Leakage and Performance Aware Resource Management for 2D Dynamically Reconfigurable FPGA Architectures

Siqi Wang, Nam Khanh Pham, Amit Kumar Singh, Akash Kumar
Department of Electrical and Computer Engineering, National University of Singapore, Singapore
{a0077818,phamnamkhanh,amit.singh,akash}@nus.edu.sg

***Abstract*—The variety of applications for field programmable gate arrays (FPGAs) is continuously growing, thus it is important to address power consumption issues during the operation. As technological node shrinks, leakage power becomes increasingly critical in overall power consumption of FPGA. The technique of configuration pre-fetching (loads configurations as soon as possible) adopted to achieve high performance is one of the major reasons of leakage waste since regions containing reconfiguration information cannot be powered down in between the time gap of reconfiguration and execution. In this work, we present a heuristic approach to minimize the leakage power consumption for two-dimensional reconfigurable FPGA architectures. The heuristic scheduler is based on list scheduling and exploits dynamic priority for sorting the tasks into schedule order and a cost function for cell allocation. Farthest placement scheme is adopted for anti-fragmentation purpose. The cost function provides control to compromise between leakage dissipation and schedule length.**

I. Introduction

Field programmable gate arrays (FPGAs) are increasingly used in current technologies. They are popular choices for digital circuits implementation because of their short design cycle, fast speed, and low cost compared to the traditional application specific integrated circuit (ASIC). The run time partial reconfigurable FPGA architecture provides even greater flexibility. Extensive studies have been done for performance improvements of FPGAs. Task execution time and power consumption are popularly discussed topics [1], [2].

The leakage power of transistor does not contribute functionally in the operation of FPGA. It is primarily the result of unwanted sub-threshold current in transistor channel. As technology advances to shrink the feature size, leakage power becomes more and more critical in the overall power consumption of FPGAs. In the recent generation of 65nm, 40nm, and 28nm FPGAs, leakage power consumption constitutes around 50% of the overall power consumption [3]. The introduction of high-threshold voltage sleep transistors enables FPGA to be power gated when not in use to manage active leakage power [4]. The technique that divides FPGA into smaller portions and uses sleep transistors to power gate the region has reduced the leakage power consumption significantly [5].

In addition, for run-time reconfigurable FPGAs, configuration pre-fetching is a well-established approach to hide the reconfiguration time in order to reduce the overall execution time. However, due to the involved task dependencies, tasks can only execute after its predecessors have finished executions. This often introduces a time gap between end of configuration and start of execution, as shown in Figure 1, which results in significant leakage waste. The region containing configuration bits cannot be power gated during this time gap since it needs to preserve the configuration. Therefore, reducing leakage waste in such scenarios is critical and highly desired.

The ability to reconfigure and power-gate (turn-off) part of the FPGA devices brings two types of architecture: one

