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
1Department of Electronics Engineering, Tsinghua University, Beijing, China
2Hong 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 makes 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 (PUs), 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 with 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 multicore 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).
2. P/G Network Modeling of MPSoC

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

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 \( R_{\text{seg}} \) and an inductance \( L_{\text{seg}} \) (Fig. 2). \( C_L \) 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.

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. Howeve...
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.

\[
\begin{array}{c|c|c|c}
\text{Attackers} & \text{Off} & \text{Settle} & \text{Active} \\
\hline
\text{Victims} & \text{Idle} & \text{Settle} & \text{Active} \\
\end{array}
\]

Fig. 4. Timing of a power on event.

\[T_{\text{clkoff}}\text{ and }T_{\text{clkon}}\text{ are the time needed to clock gate a PU and to \underline{wake it up} from the clock gated state, respectively. }T_{\text{settle}}\text{ and }T_{\text{offsettle}}\text{ are the settle time for a PU to power on and power off. In order to ensure the reliability of M\text{PSoC, here: }T_{\text{settle}}\geq\max\{T_{\text{settle}}, T_{\text{safe}}, T_{\text{settle}}\}, \forall p, q \in PU, q \neq p\}, T_{\text{offsettle}}=\max\{T_{\text{settle}}, T_{\text{safe}}, \forall p, q \in PU, q \neq p\}. \]

\[T_{\text{ON}}\text{ and }T_{\text{OFF}}\text{ are the noise protection time penalty for a victim PU when an attacker powers on and off respectively, where }T_{\text{ON}}=T_{\text{clkoff}}+T_{\text{settle}}+T_{\text{clkon}}\text{, }T_{\text{OFF}}=T_{\text{clkon}}+T_{\text{offsettle}}+T_{\text{clkoff}}\text{. Assume that the victim number of PU }p\text{ as an attacker at the moment }t\text{ is }N_{\text{victim}}(p,t)\text{. We define }P_{\text{on}}(p,t)\text{ and }P_{\text{off}}(p,t)\text{ as the total performance penalty to power on and power off attacker }p\text{, respectively, where }P_{\text{on}}(p,t)=T_{\text{ON}} \times N_{\text{victim}}(p,t), P_{\text{off}}(p,t)=T_{\text{OFF}} \times N_{\text{victim}}(p,t)\text{.}

3.2 Motivation Example

We illustrate the benefit of the noise-aware processor level power gating strategy by taking a M\text{PSoC with }4 \times 4\text{ 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-M\text{PSoC.}

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 M\text{PSoC 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 M\text{PSoC}

\textbf{Problem Definition:} Considering a homogeneous M\text{PSoC with }N\text{ PUs, and a set of real-time tasks }\text{Task, determine an assignment of tasks to PUs on-line, such that all task constraints and PU operation constraints are met, the M\text{PSoC performance is optimized and the penalty for safeguarding the victim PUs is minimized.}
i finishes right before task \( j \)'s start: 
\[
\text{toff}(i) \leq \text{ton}(j)-T_{\text{IL}}. 
\]
Here, 
\[
T_{\text{IL}} = T_{\text{initial}} + T_{\text{clkoff}}. 
\]

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 \( T \) (including DAG Link, \( t_{\text{req}}(i), \text{length}(p)(i) \)), the PU set \( P \) (including the impact relation between \( P \) and \( T \)); \( T_{\text{accept}}, T_{\text{offset}}, T_{\text{delay}}, T_{\text{show}} \)

**Output:** \( T_{\text{run}}, P \), 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 PU \( p \) to execute task \( i \);
6. if available \( P \) exists
7. \( D2: \) choose a PU \( p \) to execute task \( i \);
8. if \( \text{maximal} \) \( N \) \( (p) \)
9. \( D3: \) choose a PU \( p \) to power off;
10. if \( \text{maximal} \) \( N \) \( (p) \)
11. assign task \( i \) to PU \( p \);
12. if there are PU's waiting to power off
13. \( t=t+1; \)
14. 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 \( P \) \( p \) to execute the task. If the manager decides to power on an On \( P \) \( p \) to execute task \( i \), the clock off signal is sent to all the victims (if exist) of \( P \) \( p \). 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 \( N_{\text{off}}(p) \).

The factor options adopted by Strategy 1 in GH algorithm (GH1) are as follows. During \( D1 \), GH always runs the task with the earliest release time. If there are several tasks with the earliest \( t_{\text{req}}(i) \), 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 \( N_{\text{off}}(p) \). If there are several Off PUs as candidates, GH1 always powers on the PU with the minimal \( P_{\text{eff}}(p) \). GH1 chooses to always power on the PU with the minimal \( N_{\text{off}}(p) \). 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 safer, 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 \( \text{length}(i)=\text{length}(p)(i) \) (\( \forall i \in T \)), 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 \( P \) \( p \) will be longer than the predicted one, all the active PUs on \( P \) \( p \) 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 \( tf(i) \)).

The original static task scheduling (task starts time and execution time) for tasks after \( tf(i) \) should be postponed by \( T_{\text{initial}}+T_{\text{clkoff}}-tf(i) \).

