Stochastic Analysis of Power, Latency and the Degree of Concurrency

Yuan Chen, Isi Mitrani, Delong Shang, Fei Xia, Alex Yakovlev

Technical Report Series

NCL-EECE-MSD-TR-2009-147

September 2009

Contact: yuan.chen@rails.com.cn; fei.xia@ncl.ac.uk

Stochastic Analysis of Power, Latency and the Degree of Concurrency

Yuan Chen*, Isi Mitrani+, Delong Shang+, Fei Xia+, Alex Yakovlev+
**China Academy of Railway Science, China*
+*Newcastle University, UK*
*chenyuan@rails.com.cn,
+{isi.mitrani, delong.shang, fei.xia, alex.yakovlev}@ncl.ac.uk

Abstract

Concurrent processing has become the default mode of operation in on-chip systems. Silicon has become cheap enough for having hardware facilities to support very large scale concurrent processing on chip. As a result the availability and applicability of power is becoming more of a limiting factor than logic for on-chip systems. However, the advantage of parallelism in reducing power consumption will soon become unrealistic because of the limited scope of reducing Vdd beyond threshold voltage, leaving the reduction of concurrency (through the partial shut-down of system blocks) as a realistic means of reducing power consumption when needed. A stochastic modelling approach is presented in this paper which can integrate the **degree of concurrency** *as a parameter into power and latency analysis. This will facilitate a system design and management regime where the degree of concurrency is used as a means of control to achieve power and performance goals.*

1. Introduction

Concurrent processing has been shown to be a successful solution to improve the execution speed for on-chip and on-board systems [1]. A high degree of concurrency can distribute processing load to multiple cores in the system, improving throughput and reducing latency. Although the development in semi-conductor technology has made it possible for a chip to integrate multiple cores without much size increasing [2], the power dissipation becomes the main bottle neck to improve the performance of concurrent systems. Although increasing concurrency can sometimes be used to reduce power consumption, it is under the assumption that one could also reduce Vdd and/or clock frequency at the same time. Current technologies allow very low Vdd which could not be further reduced at run time. In this case and with systems having the hardware and software resources to support a very high degree of concurrency, power may become the limiting factor on run-time concurrency. Power applicability is limited by both supply availability (battery, scavenged power, etc.) and EMI and overheating issues. Even with a stable power supply, high peak power may cause EMI noises and overheating which may reduce system performance and lifetime [3]. Deciding on the right degree of concurrency so as to optimize the power and latency performance in a system has become a key problem.

A concurrent system can be modelled in Figure 1. Suppose there are N independent tasks in the system and each task represents an identical processing load for the processing system. A task is idled until it is triggered by a new incoming event, which comes at a certain activation rate λ. When triggered, the task becomes ready (active) and waiting to be executed by some core or processing unit in the system. Because of task independence, it is possible for multiple tasks to be active (and waiting) at the same time. We use j $(0 \le j \le N)$ to indicate the number of active tasks in the system.

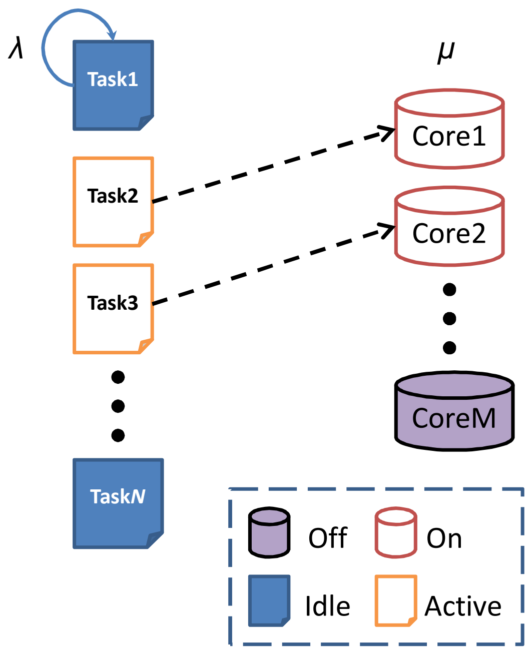

Figure 1: The system structure

Meanwhile, there are M ($1 \le M \le N$) identical cores integrated in the system (the maximum concurrency degree is M), each of which can execute a task with the rate of μ.

