Rafiev A, Xia F, Iliasov A, Gensh R, Aalsaud A, Romanovsky A, Yakovlev A.
Selective abstraction and stochastic methods for scalable power modelling of heterogeneous systems.
***In: Forum on Specification and Design Languages (FDL)*. 2017, Bremen, Germany: IEEE Computer Society.**

Selective Abstraction and Stochastic Methods for Scalable Power Modelling of Heterogeneous Systems

A. Rafiev, F. Xia, A. Iliasov, R. Gensh, A. Aalsaud, A. Romanovsky, A. Yakovlev – Newcastle University, UK
{ashur.rafiev, fei.xia, alexei.iliasov, r.gensh, a.m.m.aalsaud, alexander.romanovsky, alex.yakovlev}@ncl.ac.uk

***Abstract*—With the increase of system complexity in both platforms and applications, power modelling of heterogeneous systems is facing grand challenges from the model scalability issue. To address these challenges, this paper studies two systematic methods: selective abstraction and stochastic techniques. The concept of selective abstraction via black-boxing is realised using hierarchical modelling and cross-layer cuts, respecting the concepts of boxability and error contamination. The stochastic aspect is formally underpinned by Stochastic Activity Networks (SANs). The proposed method is validated with experimental results from Odroid XU3 heterogeneous 8-core platform and is demonstrated to maintain high accuracy while improving scalability.**

I. Introduction

Continued scaling of semiconductor technology has caused an increase of computing system capabilities, and with it, a seemingly unstoppable expansion of system application space. This is leading to a rapid increase of system complexity and diversity, exacerbating the scalability of system modelling. A typical example of such complex and diverse systems is multi-core heterogeneous platforms.

Addressing the model complexity is highly challenging due to the trade-off between *quality* (i.e. the accuracy, precision, and fidelity) of the model and its *usability* (defined by scalability, computation complexity, and design effort), as shown in Figure 1.

Modelling *non-functional properties*, e.g. power dissipation, is as crucial as modelling *functional properties*, such as operational correctness [6]. Non-functional models typically include functional representations. A widely-used method of modelling power uses Virtual Prototypes (VP) to generate states from a functional simulation for use in power simulators forming a co-simulation [17], [12].

Analogue hardware models such as SPICE models provide some of the highest representational quality, but they are not usable for studying entire computers with software running on hardware. For such studies, discrete event models, such as Instruction Set Architecture- (ISA-)accurate [7], cycle-accurate and RTL models [5], are useful when studying functional properties. Instruction Set Simulators (ISS), however, commonly have simulation speeds of the order of a few Million Instructions Simulated per Second (MISPS) [1], [17], and this puts a limit on their usability when the system scales to many-cores, especially for statistical analysis [8].

There have been, as a result, significant efforts in model simplification for power studies. One way of simplifying away from ISA- or cycle-accuracy targeting multiple cores is extrapolation [9]. In this approach, a typical subsystem (e.g. a single core) is fully characterized and represented with a complete model, and other similar cores are represented by simplified models obtained through extrapolation. However, this method tends to be less effective for highly heterogeneous systems. It is also possible to depart from ISA-accuracy through abstraction. For instance, Transaction Level Modelling (TLM) concentrates on the functional properties of larger system blocks [4], [13].

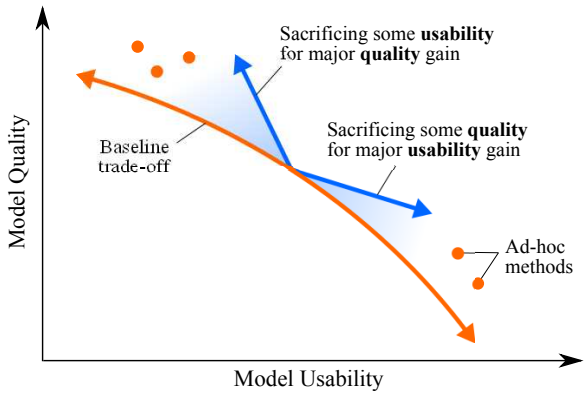

Figure 1. The realm of complexity control.

Functional modelling can be highly abstracted by using stochastic techniques, shrinking models by regarding system behaviours as stochastic, rather than deterministic [8].

Non-functional parameters like power also open up further ways to systematic model simplifications based on what may be called the "significance factor" [14]. During any particular operation of a heterogeneous system consisting of multiple parts, some parts of the system may consume more power than other parts. If a quantitative power model is to be precise to a certain degree, it makes sense to make the model power-proportional by using a simpler model for a less power hungry part and a detailed model for a more power hungry part. This approach can be broadly described as *selective abstraction*, i.e. the level of abstraction (and therefore the cost and quality) of each part of the model depends on the part's contribution to the parameter under study.

