Towards Power Elastic Systems through Concurrency Management

Fei Xia[†], Andrey Mokhov[†], Yu Zhou[†], Yuan Chen[‡],
Isi Mitrani[†], Delong Shang[†], Danil Sokolov[†], Alex Yakovlev[†]

[†] Newcastle University, UK

[‡] China Academy of Railway Sciences

Abstract

This paper describes power elastic systems, a method for designing systems whose operations are limited by applicable power. Departing from the traditional low power design approach which minimises the power consumption for given amounts of computation throughput, power elastic design focuses on the maximally effective use of applicable power. Centred on a run-time feedback control architecture, power elastic systems manage their computation loads according to applicable power constraints, effectively viewing quantities of power as resources to be distributed among units of computation. Concurrency management is demonstrated as an effective means of implementing such run-time control, through both theoretical and numerical investigations. Several concurrency management techniques are studied and the effectiveness of arbitration for dynamic concurrency management with minimal prior system knowledge is demonstrated. A new type of arbitration, called soft arbitration, particularly suitable for managing the access of flexible resources such as power, is developed and proved.

1 Introduction

Microelectronic system design is becoming more energy conscious, because of limited energy supply (scavenged energy or low battery) and excessive heat with associated thermal stress and device wear-out. At the same time, the high density of devices per die and potentially high parallelism, coupled with environmental variations, create almost permanent instability in voltage supply (cf. Vdd droop), making systems highly power variant. Conventional **low power design** was targeted merely at the reduction of capacitance, Vdd and switching activity, whilst maintaining the required system performance. In many current applications, the design objectives are changing to maximising the performance within the dynamic power constraints

from energy supply and consumption regimes. Such systems can no longer be simply regarded as low power systems, but rather as power-adaptive or power-resilient systems.

Under varying environmental conditions, with voltage and thermal fluctuations, timing tends to be the first issue affected. Most systems are still designed with global clocking and are overly pessimistic to avoid failures due to timing variations. To reduce these margins designers now consciously allow parts of systems to fail, albeit rarely, to maintain the overall balance between performance gains and reasonably low error rate [1][2]. Elsewhere designers are moving towards **timing elasticity** and a wider use of asynchronous design methods. It has been shown that the latter, materialised into 'elastic voltage scaling', can lead to 30-40% average power savings under the same level of performance [3]. The former technique suits CPU pipelines but the latter seems to be more universal and appropriate for more heterogeneous systems such as SoCs and 3D die-stacks. This trend is set to continue in a widening scope of embedded applications and multi-core and heterogeneous systems. These methods are generally based on the assumption of relatively rigid energy supply levels. Computations tend to be scheduled based on a prior knowledge of the energy requirements, with Vdd droops accommodated through reliable operation. However, the notion of elasticity can be taken further than simply stretching delays to accommodate varying conditions.

We propose to investigate elasticity in terms of energy supply and consumption. The ultimate goal is to design systems in such a way that, while preserving behavioural equivalence in computation, the execution can be altered to meet the energy mode requirements. This concept of systems being limited by applicable power and designing systems according to such limitations (called **power elastic** design in this paper) is different from conventional low power design [4]. This is becoming more widely recognised, e.g.:

> "Systems tend to be designed and optimized for peak performance. In reality, most computation nodes, networks and storage devices typically operate at a fraction of the maximum load, and do this with surprisingly low energy efficiency. If we could design systems that do nothing well (as phrased by David Culler), major energy savings would be enabled. Accomplishing **energy-proportional computing** requires a full-fledged top-down and bottom-up approach to the design of IT systems."
>
> (*from Jan Rabaey's lecture 'The Art of Green Design: Doing Nothing Well' – March 2010*)

We believe that this problem cannot be solved in its entirety without introducing a measure of energy (or power) into the system design abstraction, for example in the form of quantized resources. We also believe that this can be done very elegantly within computation behavioural models based on, for instance, Petri nets. Given that there exist powerful methods for the analysis and synthesis of Petri nets, as well as their mapping into logic circuits, the overall prospects of achieving an algorithmic and potentially automated way

of obtaining efficient power controls and their hardware implementations are realistic. This overall discipline of designing systems with dynamic power allocation is the essence of power elasticity.

In this context, traditional front-line power saving techniques [5] such as **clock gating**, **power gating** and **voltage/frequency scaling** can be regarded as means of providing multiple discrete operation modes for parts of a system with various degrees of power consumption and performance. They can then be employed as the **actuators** in a power-elastic control system which dynamically varies the computation load to maximally exploit applicable power. The clock and power gating techniques can be directly used to manage system concurrency, where individual units of computation may be scheduled in or out depending on power conditions.

In this paper we develop an approach to power elasticity suitable for deriving simple and low-cost hardware for fast response control of energy use, through the management of system concurrency. This complements the existing concept of timing elasticity based on dynamic adjustment of computational delays, also at the fine grain level, using asynchronous logic techniques [6][7]. Together they pave the way for designing systems with fine granularity of power and timing control, and thereby being significantly more robust and better optimized to the operational conditions in a wide variety of (mostly embedded) applications.

1.1 Contributions and organisation of this paper

In this paper we propose the power elastic view on system design and develop specific power elastic design and implementation techniques. These include concurrency management modelling, analysis and design as well as soft arbitration. The rest of this paper is organised as follows. In Section 2 we review existing methods of power-driven scheduling and holistic design methods of systems where power is not the limiting factor. On this baseline we clarify the main aim of power-elastic design. In Section 3 we will describe our automatic power control regime based on power profiling and feedback control concepts. In Section 4 we will model and analyse system power and latency behaviour relating to the degree of concurrency. In Section 5 we will describe investigations of power elastic design based on concurrency reduction and soft arbitration techniques; discussions and future work vision will conclude the paper in Section 6.

2 Related work

Scheduling computation activities based on power availability has been studied before. For instance, more tasks may be scheduled when more power becomes available [4]. Previous studies, however, tend to focus on specific applications with many system factors known to the designer prior to the design. This allows them to concentrate on developing targeted scheduling algorithms with power as the only variable. Dynamically

tuning the computation in run-time based on power and computation feedbacks with small controller hardware and power overhead is not a widely studied subject.

