COMPUTING SCIENCE

Studying the Interplay of Concurrency, Performance, Energy and Reliability with ArchOn – an Architecture-open Resource-driven Cross-layer Modelling Framework

Ashur Rafiev, Alexei Iliasov, Alexander Romanovsky,
Andrey Mokhov, Fei Xia and Alex Yakovlev

TECHNICAL REPORT SERIES

No. CS-TR-1408 January 2014

TECHNICAL REPORT SERIES

No. CS-TR-1408 January, 2014

Studying the Interplay of Concurrency, Performance, Energy and Reliability with ArchOn – an Architecture-open Resource-driven Cross-layer Modelling Framework

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

Abstract

The interplay between pairs of critical factors such as performance, energy and reliability within modern computing systems has always been an interesting topic of study. However, studying the interplay of all three factors together in a many-core, multi-layer design setting has been a relatively recent undertaking. This work explores the practical problems encountered in such studies and introduces the modelling framework ArchOn, which is based on a novel resource-driven graph representation. ArchOn facilitates the analysis and potentially design and synthesis of systems whose design domains are more conveniently organized into multiple layers or levels (e.g. application, OS, hardware, etc.) and potentially large scale and diverse types of concurrency. The layer-agnostic formalism helps designers reason about cross-layer issues and the resource-driven approach is advantageous for reasoning about such issues as energy and time. Example single- and multi-core case studies help explain and illustrate the method.

Printed and published by Newcastle University,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

RAFIEV, A., ILIASOV, A., ROMANOVSKY, A., MOKHOV, A., XIA, F., YAKOVLEV, A.

Studying the Interplay of Concurrency, Performance, Energy and Reliability with ArchOn – an Architecture-open Resource-driven Cross-layer Modelling Framework
[By] A. Rafiev, A. Iliasov, A. Romanovsky, A. Mokhov, F. Xia and A. Yakovlev

Newcastle upon Tyne: Newcastle University: Computing Science, 2014.

(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1408)

Added entries

NEWCASTLE UNIVERSITY
Computing Science. Technical Report Series. CS-TR-1408

Abstract

The interplay between pairs of critical factors such as performance, energy and reliability within modern computing systems has always been an interesting topic of study. However, studying the interplay of all three factors together in a many-core, multi-layer design setting has been a relatively recent undertaking. This work explores the practical problems encountered in such studies and introduces the modelling framework ArchOn, which is based on a novel resource-driven graph representation. ArchOn facilitates the analysis and potentially design and synthesis of systems whose design domains are more conveniently organized into multiple layers or levels (e.g. application, OS, hardware, etc.) and potentially large scale and diverse types of concurrency. The layer-agnostic formalism helps designers reason about cross-layer issues and the resource-driven approach is advantageous for reasoning about such issues as energy and time. Example single- and multi-core case studies help explain and illustrate the method.

About the authors

Ashur Rafiev is an RA on the EPSRC PRiME Program Grant. In this project he leads the development of the ArchOn modelling environment.

Alexei Iliasov is a Researcher Associate at the School of Computing Science of Newcastle University, Newcastle-upon-Tyne,UK. He got his PhD in Computer Science in 2008 in the area of modelling artefacts reuse in formal developments. His research interests include agent systems, formal methods for software engineering and tools and environments supporting modelling and proof .

Alexander (Sascha) Romanovsky is a Professor in the Centre for Software and Reliability, Newcastle University. His main research interests are system dependability, fault tolerance, software architectures, exception handling, error recovery, system structuring and verification of fault tolerance. He received a PhD degree in Computer Science from St. Petersburg State Technical University and has worked as a visiting researcher at ABB Ltd Computer Architecture Lab Research Center, Switzerland and at Istituto di Elaborazione della Informazione, CNR, Pisa, Italy. In 1993 he became a postdoctoral fellow in Newcastle University, and worked on the ESPRIT projects on Predictable Dependable Computing Systems (PDCS), Design for Validation (DeVa) and on UK-funded projects on the Diversity, both in Safety Critical Software using Off-the-Shelf components. He was a member of the executive board of EU Dependable Systems of Systems (DSoS) Project, and between 2004 and 2012 headed projects on the development of a Rigorous Open Development Environment for Complex Systems (RODIN), and latterly was coordinator of the major FP7 Integrated Project on Industrial Deployment of System Engineering Methods Providing High Dependability and Productivity (DEPLOY). He now leads work on fault tolerance in Systems of Systems within the COMPASS project and is Principal Investigator of Newcastle's Platform Grant on Trustworthy Ambient Systems.

Andrey Mokhov studied computing science at Kyrgyz-Russian Slavic University from 2000 to 2005. After graduation with honours he joined the Asynchronous Research Group at Newcastle University as a PhD student and in 2009 he successfully defended his PhD dissertation. Currently he is a research associate in the School of Computing Science, Newcastle University. His research interests include different levels of electronic design automation: from formal models for system specification and verification to logic synthesis and application-specific optimisation.

Fei Xia is a Senior RA on the EPSRC PRiME Program Grant. Fei is with the EE School. His research interests include Asynchronous Data Communication. Asynchrnous System Design. Systems and Networks on Chip. Energy and Power in Computing.

Alex Yakovlev received D.Sc. from Newcastle University in 2006, and M.Sc. and Ph.D. from St. Petersburg Electrical Engineering Institute in 1979 and 1982. Since 1991 he has been at the Newcastle University, where he worked as a lecturer, reader and professor at the Computing Science department until 2002, and is now heading the Microelectronic Systems Design research group (http://async.org.uk) at the School of Electrical, Electronic and Computer Engineering. His current interests and publications are in the field of modeling and design of asynchronous, concurrent, real-time and dependable systems on a chip. He has published four monographs and more than 200 papers in academic journals and conferences, has managed over 25 research contracts.

Suggested keywords

MANY CORE SYSTEMS
MODELLING
DEPENDABILITY

Studying the Interplay of Concurrency, Performance, Energy and Reliability with ArchOn – an Architecture-open Resource-driven Cross-layer Modelling Framework

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

***Abstract*—The interplay between pairs of critical factors such as performance, energy and reliability within modern computing systems has always been an interesting topic of study. However, studying the interplay of all three factors together in a many-core, multi-layer design setting has been a relatively recent undertaking. This work explores the practical problems encountered in such studies and introduces the modelling framework ArchOn, which is based on a novel resource-driven graph representation. ArchOn facilitates the analysis and potentially design and synthesis of systems whose design domains are more conveniently organized into multiple layers or levels (e.g. application, OS, hardware, etc.) and potentially large scale and diverse types of concurrency. The layer-agnostic formalism helps designers reason about cross-layer issues and the resource-driven approach is advantageous for reasoning about such issues as energy and time. Example single- and multi-core case studies help explain and illustrate the method.**

I. Introduction

Translating integration scaling to performance growth is challenged by such factors as the utilization wall [16]. Processor clock frequencies have not increased since 2005 despite increasing transistor speed [15]. Using multi-core to maintain the predictions of Moore's Law [14] will only delay the inevitable [12], as the near-threshold computing (NTC) advantages are limited by such factors as variability and reliability concerns when voltage is radically scaled down. When other issues, such as reliability, are considered in addition to performance in the context of increasing parallelism (e.g. increasing the number of cores), there is little concrete research result to be found in the literature and the picture is even more uncertain.

Modern computing systems, especially mobile and/or embedded systems, must deal with a high degree of uncertainty from within the systems themselves and without in the environment. Examples include the not-always predictable inherent physical parameters such as temperature, voltage, energy availability, external noise, variability etc. and from unpredictable user demands. In the communication-heavy devices the environment a device is communicating to can also be highly unpredictable, a natural consequence of networks based more on the concept of best effort and probabilistic behaviours than guaranteed service and deterministic behaviours.

