2009 IEEE Computer Society Annual Symposium on VLSI

On-line MPSoC Scheduling Considering Power Gating Induced Power/Ground Noise

Yan Xu[1], Weichen Liu[2], Yu Wang[1], Jiang Xu[2], Xiaoming Chen[1], Huazhong Yang[1]
[1]Department of Electronics Engineering, Tsinghua University, Beijing, China
[2]Hong Kong University of Science and Technology, Hong Kong, China
E-mail: {waterphy@gmail.com, yu-wang@tsinghua.edu.cn, jiang.xu@ust.hk }

Abstract

Power gating induced power/ground (P/G) noise is a major reliability problem facing by low power MPSoCs using power gating techniques. Powering on and off a processing unit in MPSoCs will induce large P/G noise and can cause timing divergence and even functional errors in surrounding processing units. P/G noise is different from thermal or energy which is an accumulative effect. The noise level should be predicted and victim circuits should be protected before the noise is induced. Hence, the power gating-aware scheduling problem with the consideration of P/G noise should be solved using an on-line method considering the run-time variation of tasks' execution time. In this paper, we formulate an on-line task scheduling problem with the consideration of P/G noise based on our detailed P/G noise analysis platform for MPSoC. An efficient on-line Greedy Heuristic (GH) algorithm that adapts well to real-time variations is proposed to reduce noise protection penalty and improve MPSoC performance. Our experiments show that the algorithm can achieve on average 26% performance improvement together with on average 73% noise protection penalty saving compared with the conservative stop-go method. We also compare our technique with a two-step solution that computes a static schedule at compile time and make adjustment on the schedule according to runtime variations. For benchmark with larger task number, GH method achieves impressive performance improvement comparing with the two-step solution.

1. Introduction

Power gating induced power/ground (**P/G**) noise is one of the most significant reliability threats for MPSoCs with smaller feature sizes. Tight low power requirements have forced MPSoC to aggressively adopt low power techniques such as dynamic voltage/frequency scaling, clock gating, and power gating [1,2]. While low power techniques like power gating can dramatically reduce power consumption for idle processing units (**PU**s), they exacerbate simultaneous switching noise (or *di*/*dt* noise) on the power delivery network. Such MPSoC P/G noise can result in performance degradation and even functional errors. At the same time, when process technology advances, power consumption and wire resistance have gone up while the supply voltage drops. Thus the chip noise margin will go down. As a result, to design resilient systems, design methodologies with P/G noise management become necessary to fulfill the low power and high reliability requirements of MPSoCs.

MPSoCs use multiple PUs to deliver massive parallel processing performance within a limited power budget. However, all the PUs are often not working at the same time, and some of them are idle and consume significant leakage power in deep submicron process. Power gating is one of the most effective low power techniques widely adopted to save the ever-increasing leakage power consumption of the idle PUs. When a PU finishes a task or a new task is assigned to a power-off PU, MPSoC needs to power off or on a PU. These procedures cause large P/G noise in the MPSoC power delivery network, which then propagates to other PUs and endangers their normal operations. Hence, if a new task is assigned to a power-off PU, the active PUs around it need to be *protected* from the powering-on *attack*; similarly, if a PU finishes a task to be powered off, the active PUs around it also needs to be *protected* from the powering-off *attack*. In the open literature, there are few studies addressing such noise issues in MPSoCs.

Previous works on P/G noise mitigation mainly focused on circuit level techniques for logic blocks. The proposed techniques include sleep transistor designs [3,4], decoupling capacitor insertion [5], and P/G noise-aware floorplanning [1,2,6]. Recently, power gating sequence scheduling [7-9] in a block or several blocks were proposed to tradeoff wake-up time for P/G noise reduction. These works mainly focused on block level design techniques, while in this paper, we investigate processor-level power gating scheduling based on our detailed P/G noise analysis platform for MPSoC to minimize the performance impacts due to the protection overhead during powering-on/off PUs.

On the other hand, numerous efforts have been paid to optimize the scheduling problems with power/performance/thermal objectives [10-16]. However, P/G noise is different from thermal or energy which is an accumulative effect. Recent work by Reddi *et al.* [17] based on [18] proposed a voltage emergency predictor that learns the signatures of voltage emergencies (the combinations of control flow and microarchitectural events leading up to them) and uses these signatures to prevent recurrence of the corresponding voltage emergencies. The noise level should be predicted [17] and victim circuits should be protected before the noise is induced. Hence, the power gating-aware scheduling problem with the consideration of P/G noise should be carefully modeled and solved using an on-line method considering the run-time variation of tasks' execution time; or solved off-line based on an accurate P/G noise estimation, and then assisted by a fast on-line adjustment method considering the run-time variation. Recent work from Todri [19] considered the P/G noise induced by switching current when the tasks are running on PUs to minimize the P/G noise level of multi-core systems. In this paper, we mainly focus on modeling and management of the noise induced by powering-on/off a PU when a task is assigned to or finishes on a PU.