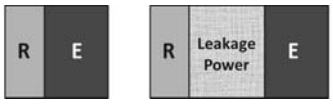

Figure 1: *Leakage power. (R: Reconfiguration, E: Execution)*

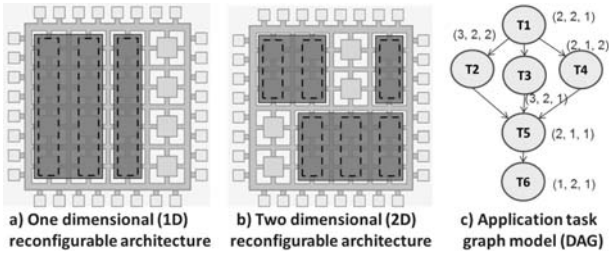

Figure 2: *FPGA architecture and application model*

dimensional reconfigurable and two dimensional reconfigurable architectures, as shown in Figure 2. The shaded area represents the task placement on FPGA . For 1D architecture, the configurable logic blocks (CLBs) are partitioned in forms of columns. Tasks are placed on predefined columns and span the whole column. One column is therefore called one Reconfigurable Unit (RU). The sleep transistors gate the power of the predefined column (RU) to power off when not in use. Recent technologies enable architectures with dynamically controlled power gating capabilities, by which logic clusters can be selectively powered-down at run time [6]. Task placements are therefore not constrained in one dimension. Such architectures allow RUs to be of any combinations of CLBs and are called 2D reconfigurable architectures. Two dimensional architecture is more flexible and can save more space and energy when compared to column-by-column one-dimensional reconfigurable architectures. The minimizations of the leakage waste are studied at different levels. In this work, we focus on the system level solution. In order to reduce the leakage power consumption, we optimize the schedule such that the reconfiguration is placed as close as possible to the execution in time in order to minimize the standby period of the RUs. *We propose list based heuristic scheduling for 2D architectures (LBHS2D) to address the leakage minimization problem for 2D architectures.* The LBHS2D adopts list scheduling skeleton. It implements dynamic priority scheme for task sorting, and cost function with farthest placement scheme for task placement.

The rest of the paper is organized as follows: Related works are reviewed and discussed in Section II, followed by the problem definition in section III. The proposed LBHS2D approach is discussed in Section IV. Experiments and results are discussed in Section V, and a conclusion in Section VI.

II. Related Works

The scheduling problem on 1D reconfigurable architectures is proven to be an NP-complete problem. Therefore, most of the research works aim at developing heuristic approaches. In [7], an efficient technique to schedule real-life applications on

FPGA is proposed, but partial reconfiguration and resource constraint has not been considered. In [2], the authors proposed a two stage approach, in which a schedule is first generated by a performance driven task scheduler, and then refined to reduce the leakage power without compromising the performance. Hsieh et al. [8] proposed an enhanced leakage-aware scheduling algorithm (ELAA) that comprises of three stages of binding the reconfiguration and execution of the task together, assigning each task block a priority, and finally placing with a split aware placer which splits the reconfiguration and execution only when the schedule cannot meet the deadline. Another work proposed in [9] considers a three stage leakage aware approach which exploits a list based scheduling and a post placement step to do the final refining without extension in the scheduled length. However, as aforementioned methods are designed for 1D reconfigurable architectures, applying them in 2D reconfigurable context will require extensive extra considerations due to different reconfiguration behavior.

For scheduling on two-dimensional reconfigurable architectures, heuristic approaches are proposed by researchers to consider improvement in execution time and anti-fragmentation. To the best of our knowledge, none of the prior works focus on leakage power reduction of 2D reconfigurable architecture. Septién et al. [10] proposed a de-fragmentation heuristic approach for 2D architectures which estimates the fragmentation status of FPGA and makes decisions that are effective for de-fragmentation purposes. However, the approach works on incoming tasks which no precedence is presented, and leakage power dissipation is not considered during the placement. Redaelli et al. [11] proposed a heuristic approach called Napoleon which adopts furthest placement and limited de-configuration scheme for anti-fragmentation purpose. Limited de-configuration suggests that all modules are left on the FPGA until other tasks require the space to maximize the chance of module reuse. However, the module reuse technique introduced is not preferable in terms of leakage consideration.

III. Problem Definition

The **targeted architecture** is the two-dimensional (2D) reconfigurable FPGA, as shown in Figure 2 (b). Dynamic partial reconfiguration is supported during run time. The reconfigurable unit can be defined as any combination of CLBs. A task can be deployed on blocks of adjacent RUs. The reconfiguration time required for a task is proportional to the number of required RUs. The reconfigurations are managed by only one configuration controller, which results in only one task reconfiguration at a particular moment of time. For such architecture, it is assumed that unused RUs can be totally powered off by the sleep transistors integrated in the device, and each RU can be independently controlled by sleep transistors [6].