Managing and adapting to on-chip conditions such as power, thermal dissipation, and computation flow, therefore, are vital for pushing computing forward. Powering off inactive cores [7], dynamically changing clock frequencies [19] and other power saving and performance boosting measures [18] already exist, but runtime monitoring, feedback, management and control are needed for systems to operate close to their power and thermal budgets under process and environment variation, and workload conditions difficult to predict at design time.

On-chip parametric sensing has been an important field of advancement, providing diverse and concrete support for feeding back the physical parameters needed by any runtime management scheme. In addition to detecting threshold crossing using voltage and thermal sensors to trigger simple actions such as throttling [26], it's also possible to sense a wider range of parameters including current/power [17], process variations [20], supply noise [4], voltage drop [23], transistor ageing [30] and electromagnetic induction [10]. More recently, reference-free sensing techniques have also been developed which would support better sensing under uncertain operating conditions, especially when no high-quality references can be had in terms of voltage, current or frequency [25].

How to make the best use of this information available at runtime, and conduct a coherent runtime management of multi-core systems where factors such as reliability, power, throughput and ageing need to be balanced with the demand from system operational requirements remains very much an open question, in spite of a substantial body of research tackling sub-problems [7], [12], [14], [25].

The main goal of the PRiME project [3] (of which this work is an integral part) is to develop a comprehensive runtime management system that takes into account all of the natural layers of computation, from application to core software (including OS) to hardware, is able to receive and process information from all layers and maintains cross-layer communications and interactions. For this to be meaningful, a solid theoretical foundation based on concrete mathematical models at all levels of detail is necessary. Layer-crossing has been a design concern for a long time and most methods concentrate on building interfaces between layers, a pragmatic approach which allows experts of different disciplines to both

work together and remain in their respective comfort zones. However, a unified mathematical foundation should not be ignored as it ultimately enables the different and diverse layers to be reasoned together.

This paper describes an initial investigation of the interplay of the essential parameters of multi-core computation, including performance (P), energy (E) and reliability (R) as well as how they relate to the scale of parallelism, i.e. the number of cores being used. This investigation leads to a set of techniques where data collected from hardware in the design process could be used to build design-time models of reliable operating regions for systems (PER models) which can be used to fully annotate the eventual system model used by the runtime management system.

An architecture-open, layer-agnostic modelling and design framework called ArchOn is then presented. ArchOn targets the reasoning, analysis, and design of systems with diverse and potentially large concurrency whilst especially taking into account the interplay between such salient metrics as PER and the related parameters power, voltage, delay, etc. Studies using ArchOn demonstrate its relevance especially with regard to different types and degrees of concurrency.

The paper is organized as follows: Section II discusses the interplay between PER parameters within a context of concurrency scaling; Sections III and IV introduce the ArchOn framework formally; Section V discusses PER with regard to concurrency between multiple but different components within a single computation core, and uses ArchOn to solve such an example problem; Section VI exemplifies ArchOn's usability in the context of larger scale concurrency, i.e. multiple cores. The paper is concluded with further discussion in Section VII.

II. The Region of Reliable Operation

In digital CMOS systems, a higher supply voltage (V) usually allows a higher operating (clock) frequency and hence a higher throughput, at the cost of higher power dissipation. When power is limited, it is possible to obtain an increase in system throughput by scaling to multiple computation units, i.e., cores, if the computation can be reasonably parallelised. Parallelisation scaling in this fashion could also be used to tackle the related problem of reducing power consumption while faced with a certain throughput requirement.

This type of parallelisation scaling has been known to provide the best advantage when V is scaled down to just above the threshold voltage of the CMOS node concerned. This is known as near threshold computing (NTC), which maximally takes advantage of the parallelisation scaling without entering the sub-threshold region, where a number of other factors limit the throughput and efficiency under further parallelisation. However, what happens to reliability in this region is not well known, because these general knowledge points were obtained usually by considering reliability as a separate issue. Here we study the inter-relationship of all these issues together, namely voltage, throughput, power, reliability and parallelisation scaling.

In order to retain reliability, a system must operate within certain constraints. For instance, a particular hardware implementation may not behave correctly if the supply voltage V goes below or above certain values. Different hardware components in a system may have different minimum and maximum V values and this means that software components, depending on how they are mapped onto hardware, could also be constrained by different V limits, even within the same system. Another type of constraint is the performance/throughput requirement specification. A system or a part of a system may be required to attain at least a certain level of throughput for the execution to be meaningful. A third type of constraint is the power supply limitation. The amount of available power limits the behaviour of the system.

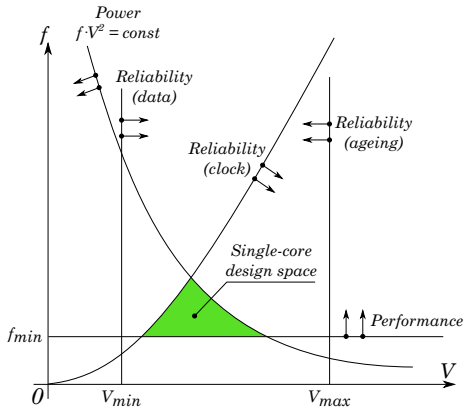

Figure 1. The region of reliable operation

The minimum latency, which is related to computation throughput, of any specific hardware logic is related to the V supplied to it. This means that if this logic is run on a clock too fast for a certain V the computation may not complete in time before the next clock pulse arrives, which usually leads to unreliable or unusable results. A common technique for determining the appropriate clock frequency for computation logic is to first determine the critical path delay of the logic under the given V condition, and add enough delay margins to account for potential effects of process, voltage and temperature (PVT) variations.

The region of reliable operation for a system within the V/throughput space is bounded by constraints on power, timing reliability, minimum throughput requirements and low and high V boundaries. The minimum and maximum V boundaries and the minimum throughput are defined as

$$\min V \leq V \leq \max V \qquad Throughput \geq \min Throughput. \quad (1)$$

The power limit and timing reliability boundaries, however, are more complex. Their general shapes can be derived from theories on semiconductor characteristics. For instance, the timing reliability boundary within a V – throughput/frequency space usually starts off at very low V with the frequency climbing from almost zero upwards exponentially until V increases to around the threshold voltage, from there upwards the increase in frequency is more or less linear, until usually when V is around the nominal V of the technology, where the frequency increase starts to saturate. The region of reliable operation is illustrated by the diagram in Figure 1.

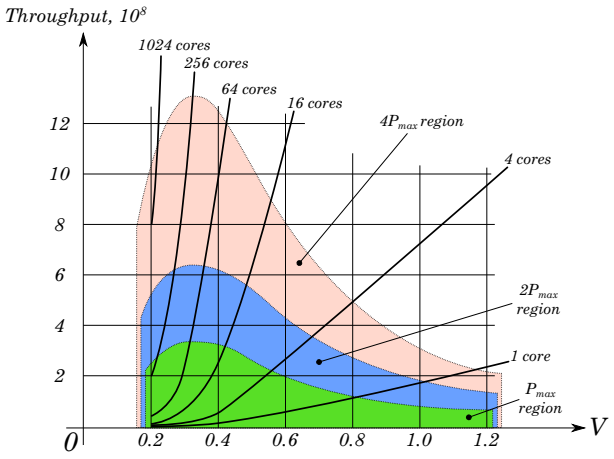

Figure 2. SRAM constant max power curve and scaling lines

For reliable operation, the system throughput must be restricted below the timing (clock) reliability and power limit lines and must be above the throughput requirement line. The system's V must also be restricted between the high and low V limits. In the super-threshold region, we can hypothesize that dynamic power dominates power consumption while leakage power and its influence can be ignored.