Based on the above discussion, our work distinguishes itself from previous works in the following aspects: **1)** We formulate an on-line task scheduling problem (Section 4) with the consideration of power gating induced P/G noise based on our detailed P/G noise analysis platform for MPSoC (Section 2/3). The PU states of P/G noise-aware on-line scheduling problem and the impact range for powering-on/off a PU in MPSoC are defined in the problem model. **2)** Based on the MPSoC P/G noise modeling considering both spatial and temporal constraints, an efficient on-line Greedy Heuristic (**GH)** algorithm (Section 4/5) that adapts well to run-time variations and real-time decision requirement is proposed to reduce noise protection penalty and improve MPSoC performance. Impacts on MPSoC performance of considering different factors during on-line scheduling decisions are also studied. **3)** We also compare on-line GH algorithm with a two-step solution: Static Scheduling method (Simulated Annealing (**SA**) algorithm here)

This work was supported by grants from 863 program of China (No. 2009AA01Z130), and NSFC (No. 60870001) and TNList Cross-discipline Foundation.

accompanied by On-Line Adjustment (**OLA**) strategy (**SSOLA**) (Section 5/6). The benefit and loss of each method in different task cases are studied (Section 7).

2. P/G Network Modeling of MPSoC

The MPSoC chip is composed of Processing Units (**PU**s). They are placed as a mesh shown in Fig. 1.

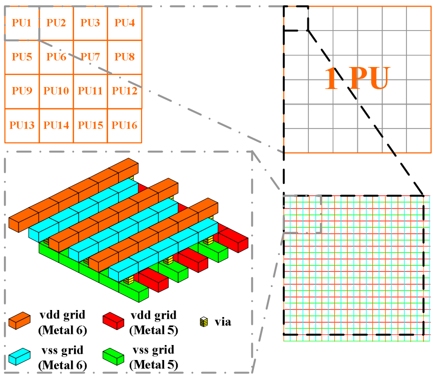

Fig. 1. P/G network architecture of MPSoC.

2.1 On-chip P/G Network model

The most common way to distribute power in a GSI (gigascale integration) chip is to distribute it through an on-chip grid made of orthogonal segments (Fig. 1) [20,21]. The horizontal and vertical segments of a grid are routed at different metal levels (e.g. at the layers of Metal 5 and Metal 6) and are connected though vias at the crossing points. A wire between two nodes is simply modeled as a lumped resistance R_{seg} and an inductance L_{seg} (Fig. 2). C_d denotes the capacitance per unit area between a power grid node and the adjacent ground grid node (including both the intentionally added on-chip decaps and the equivalent capacitance between the wires at different metal levels). C_L is the load capacitance.

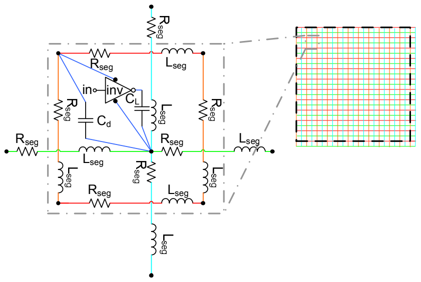

Fig. 2. In order to facilitate the P/G network analysis, each wire segment is modeled as a chain of L-type RLC equivalent circuits. An inverter with a capacitance load is used to imitate the core logic. A decap is connected to the intersection points on the vdd/vss grids.

2.2 Package Model

In flip-chip package technology, the package I/O pads are connected to the chip I/O pads through metal bumps distributed across the chip surface. The flip-chip package is more expensive than a wire bond package. However, it has smaller I/O parasitics [22]. The distributed pads also help increase total I/O count and consequently lower the P/G noise. Two-thirds of the total pads are used for power distribution [23] in our model. These power and ground pads are spread throughout the chip surface. The pad and bump are simply modeled as a package resistance and a package inductance. An off-chip decoupling capacitance is added between the virtual power node and the virtual ground node of each PU. Meanwhile, the PCB board is modeled as a lumped resistance and inductance network.

3. Power Gating induced P/G Noise in MPSoC

3.1 P/G Noise Estimation in MPSoC

Table 1. PU states in P/G noise-aware power gating scheduling

PU States	Illustration
Off state	Power gated.
ToOn state	The off to on transition. The start time of the transition for PU p to execute task i is defined as *ton(i)*.
ToOff state	The on to off transition. The end time of the transition for PU p just finished task i is defined as *toff(i)*.
Idle state	Clock gated (power is on).
ClkToOff state	Clock gate transition.
ClkToOn state	Clock wake up transition.
Free state	Both power and clock are on, but there is no task running on the PU.
Active state	A task is running on the PU.

The PU states of on-line task scheduling considering power gating induced P/G noise are defined in Table 1. The noise induced by turning on/off PUs in different locations in MPSoC is evaluated. Assuming all the PUs induce the same supply current and have identical capacitance density, an inverter is put between a power grid node and its adjacent ground grid node to represent the PU switching activity. The inverter size is chosen according to the average power consumption requirement for typical PUs. The simulation uses the 45nm bulk CMOS model [24] for transistors (V_{dd}=0.8V). The standard cell library is from the Nangate Open Cell Library [25].

1) P/G Noise Generation and Propagation: A homogeneous MPSoC is modeled with a set of PUs, $PU=PU_p$; p=1, 2, …, N. A ToOn/ToOff PU is defined as an *attacker*. A PU, which carries an active task, is defined as an active PU. An Active PU within the impact range of an attacker is defined as a *victim*. (Please note that some power-on PUs could be Idle or Free, and they are not victims in our definition.) For a PU p, R_{impact}^p is defined as the set of the victims of an attacker p, while $PV(p)$ is defined as a set of PU p's potential victims, namely $PV(p)=\{q|\ q\in PU,\ q\neq p$ and q is in the impact range of $p\}$. The number of PU p's potential victims is denoted by $N_{PV}(p)$.

