Microprocessors and Microsystems 47 (2016) 231–243

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Leakage aware resource management approach with machine learning optimization framework for partially reconfigurable architectures

Nam Khanh Pham[a,b,*], Akash Kumar[c,*], Amit Kumar Singh[d], Mi Mi Aung Khin[b]

[a] *Department of Electrical and Computer Engineering, Faculty of Engineering, NUS, Singapore*
[b] *Data Storage Institute, A*STAR, Singapore*
[c] *TU Dresden, Center for Advancing Electronics Dresden (cfaed), Germany*
[d] *School of Electronics and Computer Science, University of Southampton, UK*

A R T I C L E   I N F O

Article history:
Received 22 December 2015
Revised 30 April 2016
Accepted 28 September 2016
Available online 14 October 2016

Keywords:
Scheduling
Mapping
Resource management
Design space exploration
Machine learning

A B S T R A C T

Shrinking size of transistors has enabled us to integrate more and more logic elements into FPGA chips leading to higher computing power. However, it also brings a serious concern to the leakage power dissipation of the FPGA devices. One of the major reasons for leakage power dissipation in FPGA is the utilization of prefetching technique to minimize the reconfiguration overhead (delay) in Partially Reconfigurable (PR) FPGAs. This technique creates delays between the reconfiguration and execution parts of a task, which may lead up to 38% leakage power of FPGA since the SRAM-cells containing reconfiguration information cannot be powered down. In this work, a resource management approach (RMA) containing *scheduling, placement* and *post-placement* stages has been proposed to address the aforementioned issue. In scheduling stage, a leakage-aware priority function is derived to cope with the leakage power. The placement stage uses a cost function that allows designers to determine the desired trade-off between performance and leakage-saving. The post-placement stage employs a heuristic approach to close the gaps between reconfiguration and execution of tasks, hence further reduce leakage waste. To further examine the trade-off between performance (schedule length) and leakage waste, we propose a framework to utilize the Genetic Algorithm (GA) for exploring the design space and obtaining Pareto optimal design points. Addressing the time-consuming limitation of GA, we apply Regression technique and Clustering algorithm to build predictive models for the Pareto fronts using a training task graph dataset. Experiments show that our approach can achieve large leakage savings for both synthetic and real-life applications with acceptable extended deadline. Furthermore, different variants of the proposed approach can reduce leakage power by 40–65% when compared to a performance-driven approach and by 15–43% when compared to state-of-the-art works. It's also proven that our Machine Learning Optimization framework can estimate the Pareto front for new coming task graphs 10x faster than well-established GA approach with only 10% degradation in quality.

1. Introduction

Field-programmable gate arrays (FPGAs) are promising candidates for digital circuit implementation because of their growing density and speed, short design cycle, and steadily decreasing cost. Furthermore, most of the FPGA devices nowadays can be partially reconfigured at run time, i.e., a configuration can be loaded into part of the device while the rest of the system continues operating. This feature obviously provides greater flexibility and more powerful computing ability. However, these advantages come with additional problems related to reconfiguration time and power dissipation.

A drawback of FPGA due to its hardware redundancy is its inefficiency in term of power consumption when compared to ASIC components [1,2]. In practice, an FPGA circuit implementation may use only a fraction of the hardware resource but the power is dissipated in both the used and the unused components. The power consumption in FPGA includes static (leakage) and dynamic power and their contribution into the total power consumption heavily depends on the circuit technology. Beyond 65 nm technology, leakage power becomes an increasingly dominant component of total power dissipation [3]. This has motivated us to focus our work on reducing the leakage power dissipation.

* Corresponding author.
E-mail address: phamnamkhanh@u.nus.edu (N.K. Pham).

http://dx.doi.org/10.1016/j.micpro.2016.09.012
0141-9331/© 2016 Elsevier B.V. All rights reserved.

