School of Electrical, Electronic & Computer Engineering

Dynamic Concurrency Reduction Methods for Power Management using Petri-nets

Yu Zhou and Alex Yakovlev

Technical Report Series

NCL-EECE-MSD-TR-2010-153

May 2010

Contact:

yu.zhou@ncl.ac.uk

alex.yakovlev@ncl.ac.uk

EPSRC supports this work via EP/D053064/1

NCL-EECE-MSD-TR-2010-153

School of Electrical, Electronic & Computer Engineering,
Merz Court,
University of Newcastle upon Tyne,
Newcastle upon Tyne, NE1 7RU, UK

`http://async.org.uk/`

Dynamic Concurrency Reduction Methods for Power Management using Petri-nets

Yu Zhou and Alex Yakovlev

Abstract

Power consumption and heat radiation form the main limiting factors to high-performance microprocessors with multi-core architectures. Power management policies are the control procedures to adjust the per-core performances according to a power budget. When there is a power limitation, the cores will slow down or even shut off, reflecting the reduction of a system's concurrency relations, which are defined in this paper. Furthermore, different static and dynamic control mechanisms are proposed for concurrency reduction, with the analysis of their influences on the system performance.

1 Introduction

For high-performance microprocessors with multi-core or many-core architectures, power and heat form the main limitations on their performance. Power management policies [4, 1] are the control procedures to enforce an overall chip-level power budget with minimal cost, through intelligent adjusting of the per-core power states by techniques such as dynamic voltage frequency scaling (DVFS) and clock gating. When the chip power budget is exceeded, selective cores will slow down and run in the power states with reduced power and performance (an extreme case is the sleep mode), according to the adopted policy.

Slowing down or shutting off cores selectively on a chip can be deemed as reducing the concurrency level of the cores' operations. The policies predict the cores' performance under different power modes, and at analysis stage, the combination of the power states are chosen with the minimal performance degradation. Often the policies are based on the priorities of the applications running on the cores. These priorities are associated with the order based on memory bounds or CPU time (e.g., the MaxBIPS in [1]), or the performance degradation rate with respect to power scaling (e.g., the gradient ascent-based approach in [4]). In case of a power overshoot, the cores with lowest priorities are firstly slowed down, and when there is positive power slack the cores with the highest priorities are firstly speed up.

Different from the analytical approaches, this paper explores the formal static and dynamic control mechanisms to reduce the concurrency relations of a system embodied by the multi-core or many-core microprocessor architecture, under a power budget. An N-ary concurrency relation describes the set of N-ary tasks that can run concurrently in a system. The power resource can also be quantified as the maximum number of tasks running at the same time. When the system concurrency relations exceed the amount supported by the power resource, control mechanisms must be applied to reduce the concurrency relations.

The two types of static control mechanisms proposed in this paper are *super-linear* and *and-causal*, respectively, whereas the two dynamic control methods apply *or-causality* and *arbitrating* constraints to the systems concurrency relations, respectively. The control mechanisms schedule the task executions under a particular power budget, but without changing the system functionality. Different mechanisms involve different influences on the system concurrency relations and performance, which are analysed and compared in this paper.

2 System model and concurrency relations

In this section, description of a system's functionality is first discussed using petri-net models [2]. The *group (N-ary) concurrency relations* in a system is then discussed.

Figure 1(a) shows the system model comprising three threads (cores), denoted by a, b, and c. Each thread involves the sequential execution of two tasks, e.g., $a.1$ and $a.2$ for thread a (thread execution can be periodic, but only one iteration is modelled here). With the petri-net modelling, a bar represents the execution of a task, whereas the input places of a bar (circles with input arcs to the bar) represent the *pre-conditions* for the task. Similarly, the output places of a bar (circles with output arcs from the bar) represent the *post-conditions* of the task. The token-based playing rule is that, a task is enabled to execute (*fire*) if there is at least one tokens in each of the task's input places, and the firing of a task brings one token from each input place and generates one token for each output place. In the model, the three threads run concurrently, with the dependency relations between $(a.1, b.2)$ and $(b.1, c.2)$.