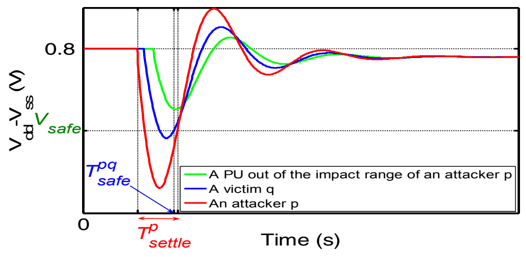

Fig. 3. A conceptual illustration of P/G noise temporal influence.

The conceptual illustration of the P/G noise temporal influence in MPSoC is shown in Fig. 3. We obtain the impact (e.g. on delay increment) of P/G noise with different amplitudes on standard logic cells and D Flip-Flop through simulation. Safe voltage levels are set for different cells to satisfy the performance (e.g. delay relaxation) requirement, and then a safe voltage level for a PU can be calculated as V_{safe}. T_{settle}^p is the minimum time required for the voltage difference between V_{dd} and V_{ss} node pairs of an attacker p to be stabilized above the safe voltage level V_{safe}. The measurement of T_{settle}^p is started at the beginning of the switching event. T_{safe}^{pq} is the earliest time point after which V_{dd}-V_{ss} of a victim q is stabilized above V_{safe}. TI_{safe}^{pq} is the time from the beginning of the switching event to T_{safe}^{pq}. These parameters are extracted from our P/G network simulation.

2) P/G Noise Protection Method: If we assign a task i to a power gated PU p, the powering-on/off noise when the task begins/finishes will attack the PUs in R_{impact}^p which is provided through our P/G model. The noise protection method is to clock gate the victim PUs before powering on or off the attacker and to wake up them when the attacker is fully turned on or off. Fig. 4 shows the timing of a power on event, and the timing of a power off event is similar.

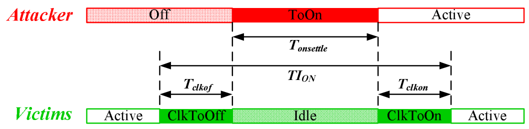

Fig. 4. Timing of a power on event.

T_{clkoff} and T_{clkon} are the time needed to clock gate a PU and to wake it up from the clock gated state, respectively. $T_{onsettle}$ and $T_{offsettle}$ are the settle time for a PU to power on and power off. In order to ensure the reliability of MPSoC, here $T_{onsettle} \geqslant \max\{T_{settle}^{p}, TI_{safe}^{pq}, \forall p, q \in PU, q \neq p\}$, $T_{offsettle} \geqslant \max\{T_{settle}^{p}, TI_{safe}^{pq}, \forall p, q \in PU, q \neq p\}$. TI_{ON} and TI_{OFF} are the noise protection time penalty for a victim PU when an attacker powers on and off respectively, where $TI_{ON} = T_{clkoff} + T_{onsettle} + T_{clkon}$, $TI_{OFF} = T_{clkoff} + T_{offsettle} + T_{clkon}$. Assume that the victim number of PU p as an attacker at the moment t is $N_{victim}(p,t)$. we define $P_{on}(p,t)$ and $P_{off}(p,t)$ as the total performance penalty to power on and power off attacker p, respectively, where $P_{on}(p,t)=TI_{ON} \times N_{victim}(p,t)$, $P_{off}(p,t)=TI_{OFF} \times N_{victim}(p,t)$.

3.2 Motivation Example

We illustrate the benefit of the noise-aware processor level power gating strategy by taking a MPSoC with 4×4 PUs as an example. The peak P/G noise levels of PUs induced by attackers located at different locations are shown in Fig. 5. Different impact ranges can be observed: for PU1 as an attacker, at most 5 PUs need protection; for PU2, at most 9 PUs need protection; for PU6, all the other active PUs need protection.

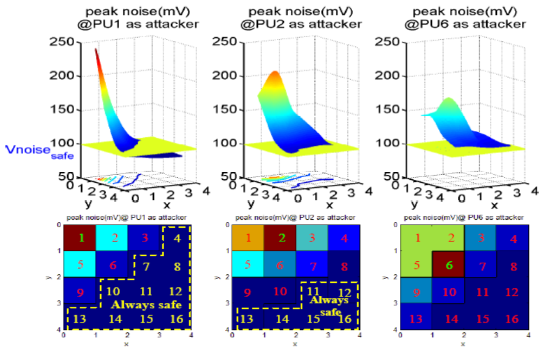

Fig. 5. Noise level and impact range of power gating induced P/G noise in 16-PU-MPSoC.

To assign a new task, the conservative strategy - **stop-go** method (introduced in Section 5.3) will protect all the active PUs (clock gating them) when turning on any PU. However, based on our P/G noise model, when the new task is assigned to different PUs, there may be some PUs which do not need protection. e.g. if the new task is assigned to PU1, PUs within the *always safe* range don't require protection, and at most 5 PUs will need to be clock gated. Hence we can improve the MPSoC performance and reduce the noise protection penalty by carefully scheduling the tasks onto PUs with different impact ranges.

4. Power Gating-Aware On-line Task Assignment and Scheduling in MPSoC

Problem Definition: Considering a homogeneous MPSoC with N PUs, and a set of real-time tasks *Task*, determine an assignment of tasks to PUs **on-line,** such that all task constraints and PU operation constraints are met, the MPSoC performance is optimized and the penalty for safeguarding the victim PUs is minimized.