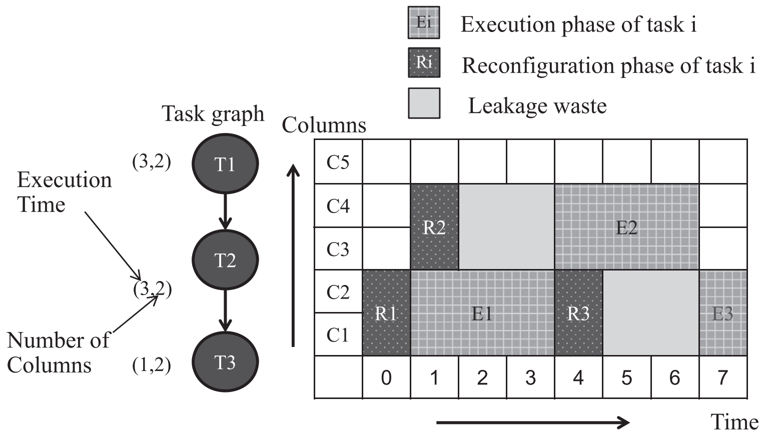


Fig. 1. Example of leakage waste caused by prefetching technique.

Configuration prefetching [4] is a widely adopted technique for reducing the reconfiguration delay in Partially Reconfigurable (PR) FPGA. In prefetching, a task is loaded into the FPGA as soon as possible and this may result in overlap between the configuration part of the waiting task (to be executed) with the execution part of operating tasks, facilitating for reduced reconfiguration overhead (time). However, even after the task is loaded (prefetched), it may not execute and has to wait until few other tasks complete due to involved dependencies. Such waiting introduces delays between the configuration and execution part of the same task. During the delay interval, the SRAM-cells of the FPGA (containing bits of the waiting task to be executed) cannot be powered down to avoid the loss of configuration data from the cells. Therefore, the cells dissipate a significant amount of power.

Motivational Example: Fig. 1 presents an example to demonstrate aforementioned issues. In this example, the task graph on the left-hand side is scheduled on an FPGA platform with prefetching technique. During the interval between R3 and E3, the logic blocks of columns 1 and 2 can be powered down to remove leakage wastes. However, since the SRAM-cells of these columns cannot be powered down as the configuration data will be lost, they consume a considerable amount of power. As SRAM cells leakage contributes ≈38% to FPGA leakage [5], reducing FPGA SRAM leakage is of paramount importance.

In order to reduce leakage, a scheduling approach needs to be developed aiming at allocating reconfiguration and execution parts as close as possible while keeping task dependencies, timing and architecture constraints into account. Several works have been proposed to solve this problem [6,7]. However, these works attempt to address the leakage problem in a single phase of the resource management process (details in later sections). As a result, the leakage power cannot be significantly reduced. It has also been observed that there exists a trade-off between leakage waste and performance [6]. However, the trade-off analysis by employing the existing approaches is not efficient. A high degradation in performance is noticed in order to achieve small amount of leakage savings. To tackle the problem in a comprehensive perspective towards achieving high leakage reductions, we propose a multi-stage **Leakage-aware resource management approach (RMA)** consisting of three stages. Our main contributions to each stage are as follows:

- **Scheduling:** A list-scheduling algorithm has been developed with a specific priority function that is customized for addressing the leakage power reduction.
- **Placement:** A cost function has been derived for the placement stage to further reduce the leakage power. This function provides designers a flexibility to manage the trade-off between performance and leakage waste.
- **Post-placement:** A post-placement heuristic has been proposed to improve the scheduling results (leakage savings) from previous stages.

As our multi-stage Leakage-aware RMA utilizes two cost functions in scheduling and placement stage with various parameters, these parameters form a multidimensional design space with 2 objectives on performance and leakage saving. To further examine the trade-off between these two objectives, we propose an Optimization Framework with Genetic Algorithm to help the designers to efficiently traverse the design space and generate a set of points that are superior in one of the objective dimensions. These points form the Pareto front, which is the Holy Grail for system designers since it not only provides the insight into the trade-off between different objectives but also allows them to choose the most efficient design for different purposes. However, the process of traversing the design space with GA is usually very time-consuming due to the exponential increase in the number of design points with the dimension of the space, which is the number of coefficients in the priority/cost functions. In attempting to solve the time consuming problem of GA optimization, we develop a Machine Learning (ML) component for our Optimization Framework that can accurately estimate the Pareto fronts of new incoming tasks in a fraction of time when compared to GA approach. To achieve such a superior performance, our ML component utilizes Linear Regression to build predictive models for Pareto fronts from a training set of task graphs (TG) at training phase and applies these predictive models together with Density-based Clustering algorithm at prediction phase. Main components of our **ML Optimization Framework** are summarized as follows:

- A comprehensive framework for integrating GA and ML techniques to optimize our **Leakage-aware RMA**: from generating data to building predictive models and predicting Pareto fronts for new TGs;
- A Linear Regression model describing the dependency between the range of Pareto front and TGs features;
- A Density-base Clustering Algorithm to generate near-Pareto-optimal design points.

Paper Organization: Section 2 presents state-of-the-art related to leakage power reduction and existing works on GA and ML techniques in scheduling domain. Section 3 provides the targeted FPGA architecture, application model and problem definition. Our **Leakage-aware RMA** are presented in Section 4. The details of our **ML Optimization Framework** are presented in Section 5. In Section 6, experimental results are reported and Section 7 provides the conclusion.

2. Related work

There are various techniques reported in literature to reduce the leakage power of FPGAs. At*architecture level*, Calhoun et al. [8] introduce a fine-grained leakage control scheme using sleep transistors [9]. With the similar idea, Gayasen et al. [10] applied the sleep transistor methods by partitioning FPGA into regions. Fei Li et al. [11] proposed the programmable supply voltage (V_{dd}) in FPGAs. Elements on critical path are provided high supply voltage to ensure high performance, while components on noncritical path are supplied with low voltage and unused part of the device is switched-off. The leakage power of FPGAs significantly depends upon the threshold voltage (V_T) and an approach using high V_T transistors is proposed in [12]. A profound survey of leakage reduction techniques for SRAMs has been provided in [13].

In [14], Hoyer et al. developed an RTL delay model with all the parameters required for *high level* leakage optimization techniques.

At*system level*, works focusing on leakage power problems are fewer than those of the architecture level [15]. Bharadwaj et al. [16] propose a design methodology that groups temporal locality design into cluster. The authors also develop a power state controller for effectively switching the states of the cluster. Addressing the leakage problem from system level as well, Zapater et al. has proposed an empirical model for leakage components and used it to design an energy-efficient control mechanism for servers in data centers [17].

Task graph scheduling for FPGA is an extensively studied topic [18–20]. Most of the scheduling methods for FPGA focus on specific problems related to reconfiguration overhead and defragmentation. Ahmadinia et al. [18] combined scheduling and placement method for 2D FPGA architecture using cluster-based method to improve the performance by 20% and task rejection by 16.2%. Christoph *et al.* [19] integrated an on-line placement into a scheduling algorithm using small tasks first and earliest deadline first techniques. However, they do not take into account prefetching technique and resource constraint due to single reconfiguration controller pertaining to PR FPGA. The first work that considered both prefetching technique and resource constraint was introduced by Banerjee *et al.* [20]. The scheduling and placement models are included with the partitioning stage to form a complete HW-SW co-design approach for PR systems. Adopting the same linear placement model and reconfiguration constraint for Partially Reconfigurable system, Ferrandi et al. [21] proposed an Ant Colony Optimization approach for mapping, scheduling and placing Task Graph on a heterogeneous platform, which might contain different type of cores (CPU, FPGA, DSP ...). While the last two works focus on system level PR platform, Yuh et al. [6] and Hsieh et al. [7] shipped their focus to HW only task graphs to address the leakage power issues for FPGA-only platform.

Yuh et al. [6] first introduced the idea of using scheduling approach to mitigate the leakage issue. The authors utilized the scheduling and placement results from [20] and on top of that they developed a post-placement heuristic to reduce the delays between execution and reconfiguration parts. They also proposed an exact ILP solution to perform the post-placement in order to verify the effectiveness of the heuristic. Since their work tackles the leakage optimization after the tasks are already allocated onto the FPGA, the existing placement results may not allow their approach to significantly eliminate the leakage power. To achieve maximal leakage saving, our work addresses the leakage problem in all phases of the resource management process: *scheduling stage, placement stage* and *post-placement stage*.