Dynamic power is known to be related to switching activity (and through which to system frequency), switching swing voltage (and through which to system V) and switching element capacitance (and through which to system size/area – which is a constant before hardware scaling). Lumping all the constants together we can say that power is related to frequency and V in the following manner:

$$P = cFV^2, \tag{2}$$

where c is a constant, P is the power and F is the frequency. In this work, we explore the issue of core scaling (increasing or decreasing the degree of parallelisation) with an assumption of perfect scaling. Multiplied hardware operating at the same frequency will provide multiplied throughput and require a multiplied amount of power based on the same constant multiplier. Non-zero scaling overheads will be investigated in the future. In perfect scaling with a scaling factor of n, the constant c is scaled in the same way, i.e.

$$c = nc_1, \tag{3}$$

where c_1 is the c for the hardware before scaling (e.g. a single core). In general, scaling with a factor of n will give a new c which is a factor of n of the unscaled c.

For each core in a new scaled set-up, the available power is also changed by a factor of $1/n$. Considering these factors, for each core, equation (2) now becomes

$$P_n = cFV^2/n, \tag{4}$$

and the overall system power equation with n cores stays the same as (2).

With the V values near threshold to below threshold, the super-threshold power relation (2) no longer describes the total power as leakage power starts to become an equally important or more important factor compared with dynamic switching power.

Instead of drawing the constant power curves from equations (2) and (4) or dealing with the much more complex power equations taking leakage power into account, observed power from experiments can be used to estimate the points along the frequency curves pertaining to any known power budget.

Figure 2 shows the scaling effects on the reliable operation region, based on experimental data collected from an asynchronous SRAM [6], with the total power from experimental data (and not calculated from (2) and (4)) including both dynamic and leakage power. As can be seen after about $n = 256$, further scaling would not improve the system throughput of the system given the constant power limit reached at the max writing frequency and voltage of $1.2V$ without scaling, but the operable region continues to be expanded to the left, although the increase becomes smaller even without considering $\min V$.

To determine the shape of power limit boundary curves, we start by setting a power limit amount, P_{lim}, usually the power consumption when operating an unscaled system at nominal V and the appropriate reliable frequency for this V (P_{max}). Then for each point i where there is experimental power data, we calculate the maximum possible scaling factor n_i for that V based on

$$n_i = P_{\text{lim}}/P_i, \tag{5}$$

where Pi is the experimental power observed at data point i. This, similar to (3) and (4), is based on the perfect scaling assumption. Plotting $Throughput = n_i Throughput_i$ gives the power limit curve for the particular P_{lim}.

Figure 3 shows a similar study on the reliable operation region and its relationship with power and parallelisation scaling. This is based on experimental data from a low-power ARM M0 core implementation [21]. Both examples demonstrate that the relationship between power limits, parallelisation scaling, and the reliable operation region within the V/throughput space is the same with both processors and memory, and the reliable operation region is reasonably straightforward to model given standard experimental data collected during any reasonable hardware design process.

III. ArchOn envisioned

Due to the unusually vast design space, our research demands a special thinking in designing the model. The modelling and simulation methods are required to be open-ended with the capability of easily extending the simulated platform's functionality. We approached the problem in three stages, as described below.

Hardware vision approach: The traditional method of creating extendible software is based on plug-in modules. In this case, the design of module interfaces is crucial and generally defines how difficult it is to create a program extension.

In order to understand the interaction between software modules, we have taken an unusual approach: "think hardware – make software". We imagined that we are to make a flexible

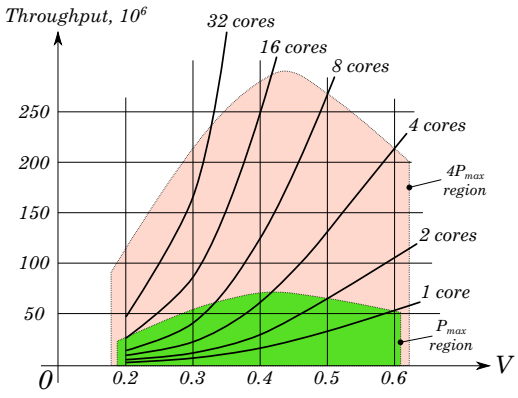

Figure 3. Scaling close to and in the sub-threshold region

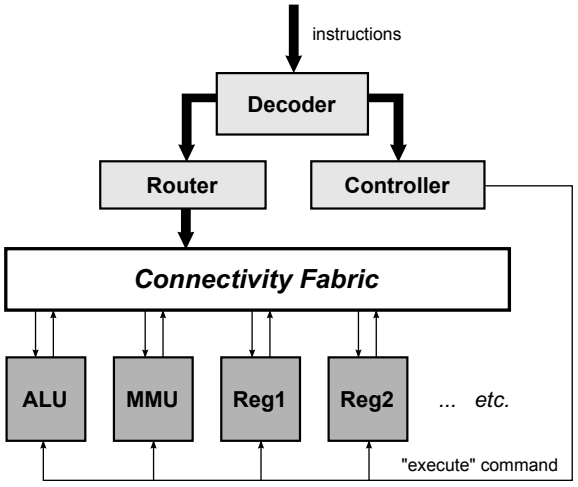

Figure 4. What a flexible architecture would look like in silicon? Making software design decisions while thinking in hardware terms.

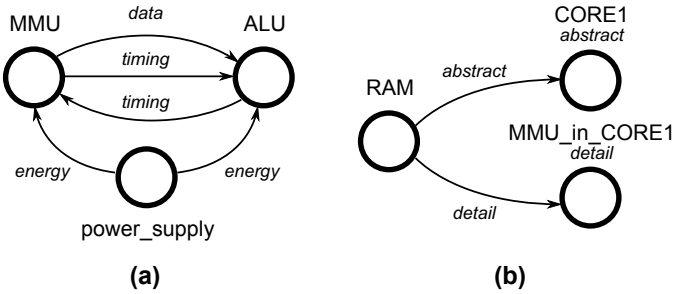

Figure 5. Examples of using labelled graphs to reason about diverse resource types and dependencies (a), and different levels of abstraction (b).

hardware architecture. What would the challenges be, and how should we overcome them? Just as FPGA allows customisation at the scale of logic gates, our hypothetical hardware must allow customisation at the scale of hardware modules, e.g. ALUs, register banks, memory units. Although in our research we do not have an actual task to deliver such a platform, this non-traditional thinking immediately paid back with a number of original design decisions, giving a better insight to the model from the simulation perspective.

Figure 4 shows a communication-based hardware architecture that could potentially emulate most cyber-physical systems. This type of architecture is called transport-triggered architecture [9]. It hasn't become popular in general purpose microprocessors, but it appeared attractive for our purposes. Assuming that the target system has an instruction set, its software can be recompiled into the connectivity fabric routing commands. The process of executing such software would have alternating phases of configuring the connectivity fabric and executing modules.

Of course, not everything can be envisaged in terms of hardware logic modules. At some point we had to introduce other types of elements like, for example, limited energy pool or time in order to explore the system capabilities. This is where our model diverged from the purely hardware view.

Resource dependency approach: 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, time.

We propose 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. Modelling different types of resources may be achieved by labelling the graph, as illustrated in Figure 5(a).

With a reference to the hardware vision approach, we also prefer to view the system as a dynamic set of resource relations. Resources may become unavailable in certain points in time, and the model must be able to capture this behaviour. The understanding of resource availability properties helps to plan ahead and orchestrate resource consumption at a high and yet sustainable rate. A dynamic model can be represented using the state transition semantic, where the states are concrete resource allocations or configurations.