Compared to this, mature holistic design methods exist for systems with other constraining factors than power. For instance, in real-time design, where systems are bounded by real-time requirements, many methodologies have been used in industry [8][9][10][11]. These design methodologies support the functional specification, synthesis and implementation of complete embedded systems and their operational control with architectures, communication protocols, computation components, real-time operating systems, etc. The normal approach to power considerations in these systems is to make use of low power techniques such as clock/power gating or voltage/frequency scaling where required in an ad-hoc manner, either during system design of after the functional design has been completed. In other words, power can be considered a parameter to be optimised after real-time constraints are met.

Power elastic design ultimately aims to develop a holistic design method for systems whose operations are limited by applicable power, where power and energy are the central factor in system specification, synthesis and operation. The targeted types of systems include embedded systems which rely on unpredictable power sources such as energy harvesters, where the computation may be variable in execution time and not pre-determined at design time like in hard real-time systems. In these systems, other parameters such as throughput, deadlines and so on, may be considered for optimisation after power constraints are met.

3 Power elastic basics

Here we explore a feedback control strategy with the applicable power as the main constraint. The aim is to derive and implement at low cost an appropriate power elastic control law for any given system.

3.1 Power profiling

Applicable power is the quantity of power that can be applied, determined by two factors. One is the availability of power from energy source(s), especially important with variable and non-deterministic sources such as batteries and scavengers. The second factor is other limitations on power application such as the operating temperature. Applicable power can be characterised as the upper bound of power as a function depending on time and space:

$$B_p = B_p(x,\ y,\ z,\ t) = B_p(S,\ t)$$

where x, y, and z are the 3-dimensional indexes of location which can be unified into a general space index S and t is time. The space factor may represent that different parts of a chip may have different temperature characteristics. In on-chip VLSI, the space factor is not continuous. In general, chips are divided into discrete areas or blocks. There is always a lower bound for block size. This block size lower bound and finite chip size imply an upper bound for the number of blocks. The space factor can then be simplified to an integer index:

$$B_p = B_p(i,\ t)$$

is the upper bound of applicable power for the i-th block.

Existing power control mechanisms in general implement coarse grain power manipulation through a limited set of operating modes. Switching among these modes is not usually applied very frequently in time because of mode switching overheads. In other words, both the values of t and B_p are also discrete. Therefore power control through switching among multiple power modes is a **discrete event system**. Figure 1(a) illustrates $B_p(i,\ t)$ as continuous and discrete concepts.

A description of applicable power in time and space like Figure 1(a) is known as a **power profile**. Many techniques can be used to obtain power profiles. These can be static methods including energy source and computation intensity predictions, or dynamic ones based on sensor data in run-time.

3.2 Architecture for power elastic circuits

Figure 1(b) depicts the generic architecture for power-elastic systems. Its idea is to extend a system with a discrete event Power Elastic Controller (PEC) which ensures that power consumption is kept within a given profile by, for example, reducing the concurrency of the system whist preserving behavioural equivalence. Figure 2 shows a more detailed synopsis of the PEC's working environment. The PEC decides which **computation units**[1] should operate to maintain the required power consumption and regulates the clock gating, power gating and voltage scaling interfaces. This decision is made based on a set of power consumption rules, data from the sensors, such as temperature, delay difference, deadlines, switching activity, etc. The PEC makes its decisions according to a concrete operation model, which is derived from the system functional specification (the **core model of computation**) and the **power model of components**.

[1] Units of computation in this context mean atomic hardware or software components that can be individually controlled, e.g. blocks and threads.

3.3 Power elastic transformation

Synthesising the PEC is of vital importance to power elastic system design. This synthesis process should take characterisation input from the core computation and power models of the system. Here the power model describes system power profiles and consumption properties, and the core computation model is the specification of the functional behaviour of the controlled system, concentrating on the control path. With these inputs, a power elastic transformation finds a concrete implementation for the computation control model. This is then applied through the PEC, resulting in a sequence of execution which satisfies the power profiles and preserves computational equivalence (Figure 2).

A unified method of modelling is needed for systematic power elastic transformations. Petri nets have been used to represent discrete event systems for their analysis and synthesis for a long time [12][13][14][15]. Petri net models can be used to directly represent such issues as causality, concurrency and synchronisation. The flow relations in Petri nets can be used to represent the relations between such computation elements as tasks, including their relative concurrency and cross-dependencies. The execution semantics [14][16] readily derivable from a Petri net core computation model can be used in the process of a power elastic transformation which preserves equivalences. In power models, power profiles can be represented by quantising $B_p(i,\ t)$ into the number of **power tokens** in a **power place**. This concept is demonstrated in the following sections.

Petri net modelling of discrete event control and asynchronous circuits allows PECs to be implemented from a collection of small circuits distributed spatially within a block to reduce operational cost and communication bandwidth needs. Generic methods of direct mapping of Petri nets to circuits [7] facilitate this spatial distribution of the PEC.

The process of power elastic transformation can be either static (design time), where a PEC is synthesised once for a system, or dynamic (run-time), where the PEC is tuned during operation, or hybrid, where the PEC synthesis will have both dynamic and static elements. Here we present relevant and useful techniques for all these choices.

4 Relating power to concurrency

Increasing concurrency can sometimes be used to reduce power consumption if one could reduce Vdd and/or clock frequency at the same time [5]. However, for ultra-low average power operations it is generally better to 'run fast then sleep' because of leakage issues [17]. In this context, power may become the limiting factor on run-time concurrency. The degree of run-time concurrency may in turn be used to control system power consumption to fit power profiles, e.g. in [18] the authors implement power management by the adaptive control of pipeline depth.

In this section we develop a general modelling approach where the system degree of concurrency is quantitatively related to power and latency performance. Such a modelling method supports qualitative and quantitative analysis of power-elastic systems for designers who may want to compare different power management algorithms under different operational situations.

Here we assume a system consisting of service providers (SPs) dealing with incoming service requests (SRs). A **task** is executed in an SP (usually a computation unit) to serve an SR and the SP only consumes power when it contains at least one **active** task.

Markovian methods have been used to model such systems for decades [19][20][21] when multiple tasks are provided by an SP to deal with non-deterministic SR arrivals. Independence is assumed across requests and tasks. Normally, λ denotes the rate of request arrival, μ stands for the rate of task completion, and P represents the power consumption when an SP is on. Because this kind of modelling relates the average power consumption P_{ave} and latency L of hardware/software designs to parameters λ, μ, and P, it can help derive optimal power-latency design tradeoffs.