With the same model and target, Hsieh et al. [7] introduced another approach to reduce the leakage waste. Their method consists of 3 phases: *binding, priority dispatching* and *split-aware placing*. First, the reconfiguration and execution parts of all tasks are combined together in the binding phase so that the leakage power is minimal. Then, each task is assigned a priority value based on the position of the task in the task graph. Finally, while placing the tasks into FPGA architecture, the split-aware placer checks for the deadline. If the deadline is violated, the placer splits the reconfiguration and the execution phase of the task. One of our main targets is to explore the trade-off between Schedule Length and Leakage Saving, therefore we consider soft-deadline during the scheduling process. However, after getting the trade-off curve, solutions satisfied hard-deadline requirements can be achieved by filtering design points with the Schedule Length smaller than the deadline. While the work in [7] tried to solve the leakage problem in the placement phase only, we propose a more complete solution having multiple stages. Furthermore, the scheduling algorithms in [7] used static priority, which is computed before the actual scheduling process takes place. The static priority is computed based on the characteristic of the task graph and remains unchanged during the scheduling process. In contrast, our algorithm dynamically recalculates the priorities of all available tasks every time a task is allocated onto the FPGA. Therefore, our algorithm updates the current available resource of the FPGA, leading to a better scheduling decision.

Table 1
Comparison of various approaches.

Features	Ref. [20]	Ref. [6]	Ref. [7]	Our work
Scheduling	Performance driven	No	Performance driven	Leakage aware
Placement	Performance driven	No	Leakage aware	Leakage aware
Post placement	No	Leakage aware	No	Leakage aware
Priority of tasks	Dynamic	No	Static	Dynamic

Table 1 summarizes the distinction of our work in comparison to the closely related works reported in the literature. As can be seen, existing works perform leakage aware optimization in scheduling, placement, or post-placement stages, whereas our approach performs optimization in all the stages. Further, unlike most of the approaches that consider static priorities of tasks, our approach considers dynamic priorities.

Genetic algorithm approaches have been used intensively for optimizing scheduling algorithms, especially in cloud computing systems [22]. However, when it comes to multiprocessor systems (MPS) with tight timing requirements, the applications of GA are quite limited because of its time-consuming behavior. Sutar et al. proposed memetic algorithm that combines GA with simulated annealing to solve the scheduling problem of precedence constrained tasks [23]. Towards using GA-based scheduling algorithm with primary-backup scheme to improve the fault-tolerance of real-time MPS, Zarinzad et al. and Samal et al. proposed their frameworks in [24] and [25] respectively. Obviously, none of the above-mentioned studies incorporates ML techniques to solve the time-consuming problem of GA methods in scheduling domain. That unique point makes our work stand out from previous studies, which applies GA approaches for solving scheduling problems. Recently, ML techniques have emerged as a promising and efficient solution for resource management problems. A comprehensive survey on existing learning-based approaches for the same problems on cloud computing systems has been conducted by Hormozi et al. [26]. More specific overview on the direction of energy minimization is presented by Berral et al. [27]. As summarized from these works, the main application of ML techniques in scheduling problem is performance modeling and Quality of Service (QoS) modeling. For performance modeling purpose, the historical data on execution trace of previous applications is used to build predictive models to forecast the performance of new coming applications [28]. On the other hand, the models for QoS are usually built based on the dependency with available resource (CPU, memory, bandwidth ...) and application requirements [29]. These models are then used to assist the scheduler at runtime to efficiently allocate the resources. The second approach applying ML techniques in resource management is to classify applications and make decisions based-on the classification results [30]. Using unsupervised learning techniques such as Reinforcement Learning to build autonomic self-management schedulers is another trend not only in cloud computing [31], but also in digital system design [32]. Amongst above-mentioned body of works, our ML optimization framework is most related to the first application of ML in resource management domain, since we also use regression techniques to build predictive models. However, the differences in purpose and the interaction between scheduling algorithms and ML techniques make our framework unique and novel. While the existing works try to assist the schedulers by predicting the performance or QoS of new applications, our framework tries to model the behavior of schedulers during GA optimization process and build predictive models for the result of that procedure (i.e. Pareto front).