The system's state diagram (Figure 1(b)) is derived from a *reachability analysis* [6] of its petri-net model. In the diagram, a token play rule is interpreted by a transition from state s_i into state s_j following transition(s) t, where $s_i(s_j)$ is encoded by a group of token-bearing places and t is labelled by the task(s) execution. We consider both *interleaving* and *step* firing semantics, where, for example, the state encoded by $\{1, 5, 11, 6\}$ is reachable from the initial state $\{1, 2, 3\}$, following the interleaving *transition sequences* (or *execution sequences*) of $(b.1, c.1)$ or $(c.1, b.1)$, or a step transition of $\{b.1, c.1\}$ (All other "step" arcs are however implicit in the diagram to reduce clutter).

Upon the transition sets, the $<$-relation is defined between two transitions a and b (i.e., $a < b$) if a always precedes b in every transition sequence, with the interleaving firing semantics. Further, based on the definitions of the $||$-relation in [3], an N-ary *concurrency relation* is defined in this paper as the set of N-tuples, where for each N-tuple, $\neg(a < b)$ holds for every pair of the tuple elements of a and b ($a \neq b$). Such an N-tuple indicates a "step" arc labelled by the set of N transitions. For example, $(a.1, b.1, c.1)$ belongs to the ternary concurrency relations, and the state $\{4, 10, 5, 11, 6\}$ is reachable from $\{1, 2, 3\}$ following a "step" arc of $\{a.1, b.1, c.1\}$ (the *diamond structure* [6]). In the example, the highest arity of the concurrency relations is ternary, which are listed in Table 1.

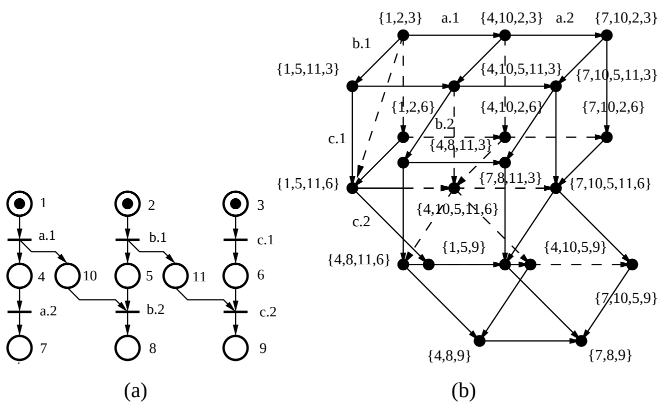

Figure 1: System petri-net model (a) and its state diagram (b)

A property of the N-ary (N>2) concurrency relations is that N-ary concurrency recursively implies M-ary concurrency for all $M \leq N$, on all M-tuples that are subtuples of the N-tuple which belongs to the N-ary concurrency relations. For example, $(a.1, b.1, c.1)$ implies three binary concurrency relations, i.e., $\{(a.1, b.1), (a.1, c.1), (b.1, c.1)\}$. Therefore, if N represents the highest arity of the concurrency relations, the concurrency relations of the system include the N-ary relations and all the M-ary relations ($M \leq N$) thereby implied. In the example, the 4 ternary concurrency tuples imply 9 binary concurrency tuples (see Table 1).

In our example, having the three binary concurrency relations of $\{(a.1, b.1), (a.1, c.1), (b.1, c.1)\}$ also implies the ternary relation of $(a.1, b.1, c.1)$. However, the generalisation of the above property does not always work in both directions. As pointed out in [3], the generalisation is only true in both directions for a subclass of systems with the *distributive* (or *or-causality free*) *concurrency*.

ternary relations	$\{(a.1, b.1, c.1), (a.2, b.1, c.1), (a.2, b.2, c.1), (a.2, b.2, c.2)\}$
binary relations	$\{(a.1, b.1), (a.1, c.1), (b.1, c.1), (a.2, b.1), (a.2, c.1), (a.2, b.2), (b.2, c.1), (a.2, c.2), (b.2, c.2)\}$