Much research, such as discrete-time [20] and continuous-time Markov processes [21] and fine-grain Markov models [19] exist for such modelling and analysis covering only single-SP cases. Here we consider the multi-SP case and investigate the optimised or permitted concurrency degree for a certain power-latency consideration.

For simplicity and following existing practice [20][21], we assume that the system being controlled consists of identical SPs which can be independently woken up or shut down and a number of them can be run concurrently[2]. When an SR arrives, the corresponding task in an SP is activated to service the SR. Given the applicable power profile, at most M $(1 \leq M \leq N)$ SPs can be on at the same time – the **maximum applicable concurrency degree** is M or there are M power tokens in the system. In other words, M power tokens are available from $B_p(i,\ t)$. These assumptions mean that we are concentrating on the simplest (Boolean) form of power control of the computation units.

Assume there are N independent tasks in the system. Because of task independence, multiple tasks can be active (including waiting) at the same time. We use j $(0 \leq j \leq N)$ to indicate the number of idle tasks (tasks whose corresponding SRs have not arrived) in the system. Without losing generality we assume the system has N SPs available. Thus the **maximum degree of concurrency** is N. With unlimited power and unlimited number of SRs the system can run N concurrent tasks at the same time $(M = N)$.

The concept of concurrency being limited by power profile is illustrated in Figure 3(a). The tasks within a logically atomic computation step could be executed fully concurrently given enough resources. However, when a resource, like power, is limited, tasks can be executed sequentially to trade latency for the resource.

[2]The assumptions made in this section do not affect generality of the modelling. Systems with differences in SPs, tasks, power and delay modes will result in more complex models but the basic modelling technique stays the same.

In this example $N = 5$ and $M = 2$.

If no more than M tasks are active ($N - j < M$), only $N - j$ SPs are needed for task execution. The other $M - N + j$ SPs can be powered off for power saving. After the completion of execution, a task becomes idled again. If at least M tasks are active ($N - j \geq M$), the system must operate at the maximum applicable degree of concurrency. However, the other $N - j - M$ tasks have to wait in a queue until some SPs have been released on task completion. As shown in Figure 3(b), power elasticity through concurrency control means modulating execution concurrency with applicable power.

4.1 Modelling of the degree of concurrency

Figure 4(a) is the stochastic model for the type of system investigated in this work. In this model, the number of idle tasks is the state variable. For example, in the state N, all tasks are idle and all SPs are powered off. Since the system moves from the state N to $N-1$ when any one of the N tasks is activated, the corresponding transfer rate is $N\lambda$ (each idle task leaves the idle state at the same rate λ). A task is executed in an SP with the completion rate μ (a task in execution leaves the execution/active state and becomes idle at the rate of μ). Both the power on and off mode switches for an SP are taken as **cost free** in both time and power (with a rate of infinity and delay of zero).

If one of the other $N - 1$ tasks becomes active before the execution of the first active task is completed, the system moves to the state $N - 2$, and another SP is powered on. With two tasks being executed, the rate of one of them leaving execution and becoming idle is 2μ. When the system is in the state $j = i$ ($N - M < i < N$), there are $N - i$ active tasks being executed, and it may move to the state $i - 1$ with the transfer rate $i \times \lambda$. With $N - i$ SPs on for processing in the state $j = i$, the execution rate becomes $(N - i)\mu$.

When the system is in the state $j = i$ ($i \leq N - M$), all M SPs are already on, and the corresponding transfer rate from the state $j = i$ to $j = i + 1$ is constant $M\mu$.

4.2 Power, latency and combined analysis

To differentiate power from probability distribution, we use Q_j to stand for the probability when the system is in state j ($j \leq N$), and P as one SP's power consumption. In this high level model, we simply assume that an SP consumes full power P when it is on and zero power when it is off. Therefore, the power dissipation when the system in the state $j = i$ is $(N - i)P$ when $N - M < i < N$ or MP when $i \leq N - M$. The average power consumption of the system $P_{ave}(M)$ is presented in (1).

$$P_{ave}(M) = P(M \sum_{i=0}^{N-M} Q_i + (M-1)Q_{N-M+1} + \cdots + 2Q_{N-2} + Q_{N-1})$$

$$= P(M \sum_{i=0}^{N-M} Q_i + \sum_{k=1}^{M-1} kQ_{N-k}) \quad (1)$$

where the probabilities can be expressed in the rates of the system (λ and μ) by first expressing all Q_i in terms of Q_0:

$$Q_i = \begin{cases} \frac{(M\mu)^i}{i!\lambda^i} Q_0 & 0 \leq i < N - M \\ \frac{M^{N-M}\mu^i \prod_{k=1}^{i-N-M} (M-k)}{i!\lambda^i} Q_0 & N - M < i \leq N \end{cases}$$

then finding Q_0 by setting the sum of all Q_i to 1:

$$\sum_{i=0}^{N} Q_i = 1$$

For the measure of latency, we use W, the average time spent by a task in both waiting and execution stages, i.e. between activation and becoming idle again. It can be derived as follows. First, if $L = Ave(j)$ is the average number of idle tasks, it can be calculated using (2):

$$L = \sum_{i=1}^{N} iQ_i \quad (2)$$

The average number of active tasks is then $N - L$. Meanwhile, the arrival rate into active states is given by λL. The average latency $W(M)$ is thus described in (3) where T is the average time for executing a single task:

$$W(M) = \frac{N - L}{\lambda L} T \quad (3)$$

If power is not a hard limiting constraint, but a factor that can be balanced with performance, an optimum M may be found for any particular power and latency balance. This type of optimisation can be described as follows.

Given a certain weight C ($0 \leq C \leq 1$), for any possible concurrency degree M ($1 \leq M \leq N$), the one which can minimise $CP_{ave}(M) + (1 - C)W(M)$ is the optimised M (M_{opt}). In other words, M_{opt} satisfies:

$$M_{opt} = \arg\min_{M} \; CP_{ave}(M) + (1 - C)W(M) \quad (4)$$

This kind of analysis can be done at design time for power-elastic systems so that at run-time, power profile permitting, the concurrency degree can be set close to M_{opt}.