This work aims towards the scalable modelling of multi-core heterogeneous systems by concentrating on stochastic modelling and selective abstraction. By doing so, we seek to support designers to systematically traverse the trade-off space between modelling quality and model scalability in "good" trajectories, shown in Figure 1 as vectors, as opposed to *ad-hoc* techniques targeting specific points in this trade-off.

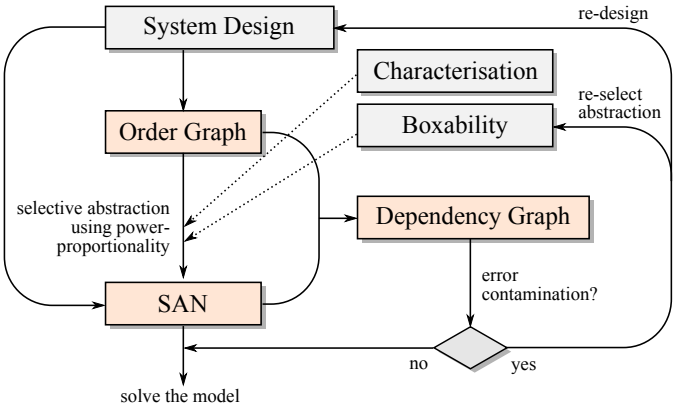

Figure 2. The flowchart of the proposed method.

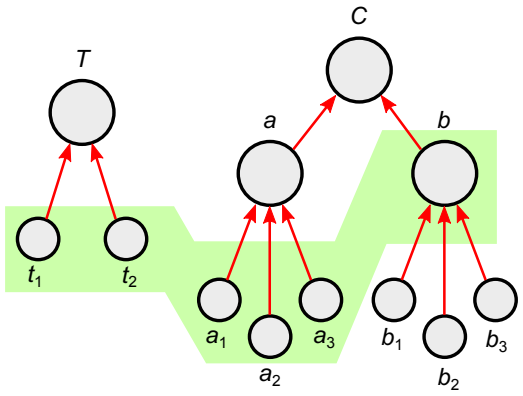

Figure 3. Example vertical view of an Order Graph with a cross-layer cut.

A. Research Methodology and Contributions

In order to validate the presented ideas, we created a model of a real hardware multi-core heterogeneous system using a mature stochastic modelling method, applied the proposed method of selective abstraction to optimise the model for scalability, and evaluated the cost and the accuracy against the actual measurements. Stochastic Activity Networks (SANs) [16] is a well-known modelling method with an extensive support by the Möbius tool [2] provides the capabilities to model power as a reward function, thus has been used to facilitate the stochastic modelling aspect of the method. Odroid XU3 development board [3] based on the ARM big.LITTLE architecture has been selected as the target system for modelling.

The following contributions have been made:

- We developed new structuring methods to tackle complexity and scalability in modelling by providing a power-proportionality metric for selective abstraction and methods to retain accuracy by avoiding error contamination.
- We validated these methods using power modelling in SANs and showed their effectiveness in improving the trade-offs between accuracy, scalability, and usability.

The paper is organised as follows. Section II described the workflow and the method. Section III presents the design of supporting experimental studies. Section IV describes the developed model and discusses the results from the simulation results. Section V concludes the work.

II. The Proposed Method

The workflow of the proposed method is illustrated in Figure 2. A hierarchical representation of the system resources in the form of an Order Graph (OG) [14] is derived from the system design knowledge. The behaviour of the system is captured in a detailed SAN model [16]. System power characteristics, usually obtained from initial experiments, are applied to the OG to compute the power-proportionality metric for selective abstraction, which determines the regions for black-boxing in the SAN model. The next step is to combine OG and SAN to capture dependencies within the model in the form of a graph, which will help identify any error contamination. If no contamination found, the SAN model can be used further in power studies, e.g. simulations. In case of error contamination, the designer has to redo the abstraction selection with the updated knowledge on the model's boxability, or even re-design the system.

A. Hierarchical Modelling and Selective Abstraction

Hierarchical representations have been used for modelling complex systems for a long time. The idea of separating the "vertical" relation between the layers of abstraction from the "horizontal" knowledge of the system at each particular layer of abstraction has been hinted in [20] and then formally defined in Zoom structures [10] as the concepts of *verticality* and *horizontality*. Zoom structures are based on partial orders and are very permissive. In contrast, OGs put a number of constraints on the modelling, which guarantee consistency between the abstraction layers.

An OG is a graph with nodes representing various system resources arranged in tree hierarchies; different concepts can be represented by separate trees. For example, in Figure 3, T can represent the hierarchy of tasks, and C – computational units. The hierarchies can be built from the knowledge of the system structure and by similarities of its constituents. The distance from the root relates to the level of abstraction. The formal definition and properties can be found in [14].