Table 1: A list of the concurrency relations in the example of Figure 1

3 Control mechanisms for concurrency reduction

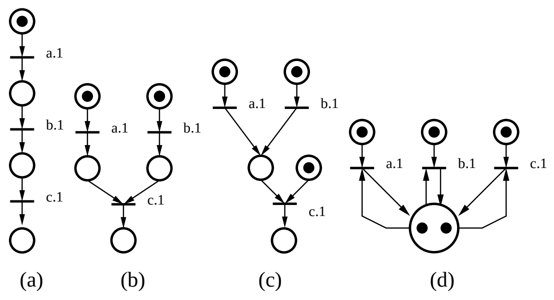

Figure 2: Control structures to reduce the concurrency relation $(a.1, b.1, c.1)$: super-linear (a), and-causality (b), or-causality (c) and arbitrating (d)

Ideally, the higher arity of a system's concurrency relations reflected from its functional model, the more able it is in running its threads in parallel. However, not all the concurrency relations are available at the run time, due to practical constraints such as the power budget limitations. For instance, the simultaneous execution of tasks $a.1$, $b.1$, and $c.1$ in Figure 1 may be restricted because there is no sufficient power resource to support the simultaneous execution of more than 2 tasks, and as a result, the concurrency relation $(a.1, b.1, c.1)$ is reduced from the system during the run time, so are the other three ternary concurrency tuples.

Concurrency reduction means the removal of a subset of the N-tuples from the N-ary concurrency relations in the system, where $1 < N \leq N'$(N'denotes the highest concurrency arity). The removal of a concurrency relation tuple will remove all its supertuples existed in the concurrency list. For example, removal of the binary relation $(b.1, c.1)$ results in the elimination of its parent ternary relations, i.e., $(a.1, b.1, c.1)$ and $(a.2, b.1, c.1)$.

Two types of control mechanisms, *static* and *dynamic*, are introduced in this paper to eliminate the tuples from a concurrency relation. Static control mechanisms impose a *partial order* on the elements of a tuple to be eliminated, and thus on the execution of the tasks in the original system. In contrast, the dynamic control mechanisms impose more than one possible partial orders, and which one of the orders takes place is nondeterministic and only decided upon during execution.

When reducing a concurrency relation, it is desirable to change the system concurrency relations in a minimal way. In particular, when eliminating a N-ary concurrency relation tuple, such as $(a.1, b.1, c.1)$ in Figure 1, using a control mechanism, we want to maintain most of the M-ary concurrency relations implied by that N-ary one. With the ascending order of their abilities to preserve the concurrency relations (or equally to minimise change after concurrency reduction), the control mechanisms are static mechanisms with *super-linear* control and *and-causality*, and dynamic mechanisms based on *or-causality* and *arbitrating*.

3.1 Static control mechanisms

Super-linear control. When the partial order imposed on the tuple elements is indeed a *total* one and the order between each pair is explicit, the control is *super-linear*. Figure 2(a) sketches out one (out of 3!) super-linear control structure to eliminate the concurrency tuple $(a.1, b.1, c.1)$, with the total order of $\{(a.1, b.1), (b.1, c.1), (a.1, c.1)\}$. With this control, the state cube formed by $a.1$, $b.1$, and $c.1$ in Figure 1(b) is replaced by the local state diagram in Figure 3(a). As a result, all the local binary concurrency relations incurred by $(a.1, b.1, c.1)$ are eliminated.

And-causality. Super-linear control is a special case of the more general static controls with the partial order that can be described by *and-causality.* And-causality expresses an *AND* enabling condition for a task execution: the task is enabled only when all its predecessors have fired. Figure 2(b) depicts one control structure using and-causality, where $c.1$ 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 state diagram is shown in Figure 3(b). With and-causality, one local binary concurrency tuple is maintained, i.e., $(a.1, b.1)$, as reflected by the “step” arc from $\{1, 2, 3\}$ to $\{4, 10, 5, 11, 3\}$.