If no more than M tasks are active ($j<M$), only j cores are needed for task execution. The other M-j cores can be powered off for power saving (How to choose j out of M cores for processing is out of the scope of this paper). After the completion of execution, a task becomes idled again.

If more than M tasks are active ($M<j \le N$), all cores in the system have to be powered on for processing. However, the executions of the other j-M tasks have to wait until some cores are available. The execution of tasks is assumed to follow the FCFS (First Come First Service) policy.

It is also possible to let tasks wait longer even when extra cores are available. This would be true for the case where there are enough computation facilities to execute more tasks, but some other resource, such as power, is the limiting factor.

Here we will try to formulate and solve the problem of finding the correct degree of concurrency M to optimize the system's performance in both power and latency. We assume that both λ and μ have

Markovian properties. This allows us to model the system using Markov chain techniques which makes the problem easier to deal with. This Markovian assumption is reasonable for multi-core, multi-task systems in a networked environment.

The rest of the paper is organized as follows: Section 2 introduces the Markov model for the system, and then describes the method for *M* optimization. In Section 3, we present a case study to demonstrate the method introduced in Section 2. Section 4 is the conclusion and future work.

The main contribution of this paper is in the fact that this is the first work to investigate and demonstrate the validity of using concurrency management to balance system power and performance performances, based on sound theoretical reasoning. We also demonstrate the potential of optimization using such techniques.

2. Stochastic Model for Current Systems

2.1 Stochastic Model Description

Figure 2 is the stochastic model for the type of system investigated in this work.

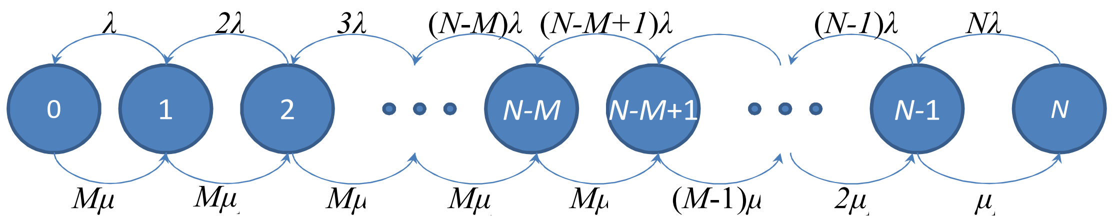

Figure 2: Stochastic System Model

In this model, we use the number of idle tasks as the state variable. For example in the state *N*, all tasks are idle. All cores are powered off accordingly. Since the system moves from the state *N* to *N*-1 when any one of the *N* task is activated, the corresponding transfer rate is *Nλ* (each idle task leaves the idle state at the same rate *λ*). A task is executed in a core with the execution rate *μ* (in other words, a task in execution leaves the execution/active state and becomes idle again at the rate of *μ*). In this first study, in order to simplify the model, both the power on and power off mode switches for a core 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 task becomes active before the execution of the first active task is completed, the system moves to the state N-2, and another core is powered on for task processing accordingly. With two tasks being executed by cores, the rate of one of them leaving execution and becoming idle at this stage is 2μ. Similarly when the system is in the state 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 $i\ \lambda$. With N-i cores on for processing in the state i, the execution rate becomes $(N\text{-}i)\mu$.

When the system is in the state i ($i \le N\text{-}M$), all M cores are already on for task execution, and the corresponding transfer rate from the state i to i+1 is fixed to $M\mu$.

The probability distribution of all states can be calculated analytically and numerically.

2.2 Power Analysis

To differentiate power from probability distribution, we use Q_j to stand for the probability when the system is in state j ($j \le N$), and P as one core's power consumption. Many low power technologies can be used to power on/off cores in a concurrent system. For example, clock gating [4] stops the propagating of clock signals and power gating [5] terminates the power propagation. In this high level model, we simply assume a core consumes full power P when it is on and has no power dissipation when it is off. Therefore, the power dissipation when the system in the state i is $(N\text{-}i)P$ when $N\text{-}M<i<N$ or MP when $i \le N\text{-}M$. The average power consumption of the system $P_{ave}(M)$ is presented in Equation 1:

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

Equation 1

2.3 Latency Analysis

We use the average latency (*W*) (the average time cost of all tasks in both waiting and processing stage, i.e. between activation and becoming idle again) as the measure of latency. It can be derived as follows:

First of all, if *L* stands for the average number of idle tasks, it can be calculated in Equation 2:

$$L = \sum_{i=1}^{N} iQ_i \qquad \text{Equation 2}$$

Therefore, the average number of active tasks is represented by *N-L*. Meanwhile, the arrival rate into active states is given by λL. The average latency *W(M)* is thus described in Equation 3:

$$W(M) = \frac{N-L}{\lambda L} \qquad \text{Equation 3}$$

2.4 M Optimization

For different implementations, system engineers may have different priority considerations about power and latency, and different priority considerations may result in different *M* optimization. Suppose a priority parameter *C* which describes how important power is relative to latency, is used to provide flexibility in *M* optimization.

Given a certain *C*(0<*C*<1), for all possible concurrency degree *M* (*1≤M≤N*), the one which can minimize $CP_{ave}(M)+(1-C)W(M)$ is the optimized *M* (M_{opt}). In other words, M_{opt} satisfies:

$$\forall M \in [1,N], CP_{ave}(M_{opt}) + (1-C)W(M_{opt}) \le CP_{ave}(M) + (1-C)W(M) \quad M_{opt} \in [1,N] \qquad \text{Equation 4}$$

3. Case Study

In order to demonstrate our analysis method, we present a case study in this section. In this case, we set *N*=15 and normalize *P* and μ to 1. Figure 3 presents the power dissipation of the system for *M*=1,2,3,4, and 5.

From Figure 3, it can be seen that for a certain task load (which is represented by λ), the higher the concurrency degree is, the more power is consumed in the corresponding concurrent system. For each M curve, the power dissipation increases with the rise of λ and saturates when λ is big enough. It can be seen that the saturate value for each curve is the corresponding concurrency degree M. It is because we normalize each core's power dissipation to 1 (P=1), and when $P_{ave}=M$, it means all cores are powered on for processing.

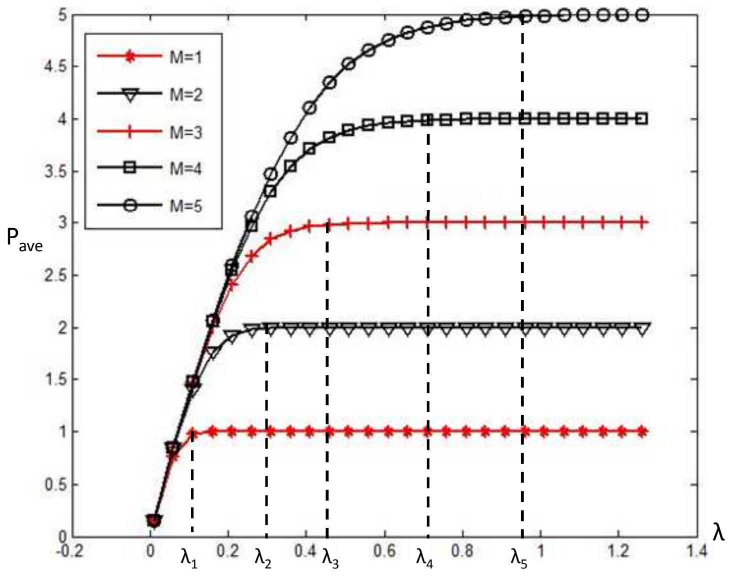

Figure 3: The average power dissipation for various Ms

In Figure 3, we mark the activation rates when each curve becomes saturated by λ_1, λ_2, λ_3, λ_4, and λ_5. It can be seen that $\lambda_1<\lambda_2<\lambda_3<\lambda_4<\lambda_5$, which reflects the higher the concurrency degree is, the more powerful processing capability the corresponding system can have. For example when $\lambda=\lambda_3$, the system has to make full use of all cores if it has no more than 3 cores. However, it may have one or more cores to be powered off from time to time when it has 4 or more cores. For the cases when M>5, power curves show the similar shapes.

On the other hand, a high concurrency degree can also bring lower latency, as shown in Figure 4. From Figure 4, it can be seen that for a certain task load, the higher the concurrency degree is, the lower the latency is. When $\lambda \to 0$, few tasks become activated for a quite long time and the system can always have some core available for task execution. In this case, the latency for a task is only its execution time. Therefore all latency curves start from the value 1 when $\lambda \to 0$.