OG contains the static knowledge of the system and needs to be paired with a dynamic model to capture the system behaviour (in our case: SANs). Each branch of a tree must have a representative element in the dynamic model, but multiple nodes from a single branch cannot be included. The nodes in OG that are included in this model relation form a *cut*. If the cut goes through different depths in the hierarchy (layers of abstraction), it is called a *cross-layer cut*. The cut containing all leaves relates to the most concrete (detail) model of the system.

Moving up in the abstraction hierarchy, thus grouping multiple nodes into one, represents grouping the corresponding elements in SANs into a single entity by averaging/totalling their parameters (known as *black-boxing*). This reduces the size of a model, but also introduces inaccuracy.

Ideally, the goal of selective abstraction is to obtain a cut that provides the minimal model while its added error

satisfies the given threshold ε: $|\Delta E| < \varepsilon$. Our proposed power-proportional metric of selecting the cut is:

$$|\Delta E| = |\Delta e_x q_x|\,, \tag{1}$$

where Δe_x is the local change of the percentage error, as a result of the black-boxing, in the part being black-boxed, and $q_x = \frac{p_x}{p}$ is the proportion of power consumed by this part in relation to the total power consumption. The values of p_x and p can be found from the model characterisation experiments, but Δe_x is typically not known before solving the model. Thus, instead of using the precise metric, one may rely on heuristic approximations. A number of methods are suggested in [15]. In this paper, we use constant Δe_x under the assumption of only black-boxing similar component sub-models. Cross-layer cuts for deeper hierarchies can be found iteratively in polynomial time.

Equation (1) assumes that black-boxing one part of the system does not effect the accuracy of the other parts in the model. However, this is not the case if the behaviour of a detail part is dependent on the behaviour of an abstract part. It is important to remember that Δe_x is a percentage error, so the total deviation from the real value is amplified when the error leaks from a part with smaller q_x to a part with larger q_x. This concept of *error contamination* has been discovered in our work with selective abstraction.

The proposed method of detecting and localising contaminating errors is done by deriving a dependency graph from the dynamic model in relation to OG. The errors from the black-boxed parts propagate along the paths in this graph: the error of a node is maximum of its own error and errors of its preset nodes. The structure of the dependency graph puts restrictions on what resources can be used in selective abstraction (i.e. are *boxable*). Section IV gives concrete examples of the models and obtained dependency graphs with and without error contamination.

It is also important to note that the method benefits from heterogeneity in the system: a bigger difference in the power consumption of the system parts provides better error tolerance in a cross-layer cut.

B. Background on Stochastic Modelling

SANs is an extension to General Stochastic Petri Nets (GSPNs) which is based on Petri Nets (PNs) [16]. It inherits the general attributes of PNs including a distributed representation of system states, making it easy to represent parts of a system directly as local subsystems, and more straightforward representations of such important issues as concurrency and synchronisation. A well-established method, it is supported by the mature software tool-kit: Möbius [2].

SANs are capable of representing both deterministic and stochastic events, and event durations in time. The elements used in this work include a) transitions whose firing speeds (rates) are specified as stochastic, following given distributions, b) transitions with multiple firing cases with specific probabilities for each case, and c) input and output gates with predicates and implications specified through logic functions.

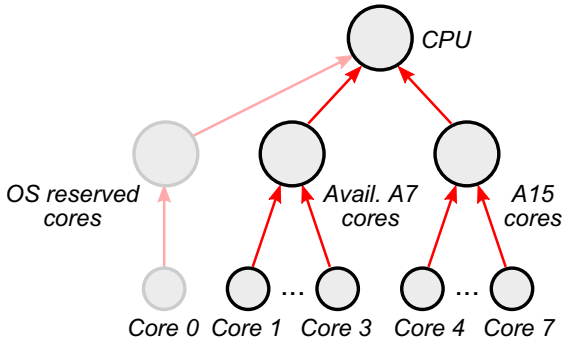

Figure 4. Hierarchical structure of Odroid XU3 CPU cores. In the experiments, Core 0 is reserved for OS and tools.

The Möbius tool, used in this paper, incorporates a set of solvers including both Monte-Carlo simulation and state-space related solvers. Numerical Markovian solutions can be done for steady-state or time averaged interval rewards, but limited to models with exponentially distributed firing rates. The tool's concept of "rewards" can be easily extended to physical parameters, such as power. In our examples, we compute time-interval average power and use average error as an accuracy metric. The method is not limited to this and can give probabilistic estimation for transient power values.

III. Case Study

In this work, we aim to evaluate the impact of selective abstraction on the total error in the model. In order to do this, we want to build three models of the same system: a detailed model without any black-boxing, a cross-layer model with selective black-boxing, and an abstract model with maximum black-boxing. The result of analysing the power consumption using each of these models has to be compared with actual measurements from the platform.