Directed acyclic graph (DAG) is used to represent the tasks of an application. An example of the task graph model is shown in Figure 2 (c). In the DAG, each node represents a task, and an edge indicates the dependency between tasks. The DAG is represented as $< S, p >$, with S being the set of all the tasks and p being the set of dependencies between tasks. Each task in task graph is characterized by 3 parameters (l_i, c_i, r_i):

- l_i : Latency (execution time) of task i;
- c_i : Number of RU columns required for task i;
- r_i : Number of RU rows required for task i.

The reconfiguration time of task is proportional to the area of the task occupied, which is $c_i \times r_i$. The reconfiguration part is mostly dependent on the architectural constraints of only one reconfigurator, while scheduling of execution part depends on the task dependencies. In the scheduling, the communication overhead between tasks is ignored due to two reasons: firstly, tasks on FPGA communicate with each other through a shared memory with equal latency and cost; secondly, the communication overhead is negligible in comparison to reconfiguration and execution time of the tasks.

Scheduling Problem: The problem targeted in this paper considers following set of input, constraints and objective.

- **Input:** The application task graph, FPGA architecture (number of RU columns and RU rows).
- **Constraints:** Task dependencies, 1 reconfigurator.
- **Objective:** Minimize leakage power dissipation, minimize schedule length.

IV. List Based Heuristic Scheduling

List scheduling is one of the fundamental methods that has been widely adopted by researchers because of its simplicity and efficiency. List scheduling in the simplest form contains two stages of operation. In the first stage, it sorts the tasks to be scheduled according to a priority scheme and makes sure that precedence constraints are respected. In the second stage, each task of the list is successively scheduled to a processor block chosen for the task. Algorithm that applies list scheduling technique has the freedom to define the two criteria: the priority scheme for sorting the task and the choice criterion for the processor block, which in FPGA context, is the placement position on FPGA.

The heuristic approach proposed in this work is based on list scheduling skeleton. It includes a leakage aware priority dispatcher in the sorting stage. The placement stage depends on a cost function based on leakage waste and execution time, and farthest placement scheme. The algorithm of the heuristic approach is presented in Algorithm 1.

In the operation of scheduling, tasks in the task graph will be selectively put into two sets: `readyNodeList`, which contains all the tasks to be scheduled, and `scheduledNodeList`, which contains the tasks that have been scheduled. A leakage aware priority dispatcher calculates the priority of tasks in the `readyNodeList` set according to Equation 1. The task with the largest priority is then selected for placement. The placement of the task is decided using Algorithm 2. After the placement, the task is removed from `readyNodeList`, and added into `scheduledNodeList`. The child tasks of this task are then added into `readyNodeList` for next round of schedule.

A. Priority Scheme

A leakage aware priority function is used in the scheduling stage to choose the next task to be placed on the FPGA from the `readyNodeList` set. The priority of each task in the `readyNodeList` set is calculated. The dynamic priority scheme used here ensures that the scheduling process can adapt with the partial schedule of the task graph as well as the current usage of the FPGA. The priority function is defined as follows.

$$F = \alpha BL - \beta LK - \gamma EEST \tag{1}$$

where LK is defined as:

$$LK = C \times R \times (EEST - (ERST + RT)) \tag{2}$$

The terms in the formula are defined as follow:
BL : bottom level of the task, represents the length of the

Algorithm 1: LBHS2D Algorithm

Input: Task graph $< S, p >$
Output: Leakage aware schedule
Put source tasks (tasks with no predecessors) $\{t_i \in S : parent(t_i) = 0\}$ into set **readyNodeList**;
while **readyNodeList** $\neq \emptyset$ **do**
  Calculate priority of tasks in **readyNodeList** (by Equation 1);
  Choose the task t with maximum priority;
  Choose the best cell (c, r) for task t (by Algorithm 2);
  if *Child tasks of* t *are not already added to* **readyNodeList** **then**
    Add the child tasks to **readyNodeList**;
  end
  Remove task t from **readyNodeList**;
  Add task t to **scheduledNodeList**;
end