Table 2. Variables in on-line task scheduling

i	Sequence number for each task $i \in Task$
DAG *Link*	In which nodes represent tasks and directed edges indicate data dependencies between pairs of tasks.
length_p(i)	Predicted ideal execution time for Task i.
length(i)	Real ideal execution time for Task i.
treq(i)	Request time for Task i.
ts(i)	Real start time for Task i.
tf(i)	Real finish time for Task i.
ton(i)	The start time of the off to on transition for PU p to execute task i.
toff(i)	The end time of the on to off transition for PU p just finished task i.
T_{end}	Finish time for all the tasks and PUs.

4.1 Task modeling for On-line Task Scheduling

Let *Task* be the set of tasks to be executed. For each task $i \in Task$, its request time and predicted ideal execution time without interruption when running on PU p are denoted by *treq(i)* and *length_p(i,p)*, respectively. (*length_p(i,p)= length_p(i)* since we are using a homogenous MPSoC.) We use a directed acyclic graph (DAG) *Link* in which nodes represent tasks and directed edges indicate data dependencies between pairs of tasks.

For each task $i \in Task$, its real start time and real finish time are denoted by *ts(i)* and *tf(i)*, respectively. During real-time operations, because there are various run-time variations in MPSoCs (e.g. the frequency variation of PUs induced by process variation), the tasks' execution time may be different from the original predicted execution time. *tf(i) - ts(i)* equals to a task i's real ideal execution time without interruption *length(i)* plus its noise protection interruption time.

4.2 Constraints for On-line Task Scheduling

Under the assumption of static task assignment problem, all the tasks request for execution at the very beginning, and the whole DAG *Link* and *length(i)=length_p(i)* ($\forall i \in Task$) are known. Different from the static problem, not all the information of the tasks are known when a task is to be assigned in the on-line task assignment problem. Supposing the current time is *tc*, the manager of the MPSoC system only knows: the existence and request time of the tasks whose request time are not later than *tc*. Once a task i is assigned and scheduled, *ts(i)* is also known. *tf(i)* and *length(i)* are not known until task i is finished. T_{end} is defined as $T_{end}=\max\{toff(i), \forall i \in Task\}$ (finish time for all the tasks and PUs). At present, T_{end} is the metric representing MPSoC performance. **CP** and **PT** denote the **C**lock gating **P**enalty and the **P**ower-on/off **T**imes, respectively.

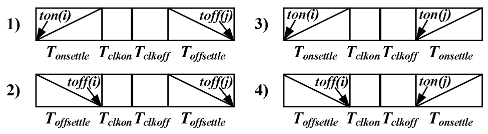
1)
ton(i)
toff(j)
$T_{onsettle}$ T_{clkon} T_{clkoff} $T_{offsettle}$
3)
ton(i)
ton(j)
$T_{onsettle}$ T_{clkon} T_{clkoff} $T_{onsettle}$
2)
toff(i)
toff(j)
$T_{offsettle}$ T_{clkon} T_{clkoff} $T_{offsettle}$
4)
toff(i)
ton(j)
$T_{offsettle}$ T_{clkon} T_{clkoff} $T_{onsettle}$

Fig. 6. Extreme conditions of two PUs' ToOn and ToOff transition events.

Special timing constraints considering protection and settling times are introduced. Fig. 6 shows the extreme conditions of two operations for powering on and off PUs. For the first condition, task i starts (turning on a PU) right before task j's finish time (turning off a PU). *ton(i)* and *toff(j)* should satisfy the timing constraints: $\forall i, j \in Task, i \neq j$: if $\exists$ *ton(i)*, *toff(j)*, then $ton(i) \leqslant toff(j)-TI_L$ so that the victim protection procedure for these two attack operation will not conflict. TI_L are defined as: $TI_L=T_{onsettle}+T_{offsettle}+T_{clkoff}+T_{clkon}$.

Similarly, for Condition 2: task i finishes right before task j's finish time: $toff(i) \leqslant toff(j)-TI_{OFF}$; for Condition 3: task i starts right before task j's start time: $ton(i) \leqslant ton(j)-TI_{ON}$; for Condition 4: task *i* finishes right before task *j*'s start: $toff(i) \leqslant ton(j)-TI_S$. Here, $TI_S=T_{clkoff}+T_{clkon}$.

5. Algorithms for On-line Power Gating Scheduling

5.1 On-line Scheduling Algorithm

Greedy Heuristic (GH) algorithm for power gating induced P/G noise-aware on-line task assignment and scheduling

Input: the task set *Task* (including DAG *Link*, *treq (i)*, *length_p(i)*); the PU set *PU* (including the impact relation between PUs); $T_{onsettle}$, $T_{offsettle}$, T_{clkoff}, T_{clkon}.
Output: T_{end}, CP, PT and the task assignment.

```
1  Initialize variables;
2  Time node $t$=0;
3  do
4      if there are requested tasks and their inputs are ready
5          D1: choose a task $i$ to assign;
6          if timing constrains are satisfied
7              D2: choose a PU $p$ to execute task $i$;
8              if available PU $p$ exists
9                  if chosen PU $p$ is Off
10                     if $N_{victim}(p,t)>0$ (PU $p$ has some victims)
11                         protect victims;
12                         power on PU $p$ to execute task $i$;
13                     else
14                         power on PU $p$ to execute task $i$;
15                 else
16                     assign task $i$ to PU $p$;
17     else if there are PUs waiting to power off
18         D3: choose a PU $p$ to power off;
19         if timing constrains are satisfied
20             if $N_{victim}(p,t)>0$
21                 protect victims;
22                 power off PU $p$;
23             else
24                 power off PU $p$;
25     $t$++;
26 while there is an un-finished task or an on PU
```

Fig. 7. On-line heuristic algorithm for P/G noise-aware scheduling.