A. Platform Description

One of the best off-the-shelf examples of a heterogeneous system for power analysis is the Odroid XU3 board [3]. The main component of Odroid XU3 is the 28nm 8-core Application Processor Exynos 5422. This System-on-Chip is based on the ARM big.LITTLE architecture [11] and consists of a high performance Cortex-A15 quad-core processor block, a low power Cortex-A7 quad-core block, a Mali-T628 GPU and 2GB LPDDR3 DRAM. The board contains four real-time current sensors allowing the measurement of power consumption on the four separate power domains: A7, A15, GPU and DRAM.

For each domain, the supply voltage and clock frequency can be tuned through a number of pre-set pairs of values. The performance-oriented A15 quad core block can scale its frequencies from 200MHz to 2000MHz, whilst the low-power A7 block has a frequency range from 200MHz to 1400MHZ. Core 0 in the A7 domain has an additional speciality of running the OS kernel and drivers, and it cannot be switched off. We avoid using this core for stress tests and benchmarks to reduce the impact from the OS on the measurements. The CPU structure is represented as an OG hierarchy in Figure 4.

B. Power modelling

The average system power consumption can be found analytically as the function of the system workload and the system's power characteristics [19]. This work uses a simplified workload-based power model, which has been shown to provide sufficient accuracy [18]. The method of selective abstraction can be applied to advanced power models as well.

In our model, the power is a function of the type of executed task T, core type C, frequency F, voltage V, and the number of cores (of this type) running n. In the experiments, the frequencies and voltages of the cores remain constant per power domain, hence the values of F and V are tied to C and do not need to be considered separately. The system workload is modelled on a per-task basis as the ratio of the task's CPU time $t(T)$ to the total duration of the experiment t_{exp}.

Additionally, the cores are never put to sleep. Because of this, there is a constant background power P_{idle} consumed regardless of the workload, called *idle power*, which depends only on the core type C (considering constant F and V). The power spent to do actual computation P_{act} is called *active power*. The total power of a power domain C is found as a time-averaged active power added to the constant idle power:

$$P_{total}(C) = P_{idle}(C) + \sum_{T} \frac{t(T)}{t_{exp}} P_{act}(n, T, C). \quad (2)$$

The values for P_{act} and P_{idle} can be characterised offline in a form of a table function. However, the exact value of $t(T)$ is known only during run-time. In our work, we use stochastic modelling to predict this value. The parameters for workload prediction models include the spawn rates for the tasks and the average CPU time required to complete a task (completion rates). It is reasonable to assume these are known to the model designer.

The presented simplified power model does not take into account system temperature. In fact, it is suspected to be the main source of error in our experimental results, as the models were characterised on fully-loaded cores, much hotter than during the actual experiments. This baseline error contributes to all layers of abstraction, including the detail model. The focus of the presented research is to investigate the additional error due to black-boxing.

C. Experiment Setup

Figure 5 shows the model evaluation framework. The fixed rates are used to generate random execution traces – list of task spawning events, affinities, and completion events. These traces are executed on the platform to produce power traces – sets of timestamped power measurements. At the same time, the rates and platform characteristics are used to parametrise the SAN models, which are analysed in the Möbius tool. The models do not know the actual execution traces. The mean power from the model analysis is then compared to the mean power obtained from the power traces to estimate the modelling error. The computational cost (time) of analysing the models is also evaluated.

Execution traces reflect the following scenarios of the system operation. During the experiment, two types of tasks

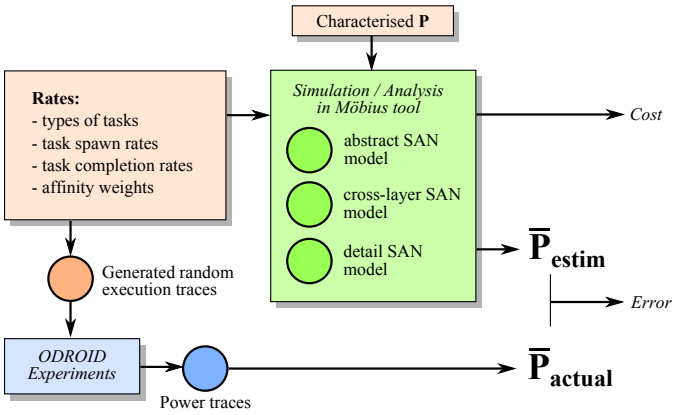

Figure 5. Model evaluation framework.