Here we use an example with $N = 15$ and normalised P, T and μ of 1 (each SP consumes one unit of power in execution at the completion time and rate of 1) to illustrate the method. Figure 5 describes the

power and latency behaviours for various values of M when $C = 0.3$. In general, with larger M the power consumption is higher and the latency is lower. When balancing these two, it is possible to find some optimal M for some weighting factor C. When the system is saturated with high λ values, the power consumption is asymptotically MP.

4.3 Uncertainty in M

It is evidently possible to control the power consumption of power elastic systems to fit power profile requirements $B_p(i,\ t)$ by adjusting the system degree of concurrency M. When a system is operating under very limited power which does not allow a high degree of concurrency and M is relatively small, it is usually quite straightforward to implement concurrency managers with simple arbitration. Hardware arbiters requiring little operating power can be designed to manage a small number of resources. For such simple arbitration cases the models above are sufficient for analysis.

With large M however the arbiters can become complex with performance and cost penalties. Fortunately, unlike hard enumerable resources such as software threads and hardware components, power allows a degree of **softness** in arbitration. Power tokens are discretised from an ultimately analogue value and a degree of imprecision in M (occasionally allowing more than M SPs to run at the same time) may be tolerable, causing a slowdown but not a catastrophic failure. This permits the use of **soft arbiters** which are much cheaper to implement and run [22].

However, whether this kind of intuitive reasoning will be applicable for any system needs to be investigated and analysed at design time. Here we extend the model to cover softness in M.

Figure 6 includes representation for cases where, at a probability of $1 - \alpha$, soft arbitration allows $M + 1$ SPs to execute at the same time. The *-marked branch can have its own rate of execution (μ^*) and power cost (P^*) because of potential Vdd droop in such cases. The average power consumption can be calculated as:

$$P_{ave} = \sum_{j=0}^{N-M-1} P^*(M+1)Q_{j^*} + \sum_{j=0}^{N-M-1} PMQ_j + \sum_{j=N-M}^{N} P(N-j)Q_j \tag{5}$$

In (5), we use Q_j and Q_{j^*} to represent the probabilities when the system is in the states j or j^*. Latency behaviour can be similarly derived.

Extending the model with μ rates and P quantities not as constants but as functions of the number of running SPs can better reflect the effect on latency and power by allowing different degrees of power token overflow. Such functions can be established with design-time analysis. These models can also be extended to cover the cases where the wakeup and shutdown transitions are not overhead-free [23], e.g. the model in Figure 4(a) can be extended with the elements in Figures 4(b,c). Such extensions increase the number of

states in a model and may result in the loss of closed form analytic solutions, however, numerical solutions can still be obtained. More detailed explanations can be found in [23].

With these types of analysis, system designers can discover such operating conditions and incorporate allowances for them when designing power control algorithms. High-level design time explorations using the modelling and analysis techniques presented in this section should ultimately help reduce the design effort.

These modelling and analysis methods can be generalised further to cover more complex and non-Boolean power control modes such as DVS and DFS. This may however make closed-form solutions impossible, although numerical analysis will still be straightforward.

Most of this section is based on [23].

5 Power elasticity through concurrency management

Here we further investigate power elastic techniques based on concurrency reduction and the simplest form of power control – Boolean power modes for computation units. Generalising these techniques to more complex operation modes is a subject for future exploration.

5.1 Concurrency relations and concurrency reduction

As a concurrency reduction example, Figure 7(a) depicts a Petri net core computation model consisting of three concurrent tasks a, b, and c. Each task involves the sequential execution of two subtasks, e.g. $a.1$ and $a.2$ for task a. Cross-dependency relations exist between $(a.1,\ b.2)$ and $(b.1,\ c.2)$. Suppose that applicable power is quantised into two power tokens, and the execution of a task requires one power token. The consumption of power, different from energy consumption, is a temporary occupation of a resource. Once the task is completed the power token is recycled to the system. The concrete control model in Figure 7(b) illustrates dynamic scheduling based on arbitration, whereby at most two tasks can be scheduled simultaneously during run-time.

The behaviour semantics of a Petri net can be described by its **Reachability Graph** (RG): $RG = (S,\ T,\ F,\ M_0)$. S is the set of all possible markings of the net; T is the set of transitions when considering both **interleaving** and **step** firing semantics; F are the transition (or next-state) functions of $s' = f(s,\ t)$ where $s',\ s \in S$ and $t \in T$; and M_0 is the initial marking of the net. Figure 7(c) shows the RG of the example net. The marking $\{1,\ 5,\ 11,\ 6\}$ is reachable from the initial marking $\{1,\ 2,\ 3\}$, following the interleaving transition sequences of $(b.1,\ c.1)$ or $(c.1,\ b.1)$, or a step transition of $\{b.1,\ c.1\}$. Other 'step' arcs are not explicitly shown to reduce clutter.

For two transitions t_1 and t_2, $t_1 < t_2$ if t_1 precedes t_2 in every transition sequence of the RG. An N-ary

concurrency relation upon T is defined as the set of N-tuples, where for each tuple, $\neg(t_1 < t_2)$ holds for every pair of tuple elements t_1 and t_2 ($t_1 \neq t_2$). In this example the highest number of concurrent tasks is three, hence $N = 3$.

An N-ary concurrency relation recursively implies M-ary concurrency relations for all $M \leq N$, e.g. the ternary relation tuple ($a.1,\ b.1,\ c.1$) implies three binary tuples ($a.1,\ b.1$), ($a.1,\ c.1$), and ($b.1,\ c.1$). Our example contains 4 ternary and 9 binary relations, as listed in Table 1. The generalisation of this property is only true in both directions for a subclass of systems with distributive concurrency [24].

Concurrency reduction means the removal of a subset of the N-tuples from an N-ary concurrency relation. Removing a tuple eliminates all its supertuples, e.g. removing ($b.1,\ c.1$) eliminates its parent tuples of ($a.1,\ b.1,\ c.1$) and ($a.2,\ b.1,\ c.1$) in the concurrency relation list. Our task is to derive concurrency reductions in the system with $N = 3$ when only two power tokens ($M = 2$) exist.