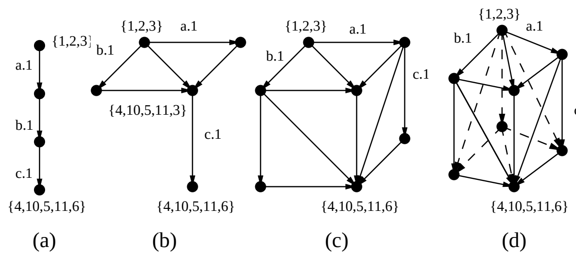

Figure 3: Local state diagrams after concurrency reduction using super-linear control (a), and-causality (b), or-causality (c) and arbitrating (d)

3.2 Dynamic control mechanisms

Or-causality. Different from the and-causality, or-causality [5] expresses an *OR* enabling condition for a task’s execution. Figure 2(c) shows a control structure using or-causality for concurrency reduction, where $c.1$ is enabled when either $a.1$ or $b.1$ has fired. The speciality of or-causality is that it imposes two possible (mutually exclusive) partial orders: $\{(a.1, c.1)\}$ and $\{(b.1, c.1)\}$. It is only known at the run time which order takes place. With or-causality, the local state diagram is shown in Figure 3(c), where all local binary concurrency relations are maintained.

Arbitrating. With the arbitrating structure shown in Figure 2(d) (a *2-of-3 arbitration*), all the three tasks are enabled, but at most two of them can be executed simultaneously. This arbitrating control imposes 6 possible partial orders on the system. Figure 3(d) shows its local state diagram, by which it is evident that all the three binary concurrency relations implied by $(a.1, b.1, c.1)$ are maintained.

In this example, both or-causality and arbitrating mechanisms result in *non-distributive concurrency* [3] and maintains all the local binary concurrency relations. However, with arbitrating all the binary "step" arcs are allowed at the initial state, i.e., $\{1, 2, 3\}$, where only one binary step is allowed with the or-causality. As a result, we regard arbitrating more efficacious than or-causality in minimising the change to the group concurrency relations.

Finally, we point out that, dynamical concurrency reduction techniques, such as those based on arbitrating, are much simpler to implement than "program" the same set of possible schedules statically, thus avoiding combinatorial explosion.

3.3 Performance degradation by the concurrency reduction mechanisms

control scheme		structure	partial order imposed	concurrency reduction effect	performance degradation
static	super-linear	$M!$ arrangements	single total order	All concurrency relations removed	largest
	and-causality	AND enabling conditions between $\lceil \frac{N}{M} \rceil$ groups of M-ary subtuples	single partial order between groups	M-ary concurrency relations maintained but restricted to a group	medium
dynamic	or-causality	OR conditions on a M-ary subtuple to enable the next new tuple element	multiple	All M-ary concurrency relations maintained	dynamic, smaller than and-causality
	arbitrating	*M-of-N* arbitration	multiple	All M-ary relations maintained	dynamic

Table 2: Comparison of different control mechanisms for concurrency reduction, based on the elimination of an N-ary concurrency relation tuple

The performance of a system is degraded when using the proposed control mechanisms to reduce its concurrency. The performance degradation is due to two factors: the extended task execution time as the direct consequences of the concurrency reduction, and the extra delay imposed by the control mechanisms. The former can be determined from the petri-net control structures, and the latter is related with the actual implementation of the controllers. The performance degradation is evaluated considering the example of eliminating the ternary tuple of $(a.1, b.1, c.1)$, in Figure 1(a).

Suppose the execution delays of tasks $a.1$, $b.1$, and $c.1$ are t_{a1}, t_{b1}, and t_{c1}, respectively, and the delays are positive and bounded. Further suppose the delay required by implementing the super-linear, and-causality, or-causality, and arbitrating are denoted by d_{sl}, d_{ac}, d_{oc}, and d_{ab}, respectively.

Without the elimination of the ternary concurrency tuple, the triple tasks can be executed within a time period of $max(t_{a1}, t_{b1}, t_{c1})$.

With the super-linear control (Figure 2(a)), the time to finish executing the three tasks is $t_{a1} + t_{b1} + t_{c1} + d_{sl}$.