The on-line task scheduling algorithm (GH) is shown from line 4 to 24 in Fig. 7. We assume that there is a manager to perform reliability operations in MPSoCs. Once the on-chip reliability manager receives the signal which reports requested tasks' input data are ready, it chooses a task *i* to assign. According to the PUs' states, the manager tries to choose a PU *p* to execute task *i*, and synchronously, it records the victim information if there are victims. If the timing constraints (the last two paragraphs in Section 4.2) are not satisfied, the task assigning operation will be postponed and decided until a proper time node. The Free and Off PUs are both available PUs to execute a task. If the chosen PU *p* is Free, the manager directly assign task *i* to PU *p* to execute the task. If the manager decides to power on an Off PU *p* to execute task *i*, the clock off signal is sent to all the victims (if exist) of PU *p*. Once all the victims are clock gated, the attacker *p* begins to be powered on. After powering on attacker *p*, the victims protected procedure ends. The manager start to clock on them, and then victims come back to resume their tasks.

If there are no tasks waiting to assign, the manager will consider choosing a PU *p* to power off. Free PUs and PUs that just finish task execution are both candidates. If the timing constrains (the last two paragraphs in Section 4.2) are not satisfied, the powering off operation will be postponed and decided until a proper time node. In a similar way of powering on an Off PU and protecting its victims, the system will protect victims (if exist) and power off PU *p*.

As introduced above, in our on-line scheduling algorithm, assigning a new task has higher priority than powering off a PU. However, for low power design, if there are no tasks waiting to assign for the moment and timing constrains are satisfied, a PU which is Free or just finishes a task will be power gated. The strategies of key decision D1, D2, and D3 operations (line 5, 7 and 18 in Fig. 7) are detailed in Section 5.2. These decision procedures are simple enough to satisfy the real time operations in on-line scheduling problem. Hence, we ignore the decision time and the communication time for simplicity.

5.2 Detail Strategies in Greedy Heuristic (GH) Algorithm

GH algorithm is proposed for the on-line power gating-aware task assignment and scheduling problem. Potential differences occur during the three key decision operations D1, D2, and D3 (line 5, 7 and 18 in Fig. 7). The factor options adopted by different strategies in following GH algorithm are described as follows. Basically, assigning new tasks to PUs has higher priority of execution than powering off PUs; PUs are selected for task allocation based on their impacts on the running tasks; tasks are scheduled in the First In First Out (FIFO) [26] order; the frequency of powering on/off a PU is decided by $N_{PV}(p)$.

The factor options adopted by Strategy 1 in GH algorithm (GH1) are as follows. During ***D1***, GH1 always runs the task with the earliest release time. If there are several tasks with the earliest *treq(i)*, GH1 chooses to always run the longest predicted task. When we choose a PU to execute the new task (***D2*** in line 7 of Fig. 7), GH1 gives first priority to Free PUs. If there is no Free PU, consider Off PUs. If there are several Free PUs, GH1 chooses the PU with maximal $N_{PV}(p)$. If there are several Off PUs as candidates, GH1 always powers on the PU with the minimal $P_{on}(p,t)$. GH1 chooses to always power on the PU with the minimal $N_{PV}(p)$. If there are several PUs waiting for powering off (***D3*** in line 18 of Fig. 7), the choosing rule of GH1 is similar to the rule of powering on a PU.

We also adopt another Strategy 2 in GH algorithm (GH2) similar to GH1. The only difference is that GH2 chooses to always run the shortest predicted task during ***D1***.

5.3 GH's corresponding Stop-go Algorithm

During powering on or off a PU, on-line stop-go algorithm protects all the other active PUs, while GH algorithm only protects the active PUs in the attacker's impact range (see also Section 3.2). The stop-go algorithm is simpler and safe, but it is conservative according to our P/G noise model. We implement two stop-go algorithms stop-go(GH1) and stop-go(GH2) corresponding to GH1 and GH2 respectively.

6. Static Scheduling+On-Line Adjustment (SSOLA)

If $length(i)=length\_p(i)$ ($\forall\ i \in Task$), the task scheduling problem can be solved off-line by static scheduling algorithm. However, because of various run-time variations, the static task scheduling may be not applicable for real time implementation. Increasing and decreasing execution time of an attacker will cause potential reliability threaten to its victims. If we still want to use the static scheduling results (here we obtain the static scheduling results using a Simulated Annealing (**SA**) algorithm), an On-Line Adjustment (**OLA**) strategy should be adopted to ensure the reliability of MPSoC. A light weight OLA method is described as follows:

The real tasks' execution time will change in two ways compared with the predicted one: increasing or decreasing.

(1) If the on-line monitors find the real execution time of task *i* on PU *p* will be longer than the predicted one, all the active PUs on chip will be protected at the predicted finish time of task *i* (*tf(i)* in static scheduling) until the PU executing task *i* is really powered off when task *i* is finished (the real finish time is denoted as $tf_{real}(i)$). The original static task scheduling (task starts time and protection time) for tasks after *tf(i)* should be postponed by $T_{clkoff}+ tf_{real}(i)- tf(i)$.

(2) If the real execution time of task *i* on PU *p* is shorter than the predicted one, the conservative on-line adjustment strategy is to keep PU *p* in Free state and power it off at the predicted power off time *toff(i)*. The original static task scheduling will not be influenced.

