chip simulator. In *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 162–163, July 2015.

[6] L. S. Indrusiak, J. Harbin, and O. M. Dos Santos. Fast simulation of networks-on-chip with priority-preemptive arbitration. *ACM Trans. Des. Autom. Electron. Syst.*, 20(4):56:1–56:22, Sept. 2015.

[7] N. Jiang, D. U. Becker, G. Michelogiannakis, J. D. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally. A detailed and flexible cycle-accurate network-on-chip simulator. In *ISPASS*, pages 86–96. IEEE Computer Society, 2013.

[8] A. B. Kahng, B. Lin, and S. Nath. ORION3.0: a comprehensive NoC router estimation tool. *IEEE Embedded Systems Letters*, 7(2):41–45, June 2015.

[9] A. Rafiev et al. Studying the interplay of concurrency, performance, energy and reliability with ArchOn – an architecture-open resource-driven cross-layer modelling framework. In *Proc. to ACSD*, 2014.

[10] A. Rafiev et al. Selective abstraction and stochastic methods for scalable power modelling of heterogeneous systems. In *Proc. to FDL*, Sept. 2016.

[11] W. Sanders and J. Meyer. *Lectures on Formal Methods and Performance Analysis*, volume LNCS2090, chapter Introduction to Generalized Stochastic Petri Nets, pages 315–343. Springer, 2001.

[12] A. Stevens. Introduction to AMBA 4 ACE and big.LITTLE processing technology. Technical report, ARM, 2011-2013.

[13] C. Sun, C. H. O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L. S. Peh, and V. Stojanovic. DSENT - a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, pages 201–210, May 2012.