With the and-causality based control (Figure 2(b)), the time to finish executing the three tasks is $max(t_{a1}, t_{b1}) + t_{c1} + d_{ac}$.

With the or-causality based control (Figure 2(c)), the execution time is $max(min(t_{a1}, t_{b1}) +$

$t_{c1}, max(t_{a1}, t_{b1})) + d_{oc}$. In addition, it is $max(t_{a1} + t_{c1}, t_{b1}) + d_{oc}$ should the run-time order is $\{(a.1, c.1)\}$, or otherwise $max(t_{b1} + t_{c1}, t_{a1}) + d_{oc}$.

Finally, the execution time using the arbitrating based control (Figure 2(d)) depends on the run time token-game results, as the or-causality one. If the imposed partial order turns out to be $\{(a.1, b.1)\}$ (or $\{(b.1, a.1)\}$) during the run time, the execution time is $max(t_{a1} + t_{b1}, t_{c1}) + d_{ab}$. The execution times with other partial orders at the run time can be similarly derived.

Among the control schemes, if not considering the extra delay caused by the controllers, super-linear control has the largest performance degradation, followed by the and-causality, whereas or-causality incurs the smallest performance degradation with dynamically determined execution delays. This order can be changed when considering the extra control delays though. The arbitrating based control mechanisms can have more delay choices and may result in even lower performance degradation compared with the or-causality based ones. Again, the actual performance gains depend on the arbitration time.

3.4 A comparison of the concurrency reduction mechanisms

We generalise the comparison results in Table 2 for the different control mechanisms in their reducing of a N-ary concurrency relation tuple (and all its lower-arity concurrency relations) to the concurrency relations with highest arity of M ($M < N$). In this table, the control schemes are described, as well as the order relations they impose on the tuple elements, the effects in maintaining the concurrency relations, and the associated performance degradation (without considering the extra controller delays).

In the table, the and-causality applies the AND enabling conditions between the groups of M-ary subtuples, and its concurrency relations are restricted to the subtuples within a group. The or-causality control applies the OR condition on a M-ary subtuple, such that a new tuple element is enabled when the earliest one in the M-ary tuple fires. The arbitrating assumes a *M-of-N* arbitration structure. Both or-causality and arbitrating have multiple partial orders imposed on their tuple elements, and maintain all the M-ary concurrency relations after the reduction.

Finally, we mention that the dynamic control mechanisms can be made *adaptive* with variable numbers of M. With the adaptive approaches, the arity of a system's concurrency relations are dynamically adjustable.

4 Conclusions

This paper proposes different static and dynamic control mechanisms to reduce the system concurrency relations. These concurrency reduction mechanisms can be applied during the power management for a multi-core microprocessor architecture. The control mechanisms are analysed and compared for their abilities in maintaining the system concurrency relations, and their influences on the system performance.

5 Acknowledgements

The authors would like to thank Danil Sokolov, Fei Xia and Delong Shang for their helpful discussions.

References

[1] C. Isci, A. Buyuktosunoglu, C. Cher, P. Bose, and M. Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *MICRO 39: Proceedings of the 39th International Symposium on Microarchitecture*, pages 347–358, 2006.

[2] James Lyle Peterson. *Petri Net Theory and the Modelling of Systems.* Prentice Hall, 1981. ISBN 0-13-661983-5.

[3] L. Y. Rosenblum, A. V. Yakovlev, and V.B.Yakovlev. A look at concurrency semantics through "lattice glasses". In *Bulletin of the EATCS (European Association for Theoretical Computer Science)*, volume 37, pages 175–180, 1989.

[4] J. Sartori and R. Kumar. Distributed peak power management for many-core architectures. In *Proc. Design, Automation and Test in Europe (DATE)*, 2009.

[5] 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*, 9(3):189–233, 1996.

[6] A. V. Yakovlev and A. M. Koelmans. Petri nets and digital hardware design. In *Lectures on Petri Nets II: Applications. Advances in Petri Nets*, volume 1492 of *Lecture Notes in Computer Science*, pages 154–236, 1998.