Here we consider both static and dynamic control mechanisms. Static control establishes a single partial order over the tuple elements (i.e. execution of tasks) whereas dynamic control permits multiple orders (the actual order is only determined during run-time). Static control is further divided into **super-linear** and **and-causal** cases, whereas dynamic control is divided into **or-causal** and **arbitrating** ones. Figure 8 lists these control structures by Petri net models in reducing the applicable degree of concurrency of ($a.1,\ b.1,\ c.1$) to binary.

Super-linear control imposes a complete order on the tuple elements. With this control, the state cube formed by $a.1$, $b.1$, and $c.1$ in Figure 7(c) is replaced by the local RG in Figure 9(a). As a result, all the local binary concurrency tuples incurred by ($a.1,\ b.1,\ c.1$) are eliminated. This complete sequentialisation is suitable for any situation where there is at least a single power token (i.e. $M > 0$).

And-causal control expresses an AND enabling condition for a task. With Figure 8(b), $c.1$ is enabled when both $a.1$ and $b.1$ have fired. The partial order in this example is {($a.1,\ c.1$), ($b.1,\ c.1$)}, and the corresponding local RG is shown in Figure 9(b). With and-causality, only one local binary concurrency tuple is maintained, i.e., ($a.1,\ b.1$).

Or-causal control [25] expresses an OR enabling condition for a task. With Figure 8(c), $c.1$ is enabled when either $a.1$ or $b.1$ has fired. Or-causality imposes two (mutually exclusive) partial orders: ($a.1,\ c.1$) and ($b.1,\ c.1$). It is only known at run-time which order takes place. The local RG with or-causal control is shown in Figure 9(c), where all local binary concurrency tuples are maintained.

Finally, with the arbitrating control of Figure 8(d) (2-of-3 arbitration), all three tasks are enabled, but at most two of them can be executed simultaneously. Figure 9(d) shows the local RG with all three binary tuples maintained. In addition, all three 'step' arcs corresponding to the tuples are enabled at the initial state, whereas only one 'step' arc is allowed in the or-causal control.

Concurrency reduction causes **performance degradation** because of two factors: the stretched execution time (a direct consequence), and the extra delays incurred by concurrency control (i.e. PEC). The former factor can be determined from the Petri net control structures, whereas the latter is related to the controller implementation.

Suppose the execution delays of tasks $a.1$, $b.1$, and $c.1$ are t_{a1}, t_{b1} and t_{c1}, respectively, while the delays of the super-linear, and-causal, or-causal, and arbitrating controllers are d_{sl}, d_{ac}, d_{oc}, and d_{ab}, respectively.

Before concurrency reduction, the tasks in the example can be executed within a period of $max(t_{a1},\ t_{b1},\ t_{c1})$. With super-linear control, the execution period is $t_{a1} + t_{b1} + t_{c1} + d_{sl}$. With and-causal control, the execution period is $max(t_{a1},\ t_{b1}) + t_{c1} + d_{ac}$. With or-causal control, the execution time is $max(min(t_{a1},\ t_{b1}) + t_{c1},\ max(t_{a1},\ t_{b1})) + d_{oc}$, which can be further refined to $max(t_{a1} + t_{c1},\ t_{b1}) + d_{oc}$ should the run-time order be $(a.1,\ c.1)$, or otherwise to $max(t_{b1} + t_{c1},\ t_{a1}) + d_{oc}$.

Arbitration-based control also has an execution period dependent on run-time decision results. If the imposed partial order turns out to be $(a.1,\ b.1)$ or $(b.1,\ a.1)$ during run-time, the execution time is $max(t_{a1} + t_{b1},\ t_{c1}) + d_{ab}$. Other cases can be similarly derived.

Not considering controller delays, static controls degrade performance more than dynamic controls. Arbitration-based control has a superset of execution paths of or-causal control and can render even lower performance degradation. The control structures, the orders they impose, and the effects on concurrency relations and performance degradation are described in Table 2.

In the context of Figure 2, M would be derived from the power model and direct power place/token modelling exists in the arbitration case. Petri net techniques can be used to derive concrete models for the PEC in the form of Figure 8 from core computation models in the form of Figure 7(a). More details, including discussions on the distribution of concurrency reducing PEC algorithms among small circuits across a block, can be found in [26].

In principle, PEC based on concurrency reduction can make use of all the techniques discussed above, but the actual choice depends on the overhead-performance tradeoff. In the next section we concentrate on further development of the arbitration technique.

5.2 Soft arbiters

The previous section demonstrated that arbitration-based concurrency reduction is simple to design and potentially efficient in operation. However, as current and future on-chip and 3D systems are likely to be highly concurrent, large M and N numbers will prevail in real systems. This could prohibit the use of dynamic concurrency reduction for cost reasons, unless efficient multi-client, multi-resource arbiters are

found. We demonstrated a distributed arbiter architecture for large (10x10) implementations with good scalability in [27], but issues like performance and cost persist.

Power, unlike hard enumerable system resources, can occasionally be shared among more than M computation units without catastrophic failure. This fact may be used to simplify the problem of arbiter design, resulting in cheaper and faster arbiters.

The classical case of 'hard' arbitration is when a system does not have enough **hard resources** to serve more than M clients, e.g. it has only M processing units. However, power resource allows more flexibility: random, occasional events have little effect on the inertial, statistical characteristics of power consumption. Therefore, in this case the bound may be relaxed and granting access to more or less than M clients may be allowed as long as the rate of such imprecise granting is acceptable. We call arbiters with relaxed bounds **soft arbiters**.

Implementations of soft arbiters can in general be smaller and faster than strict ones. For example, the implementation of a 1-of-3 strict arbiter presented in [28] is given in Figure 10. It has a construction with four layers: pair-wise arbitration, reset filters, computation of the winner, and finally completion detection. But if it is occasionally allowed to issue two grants (instead of always at most one) then its implementation can be simplified [22] dramatically as shown in Figure 11.

Let us study the behavior of the simplified arbiter. Table 3 shows the grants issued under different request orders. The arbiter always gives the grant to the first request, plus to request ra if it came second. Thus, for a single burst of incoming requests, the probability for the arbiter to issue two grants is $1/3$. On the other hand, if there is a constant flow of incoming requests *abcabcabc*... then the arbiter will give one and two grants in an alternating fashion, effectively behaving as an 1.5-of-3 soft arbiter on average (the series of grants will be $ga - gb - \{ga,\ gc\} - gb - \{ga,\ gc\} - ...$ etc).