(2) If the real execution time of task \( i \) on \( P \) \( p \) is shorter than the predicted one, the conservative on-line adjustment strategy is to keep \( P \) \( p \) 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 660 μm×660 μm. Based on the performance results of standard logic cells and D Flip-Flops with different P/G noise, the noise toleration of Vd−Val is set as 100mV, hence Vd+Val defined in Section 3.1 is set to 700mV. Then the corresponding Rimpact for each victim is derived for 4×4 to 8×8 PU mesh MPSoCs. The Tcl and Tchell are set to 100 clock cycles for the time penalty to protect/resume the data and clock-on/off the victim PU. Tend 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) TASKNC: tasks with No Correlation, (2) TASKNC: several Sequential tasks in Parallel, (3) TASKFC: Tree-connected Tasks, (4) TASKFC: 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∈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. Tend,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) ( ∀ i∈ 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. TASKNC with different task numbers on 4×4 MPSoC.](image)

In Fig. 8(a), for TASKNC, 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: Tend(SA)<Tend(GH)<Tend(SSOLA)<Tend(stop-go(GH1)). In Fig. 8(b), for TASKNC, 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: Tend(GH)<Tend(stop-go(GH1))<Tend(SA)<Tend(SSOLA).

PT of on-line algorithms is much less than PT of SA/SSOLA. For large task number, GH1 method achieve impressive Tend improvement comparing with SSOLA: from 7% to 220% of Tend,i, and this improvement is especially large for TASKNC 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 Tend and PT trends generally match TASKNC’s. The results of SSOLA are the worst. However, the relatively value of Tend(SA) to Tend(GH1) and Tend(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 Tend,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 TASKNC and TASKFC with different task numbers (60 and 80) on 4×4×8 MPSoC.

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

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

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

<table>
<thead>
<tr>
<th>Task</th>
<th>Task Structure</th>
<th>Tend 1</th>
<th>Static(SA)</th>
<th>SSOLA</th>
<th>GH1</th>
<th>Stop-go(GH1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>NC</td>
<td>19114</td>
<td>21149</td>
<td>16 4</td>
<td>12</td>
<td>268</td>
</tr>
<tr>
<td>8</td>
<td>NC</td>
<td>13107</td>
<td>13997</td>
<td>14 16</td>
<td>17313</td>
<td>33 16</td>
</tr>
<tr>
<td>10</td>
<td>NC</td>
<td>16658</td>
<td>18058</td>
<td>28 18</td>
<td>20696</td>
<td>45 18</td>
</tr>
<tr>
<td>12</td>
<td>NC</td>
<td>17882</td>
<td>20025</td>
<td>42 20</td>
<td>22648</td>
<td>75 22</td>
</tr>
<tr>
<td>20</td>
<td>NC</td>
<td>17510</td>
<td>21112</td>
<td>64 24</td>
<td>24134</td>
<td>151 30</td>
</tr>
<tr>
<td>30</td>
<td>NC</td>
<td>18002</td>
<td>34669</td>
<td>138 42</td>
<td>39042</td>
<td>333 46</td>
</tr>
<tr>
<td>40</td>
<td>SP</td>
<td>20032</td>
<td>46392</td>
<td>32 34</td>
<td>54559</td>
<td>386 52</td>
</tr>
<tr>
<td>50</td>
<td>SP</td>
<td>68875</td>
<td>74084</td>
<td>120 64</td>
<td>84797</td>
<td>172 66</td>
</tr>
<tr>
<td>70</td>
<td>TT</td>
<td>57732</td>
<td>64395</td>
<td>164 58</td>
<td>74243</td>
<td>372 62</td>
</tr>
<tr>
<td>80</td>
<td>FC</td>
<td>73239</td>
<td>76314</td>
<td>98 68</td>
<td>88909</td>
<td>300 68</td>
</tr>
<tr>
<td>60</td>
<td>NC</td>
<td>21819</td>
<td>65871</td>
<td>517 86</td>
<td>80089</td>
<td>735 80</td>
</tr>
<tr>
<td>80</td>
<td>SP</td>
<td>99583</td>
<td>112754</td>
<td>206 110</td>
<td>130163</td>
<td>172 66</td>
</tr>
<tr>
<td>60</td>
<td>TT</td>
<td>70513</td>
<td>90672</td>
<td>421 88</td>
<td>107019</td>
<td>485 89</td>
</tr>
<tr>
<td>80</td>
<td>FC</td>
<td>114088</td>
<td>127558</td>
<td>150 100</td>
<td>140566</td>
<td>548 108</td>
</tr>
<tr>
<td>80</td>
<td>NC</td>
<td>20632</td>
<td>78089</td>
<td>722 102</td>
<td>104635</td>
<td>462 106</td>
</tr>
<tr>
<td>60</td>
<td>SP</td>
<td>131606</td>
<td>150433</td>
<td>296 144</td>
<td>173831</td>
<td>873 142</td>
</tr>
<tr>
<td>80</td>
<td>TT</td>
<td>72066</td>
<td>109376</td>
<td>740 124</td>
<td>138071</td>
<td>527 116</td>
</tr>
<tr>
<td>60</td>
<td>FC</td>
<td>111606</td>
<td>154669</td>
<td>282 138</td>
<td>177013</td>
<td>949 144</td>
</tr>
</tbody>
</table>

Our experiments above show that for different kinds of tasks and with large task number, GH is more efficient than stop-go method on 4×4 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