Algorithm 2: Best Placement Cell Search Algorithm

Input: FPGA, task t
Output: Best placement cell (c, r)
for *each cell:* $c_i \in C, r_i \in R$ **do**
  Put task t on the block begin with (c, r);
  Calculate the cost $cost\_new$ for task t on the block;
  if $cost\_new > cost\_min$ **then**
    Update minimum cost value $cost\_min$;
    Update best placement cell (c, r);
  end
  if $cost\_new = cost\_min$ **then**
    Check the distance of the cells (c, r) to the boundary of FPGA;
    if *new cell* (c, r) *is nearer to the boundary than old cell* **then**
      Update best placement cell (c, r);
    end
  end
end

longest path in the task graph from this task to the exit task;
LK : leakage waste caused by scheduling the task.
C : number of columns required by the task;
R : number of rows required by the task;
EEST : earliest execution start time of the task;
ERST : earliest reconfiguration start time of the task;
RT : the reconfiguration time of the task, which is proportional to $C \times R$;
The metrics to be considered in deciding which task to be scheduled next are the overall execution time and the leakage power consumption. Therefore here we include the following:

- Bottom level (BL): to schedule the tasks with more descendants first.
- Leakage power (LK): not to schedule the tasks those have more leakage power in current FPGA condition.
- Earliest execution time (EEST): to schedule the tasks that can start early first.

α, β, γ are the coefficient that are adjustable for better schedule results for different parallelism of task graphs.

B. Farthest Placement Placer

The algorithm to find the best RUs for a task is presented in Algorithm 2. To find the best cell, the placer checks all the RUs and choose the cell (c, r) proving least cost K, which is computed as follows.

$$K = a \times LK + (1 - a) \times EEST \tag{3}$$

where LK represents leakage power, EEST represents the earliest execution starting time for the task to be placed on the block of cells. a is a coefficient to compromise between amount of leakage power and scheduled length which can be set to any value between 0 and 1. Large value of a will result in more concentration of leakage power minimization in the scheduling, while a small value of a will concentrate more on the scheduled length optimization. This parameter gives designer flexibility to compromise between leakage power dissipation and scheduled length.

The algorithm checks all the blocks of the task size and find the block with the least cost K. If more blocks are available with the same least cost, the algorithm chooses the block that is nearer to the boundary of the FPGA. This farthest placement criterion is the anti-fragmentation technique used to facilitate better future placements such as easier fitting of larger tasks into the middle part of FPGA.

V. Experiments and Results

A series of task graphs are generated and scheduled by our proposed approaches in order to evaluate the quality of obtained results. The approaches are compared with existing performance driven scheduler (PDA) proposed in [1], Enhanced Leakage Aware Algorithm (ELAA) proposed in [8], and the Napoleon (NAP) heuristic proposed in [11]. The experiments are designed to test the performance of the scheduler; therefore no hard deadline is specified in the experimental settings.

PDA [1] algorithm considers only the scheduled length, while ELAA [8] makes sure the minimum leakage dissipation when meeting the deadline constraints. Since no deadlines are specified in the context, ELAA will always return zero leakage dissipation. The parameter a in Equation 3 is set to different values in the experiments to explore the trade-offs between scheduled length and leakage power dissipation. NAP algorithm proposed in [11] exploits limited de-configuration method which introduces large amount of leakage dissipation. Therefore the implementation of the algorithm omits the limited de-configuration scheme for fair comparison.

The algorithms are implemented in Java language, and the experiments are performed on an Intel Core i5 2.30 GHz CPU with 4GB RAM. A series of synthetic task graphs are generated by the TGFF tool [12]. 5 sets of task graphs with 10 task graphs in each set are generated. The number of tasks in a task graph in each sets is 10, 20, 30, 40 and 50. The tasks have l_i in range of 15 ± 10 time units, c_i in range of 4 ± 3 cells and r_i in range of 3 ± 2 cells. In order to see the effectiveness of approaches to real-life applications, task graphs of JPEG encoder [1], MPEG4 decoder [13] and MP3 decoder [14] are also considered with their specifications provided in respective references.