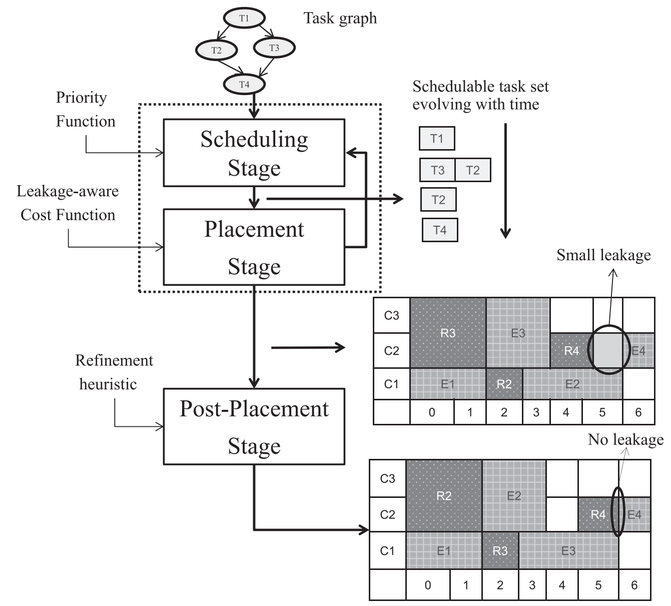


Fig. 2. Multi-stage scheduling scheme.

3. System model and problem definition

The **targeted architecture** used in this work is 1 dimensional (1D) FPGA, where the configurable logic blocks (CLBs) are arranged in fixed vertical columns, and a task occupies an integer number of columns. The basic configuration unit is a column. A task can be deployed on an adjacent set of columns, and the reconfiguration time of the task is proportional to the number of columns. Such an architecture is similar to Xilinx FPGA Virtex family [33]. The device can be configured by a bitstream through configuration ports like JTAG or ICAP. However, both configuration ports are managed by only one configuration controller. Therefore, two different tasks cannot be reconfigured at the same time. Such architectural constraint plays a critical role in the process of scheduling and placement. Moreover, there are 2 main requirements that make the system fully beneficial from our RMA:

- The device needs to support dynamic partial reconfiguration at runtime: a part of the platform can be configured while other parts operate without interruption.
- Another key element realizing the benefits of scheduling algorithm on FPGA is sleep transistors. It is assumed that unused CLBs can be totally powered off by the sleep transistors integrated into the device. Based on this assumption, each column can be independently controlled by a sleep transistor [6].

These conditions constraint the targeted architecture to 1D FPGA only.

Task model: In this work, we consider problem of scheduling only hardware tasks, i.e., a task can be synthesized and implemented on the FPGA platform. In comparison to software tasks, hardware tasks have some additional parameters related to the required hardware area and configuration time. Directed acyclic graph (DAG) is used to represent the task set of an application. An example of the task graph model is presented in Fig. 1. In the DAG, each node u represents a task, while an edge $e(u; v)$ indicates the data dependency between tasks u and v (task v can start only after task u finishes).

A task has two components: reconfiguration and execution. Reconfiguration part is scheduled under the architectural constraint (only one reconfiguration controller) while scheduling of execution part depends on the data dependencies, where a linear task placement model as that of [20] has been adopted. In the scheduling process, the communication overhead between tasks is ignored due to two reasons: *(1)* since all tasks are executed on the same hardware device (FPGA chip), they can communicate with each other through a shared memory (BRAM or DDRAM) with the same latency and cost; and *(2)* this latency is negligible in comparison to runtime reconfiguration overhead (time) and execution time. Similar assumption has been undertaken in previous works [20], [6] and [7], which adopt the same hardware model and application model as ours.