7. Implementation and Experimental Results

7.1 Implementation and Experiment Setup

The P/G noise analysis platform is built up with HSPICE and C. Scheduling algorithms are implemented with C and MATLAB. The experiments are performed on a server with 2 Intel Core2 Xeon and 8GB memory.

For different MPSoCs, the average power consumption of a single PU is kept as a constant around 30mW, and the area of a single PU is set as a constant 660μm×660μm. Based on the performance results of standard logic cells and D Flip-Flops with different P/G noise, the noise toleration of V_{dd}-V_{ss} is set as 100mV, hence V_{safe} defined in Section 3.1 is set to 700mV. Then the corresponding R_{impact} of each attacker i is derived for 4×4 to 8×8 PU mesh MPSoCs. The T_{clkon} and T_{clkoff} are set to be 100 clock cycles for the time penalty to protect/resume the data and clock-on/off the victim PU. T_{settle} for both powering on and off are set to 200 clock cycles. All the time units in the results are measured by clock cycles. The P/G network RLC parameters are extracted from PTM interconnect model [24].

Four task structures are generated as test benchmarks: (1) $TASK_{NC}$: tasks with No Correlation, (2) $TASK_{SP}$: several Sequential tasks in Parallel, (3) $TASK_{TT}$: Tree-connected Tasks, (4) $TASK_{FC}$: Fully Correlated tasks (a connected DAG with multiple inputs and multiple outputs). The real execution time of each task is assumed to vary between -20% and 10% of the predicted execution time. The predicted execution time (less than 20000 clock cycles) of each task is random generated with a uniform distribution. In the following experiment, we assume every task is released at time 0: for each task $i \in Task$, $treq(i)=0$.

7.2 Results for SA/SSOLA/GH/stop-go(GH)

We first evaluate task bench on 4×4 MPSoC to compare the SA/SSOLA/GH/stop-go(GH) methods shown in Table 3. $T_{end\_}i$ is defined as ideal finish time for all the tasks assuming that power gating is not adopted.

The static method is used in an ideal case, assuming *length_p(i)=length(i)* ($\forall$ $i \in Task$). Here, we use a Simulated Annealing (SA) algorithm as the static method. GH algorithm adopts Strategy 1 (denoted by GH1), and stop-go(GH1) algorithm is its corresponding stop-go method. In the experiment, the runtime of SA in SSOLA is from 20 minutes to 20 hours, and on average the runtime is about 3 hours.

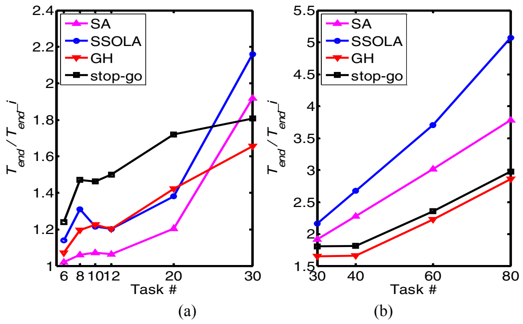

Fig. 8. $TASK_{NC}$ with different task numbers on 4×4 MPSoC.

In Fig. 8(a), for $TASK_{NC}$, with small task number (6 to 20), generally, the static method gets better results than the on-line ones, while GH performs best among all the on-line methods: T_{end}(SA)< T_{end} (GH)< T_{end} (SSOLA)< T_{end} (stop-go(GH1)). In Fig. 8(b), for $TASK_{NC}$, with large task number (30 to 80), our GH still gets the best performance among the on-line methods, and is even better than the static method since the static method is hard to get optimal results: T_{end} (GH)< T_{end} (stop-go(GH1)) < T_{end}(SA)<T_{end} (SSOLA).

PT of on-line algorithms is much less than PT of SA/SSOLA. For large task number, GH1 method achieve impressive T_{end} improvement comparing with SSOLA: from 7% to 220% of $T_{end\_}i$, and this improvement is especially large for $TASK_{NC}$; CP and PT of GH1 are from 22% to 96% and from 16% to 88% less than CP and PT of SSOLA, respectively.

From Table 3, for other three task structures with large task number, the T_{end} and PT trends generally match $TASK_{NC}$'s. The results of SSOLA are the worst. However, the relatively value of T_{end}(SA) to T_{end} (GH1) and T_{end} (stop-go(GH1)) fluctuates a little.

In order to get results good enough, SA and SSOLA need to run hours off-line. While GH and stop-go algorithms can be used at real-time. Besides, SA only can solve static problem which is an ideal case. Only using static method can not solve on-line scheduling problem. The performance gain from SSOLA is not obvious compared with the online algorithms proposed in this paper (up to 4.1% of $T_{end\_}i$), and as the problems size goes large (number of tasks larger than 30), it is harder for SSOLA to find solutions of high quality, and our online algorithms show superior capability in terms of quality of solution and algorithm running time.

In a word, for on-line task scheduling problem, the purely on-line algorithms like GH and stop-go(GH) have more advantages. Hence, the following experiments will focus on on-line algorithms only.

7.3 Results for Purely On-line Task Scheduling

Our experiment gets GH1/GH2/stop-go(GH1)/stop-go(GH2) results for 40 tasks of four task structures on different MPSoCs (from 4×4 to 8×8 PUs) and GH1/GH2/stop-go(GH1)/stop-go(GH2) results for $TASK_{NC}$ and $TASK_{FC}$ with different task numbers (60 and 80) on 4×4 to 8×8 MPSoC.