The comparison of different approaches are mainly based on the overall scheduled length and the leakage power dissipation. The scheduled length is measured in time units, and the leakage dissipation is measured in energy units. One energy unit is defined as the leakage power dissipation of 1 cell (RU) during 1 time unit. The leakage power of the task is calculated as defined in equation 2. The overall leakage power of a task graph is the sum of all individual tasks. The leakage power of the task set is the average of all the task graphs in the set.

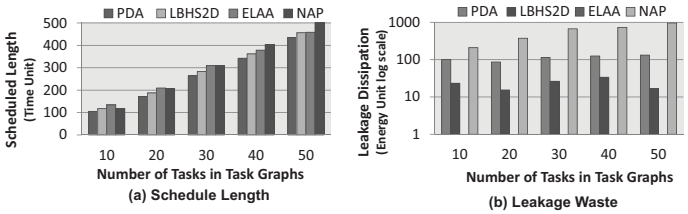

Figure 3: *Scheduled length and leakage dissipation comparison*

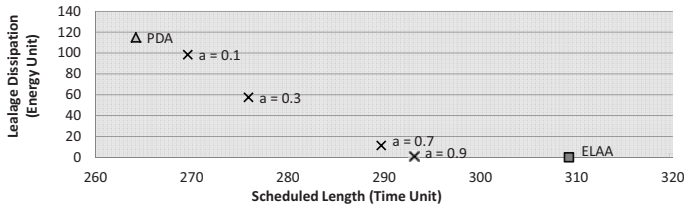

Figure 4: *Leakage dissipation and scheduled length trade-offs*

A. Schedule Length and Leakage Dissipation

Figure 3 (a) shows schedule length for application sets for an FPGA having 10 row and 10 columns of RUs. It can be observed that as the number of tasks in the task graph increases, the overall scheduled length increases. Further, LBHS2D provides scheduled length in between best performance scheduler (PDA) and best leakage scheduler (ELAA). Figure 3 (b) shows corresponding leakage results. It can be observed that LBHS2D reduces the leakage waste significantly compared to PDA. On an average, LBHS2D reduces leakage waste by 33.72% when compared to PDA.

B. Leakage Dissipation and Scheduled Length Trade-offs

The parameter α in Equation 3 gives the designers flexibility to control over the scheduled length and leakage power dissipation. Figure 4 shows the results of LBHS2D for different values of α, and existing approaches PDA and ELAA. The results shown are for task graphs of 30 tasks on an FPGA of size 10×10. The figure shows the experimental Pareto-front, which indicates that designers can choose different values of α to achieve different objectives. Additionally, it can be observed that when parameter α approaches 0.9, the LBHS2D algorithm can achieve zero leakage power dissipation with scheduled length shorter than ELAA. Similar behavior has been observed for other application sets.

C. Real-life Applications Case Study

Table I shows schedule length and leakage dissipation results for real-life applications JPEG encoder, MPEG4 decoder and MP3 decoder when different approaches are employed. The shown results are for an FPGA size of 6×6. The parameter α is set to 0.5 in our LBHS2D approach. It can be observed from the table ELAA always provides results with no leakage dissipation, but at the cost of higher schedule length. The LBHS2D provides a good compromise between schedule length and leakage, and low leakage in most of the cases. Therefore, based on the requirements to trade between leakage and performance, one can employ LBHS2D to achieve good values for leakage dissipation and schedule length.

VI. Conclusion

In this work, we presented a heuristic approach to tackle the leakage power minimization problem on two dimensional reconfigurable FPGAs. The proposed LBHS2D approach includes a leakage aware priority function at the scheduling stage and cost function with farthest placement scheme at the placement stage. The experimental results indicate effectiveness of the proposed approach. We can also see from the results that further improvement of the approaches is possible. In the future, we plan to explore on post placement adjustment of the approach to further reduce the leakage power consumption by taking fixed schedule length.