Scheduling Problem

The problem targeted in this paper is an online scheduling problem where the new incoming task graph can be handled at runtime without prior information about it and the scheduling process starts as soon as all tasks of the application (task graph) are read. Following are the set of input, constraints and objective:

- **Input:** The application task graph and FPGA architecture (number of columns, 1 reconfiguration controller and 1D architecture).
- **Constraints:** Task graph dependency for execution parts, reconfiguration controller constraint for reconfiguration part and sequential relation between the reconfiguration and execution parts of the same task.
- **Objective:** Minimize leakage power dissipation because of the delays between the reconfiguration and execution parts, minimize schedule length.

4. Proposed leakage-aware resource management approach

An overview of the proposed resource management approach is provided in Fig. 2. The approach has 3 stages: *Scheduling, Placement* and *Post-placement*. At first, the application task graph is processed iteratively in the first two stages (Scheduling and Placement). In each iteration, the Scheduler will define the next task coming to the Placer by a dynamic priority scheme, which means that the priorities of all the schedulable tasks are changed after each iteration. The Placer then decides the column where the task should be mapped and update the current status of the platform for the Scheduler. Once all the application tasks are mapped, the post-placement stage tries to reduce gaps between configuration and execution parts for all the tasks in order to improve results obtained from earlier stages.

Algorithm 1 Leakage aware task scheduling algorithm.

```
Require: Task graph G=(U,V)
Ensure: Schedule with minimal LK
 1: Put source tasks $\{t_i \in U : \mathbf{pred}(t_i) = \emptyset\}$ into set S
    {S˜– Set of schedulable tasks}
 2: while $S \neq \emptyset$ do
 3:   Calculate priorities of unscheduled tasks in S (by Equation 1)
 4:   Choose the task $t$ with maximum priority
 5:   Choose the best column $C$ for task $t$(by Algorithm 2)
 6:   Schedule task $t$ starting from column $C$
 7:   if child tasks of $t$ are not already added to S then
 8:     Add new available tasks to S
 9:   end if
10:   Remove task $t$ from S
11: end while
```

4.1. Scheduling stage

Algorithm 1 presents our algorithm for the scheduling phase. At each step, all schedulable tasks whose parents have been scheduled are stored in a set of ready task –*S*. Then, the scheduler calculates the dynamic priorities of all tasks in set *S* according to a priority function defined by Eq. (1). Thereafter, it chooses the task with highest priority to pass to the placer. As mentioned in Section 2, we use a dynamic priority function so that the scheduling process can adapt with the current status of the FPGA. Since the priority function has a strong impact on the schedule quality, it is carefully designed to address both leakage saving and performance requirement. The function includes different components that reflect the affection of constraints (FPGA architecture and task graph dependency) as well as optimization targets (leakage saving and schedule length) on scheduling decision. Our priority function is described as follows:

$$F = \alpha BT + \sigma C - \beta EET - \gamma ERT - \mu LK \tag{1}$$

$$LK = C^*(EET - (RT + ERT)) \tag{2}$$

where,

BT:	bottom level of the task that represents the length of the longest path in task graph starting from this task;
EET:	earliest execution time of the task;
ERT:	earliest reconfiguration time of the task;
C:	number of columns required by the task;
RT:	the reconfiguration time of the task;
LK:	leakage waste caused by scheduling the task. The leakage waste is the product of the used columns and the delay between reconfiguration and execution parts.

EET, ERT and *LK* are dynamic factors and are computed in scheduling process based on the current status of the partial schedule. Since these variables are fundamentals for scheduling problem, the details of their calculation can be found in basic textbook about task scheduling, such as [34]. α, β, γ, σ, μ are parameters related to each factor and used to determine the intensity of their impact on the cost function. The signs of elements in the function are given based on their impact on the schedule: tasks requiring larger columns should be placed earlier to increase the space for other tasks; tasks with higher bottom level (close to leaf tasks) should be scheduled first because they strongly affect the schedule length. Additionally, tasks with minimal *EET, ERT* and *LK* should be chosen for the desired optimization objective. As shown in Fig. 2 the output of the scheduling stage is a set of schedulable tasks with the task of the highest priority in the front of the set. This highest priority task is then transferred to Placement Stage to be allocated onto the FPGA. Since we are using a dynamic priority scheme, both the schedulable task set and the priorities of tasks in the set are changed every time a task is placed in FPGA. The feasibility of a task placement is considered while computing its Earliest Execution Time (EET) and Earliest Reconfiguration Time (ERT) for the priority function in **Line 3**, Algorithm 1. Given the current partial scheduling status (for example when task 1 and task 2 are already placed on FPGA), the next task can always be placed on the FPGA at some moment in the future. In case the new task is very large, i.e., it needs lot of space for placement, it can be placed only when earlier placed tasks finish their execution. The EET and ERT capture the moment when the task is feasible to be placed on FPGA platform. Therefore, when a task is transferred from scheduling stage to placement stage, it should be sent along with its EET and ERT hence it will always be feasible to be placed on the FPGA platform from the latest time of EET and ERT onward. The loop from placement stage to scheduling stage ensures the scheduler captures the latest status of FPGA platform after each new task placement so that it can dynamically compute *EET, ERT* and *LK* for the priority function (1) of each task in the Schedulable Set *S*.