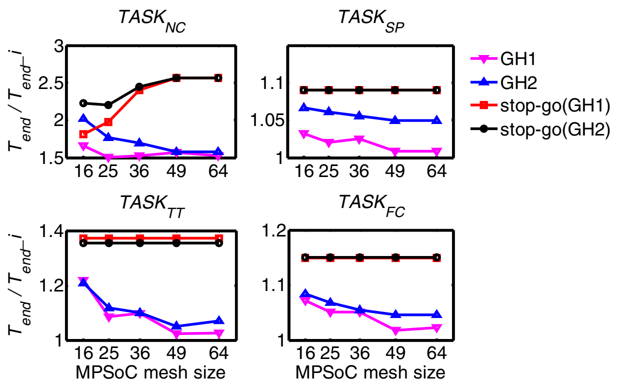

Fig. 9. GH1/GH2/stop-go(GH1)/stop-go(GH2) methods on 40 task scheduling on 4×4~8×8 MPSoC.

Both GH1 and GH2 methods achieve impressive T_{end} improvement compared with their corresponding stop-go methods: from 5.8% to 117.1% and from 2.3% to 159.0% of $T_{end\_}i$, respectively. Especially, the improvement is obvious for task structures that many tasks run in parallel like $TASK_{NC}$. In Fig. 9 and Fig. 10, more PUs will lead to larger T_{end} improvement, but the trends are different for different task structures. For $TASK_{FC}$ and $TASK_{SP}$ in which tasks are highly correlated, the improvement is limited. For $TASK_{NC}$ with a larger solution space, T_{end} improvement increases quickly until the task number is not large enough to exert the advantage of a larger PU number.

Table 3. Simulated Annealing/Static Scheduling+On-Line Adjustment/Greedy Heuristic 1/stop-go(GH1) methods on 4×4 MPSoC

Task#	Task structure	Tend_i	Static(SA)			SSOLA			GH1			Stop-go(GH1)		
			Tend	CP	PT	Tend	CP	PT	Tend	CP	PT	Tend	CP	PT
6	NC	19144	19544	4	12	21819	12	12	20544	8	12	23744	30	12
8	NC	13197	13997	14	16	17313	33	16	15797	20	16	19397	56	16
10	NC	16858	18058	28	18	20496	45	18	20658	40	20	24658	90	20
12	NC	18825	20025	42	20	22648	75	22	22625	64	24	28225	132	24
20	NC	17510	21112	64	28	24194	151	30	24910	171	32	30110	240	32
30	NC	18082	34669	183	42	39042	333	46	29924	154	32	32668	240	32
40	NC	20382	46392	238	54	54559	386	52	33834	125	32	37015	240	32
40	SP	68875	74084	120	64	84797	172	66	71075	20	16	75075	56	16
40	TT	57501	64395	164	58	74243	372	62	70101	287	52	78901	450	54
40	FC	73239	76314	98	64	88090	300	68	78540	40	28	84140	114	28
60	NC	21819	65871	517	86	80809	735	80	48567	159	32	51435	240	32
60	SP	98833	112754	206	110	130163	524	110	101433	20	16	105033	56	16
60	TT	70513	90672	421	88	107019	485	90	82515	221	40	94026	540	56
60	FC	114088	127358	150	100	140506	548	108	119146	56	36	128189	156	36
80	NC	20632	78089	722	102	104635	462	106	59050	131	32	61428	240	32
80	SP	131696	150433	296	144	178381	873	142	134296	20	16	137896	56	16
80	TT	72606	109376	740	124	130751	527	116	91144	189	36	95936	278	36
80	FC	131696	154669	282	138	177013	949	144	136696	86	52	152296	252	52

CP and PT of GH1 is 33.8% to 100% and up to 11.1% less than CP and PT of stop-go(GH1) method, respectively. Meanwhile, CP and PT of GH2 is 30.8% to 100% and up to 38.3% less than CP and PT of stop-go(GH2) method, respectively.

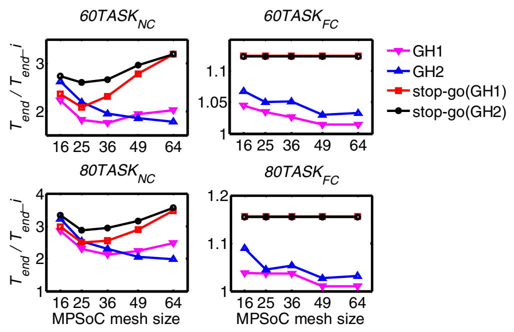

Fig. 10. GH1/GH2/stop-go(GH1)/stop-go(GH2) methods on 60/80 task scheduling on 4×4~8×8 MPSoC.

T_{end} improvement of GH1 is better than that of GH2 except five cases (40 $TASK_{TT}$ on 4×4 MPSoC, 60 and 80 $TASK_{NC}$ on 7×7 and 8×8 MPSoC). The T_{end} improvement of GH1 comparing with GH2 is up to 40.4% of T_{end}_i, and the degradations in the five cases are 1%, 8.8%, 17.9%, 23.9% and 50.2% of T_{end}_i in-order. Comparing with GH2, the benefit of GH1 is that the longer task will start earlier, however the expense is that the longer task may be interrupted more frequently. When the PU number is large enough, T_{end} of stop-go(GH1) is closed to that of stop-go(GH2).

Our experiments above show that for different kinds of tasks with large task number, GH is more efficient than stop-go method on 4×4 to 8×8 MPSoC. The improvement of GH is increasing along with the increase of PU number. For most task test benches, GH1 gets better performance than GH2.