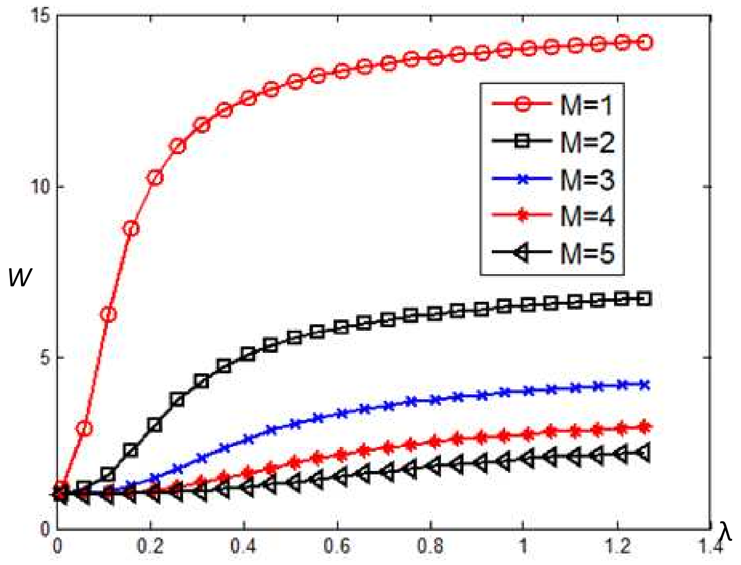

Figure 4: the latency for various *M*s

Different from M/M/1 model [5], λ in this model represents the activation rate and this model has no restriction of $\lambda<\mu$. A big λ ($\lambda>>\mu$) means a task becomes active almost immediately after it has been executed. The system always has *N*-*M* tasks waiting to be executed. When *M*=1, a task can only be executed after the completion of the execution of the other (*N*-1). The latency in this case is *N* (including its own execution time cost). Therefore in Figure 2, the latency curve for *M*=1 saturates to 15. With the increase of concurrency degree, the saturation value decreases uniformly since the system has more computing capability. When *N*=*M*, no task needs to wait when it is activated, so the corresponding latency curve is a straight line with value 1.

As stated in Section 2, we may try to optimize the concurrency degree to balance power and latency. We first set *C* to 0.7, which means low power is the main concern for the system design. We tried *M*=1, 2, 3, 4, and 5 cases and the result is given in Figure 5. It can be seen that when $0<\lambda<0.3$, *M*=2,3,4,5 shows little difference because most cores can be off from time to time when the load is light. When *λ*>0.3, *M*=3 shows as the optimized concurrency degree since it has the best power with latency balance compared to others.

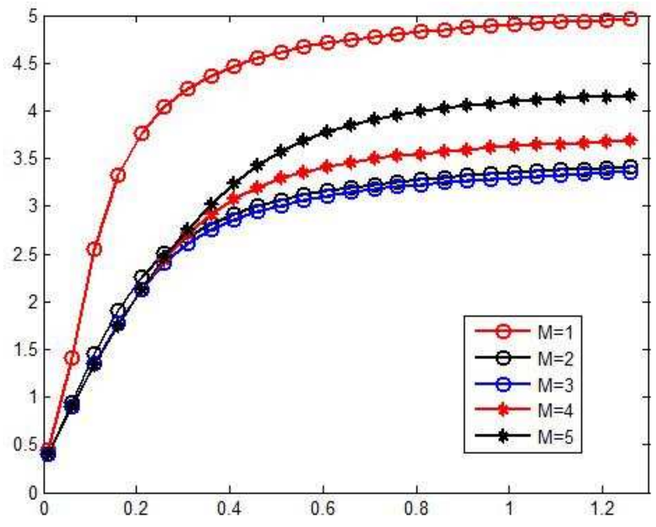

Figure 5: *M* Optimization (*C*=0.7)

If we reset *C* to 0.3, which means latency draws more attention than power dissipation, the corresponding result is given in Figure 6. It shows *M*=5 is the best concurrency degree for the entire scope of *λ*.