5.3 Run-time control of softness

It is possible to build soft arbiters that have a variable soft bound on the number of issued grants, effectively making them behave like a valve which can restrict the flow of requests to a specified degree (including the scenario when it is 'fully open' or transparent to all the requests). At any moment of time such an arbiter lets a discrete number of requests through but using a feedback control from the PEC it is possible to obtain continuous transmission characteristics.

Figure 12 shows a possible implementation of such an arbiter. It is built on the same principle as the previously described 3-way soft arbiter but instead of using either C-element or OR-gate as a threshold element we use a generic **threshold gate** [29] with some of its inputs controlled by the **softness control**

register (SCR)[3]. The layer of ME-elements provides pairwise ranking of the arrived requests, while the threshold layer decides how many wins a particular request must collect to be granted. For example, if SCR=10101010 then a request has to win against at least two other requests, thereby resulting in a 2-of-4 soft arbiter as demonstrated in Table 4. The arbiter can issue 3 grants only if all the requests arrive simultaneously so that the layer of ME-elements cannot decide between them, producing a random ranking table. There are 64 different ranking tables and only 8 of them have such cyclic ranking that 3 requests are granted. Therefore only in 12.5% of the problematic cases (which are very rare themselves) the arbiter is 3-of-4 and in other 87.5% it is 2-of-4. The fairness of this type of arbiter can be controlled by the SCR, in particular, if SCR=0 the arbiter is bound to be fair due to symmetry. In comparison, the arbiter from Figure 10 is not fair (grant ga has a lower threshold).

The 2-of-4 soft arbiter in Figure 12 is readily scalable to any m-of-n arbitration because the basic architecture is independent of values m and n; the only non-trivial issue is the decomposition of the threshold gates when these values are large. The size of these gates linearly increase with n and SCR length and may bring power and latency overheads.

6 Conclusions

In this paper the power elastic view of system design is proposed. These systems are based on a run-time feedback control architecture around the Power Elastic Controller, where the main technique for fitting a power profile is concurrency management. Methods for modelling, analysis and design of concurrency management systems have been developed and described in this paper. In particular, a promising implementation technique, soft arbitration, has been discussed in detail.

These components and theoretical basis will facilitate the further development of a complete methodology for power elastic design. We are currently further developing and applying these methods in the context of energy harvesting systems.

Acknowledgement

The work has been supported by EPSRC grants – 'Self-Timed Event Processor' (EP/E044662/1) and 'Next Generation Energy-Harvesting Electronics – Holistic approach' (EP/G066728)

[3]The value in SCR should only be changed when the system is idle (no requests are active).

Tables

Ternary concurrency	$(a.1,\ b.1,\ c.1)$, $(a.2,\ b.1,\ c.1)$ $(a.2,\ b.2,\ c.1)$, $(a.2,\ b.2,\ c.2)$
Binary concurrency	$(a.1,\ b.1)$, $(a.1,\ c.1)$, $(b.1,\ c.1)$ $(a.2,\ b.1)$, $(a.2,\ c.1)$, $(a.2,\ b.2)$ $(b.2,\ c.1)$, $(a.2,\ c.2)$, $(b.2,\ c.2)$

Table 1: Concurrency relations in the system of Figure 7

Control Scheme	Structure	Partial Order Imposed	Concurrency Reduction Effects	Performance degradation
super-linear	$M!$ arrangements	single total order	all relations removed	largest
and-causal	AND enabling conditions between $\lceil \frac{N}{M} \rceil$ groups of M-ary subtuples	single partial order between groups	up to M-ary relations maintained but restricted to within a group	second largest
or-causal	OR conditions on a M-ary subtuple to enable the next new tuple element	multiple	all M-ary relations maintained	dynamic, smaller than static controls
arbitrating	M-of-N arbitration	multiple	all M-ary relations maintained	dynamic, more flexible than or-causal control

Table 2: Comparison of different control mechanisms for concurrency reduction

Request order	ab/ba	ac/ca	bc/cb	Issued grant(s)
abc	ab	ac	bc	ga
acb	ab	ac	cb	ga
bac	ba	ac	bc	ga, gb
bca	ba	ca	bc	gb
cab	ab	ca	cb	ga, gc
cba	ba	ca	cb	gc

Table 3: Analysis of 3-way soft arbiter with respect to request orders

Number of wins	Combinations	Number of issued grants
3, 2, 1, 0 (total order)	$4! = 24$ 37.5%	2 (2-of-4 arbitration)
3, 1, 1, 1 (one winner, one 3-cycle)	$4 \times 2 = 8$ 12.5%	1 (1-of-4 arbitration)
2, 2, 2, 0 (one loser, one 3-cycle)	$4 \times 2 = 8$ 12.5%	3 (3-of-4 arbitration)
2, 2, 1, 1 (one 4-cycle)	$4! = 24$ 37.5%	2 (2-of-4 arbitration)
Total	$2^6 = 64$ 100%	2 (average) (2-of-4 soft arbitration)