(T0 and T1) spawn continuously. The spawn intervals, in ms, are exponentially distributed with rates respectively $\lambda_{spawn,T0} = 0.003$ and $\lambda_{spawn,T1} = 0.002$. Both tasks are CPU-heavy: T0 performs a floating point square root computation in a loop; T1 performs integer multiplication and addition. The tasks are then scheduled on the available cores. In order to increase the heterogeneity of the system, we assigned different probabilities to scheduling onto different cores (affinity weights) providing three possible scenarios: Scenario EQ represents equal scheduling chances for all cores regardless on their type, Scenario CA differentiates between cores, while Scenario TCA has weights dependant on the type of a task as well as the core.

Each tasks is executed on a core until it is finished and then removed. Completion times are random and exponentially distributed[1] with constant rates (the frequencies of the cores are kept unchanged during the experiments). In real-life situations, low-power A7 cores typically operate on lower frequencies than A15, so their performance is reduced. To mimic this behaviour in the experiments, A15 is set to work approximately twice as fast as A7, and the completion rates for A7 domain in the model are halved. The rate values are shown in Table I as "target completion rates". During the model characterisation, these values are re-adjusted for more precision, as discussed in Section III-D.

D. Characterisation Experiments

Platform characteristics required to parametrise the models include per-core power consumption for each core type when idle and when running each type of a task. We set A7 cores to run at 1000MHz, and A15 cores are set to run at 1800MHz. The core frequencies are set below the highest mark to avoid throttling. Since the power can be measured only per domain, which consists of 4 cores, it requires additional experiments and calculations to be performed.

Idle power is measured directly for each power domain. Core0 is reserved for running the OS, the scheduler, and the power trace logger, and is not used directly for running

[1]We had to limit our examples to exponential distribution in order to test Markovian solvers. In simulation-only studies, any other random distribution can be programmed in.

Table I
POWER AND TIME CHARACTERISTICS

domain	A7		A15	
number of cores N	3		4	
idle power, W	0.0737		0.6211	
task	T0	T1	T0	T1
1 core active power, W	0.1392	0.1427	1.4684	1.4281
N cores active power, W	0.2703	0.2771	4.0101	3.9066
exec time, ms	8745	9730	9827	8184
target completion rate	0.001	0.0015	0.002	0.003
adjusted comp. rate	0.00114	0.00154	0.00204	0.00367

the experiments. Its impact on power consumption is viewed as a background noise included in the idle power of the A7 domain. To measure the active power we ran each task separately on each domain, i.e. providing characterisation data for each $\langle T, C\rangle$ pair. Also, since we didn't know if the power consumption scales linearly by adding more cores, we ran each set on $1 \leq n \leq N$ active cores. The active power of a single core is then related to the measured power as $P_{act}(1) = (P_{meas} - P_{idle})/n + P_{idle}$. Similarly, the power of running all cores in the domain is $P_{act}(N) = (P_{meas} - P_{idle}) \cdot N/n + P_{idle}$. All instances of measured or computed values are within 3% range from the respective mean values across all experiments with the exception of a single A7 core executions, which deviate by 5%. This is within the acceptable error range, so we can still assume linear power scaling: $P_{act}(n) = n \cdot P_{act}(1)$. The final values used in the model are shown in Table I.

Since rates and characteristics are the only things connecting the models with the actual experiment, we take extra care when generating traces and executing them on the platform. A specialised scheduler has been designed to address this issue.

Each entry of the execution trace contains the timestamp of spawning a task, the task type, its affinity and input data (a single integer value T), which affects the task completion time. To guarantee that the execution times follow the exponential distributions with the given rates and to simplify the trace generation, T is equal to the requested execution time in ms. The tasks T0 and T1 henceforth are required to match their execution time to T as close as possible. It is possible to achieve by reading the system timer, but calling kernel functions may cause unwanted interference. Instead, we achieve this by doing a task in a loop and calibrate the number of iterations to complete in the given time. The task calibration function for some core i and task j is $f(i, j, T) = x_i \cdot y_j \cdot T$; the constants x_i and y_j are found experimentally: $x_{A7} = 21, x_{A15} = 40$ (confirming that A15 running at 1800MHz is roughly twice as fast as A7 at 1000MHz), $y_{T0} = 95, y_{T1} = 1700$. Thus, for example, in order to run T1 on A7 for 100ms, we need to do 3,570,000 loop iterations. However, this calibration is not perfect and requires further adjustment.

During the characterisation experiment, we request the tasks to run for 10s by specifying $T = 10,000$ and then measure the actual completion time. Considering that the target completion rates are used for generating the traces, but the actual times are skewed, as shown in Table I, we apply a simple proportion to calculate the adjusted completion rates to be used in the model.