Cross-layer approach: Organising systems, both practically and conceptually, as hierarchies is a popular way of thinking and engineering. The practical motivation for this is manageability. This is the "natural" way for humans to reason about, design, and organize most of our systems. Since in this project we emphasize the cross-layer aspects of our work, having a flat graph model as the foundation may seem counter-intuitive.

In fact, the flat labelled graph approach facilitates the cross-layer way of thinking, as Figure 5(b) demonstrates. A label can be viewed as a condition that includes or excludes an edge or a vertex, giving a graph *projection* onto that label. The complexity of the system can be dealt with using projections of the resource graphs. With resources as diverse as a software instruction or a single hardware gate, within the same single graph executed in a transition, we could reason about different parts of the system at different abstraction layers. This helps a designer focus their attention on any particular details of a system they want, and build a system either top down or

bottom up or with mixed-level components at different stages of development. This layer-agnosticism also helps a runtime management scheme to focus attention on different layers at the same time. This versatility is not available for methods which use explicit hierarchy within their frameworks.

At the same time, this does not prevent designers to isolate concerns and concentrate on some layers only. For instance, all resources in one transition could be elements of the same layer, or a software engineer could arrange complex low-level software resources for detailed study with coarse-grain hardware resources provided by hardware colleagues (which are not the specific target of concern) in the same transition.

Recently, in the area of modelling biological systems and chemical processes, the authors of [11] developed "exploration systems" which are based on describing system dynamics as transitions among graphs, which contain the static knowledge of system components and their inter-relations including dependencies. This division of system dynamics and static knowledge allowed the use of an essentially flat model for the dynamic evolution of systems whilst retaining a potential view of system static knowledge in a hierarchy of unbounded number of layers. Compared with these exploration systems, ArchOn is more targeted at computation systems without a biological or chemical specialization, allowing a richer representation of system hierarchy from its specifically layer-agnostic approach. ArchOn also emphasizes the issue of concurrency and its active management which is not a focal point for exploration systems.

IV. Model fundamentals

The ArchOn framework is a further development of Conditional Partial Order Graphs [22] that have been used for modelling low-level multimode asynchronous microcontrollers.

An ArchOn system is defined by

- set R of resources;
- *communication fabric* relation $\mathscr{A}$ defining feasible dependencies (communication channels), $\mathscr{A} \subseteq \mathsf{R} \times \mathsf{R}$;
- set A of currently present inter-resource dependencies, $\mathsf{A} \subseteq \mathscr{A}$;
- indexed set $(\mathsf{U}_i)_{i\in\mathsf{R}}$ of resource states; set $\mathsf{U} = \times_{i\in\mathsf{R}}\mathsf{U}_i$ denotes the overall resource state;
- resource and resource dependency labelling function $\mathsf{L} \in (\mathsf{A}\cup\mathsf{R}) \to \mathscr{L}$ attaching uninterpreted labels from set $\mathscr{L}$;
- initial resource state $\mathsf{U}_0 \in \mathsf{U}$;
- finite set I of configurations;
- indexed set $(\varepsilon_i)_{i\in\mathsf{I}}$ of idempotent reconfiguration functions $\varepsilon_i \in \mathsf{A}\times\mathsf{L} \to \mathsf{A}\times\mathsf{L}$; it must hold that $\varepsilon_i(a\mapsto l) = \varepsilon_i(\varepsilon_i(a\mapsto l))$ for all $i \in \mathsf{I}$;
- indexed set $(\varphi_i)_{i\in\mathsf{R}}$ of resource evolution functions $\varphi_i \subseteq \mathsf{U}_i \times \mathsf{U}_i$;
- strict partial order relation $\prec$ on set I parametrised by U: $\prec \in \mathsf{U} \to \mathbb{P}(\mathsf{I}\times\mathsf{I})$ where for every $u \in \mathsf{U}$, $\prec_u \subseteq \mathsf{I}\times\mathsf{I}$ is irreflexive and transitive.

A resource is deemed *ready* when all its input dependencies are connected (enabled). For some resource $r \in \mathsf{R}$ we define predicate $\mathrm{ready}(r)$ to be

$$\mathrm{ready}(r) \equiv r \in \mathsf{R} \wedge \mathsf{A}^{-1}[\{r\}] = \mathscr{A}^{-1}[\{r\}].$$

Here $\mathsf{A}^{-1}[\{r\}]$ denotes the pre-set of r (set of resources on which r has a dependency). In the paper we also use short-cut notation $\bullet r$ and $r\bullet$ to denote pre-set and post-set (which is $\mathsf{A}[\{r\}]$, or set of resources dependent on r) of resource r. A ready resource r may *fire* resulting in a new resource state u'_r; resources that are not ready do not change their state:

$$\begin{cases} u'_r \in \varphi_r[u_r], \text{ for } \mathrm{ready}(r), \\ u'_r = u_r, \text{ otherwise}, \end{cases}$$

where $u'_r \in \mathsf{U}_r$. Note the non-deterministic computation of new resource state as expressed by $u'_r \in \varphi_r[u_r]$. This allows a designer to abstract from the specifics of resource state evolution or abstract a complex of a resources by a single non-deterministic resource. All ready resources fire concurrently and the overall new state u' is

$$u' = \times_{r\in\mathsf{R}} u'_r.$$

The use of a primed identifier to denote a new state is merely a syntactic convention. Conceptually, new state u' instantaneously replaces some previous state u without observation of intermediate states u'_r computed by individual resources.

In parallel with resource state updates, the communication fabric connecting resources (configuration) may also change independently. The way of change is defined in the reconfiguration functions $(\varepsilon_i)_{i\in\mathsf{I}}$; the overall direction of change is controlled by order $\prec_u$ sensitive to the current resource state u. Communication fabric changes in steps, each such step defined by a transformation encoded in an applicable reconfiguration functions ε_i:

$$\mathsf{A}' \mapsto \mathsf{L}' \in \biguplus_{i\in(\mathsf{I},\prec_u)} \varepsilon_i(\mathsf{A}\mapsto\mathsf{L}),$$

where $\uplus$ is a generalised version of the relational override operator $\uplus$ defined as $f \uplus g \equiv \{s\mapsto t \mid (s\in\mathrm{dom}(f)\setminus\mathrm{dom}(g) \wedge s\mapsto t\in f) \vee (s\in\mathrm{dom}(g)\wedge s\mapsto t\in g)\}$. Expression $\biguplus_{i\in(D,<)} f_i$ defines a set of relations each of which is a relational override defined over indexed set $(\varepsilon_i)_{i\in\mathsf{I}}$ with the priority given by a partial order $(D,<)$:

$$\biguplus_{i\in(D,<)} f_i \equiv \{f_k \uplus f_i \uplus \cdots \uplus f_j \uplus f_n \mid n < j < \cdots < i < k\}.$$

An ArchOn system is *consistent* if eventual resources state do not depend on the choice of relation from set $\biguplus_{i\in(\mathsf{I},\prec_u)} \varepsilon_i(\mathsf{A}\mapsto\mathsf{L})$. Note that reconfiguration of communication fabric and resource state updates are only weakly synchronised through observation of resource state by the reconfiguration activity. There is no assumption made about the relative rate of progress of these two activities.