Table 4: Analysis of 2-of-4 soft arbiter with respect to request orders

Figures

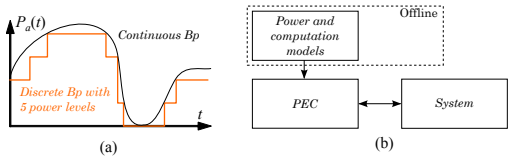

Figure 1: Continuous and discrete power bounds

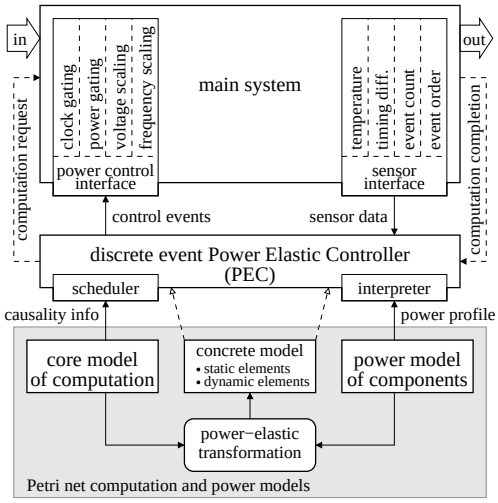

Figure 2: Power elastic architecture

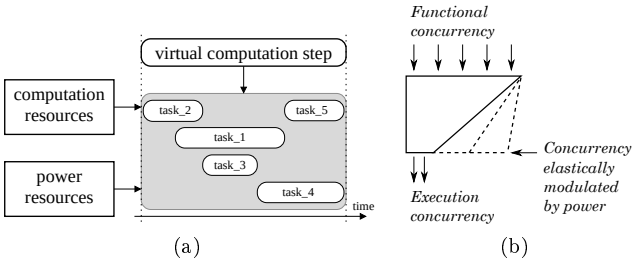

Figure 3: Concurrency reduction

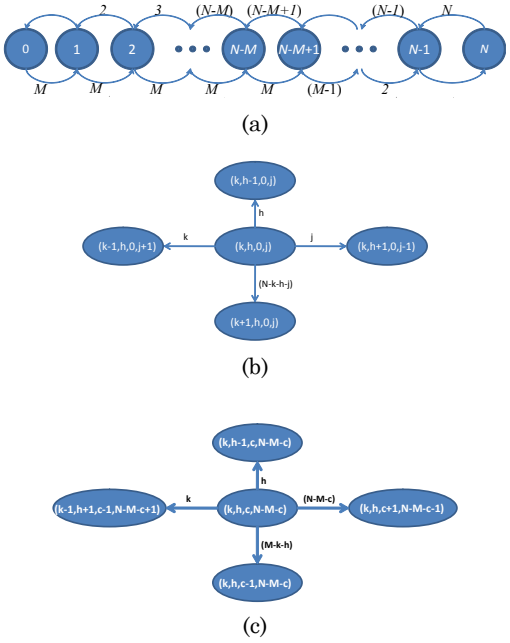

Figure 4: System Markov chain models

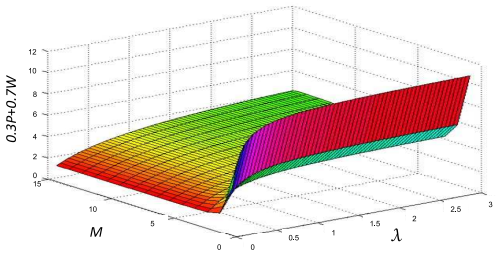

Figure 5: Power and latency for $C = 0.3$

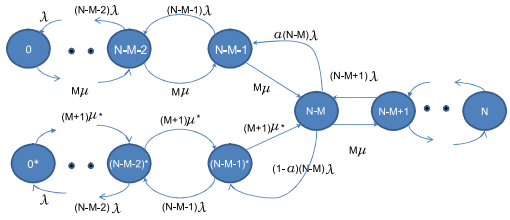

Figure 6: Uncertainty in M

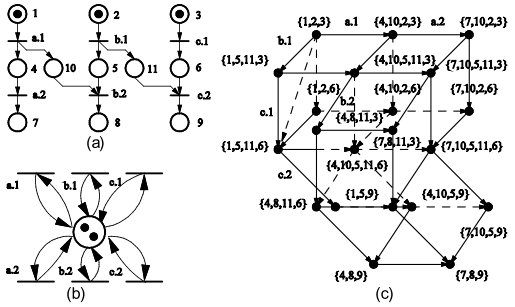

Figure 7: Core PN model (a), its RG (c), and a control model (b)

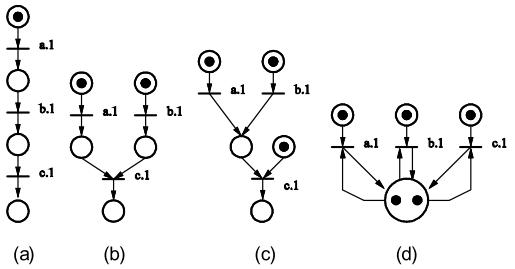

Figure 8: Concrete control models to eliminate ($a.1$, $b.1$, $c.1$): super-linear (a), and-causality (b), or-causality (c), and arbitrating (d)

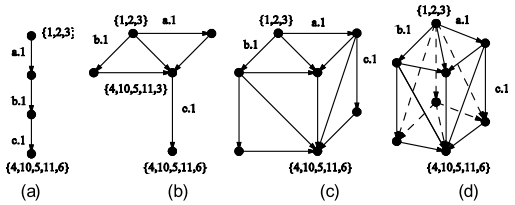

Figure 9: Local RGs after concurrency reduction: super-linear control (a), and-causality (b), or-causality (c), and arbitration (d)

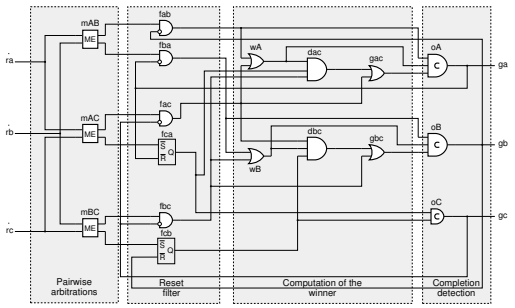

Figure 10: Strict 1-of-3 arbiter

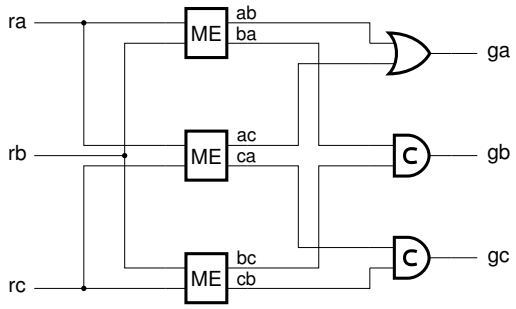

Figure 11: Soft 1-of-3 arbiter

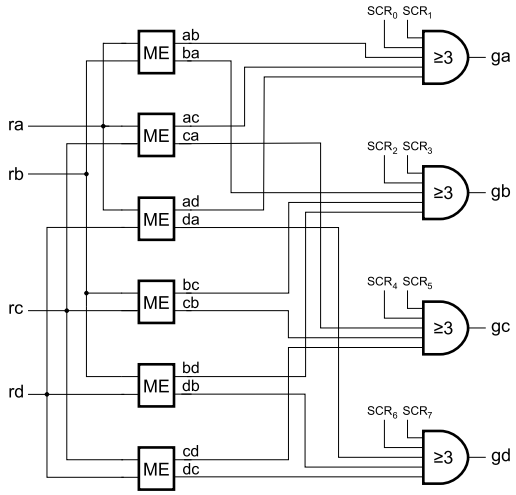

Figure 12: 2-of-4 soft arbiter with run-time control of softness

References

[1] K. BOWMAN, J. TSCHANZ, C. WILKERSON, S. LU, T. KARNAK, V. DE, and S. BORKAR. Circuit techniques for dynamic variation tolerance. In *DAC '09: Proceedings of the 46th Annual Design Automation Conference*, pages 4–7. ACM, 2009.