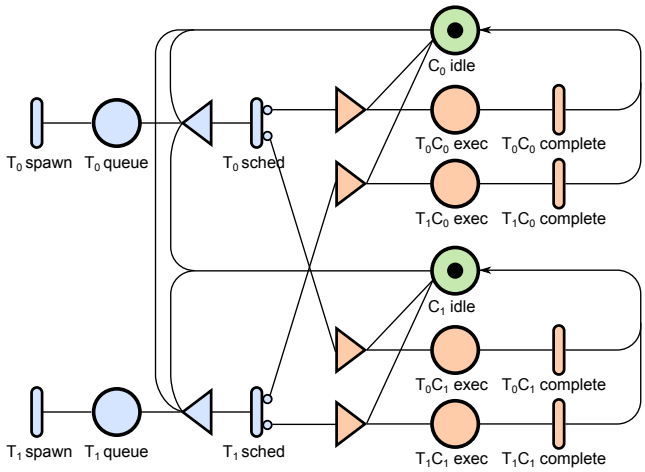

Figure 6. Model for the naïve scenario leading to error contamination.

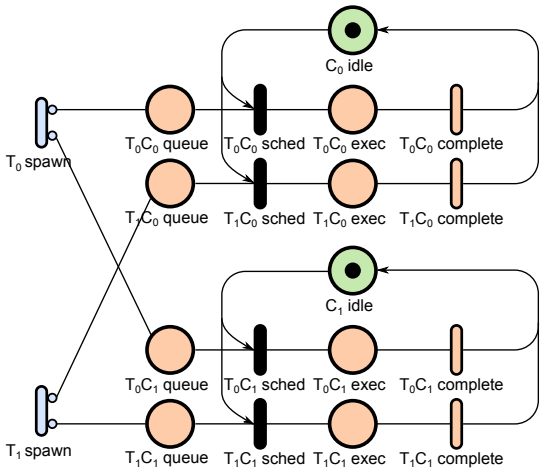

Figure 7. Fixed affinities in the scenario produce a better model.

IV. MODELS AND RESULTS

Following Section II-A, by using black-boxing some x cores can be grouped into more abstract meta-cores combining the performance, power consumption, and scheduling probabilities of constituent cores. We sacrifice the accuracy by considering the task completion in the meta-core to be exponentially distributed with the rate λ^x_{exec}, while it is in fact the sum of exponential distributions.

The model template for scaling is a parametrised SAN where the elements are replicated to a given target number of system resources. In our case, the templates scale to n cores and m tasks. Figures 6 and 7 show example models scaled to 2 tasks and 2 (meta-)cores. Hence, some model elements are added per task (prefixed with T_i, $0 \leq i < m$), some appear per core (prefixed with C_j, $0 \leq j < n$); the others are instanced $n \times m$ times: per-core and per-task (interfaces). Different types are shown in different colours. Colour is not a part of an actual model and is used solely for visual aid.

All the scenarios in Section III use two types of tasks ($m = 2$), and the scaling is done only on the number of cores. The models representing different levels of abstraction are:

3+4 model ($n = 7$) is the most detailed model considering

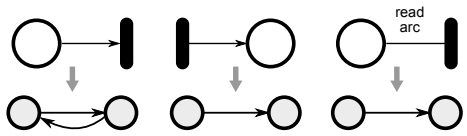

Figure 8. Rules for SAN mapping to dependency graphs.

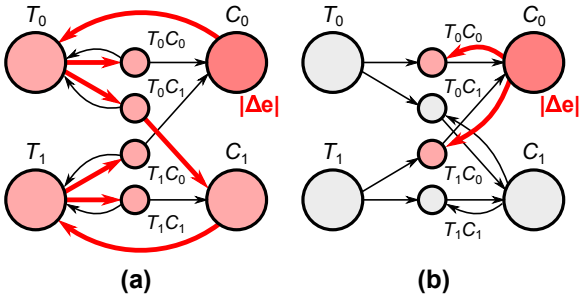

Figure 9. Dependency graphs with highlighted error contamination paths: (a) for Figure 6, (b) for Figure 7.

each core separately.

1+4 model ($n = 5$) is the model obtained using the proposed method of selective abstraction. Here, three A7 cores are grouped into a single meta-core representing the entire domain.

1+1 model ($n = 2$) is the most abstract model with two meta-cores, one representing the A7 domain, the other representing A15.

Table I gives $q_{A7} = 0.065$, and from (1), under the assumption of constant Δe_x, we have $\Delta E = 0.065 \cdot \Delta e_x$ for the (1+4) model. This assumption can be justified by simulation results if

$$\left|E_{(1+4)} - E_{(3+4)}\right| \approx q_{A7} \cdot \left|E_{(1+1)} - E_{(3+4)}\right|, \qquad (3)$$

where $E_{(1+1)}$, $E_{(1+4)}$, and $E_{(3+4)}$ are the total percentage errors in the respective models.