The dynamic part of ArchOn is defined by the tuple $(\mathsf{A},\mathsf{L},\mathsf{U})$ of inter-resource dependencies, labels and state. The initial state σ_0 defines no dependencies or labels and resources in the initial state U_0: $\sigma_0 = (\emptyset,\emptyset,\mathsf{U}_0)$. All subsequent states are computed by state update and reconfiguration transitions $\xrightarrow{s}$ and $\xrightarrow{c}$ defined as follows:

state update:

$$\frac{\forall r\in\mathsf{R}\cdot(u'_r\in\varphi_r[u_r]\wedge\mathrm{ready}(r)) \vee (u'_r = u_r \wedge \neg\,\mathrm{ready}(r))}{(\mathsf{A},\mathsf{L},u) \xrightarrow{s} (\mathsf{A},\mathsf{L},\times_{i\in\mathsf{R}} u'_i)},$$

configuration update:

$$\frac{\mathsf{A}' \mapsto \mathsf{L}' \in \biguplus_{i\in(\mathsf{I},\prec_u)} \varepsilon_i(\mathsf{A}\mapsto\mathsf{L})}{(\mathsf{A},\mathsf{L},u) \xrightarrow{c} (\mathsf{A}',\mathsf{L}',u)}.$$

To define reconfiguration functions ε_i and relation $\prec$ we employ a mixture of textual and graph notations. There is, of course, a tight interplay between the two as $\prec$ controls which of ε_i is applicable at any given moment. The definition ε_i for some concrete $i \in \mathsf{I}$ is given by *ArchOn expression* ξ of the following form

$$\xi = \xi\ \ \xi \mid a \xrightarrow{\ell} b \mid a \not\to b \mid a : \ell,$$

where $a, b \in \mathsf{R}$ are some resources and $\ell \in \mathscr{L}$ is a label. Statements $a \xrightarrow{\ell} b$ and $a \not\to b$ signify, correspondingly, the addition and removal of dependencies arc from the communication fabric; $a : \ell$ attaches label ℓ to resource a. The juxtaposition $\xi\ \xi$ chains statements in the left to right order. The following structural induction converts ξ statements into a value of type $\mathscr{A} \times ((\mathsf{A}\cup\mathsf{R}) \to \mathscr{L})$, given some current connectivity $\mathsf{A} \mapsto \mathsf{L}$:

$$\begin{array}{lcl}
[\![a \xrightarrow{\ell} b]\!]_{\mathsf{A}\mapsto\mathsf{L}} & \equiv & (\mathsf{A}\cup\{a\mapsto b\}) \mapsto (\mathsf{L}\cup\{(a\mapsto b)\mapsto\ell\}), \\
[\![a \not\to b]\!]_{\mathsf{A}\mapsto\mathsf{L}} & \equiv & (\mathsf{A}\setminus\{a\mapsto b\}) \mapsto (\{a\mapsto b\} \lhd \mathsf{L}), \\
[\![a : \ell]\!]_{\mathsf{A}\mapsto\mathsf{L}} & \equiv & \mathsf{A} \mapsto (\mathsf{L}\cup\{a\mapsto\ell\}), \\
[\![x\ y]\!]_{\mathsf{A}\mapsto\mathsf{L}} & \equiv & [\![y]\!]_{[\![x]\!]_{\mathsf{A}\mapsto\mathsf{L}}}.
\end{array}$$

Every instance of ξ expression must occur in the context (explicit or implied) of some $i \in \mathsf{I}$. It is then assumed that the expression is converted into a statement of the form $\varepsilon_i = [\![\xi]\!]_{\mathsf{A}\mapsto\mathsf{L}}$. The definition of $[\![\xi]\!]$ clearly satisfies the idempotence requirement of ε_i.

To represent relation $\prec$ we found it convenient to use an acyclic directed graph defined over set I. The edges of a graph define a relation $q \subseteq \mathsf{I}\times\mathsf{I}$ that is asymmetric. A reflexive and transitive closure of q yields a partial order on I. To be able to control the direction of system evolution, it is necessary to annotate the edges of such graph with predicates over U. Such a predicate conditions the existence of an annotated edge by the current resource states. This fits well with state parametrisation of $\prec_u$. More concretely, let (I, q) be an acyclic directed graph and $P \in q \to \mathscr{P}(U)$ be a function mapping edges of the graph to predicates on U. Then for each $u \in \mathsf{U}$, $\prec_u$ is computed as follows

$$\prec_u = \{a \mapsto b \mid a \mapsto b \in q \wedge P(a\mapsto b)(u)\}^+,$$

where f^+ denotes a transitive closure of relation f.

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 6. In this case, the resources are concrete hardware units: registers `reg_a`, `reg_b`, and two ALUs: `cmp` and `sub`. Resource states store unit data, and the resource dependencies represent data transfer between the units. A register is simply an identity function that copies its pre-set state, therefore in order to "maintain" the state it requires an explicit self-loop. This rule may be useful if we need to estimate the energy of the system: a self-dependent element remains in the graph as an active resource. Although comparison and subtraction can be done in just one ALU, for the sake of example we consider `cmp` and `sub` to be different resources. 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 memoryless combinational logic element, so it has no state in the model: the result is propagated to the output (post-set) node.

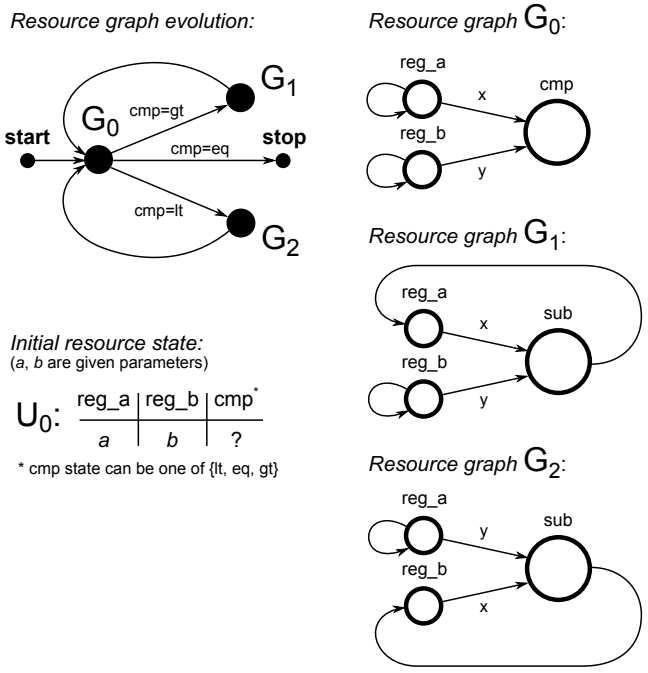

Figure 6. Simulating Euclid's algorithm for GCD(a,b). In this example, resources are hardware units with data dependencies between them.

V. Exploring concurrency in a single-core system

In this section we demonstrate the application of the ArchOn framework to modelling PER trade-offs in a single-core processor. As our example we take the basic computational step in the 3×3 matrix convolution that is used in most image processing applications. Given two 3×3 matrices A and B the goal is to multiply them element-wise and sum up the results, denoted by $A \boxtimes B$:

$$\begin{pmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{pmatrix} \boxtimes \begin{pmatrix} y_1 & y_2 & y_3 \\ y_4 & y_5 & y_6 \\ y_7 & y_8 & y_9 \end{pmatrix} = \sum_{1\le k\le 9} x_k y_k. \quad (6)$$

Usually, one of the matrices is a 3×3 sub-matrix of an image being processed and the other matrix, called a *kernel* or a *mask*, represents the required image transformation, e.g., sharpening or edge detection. This step is applied to all 3×3 sub-matrices of a given image, each time producing a value for a pixel in the resulting image. This is an embarrassingly parallelisable computation task: one can cut an image into pieces and process them in parallel on different cores. In this section, however, we focus on a single processing core, in particular on different processor microarchitectures that can be used to accelerate the

computation of a single convolution step (6). Concurrency and PER trade-offs of the parallel many-core implementation will be discussed in Section VI.