8. Conclusions

In this paper, we for the first time formulate an on-line task scheduling problem with the consideration of power gating induced P/G noise based on our detailed P/G noise analysis platform for MPSoC. An efficient on-line Greedy Heuristic algorithm that adapts well to run-time variations and real-time decision requirement is proposed to reduce noise protection penalty and improve MPSoC performance. The experiment results show that the GH algorithm can achieve on average 26% performance improvement together with on average 73% noise protection penalty saving compared with the corresponding stop-go method. Impacts on MPSoC performance of considering different factors during on-line scheduling decisions are also studied.

References

[1] F. Mohamood, M. Healy, S. K. Lim, and H.-H. Lee, "Noise-direct: A technique for power supply noise aware floorplanning using microarchitecture profiling," *ASP-DAC'07*, pp. 786–791, Jan. 2007.
[2] M. Healy, F. Mohamood, H.-H. Lee, and S. K. Lim, "A unified methodology for power supply noise reduction in modern microarchitecture design," *ASP-DAC'08*, pp. 611–616, March 2008.
[3] K. Shi and D. Howard, "Challenges in sleep transistor design and implementation in low-power designs," *DAC'06*, pp. 113–116, July 2006.
[4] S. Kim, S. Kosonocky, and D. Knebel, "Understanding and minimizing ground bounce during mode transition of power gating structures," *ISLPED '03*, pp. 22–25, Aug. 2003.
[5] J. Gu, H. Eom, and C. Kim, "A switched decoupling capacitor circuit for on-chip supply resonance damping," *IEEE Symposium on VLSI Circuits*, pp. 126–127, June 2007.
[6] H. Jiang and M. Marek-Sadowska, "Power-gating aware floorplanning," *ISQED'07*, pp. 853–860, March 2007.
[7] A. Davoodi and A. Srivastava, "Wake-up protocols for controlling current surges in mtcmos-based technology," *ASP-DAC*, vol. 2, pp. 868–871, Jan. 2005.
[8] A. Ramalingam, A. Devgan, and D. Z. Pan, "Walk-up scheduling in mtcoms circuits using successive relaxation to minimize ground bounce," *J. of Low Power Electronics*, vol. 3, no. 1, pp. 1–8, 2007.
[9] H. Jiang and M. Marek-Sadowska, "Power gating scheduling for power/ground noise reduction," *DAC'08*, pp. 980–985, June 2008.
[10] Y. Zhang, X. Hu, and D. Chen, "Task scheduling and voltage selection for energy minimization," *DAC'02*, pp. 183–188, 2002.
[11] J. Liu, P. Chou, N. Bagherzadeh, and F. Kurdahi, "Power-aware scheduling under timing constraints for mission-critical embedded systems," *DAC'01*, pp. 840–845, 2001.
[12] J. Hu and R. Marculescu, "Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints," *DATE'04*, vol. 1, pp. 234–239 Vol.1, Feb. 2004.
[13] P. Rong and M. Pedram, "Power-aware scheduling and dynamic voltage setting for tasks running on a hard real-time system," *ASP-DAC'06*, Jan. 2006.
[14] A. Coskun, T. Rosing, and K. Whisnant, "Temperature aware task scheduling in MPSoCs," *DATE '07*, pp. 1–6, April 2007.
[15] A. Coskun, T. Rosing, K. Whisnant, and K. Gross, "Temperature-aware MPSoC scheduling for reducing hot spots and gradients," *ASPDAC*, pp. 49–54, March 2008.
[16] T. Chantem, R. Dick, and X. Hu, "Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs," *DATE '08*, pp. 288–293, March 2008.
[17] V. J. Reddi, M. S. Gupta, G. Holloway, M. D. Smith, G.-Y. Wei, and D. Brooks, "Voltage emergency prediction: a signature-based approach to reducing voltage emergencies," *HPCA-15*, Raleigh, NC, February, 2009.
[18] M. S. Gupta, K. K. Rangan, M. D. Smith, G.-Y. Wei, and D. Brooks, "DeCoR: a delayed commit and rollback mechanism for handling inductive noise in microprocessors, " *HPCA-14*, Salt Lake City, UT, February 2008.
[19] A. Todri, M. Marek-Sadowska, and J. Kozhaya, "Power supply noise aware workload assignment for multi-core systems," *ICCAD'08*, pp. 330–337, Nov. 2008.
[20] K. Shakeri and J. Meindl, "Compact physical IR-drop models for chip/package co-design of gigascale integration (GSI)," *Electron Devices, IEEE Transactions on*, vol. 52, no. 6, pp. 1087–1096, June 2005.
[21] S. Pant and E. Chiprout, "Power grid physics and implications for cad," *DAC'06*, pp. 199–204, 2006.
[22] G. Huang, D. Sekar, A. Naeemi, K. Shakeri, and J. Meindl, "Compact physical models for power supply noise and chip/package co-design of gigascale integration," *Electronic Components and Technology Conference*, pp. 1659–1666, May 29-June 1 2007.
[23] ITRS 2007. [Online]. Available: http://www.itrs.net/
[24] Nanoscale Integration and Modeling (NIMO) Group, ASU. Predictive Technology Model (PTM). [Online]. Available: http://www.eas.asu.edu/˜ptm/
[25] Nangate Open Cell Library. [Online]. Available: http: //www. opencelllibrary.org
[26] K. Pruhs et al, "Handbook of scheduling algorithms, models, and performance analysis," 2004 by CRC Press.