Application	Performance	PDA	LBHS2D	ELAA	NAP
JPEG	Scheduled Length	23	22	25	28
	Leakage dissipation	3	2	0	16
MPEG4	Scheduled Length	37	47	47	45
	Leakage dissipation	25	0	0	37
MP3	Scheduled Length	44	53	53	59
	Leakage dissipation	30	1	0	40

Table I: *Leakage waste and scheduled length for real-life applications*

References

[1] S. Banerjee, E. Bozorgzadeh, and N. Dutt, "Physically-aware hw-sw partitioning for reconfigurable architectures with partial dynamic reconfiguration," in *Proceedings of Design Automation Conference (DAC)*, 2005, pp. 335–340.

[2] P.-H. Yuh, C.-L. Yang, C.-F. Li, and C.-H. Lin, "Leakage-aware task scheduling for partially dynamically reconfigurable FPGAs," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 14, no. 4, p. 52, 2009.

[3] A. Nafkha, J. Palicot, P. Leray, and Y. Louët, "Leakage power consumption in fpgas: thermal analysis," in *Wireless Communication Systems (ISWCS), 2012 International Symposium on*. IEEE, 2012, pp. 606–610.

[4] J. W. Tschanz, S. G. Narendra, Y. Ye, B. A. Bloechel, S. Borkar, and V. De, "Dynamic sleep transistor and body bias for active leakage power control of microprocessors," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 11, pp. 1838–1845, 2003.

[5] A. Gayasen, Y. Tsai, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and T. Tuan, "Reducing leakage energy in FPGAs using region-constrained placement," in *Proceedings of International Symposium on Field Programmable Gate Arrays (FPGA)*, 2004, pp. 51–58.

[6] A. A. Bsoul and S. J. Wilton, "An FPGA architecture supporting dynamically controlled power gating," in *Proceedings of Field-Programmable Technology (FPT)*, 2010, pp. 1–8.

[7] A. K. Singh, A. Kumar, T. Srikanthan, and Y. Ha, "Mapping real-life applications on run-time reconfigurable NoC-based MPSoC on FPGA," in *Proceedings of Field-Programmable Technology (FPT)*, 2010, pp. 365–368.

[8] J.-W. Hsieh, Y.-H. Chang, and W.-L. Lee, "An enhanced leakage-aware scheduler for dynamically reconfigurable FPGAs," in *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2011, pp. 661–667.

[9] N. K. Pham, A. K. Singh, and A. Kumar, "A Multi-stage Leakage Aware Resource Management Technique for Reconfigurable Architectures," in *Proceedings of Great Lakes Symposium on VLSI (GLSVLSI)*, 2014, pp. 63–68.

[10] J. Septién, H. Mecha, D. Mozos, and J. Tabero, "2d defragmentation heuristics for hardware multitasking on reconfigurable devices," in *International Parallel and Distributed Processing Symposium (IPDPS)*, 2006, pp. 7–pp.

[11] F. Redaelli, M. D. Santambrogio, and S. O. Memik, "An ILP formulation for the task graph scheduling problem tailored to bi-dimensional reconfigurable architectures," *International Journal of Reconfigurable Computing*, vol. 2009, p. 7, 2009.

[12] R. P. Dick, D. L. Rhodes, and W. Wolf, "Tgff: task graphs for free," in *Proceedings of Hardware/software codesign*, 1998, pp. 97–101.

[13] P. Kumar and L. Thiele, "Thermally optimal stop-go scheduling of task graphs with real-time constraints," in *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2011, pp. 123–128.

[14] J. Cong and K. Gururaj, "Energy efficient multiprocessor task scheduling under input-dependent variation," in *Proceedings of Design, Automation and Test in Europe (DATE)*, 2009, pp. 411–416.