At least two hardware resources are required for performing a single convolution step: an adder and a multiplier. In this section we use time and power estimates of 16-bit circuits generated by PRIMETIME simulation tool for 130nm STMicroelectronics technology library at the nominal supply voltage of 1.2V:

Component	Latency, ns	Power, mW
Adder (Brent-Kung [8])	1.37	0.49
Multiplier (Wallace tree [32])	2.25	6.7

See Section 4.1.1 of [27] for more details on hardware implementation of adders and multipliers with different PER characteristics.

A. ArchOn modelling

Adder and multiplier are hardware resources. There are a number of ways to connect these resources in order to perform the computation of a convolution step. Each way has certain advantages and disadvantages in terms of the PER characteristics. In order to reason in PER terms, we need the ability to retrieve physical parameter estimates from our models. ArchOn models of these different configurations can be executed effectively forming simulations of system executions to provide this data.

Time evaluation: Traces extracted from ArchOn model reflect the ordering of the events, but did not capture the exact execution time. In order to add numeric time value to the model, we need to expand the resource state to store a timestamp t, so the state of a resource becomes a time and data tuple. The node function also becomes compound and computes both data and the time needed to process it.

Time dependencies between resources can be implied from data dependencies. While computing the next resource state, the timestamps of a node and its pre-set are synchronised and set to the maximum value. Delay function computes Δt and adds it to the synchronised timestamp value. It is important that time is synchronised both ways: the destination node must wait for the source to finish the computation, and the source node must retain data until the destination node is ready to use it. This is similar to request/acknowledge signals in asynchronous circuits [29]. Synchronisation and tracking of timestamps between different resources is achieved using a specialised time keeper resource.

Energy and power evaluation: The simplest way to estimate energy is by resource counting. One can add another element to the resource state – the number of invocations of this resource. When the simulation is finished, the resource access vector can be extracted from the final state of the system. These numbers can be multiplied by resource energy costs to give a rough energy estimate. This method uses constant energy cost values and does not consider leakage while the resource is idle.

A better way to model energy consumption is to associate each resource with an energy function. There is no restriction on the type of information that can be used for computing the energy. For a resource r, empty pre-set means that the resource is not enabled in the current resource graph. This can be used to capture a leakage energy:

$$E'(r) = E(r) + \begin{cases} \Delta e_{dyn}(r) + \Delta e_{leak}(r) & , \text{ if } \bullet r \neq \emptyset \\ \Delta e_{leak}(r) & , \text{ if } \bullet r = \emptyset \end{cases}$$

where $\Delta e_{dyn}(r)$ computes the dynamic energy of the resource r, and $\Delta e_{leak}(r)$ computes static (leakage) energy. $E(r)$ is the accumulated amount of energy stored as a part of the resource state. Similarly to time estimation, energy estimation requires a specialised resource node that collects information about energy usage from other resources. Power is derived from energy $\Delta E(r)$ and delay Δt.

B. Microarchitectural solutions

To accelerate the computation of the convolution step one can implement a specialised processor instruction, e.g., as in Application Specific Instruction set Processors (ASIPs) [31]. Let us study three possible microarchitectures for the instruction.

An obvious way to compute (6) is to perform the required multiplications concurrently (as they have no data dependencies) and then sum up the results with an adder tree, thus allowing as much Instruction Level Parallelism (ILP) [13] as possible. The corresponding computation tree, which reflects the unfolding of the ArchOn model, is shown in Figure 7(a, left). As one can see this implementation requires at least 9 multipliers and 4 adders, therefore being quite expensive in terms of silicon area. Another disadvantage of this microarchitectural solution is a big power consumption spike which occurs at the very start of the instruction execution when all 9 multiplications are running concurrently, as shown in Figure 7(a, right). This may cause a local voltage drop affecting the reliability of the multipliers as well as surrounding hardware components. Such voltage drops are particularly dangerous in the near- and sub-threshold operating modes, when drops of 10–20mV amplitude can cause timing failures [6], [21], [25], [28].

To mitigate the above reliability issues and balance the power profile, one can reduce the degree of concurrency. Let us halve the number of multipliers that are allowed to run concurrently. The resulting computation tree and power profile are shown in Figure 7(b). As one can see the power peak is significantly smoothed out at the cost of 23% latency increase. As a (perhaps insignificant) side benefit, this implementation also requires one fewer adder. In terms of the throughput, however, the 9-multiplier version has a significant advantage (more than 2x) if the microarchitecture is pipelined.

Finally, consider the 2-multiplier version shown in Figure 7(c). It brings further power balancing and area benefits albeit at the cost of 81% latency overheads and 4.5x lower throughput in comparison to the 9-multiplier microarchitecture.

Note that the consumed dynamic energy is the same in all three microarchitectures. However, the smallest (with respect to the occupied area) implementation will generate the least

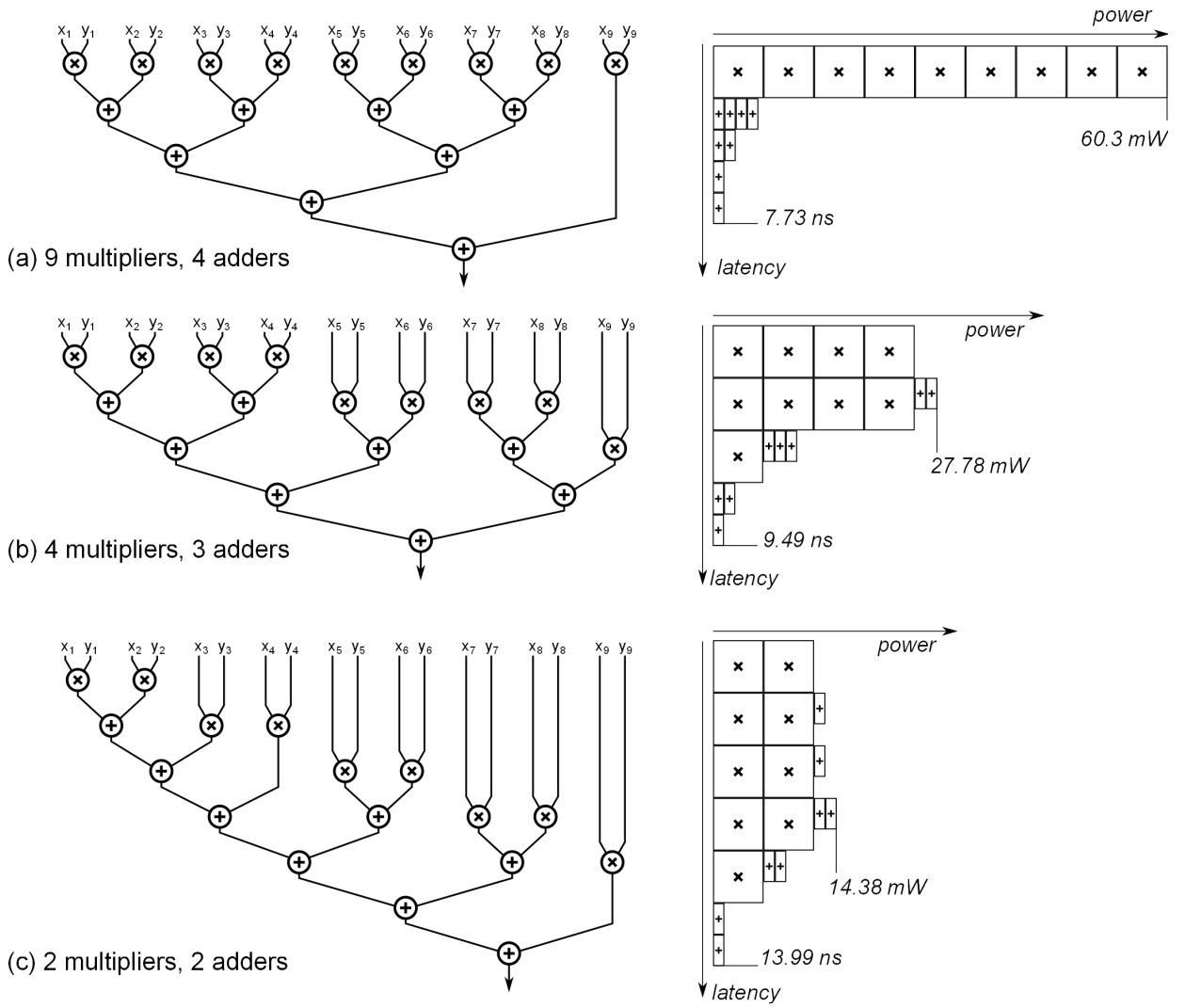

Figure 7. Computation trees and power profiles of three implementations of 3×3 convolution instruction: trading concurrency (latency) for power consumption