The initial experiment setup was not specific on how exactly task affinities are realised in the scheduler, so we implemented two variants of the system and produced two models respectively. Figures 6 shows the model of a system with task-exclusive queues, where the scheduling weights are applied after the queueing, which means that the scheduler needs to check every core for availability. Figure 7 models tasks with fixed affinities, i.e. the task is paired with a core once it is spawned and then waits in the queue for this core to become available (idle).

Unfortunately, the first implementation proved to be of a poor quality because of error contamination due to a high inter-dependence of its elements. Figure 9a shows its dependency graph, obtained by the mapping rules from Figure 8. Let's assume C_0 elements produce extra $|\Delta e|$ due to black-boxing, and C_1 must be protected from error contamination. From the graph, it is evident that there is a path connecting C_o to every other element in the model propagating the error. Figure 9b, showing the second implementation, is able to contain the error locally: it pollutes only the adjacent nodes, but keeps C_1 unaffected.

Table II
SIMULATION RESULTS COMPARED TO MEASURED VALUES

scen	model	pwr, W	pwr var	error	sim, s
EQ	1+1	1.7662	0.0470	6.53%	2.535
	1+4	1.7287	0.0424	4.27%	2.546
	3+4	1.7215	0.0423	3.84%	2.764
	meas.	*1.6579*	*0.0572*		
CA	1+1	2.0205	0.0619	7.99%	2.545
	1+4	1.9470	0.0468	4.06%	2.610
	3+4	1.9421	0.0468	3.79%	2.764
	meas.	*1.8711*	*0.0385*		
TCA	1+1	2.0038	0.0608	10.41%	2.547
	1+4	1.9274	0.0439	6.21%	2.613
	3+4	1.9245	0.0440	6.05%	2.771
	meas.	*1.8148*	*0.0279*		

We were unable to find a way to resolve the contamination by model transformation without changing the system's behaviour, as the problem has roots in the functional properties of a modelled system. This means that the algorithm modelled by Figure 6 has no viable selective abstraction and system redesign is needed (Figure 2). The main contribution of this work is to detect and identify error contamination in models by the structural analysis, as described above, before the model is simulated. The rest of the paper uses the template shown in Figure 7.

A. State-Space Analysis

Our investigation showed that state-space analysis has a number of limitations compared to simulations. The presented models have unbounded state spaces, and, unless we put a hard constraint on the total number of tasks in the system, the state-space analysis is not possible. From the task spawn rates we know that on average the number of tasks spawned during a 15s experiment is 75. The lack of scalability in state-space analysis makes even this number infeasibly large. The highest number that doesn't fail to compute is 50 for the (1+1) model, taking 5,729s of computation time on a 2-core Intel i7 5500U machine. For more detailed models, the feasible number of tasks goes down to 12 for (1+4) and 9 for (3+4), which is not enough for trustworthy results.

Consequently, with the added restriction to exponentially distributed rates, the state-space analysis methods appear rather impractical for the presented application. We recommend using simulations instead, reserving state-space analysis only for model verification purposes.

B. Simulation Results

Table II present the power values obtained from simulations in Möbius tool and compares them against the actual power measurements. The simulation time is given for 1500 simulation batches, which are required for calculating 0.01 relative interval with 95% confidence. The variances have been calculated separately over 20,000 simulation batches. Experimental results are averaged over 50 random trace runs for each scenario, and the variance is also calculated.

The achieved 4–6% accuracy meets the typical requirement for system-wide power modelling and shows a good potential

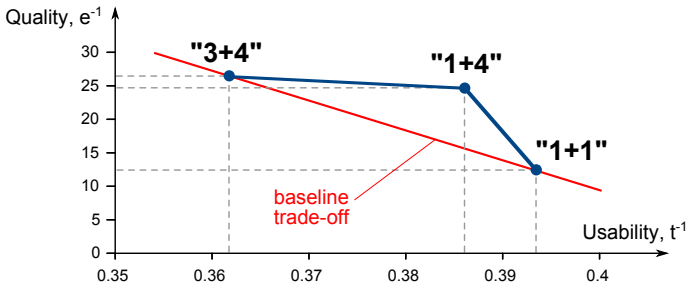

Figure 10. Simulation results in the Quality/Usability trade-off diagram

in using stochastic methods. The results justify (3), thus confirming the expectations of selective abstraction metric: the error added by moving from (3+4) to (1+4) model in comparison to going from (3+4) straight to (1+1) is proportional to the power output of A7 domain in relation to the total power. The difference between simulation times, however, is not large and appears to grow linearly with the model size in the presented sample.

Figure 10 plots the simulation results in a Quality/Usability trade-off space with the inverse of error e^{-1} representing Quality, and the inverse of simulation time t^{-1} being the metric for Usability. The results demonstrate that the method allows trading little accuracy for the steady increase in usability, and demonstrates the scalability of SAN models for simulation studies.