4.2. Placement stage

After getting the task with highest priority, the placer applies the steps in Algorithm 2 to allocate the task into physical column(s) of FPGA. When a task comes to this stage, the algorithm scans all the columns to find available positions for the task and for each available position, the cost function is computed. Then, the task is placed into the position with minimal cost value. Here, also the cost function is also designed to optimize for both performance and leakage waste, which is presented as follows:

Algorithm 2 Leakage aware placement algorithm.

```
Require: Task $t$, set of columns P
Ensure: column $C$- with minimal $LK$
1: for each column $c_i \in$ P do
2:   Schedule task $t$ starting from column $c_i$
3:   Calculate cost of placing $t$ on $c_i$ (by Equation 3)
4: end for
5: Choose the column $C$ with minimal cost function
```

$$G = \frac{a}{10} * LK + \left(1 - \frac{a}{10}\right) * EST \tag{3}$$

where, *LK* and *EST* represent leakage power and earliest start time for a placement; *a* is the leakage-schedule length trade-off coefficients, which can be used to provide a balance between the two optimization goals. Therefore, the cost function not only facilitates to reduce the leakage dissipation but also provides designer the ability to manage the trade-off between performance (schedule length) and leakage saving. The trade-off values can be achieved by adjusting the value of *a* in Eq. (3). By increasing the value of *a*, designer can save more leakage power with a longer schedule length.

Fig. 2 demonstrates the placement results from the first 2 stages of our approach. It is expected to have small leakage power as a result of above optimization techniques as shown in the figures.

4.3. Post-placement heuristic

Our post-placement heuristic is presented in Algorithm 3. The heuristic takes task graph & tasks' placement as input and provides optimized placement of tasks so that leakage power due to delays between reconfigurations and executions is further minimized. The heuristic first schedules leaf tasks to maintain the same finish time towards meeting the timing deadline. For each leaf task, it's parent tasks are evaluated for their reconfiguration costs and

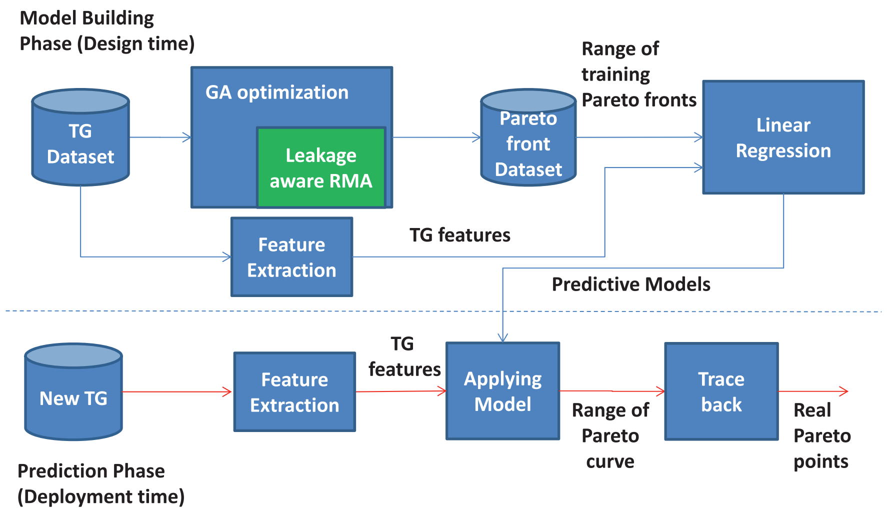


Fig. 3. ML framework.