constant leakage current thus being more energy-efficient (unless inactive components are shut down by fine-grain power gating, in which case all three implementations will have the same energy-efficiency).

VI. Exploring concurrency in a multi-core system

Although having a specialised hardware to do the entire convolution step in a single instruction is attractive and efficient, it implies considerable production costs. It might be more practical to consider using existing hardware to do the job. As a many-core test platform we used simplified ARM architecture. The mainstream ARM processors to date are mainly dual- and quad-core, however there are concrete plans to increase the number of cores to 16, or even 32 [1].

In terms of ArchOn, the difference between multi-core and single core architectures is in the restrictions on certain connections between the resources belonging to different cores, thus fundamentally the model is still the same.

The method to supply resource graphs to the simulation software has been derived from our hardware vision, described in Section III. We view the simulator modules as connected via the connectivity fabric, and the simulator input parser works as a router. Table I shows some commands for this "router", which provide step-by-step graph configurations as well as explicit invocations of resource state transitions. Applying this method to sparsely connected graphs with many vertices gives more compact specifications than traditionally used adjacency matrices. We call it *graph assembly language*.

Table I
Some commands of graph assembly language

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

Convolution filter software is written in ARM assembly lan-

Algorithm 1 ARM instruction `MLA r8,r9,r10,r8` in graph assembly language.

$G = \emptyset$
r9 $\xrightarrow{n}$ mul
r10 $\xrightarrow{m}$ mul
go!
$G = \emptyset$
mul $\xrightarrow{n}$ alu_add
r8 $\xrightarrow{m}$ alu_add
go!
$G = \emptyset$
alu_add $\rightarrow$ r8
go!

guage. Here, a 256×256 image is divided between processing cores, each working on a separate set of pixels (with single pixel wide overlaps). Each pixel is a 32-bit integer representing grey-scale colour. For every ARM instruction we derive a resource evolution and translate it into graph assembly language. This is a routine task since all instructions follow a common pattern. The process of translation can be done automatically. An example instruction is shown in Algorithm 1.

With this example we start exploring non-ideal concurrency scaling and non-zero overheads. Since shared memory would become the bottleneck while scaling to many-cores, we added control over the "criticality" of this resource, so the program can be executed in three different modes: 1) simultaneous read and write access to the memory is allowed, 2) simultaneous read is allowed, but only one writer is allowed at a time, 3) all memory access is exclusive and must be done sequentially.

By Amdahl's law [5], the theoretical speed-up that can be achieved by executing a given algorithm on a system capable of executing n threads is:

$$\frac{T(1)}{T(n)} = \frac{1}{s_s + \frac{s_p}{n}}, \tag{7}$$

where $T(n)$ is the time an algorithm takes to finish when being executed on n cores, and $s_s \in [0,1]$ is the fraction of the algorithm that is strictly serial, $s_p = 1 - s_s$ is the fraction of the algorithm that runs in parallel.

For our algorithm s_s and s_p are not known in advance, and actually depend on the memory mode and the number of cores running, i.e. are not constants. ArchOn time estimation enables analysis of this factor. Table II gives the estimates for execution time. From (7) we can find s_p, which will be an estimate of parallelisation for our example. In Mode 1 the scaling is nearly perfect, $s_p \approx 9.999999$, however in Mode 3 the memory becomes such a narrow bottleneck that there is no performance gain for more than two cores (performance cap is shown in bold). The most illustrative example is Mode 2, when multiple cores are allowed to simultaneously read, but forbidden to simultaneously write to the memory. The performance cap is reached at 4 cores, and s_p varies from 9.96 at 2 cores to 0.4 at 4 cores and decreasing.

The main goal of this section is to use ArchOn simulation to draw PER diagrams, described in Section II. Since the actual power data for ARM cores is implementation dependant and proprietary, we use the power profile for an asynchronous SRAM [6], Figure 2. This does not affect the generality of

Table II
EXECUTION TIME (IN CYCLES) VERSUS THE NUMBER OF CORES RUNNING FOR DIFFERENT MEMORY ACCESS MODELS.

N cores	multiple read multiple write	multiple read single write	single read single write
1	26607635	26607635	26607635
2	13303827	13303947	**19660819**
3	8938515	8938755	**19660819**
4	6651923	**7864625**	**19660819**
5	5404691	**7864625**	**19660819**
6	4469267	**7864625**	**19660819**

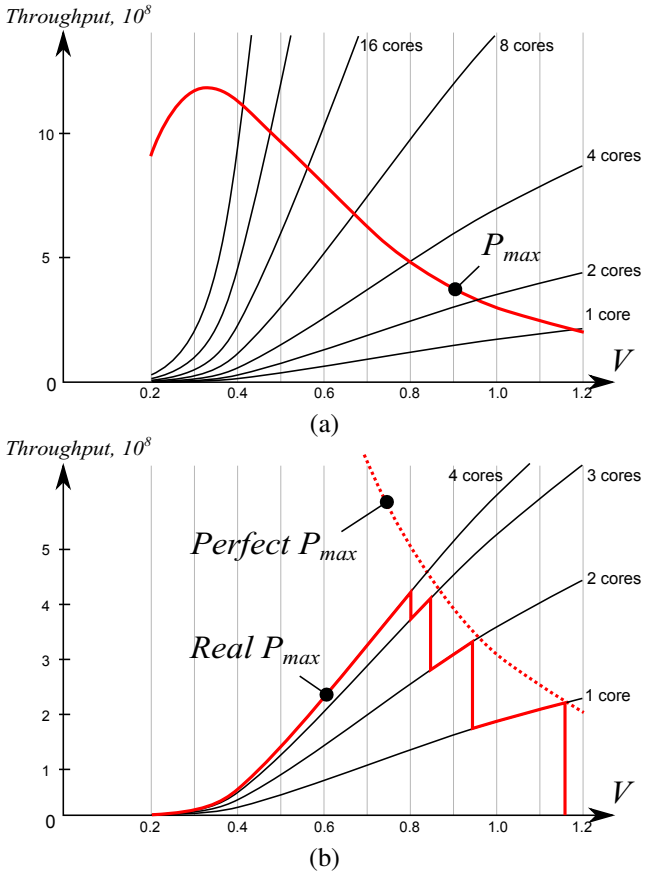

Figure 8. Computed power limit for perfect scaling (a) and for an actual scaling to many cores in the simulated example (b).

the approach. Figure 8(a) illustrates perfect scaling (memory Mode 1) with applied power limit of 2mW. Figure 8(b) shows the PER diagram for the same system with the same power budget after applying actual metrics for scalability to many cores in Mode 2. The diagram considers only integer numbers of cores, hence the performance line looks jagged. Please note that the line for 4 cores in Figure 8(b) is lower than in Figure 8(a) due to imperfect scaling. The data for this graph is computed automatically. One can see that the performance cap is clearly reflected in the power limit.