V. Conclusions

This work aims to produce a method towards scalable power models for multi-core heterogeneous systems. We concentrate on rationalising model sizes based on power-proportional representation and stochastic modelling. A systematic approach to selective abstraction using OGs is developed. The method includes ways of identifying error contamination and determining boxability. Stochastic techniques are investigated with SAN and Möbius. Selective abstraction is shown to be effective for model size and designer effort reduction, and SAN models are demonstrated to have excellent scalability for simulations. In these ways our method supports the systematic discovery of good trade-offs between modelling quality and model scalability.

In addition to further improvements to the SAN model of the platform by adding memory and cache, the future work may include the application of cross-layer cuts to other modelling methods.

Acknowledgement The authors want to thank Rishad Shafik and Mohammed Al-Hayanni for useful discussions. This work is supported by EPSRC as a part of PRiME project EP/K034448/1. A. Aalsaud is supported by a postgraduate studentship from the Education Ministry of Iraq

References

[1] The gem5 simulator system. http://www.m5sim.org.
[2] The Möbius modelling tool. https://www.mobius.illinois.edu.
[3] Odroid XU3. http://www.hardkernel.com/main/products.
[4] Accellera systems initiative. IEEE 1666 standard: SystemC language reference manual. http://www.accellera.org/, 2011.
[5] Synopsys. Primetime PX. https://www.synopsys.com/apps/support/training/primetime_fcd.html, 2015.
[6] A. Bartolini, M. Cacciari, A. Tilli, and L. Benini. Thermal and energy management of high-performance multicores: Distributed and self-calibrating model-predictive controller. *Parallel and Distributed Systems, IEEE Transactions on*, 24(1):170–183, Jan 2013.
[7] M. Caldari, M. Conti, P. Crippa, G. Nuzzo, S. Orcioni, and C. Turchetti. Instruction based power consumption estimation methodology. In *Intl. Conf. on Electronics, Circuits and Systems '02*, volume 2, pages 721–724, 2002.
[8] M. Chen, D. Yue, X. Qin, X. Fu, and P. Mishra. Variation-aware evaluation of MPSoC task allocation and scheduling strategies using statistical model checking. In *Proc. to DATE '15*, pages 199–204, 2015.
[9] X. Chen, G. Zhang, H. Wang, R. Wu, P. Wu, and L. Zhang. MRP: mix real cores and pseudo cores for FPGA-based chip-multiprocessor simulation. In *Proc to DATE '15*, pages 211–216, 2015.
[10] A. Ehrenfeucht and G. Rozenberg. Zoom structures and reaction systems yield exploration systems. In *IJFCS*, pages 275–306, 2014.
[11] P. Greenhalgh. *big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7 – Improving Energy Efficiency in High-Performance Mobile Platforms*. ARM, 2011. White Paper.
[12] S. Kaiser, I. Materic, and R. Saade. ESL solutions for low power design. In *Proc. of the ICCAD*, pages 340–343, 2010.
[13] V. Narayanan, I.-C. Lin, and N. Dhanwada. A power estimation methodology for SystemC transaction level models. In *Proc. to CODES+ISSS '05*, pages 142–147, 2005.
[14] A. Rafiev, F. Xia, A. Iliasov, R. Gensh, A. Aalsaud, A. Romanovsky, and A. Yakovlev. Order graphs and cross-layer parametric significance-driven modelling. In *Proc. to ACSD*, 2015.
[15] A. Rafiev, F. Xia, A. Romanovsky, and A. Yakovlev. Error-based metric for cross-layer cut determination. Technical Report NCL-EEE-MICRO-TR-2016-201, School of EEE, Newcastle University, 2016.
[16] 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.
[17] T. Sassolas, C. Sandionigi, A. Guerre, J. Mottin, P. Vivet, H. Boussetta, and N. Peltier. A simulation framework for rapid prototyping and evaluation of thermal mitigation techniques in many-core architectures. In *Proc. in ISPLED '15*, 2015.
[18] M. J. Walker, A. K. Das, G. V. Merrett, and B. Hashimi. Run-time power estimation for mobile and embedded asymmetric multi-core cpus. In *HIPEAC Workshop on Energy Efficiency with Heterogenous Computing*. HiPEAC, Jan 2015.
[19] S. Yang, R. A. Shafik, G. V. Merrett, E. Stott, J. M. Levine, J. Davis, and B. M. Al-Hashimi. Adaptive energy minimization of embedded heterogeneous systems using regression-based learning. In *Proc. to PATMOS '15*, 2015.
[20] F. W. Zurcher and B. Randell. Iterative multi-level modeling - a methodology for computer system design. In *in Proc. IFIP Congress 68*, pages 138–142. Press, 1968.