4. Conclusion and Future Work

A stochastic analysis based concurrency degree optimization method is presented in this paper. This modelling approach explicitly represents a concurrent system with various concurrency degrees. With the power and latency measure in Section 2, this approach can easily derive the optimized concurrency degree (*M*) based on the amount of embedded tasks (*N*), the task activation rate (*λ*) and the execution speed of each integrated core (*μ*). A case study is presented to demonstrate the usage of this analysis.

In our analysis, we simply assumed that the power dissipation and execution speed for each integrated core is independent from the concurrency degree, which is not always true in the real world. With the restriction of on-chip or on-board size, the peak power provided by the battery is limited. In this case, the system may have to lower the supply voltage if further more cores are needed to be integrated, so as to meet the power restriction, or suffer issues like Vdd droop. The reduction in the supply voltage results in slower execution speed in each core. Therefore, a concurrency related power dissipation P_M and execution speed μ_M can give more realistic representation in the model, and the new stochastic model is shown in Figure 7. New case study will be given in the near future to present the usage of the new model.

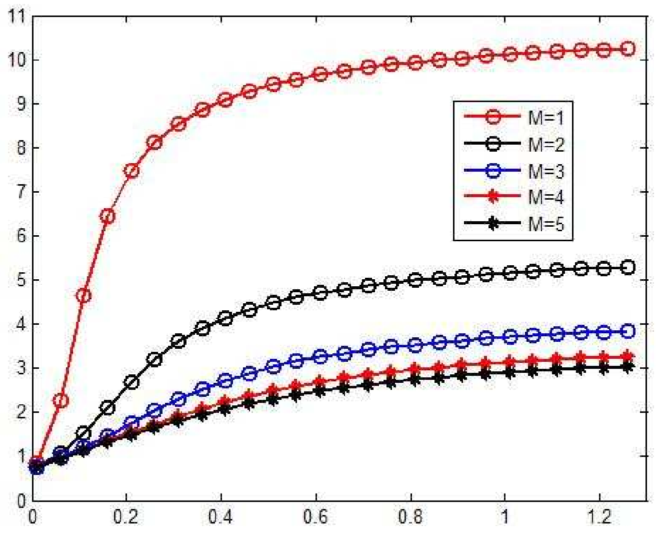

Figure 6: *M* optimization (*C*=0.3)

Furthermore, a concurrent on-chip system with optimized concurrency degree will be put into VLSI design and its performance will be tested in realistic environments for portable system applications.

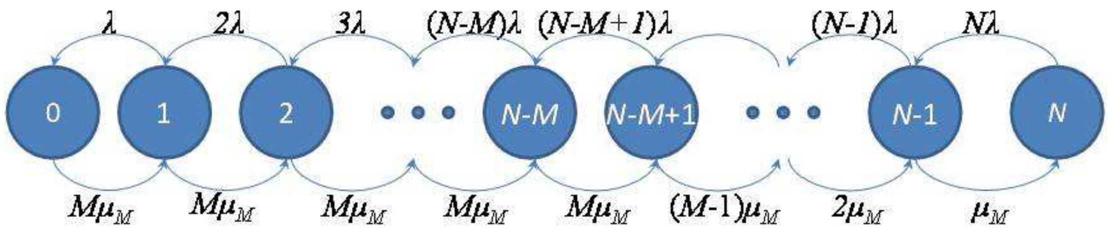

Figure 7: Stochastic Model when execution speed is concurrency degree related

References

[1] Santanu Dutta, Rune Jensen, Alf Rieckmann, "Viper: A Multiprocessor SOC for Advanced Set-Top Box and Digital TV Systems," IEEE Design and Test of Computers, vol. 18, no. 5, pp. 21-31, Sep./Oct. 2001.

[2] International technology roadmap for semiconductors 2008 update, http://www.itrs.net/Links/2008ITRS/Home2008.htm.

[3] Chinhung Chan, Yucheng Chang, Hsichi Ho, Herming Chiueh, "A thermal-aware power management soft-IP for platform-based SoC designs", IEEE SOC proceeding, page(s): 181-184,2004.

[4] H. M. Jacobson, "Improved Clock-Gating through Transparent. Pipelining", ISLPED04, Pages 26-31, 2004.

[5] S.Henzler, "Power Management of Digital Circuits in Deep Sub-Micron CMOS Technologies", Springer 2007.