Such diagrams can be used in a runtime management system in order to predict the best voltage and the number of cores for a particular software with regard to the power restrictions. In our example, for memory Mode 2, if the system is limited to 2mW, the best number of cores is 4 running at 0.8V. Of course, if we consider a general purpose processor in a wide

range of devices, it is not possible to build scaling profiles for all variety of applications they can run. The next step for our research is to build-in an adaptation cycle, then model and simulate it.

VII. Conclusion

The PER interplay relations are studied in a parallelisation scaling context, yielding models which can be incorporated into the ArchOn modelling framework.

The ArchOn method is developed to help designers of complex systems. Its unique resource-graph based approach represents the first known attempt at layer-crossing friendliness by being explicitly and implicitly layer- and level-agnostic. Fundamentally, resource dependencies are represented as graphs, and in any one such graph, resources could be diverse elements including components at all levels of detail and from all different abstraction layers. Resources could also include items outside of hardware and software components, such as power, energy, reliability, time available, thermal budget, etc. This allows large design teams of experts from different disciplines to both concentrate on detailed problems explicitly and reason about inter-related issues at more abstract levels.

Being a graph-based model, ArchOn presents users with a friendly interface for understanding. Its usability in system simulation is demonstrated through a number of case study examples in this paper. The first software implementation of the ArchOn method, a simulator with PER model capabilities, has been developed and demonstrated at ES4CPS '14 workshop and further developments of the method including tool support is ongoing.

The development of the method is at an initial exploratory stage. Future topics of investigation include integration with tools like gem5 – a popular processor simulation platform [2]. Since ArchOn's foundation matches the traditional philosophy comfortable for discrete event system designers, we would also like to exploit potential mappings onto the problem and solution spaces of existing modelling methods such as CPOG's [22] and Petri nets

Acknowledgement: This work is supported by EPSRC research grant EP/K034448/1.The authors would like to thank Maxim Rykunov and Danil Sokolov for their valuable help in conducting hardware simulations and experiments. Parts of Sections II and III have been previously reported in [24], [33].

References

[1] ARM. http://www.arm.com.
[2] The gem5 simulator system. http://www.m5sim.org.
[3] The PRiME project. http://www.prime-project.org.
[4] E. Alon et al. Circuits and techniques for high-resolution measurement of on-chip power supply noise. *IEEE JSSC*, 40(4), 2005.
[5] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the Spring Joint Computer Conference*, AFIPS '67 (Spring), pages 483–485. ACM, 1967.
[6] A. Baz, D. Shang, F. Xia, and A. Yakovlev. Self-timed sram for energy harvesting systems. *J. Low Power Electronics*, 7(2):274–284, 2011.
[7] A. Branover, D. Foley, and M. Steinman. Amd fusion apu: Llano. *IEEE Micro*, 32(2):28–37, 2012.
[8] R. P. Brent and H.-T. Kung. A regular layout for parallel adders. *IEEE Transactions on Computers*, 100(3):260–264, 1982.
[9] H. Corporaal. Design of transport triggered architectures. In *Proc. to Design Automation of High Performance VLSI Systems*, pages 130–135, 1994.
[10] S. B. Dhia et al. On-chip noise sensor for integrated circuit susceptibility investigations. *IEEE TIM*, 61(3):696–707, 2012.
[11] A. Ehrenfeucht and G. Rozenberg. Zoom structures and reaction systems yield exploration systems. In *IJFCS, to appear*, 2014.
[12] H. Esmaeilzadeh et al. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 365–376, New York, NY, USA, 2011. ACM.
[13] J. A. Fisher. Very Long Instruction Word architectures and the ELI-512. *SIGARCH Comput. Archit. News*, 11:140–150, June 1983.
[14] S. H. Fuller and L. I. Millett. Computing performance: Game over or next level? *Computer*, 44(1):31–38, 2011.
[15] N. Goulding-Hotta et al. The greendroid mobile application processor: An architecture for silicon's dark future. *IEEE Micro*, 31(2):86–95, 2011.
[16] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Toward dark silicon in servers. *IEEE Micro*, 31(4):6–15, 2011.
[17] J. Kruppa and D. Hesidenz. High speed, high bandwidth on-chip current and voltage sensor. In *Sensors, 2006. 5th IEEE Conference on*, pages 1337–1340, 2006.
[18] H.-Y. McCreary et al. Energyscale for ibm power6 microprocessor-based systems. *IBM Journal of Research and Development*, 51(6):775–786, 2007.
[19] R. McGowen et al. Power and temperature control on a 90-nm itanium family processor. *IEEE JSSC*, 41(1):229–237, 2006.
[20] M. Meterelliyoz, P. Song, F. Stellari, J. P. Kulkarni, and K. Roy. Characterization of random process variations using ultralow-power, high-sensitivity, bias-free sub-threshold process sensor. *Trans. Cir. Sys. Part I*, 57(8):1838–1847, Aug. 2010.
[21] J. N. Mistry. *Leakage power minimisation techniques for embedded processors*. PhD thesis, University of Southampton, 2013.
[22] A. Mokhov and A. Yakovlev. Conditional Partial Order Graphs: Model, Synthesis and Application. *IEEE Transactions on Computers*, 59(11):1480–1493, 2010.
[23] A. Muhtaroglu et al. ., on-die droop detector for analog sensing of power supply noise. *IEEE JSSC*, 39(4):651–660, 2004.
[24] A. Rafiev, A. Iliasov, A. Romanovsky, A. Mokhov, F. Xia, and A. Yakovlev. ArchOn: Architecture-open resource-driven cross-layer modelling framework. Technical Report NCL-EEE-MICRO-TR-2014-184, School of EEE, Newcastle University, January 2014.
[25] R. Ramesani and other. Voltage sensing using an asynchronous charge-to-digital converter for energy-autonomous environments. *IEEE JETCAS*, 3(1):35–44, 2013.
[26] E. Rotem, A. Naveh, M. Moffie, and A. Mendelson. Analysis of thermal monitor features of the intel pentium m processor. In *in Workshop on Temperatureaware Computer Systems*, 2004.
[27] M. Rykunov. *Design of Asynchronous Microprocessor for Power Proportionality*. PhD thesis, Newcastle University, 2013.
[28] M. Rykunov. *Design of Asynchronous Microprocessor for Power Proportionality*. PhD thesis, University of Newcastle upon Tyne, School of Electrical, Electronic and Computer Engineering, 2013.
[29] J. Sparso and S. Furber. *Principles of asynchronous circuit design : a systems perspective*. European Low-Power Initiative for Electronic System Design. Kluwer Academic Publishers, Boston, Dordrecht, London, 2002.
[30] J. Tschanz et al. Tunable replica circuits and adaptive voltage-frequency techniques for dynamic voltage, temperature, and aging variation tolerance. In *Symp. on VLSI Circuits*, 2009.
[31] J. Van Praet, G. Goossens, D. Lanneer, and H. De Man. Instruction set definition and instruction selection for ASIPs. In *Proc. of the Int'l Symposium on High-Level Synthesis*, pages 11–16, 1994.
[32] C. S. Wallace. A suggestion for a fast multiplier. *IEEE Transactions on Electronic Computers*, (1):14–17, 1964.
[33] F. Xia, A. Mokhov, A. Yakovlev, A. Iliasov, A. Rafiev, and A. Romanovsky. Adaptive resource control in multi-core systems. Technical Report NCL-EEE-MICRO-TR-2013-183, School of EEE, Newcastle University, December 2013.