[2] S. DAS et al. Razor II: In Situ Error Detection and Correction for PVT and SER Tolerance. *IEEE J. Solid-State Circuits*, pages 32–48, 2009.

[3] E. TUNCER, J. CORTADELLA, and L. LAVAGNO. Enabling adaptability through elastic clocks. In *DAC '09: Proceedings of the 46th Annual Design Automation Conference*, pages 8–10. ACM, 2009.

[4] J. LIU, P. H. CHOU, N. BAGHERZADEH, and F. KURDAHI. Power-aware scheduling under timing constraints for mission-critical embedded systems. In *DAC '01: Proceedings of the 38th Annual Design Automation Conference*, 2001.

[5] S. HENZLER. *Power Management of Digital Circuits in Deep Sub-Micron CMOS Technologies (Springer Series in Advanced Microelectronics)*. Springer-Verlag New York, Inc., 2006.

[6] J. SPARSØ and S. FURBER. *Principles of Asynchronous Circuit Design: A Systems Perspective*. Kluwer Academic Publishers, 2001. ISBN: 0792376137.

[7] D. SOKOLOV and A. YAKOVLEV. Clock-less circuits and system synthesis. In *IEE Proceedings, Computers and Digital Techniques*, volume 152, pages 298–316, 2005. ISBN: 1350-2387.

[8] H.R. SIMPSON. Protocols for process interaction. *Computers and Digital Techniques, IEE Proceedings*, 150(3):157 – 182, May 2003.

[9] H.R. SIMPSON and E.R. CAMPBELL. Data interaction architecture for real-time embedded multi processors. international patent application wo 91/16681., November 1991.

[10] Altreonic OpenComRTOS homepage. http://www.altreonic.com/taxonomy/term/2.

[11] Eric VERHULST and Gjalt G. de JONG. OpenComRTOS: An Ultra-Small Network Centric Embedded RTOS Designed Using Formal Modeling. In *SDL Forum*, pages 258–271, 2007.

[12] J. L. PETERSON. *Petri Net Theory and the Modeling of Systems*. Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1981.

[13] A. BENVENISTE, E. FABRE, C. JARD, and S. HAAR. Diagnosis of asynchronous discrete event systems, a net unfolding approach. *IEEE Trans. on Automatic Control*, 48:714–727, 2001.

[14] P. DARONDEAU, M. KOUTNY, M. PIETKIEWICZ-KOUTNY, and A. YAKOVLEV. Synthesis of nets with step firing policies. *Fundam. Inf.*, 94(3-4):275–303, 2009.

[15] B. CAILLAUD, P. DARONDEAU, L. LAVAGNO, and X. XIE. *Synthesis and Control of Discrete Event Systems*. Springer, 2002.

[16] E. BEST and M. KOUTNY. Petri net semantics of priority systems. In *Selected papers of the Second Workshop on Concurrency and compositionality*, pages 175–215. Elsevier Science Publishers Ltd., 1992.

[17] M. ALIOTO. Understanding dc behavior of subthreshold cmos logic through closed-form analysis. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 2010.

[18] A. EFTHYMIOU and J. D. GARSIDE. Adaptive pipeline depth control for processor power-management. In *In Proceedings of International Conference on Computer Design*, pages 454–457. IEEE Computer Society Press, 2002.

[19] Y. CHEN, F. XIA, D. SHANG, and A. YAKOVLEV. Fine-grain stochastic modelling of dynamic power management policies and analysis of their power - latency tradeoffs. *Software, IET*, 3(6):458 –469, december 2009.

[20] L. BENINI, A. BOGLIOLO, G. A. PALEOLOGO, and G. MICHELI. Policy optimization for dynamic power management. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 18(6):813–833, 1999.

[21] Q. QIU and M. PEDRAM. Dynamic power management based on continuous-time markov decision processes. In *DAC '99: Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, pages 555–561. ACM, 1999.

[22] A. MOKHOV and A. YAKOVLEV. Soft arbiters. Technical report, Newcastle University, 2009.

[23] Y. CHEN, I. MITRANI, D. SHANG, F. XIA, and A. YAKOVLEV. Stochastic analysis of power, latency and the degree of concurrency. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2010.

[24] L. ROSENBLUM, A. YAKOVLEV, and V. YAKOVLEV. A look at concurrency semantics through "lattice glasses".

[25] A. YAKOVLEV, M. KISHINEVSKY, A. KONDRATYEV, L. LAVAGNO, and M. PIETKIEWICZ-KOUTNY. On the models for asynchronous circuit behaviour with OR causality. *Formal Methods in System Design*, pages 189–234, 1996.

[26] Y. ZHOU and A. YAKOVLEV. Dynamic concurrency reduction methods for power management using petri-nets. Technical report, Newcastle University, May 2010.

[27] D. SHANG, F. XIA, S. GOLUBCOVS, and A. YAKOVLEV. The magic rule of tiles: Virtual delay insensitivity. In *PATMOS*, pages 286–296, 2009.

[28] A. MOKHOV, V. KHOMENKO, and A. YAKOVLEV. Flat arbiters. In *Proc. of 9th International Conference on Applicatioon of Concurrency to System Design (ACSD'09)*, pages 99–108, 2009.

[29] V. BEIU, J. QUINTANA, and M. AVEDILLO. VLSI implementations of threshold logic – a comprehensive survey. *IEEE Transactions on Neural Networks*, 14(5):1217–1243, 2003.