Algorithm 3 Leakage aware post-placement algorithm.

```
Require: Task graph G=(U,V), Tasks' placement after placement stage
Ensure: Optimized placement of tasks
 1: for each leaf task $t_i \in$ U do
 2:   Schedule configuration and execution of task $t_i$ by considering architectural constraint
 3:   while $parents~of~t_i \neq \emptyset$ do
 4:     Find reconfiguration costs for parent tasks of $t_i$ by Equation 4
 5:     Sort reconfigurations in descending order based on cost
 6:     Schedule reconfigurations considering architectural constraints
 7:     Select parents one by one from maximum to minimum cost as $t_i$
 8:   end while
 9:   Move executions close to reconfigurations if dependencies do not violate
10: end for
```

scheduled by taking architectural constraints into account. The cost is computed as follows

$$C = lw * NC - sw * SP \tag{4}$$

where, NC and SP are the number of occupied columns and range of reconfiguration space, respectively. The lw and sw are the weights to be given to NC and SP respectively, which determine the leakage power dissipation. After all the tasks are scheduled, the executions are tried to place close to the respective reconfigurations if dependencies are not violated. This helps us to achieve placement that contains reconfigurations and executions close to each other as shown in Fig. 2, leading to reduced leakage power.

5. Proposed ML optimization framework

In this section, we provide an overview of the working flow and the general functionality of the components in our ML Optimization framework. We explain how the Genetic Algorithm (GA) and Machine Learning (ML) techniques are utilized to optimize our Leakage-aware RMA. As can be seen from Fig. 3, our framework has two main phases, the Model Building Phase executes at design time, while the other Prediction Phase executes at deployment time (after coming of new TG and before executing Leakage-aware RMA).

Phase 1: Building the predictive models

In the first phase, the Leakage-aware RMA is wrapped by the *GA optimization* process, which takes a bunch of previously generated task graphs (TG) as the input, iterates through their design spaces and generates the optimal Pareto front for each TG. The generated Pareto fronts are stored in a database and fed to the *Linear Regression* block. This module receives historical data, which is the range of Pareto front from previous block, as well as the Features of respective TGs in the TG dataset. From these inputs, it builds Linear Regression models that characterize the dependences of Pareto range on the TG features. The Predictive models output from this module are sent to Phase 2 for use at deployment time. The last component of Model Building Phase is *Feature Extraction* block, which computes the most important metrics of TG such as: number of tasks, maximum bottom level, maximum top level [35], mean of task size, variance of task size, mean of columns, variance of columns. In the training phase, this block processes the TG from historical training set and sends the features to *Linear Regression* block; while in Prediction Phase, it computes features for new TGs and feeds them to the *Applying Model* block. The components of Model Building Phase are further presented in Subsection 5.1.

Phase 2: Prediction at deployment time

The second phase in our ML framework utilizes the results from previous stages to generate the Pareto front for a new TG at deployment time. The first building block in this phase is *Applying Model* component, which takes the Linear Regression models and features of the new TG to estimate the range of Pareto curve. The *Trace back* block produces the real design points on Pareto front using Density-based clustering algorithm. The detail implementation of this phase is discussed in Section 5.2

5.1. Phase 1: Building the predictive models

5.1.1. Generating Pareto front with GA

To apply the GA for our Leakage-aware RMA, we first define the hyper-parameter set fed to the GA as the combination of parameters (α, β, γ, σ, μ) of scheduling priority function (Eq. (1)) and parameter a of placement cost function (Eq. (3)). Varying the hyper-parameter set (α, β, γ, σ, μ, a) gives various solutions in term of schedule length and leakage waste. Thereafter, the GA is used to explore the design space of these parameter set to find the Pareto front in the objectives space. In general, the GA encodes the parameters in the form of chromosome and uses the objectives as criteria to heuristically search for better parameters by iterating

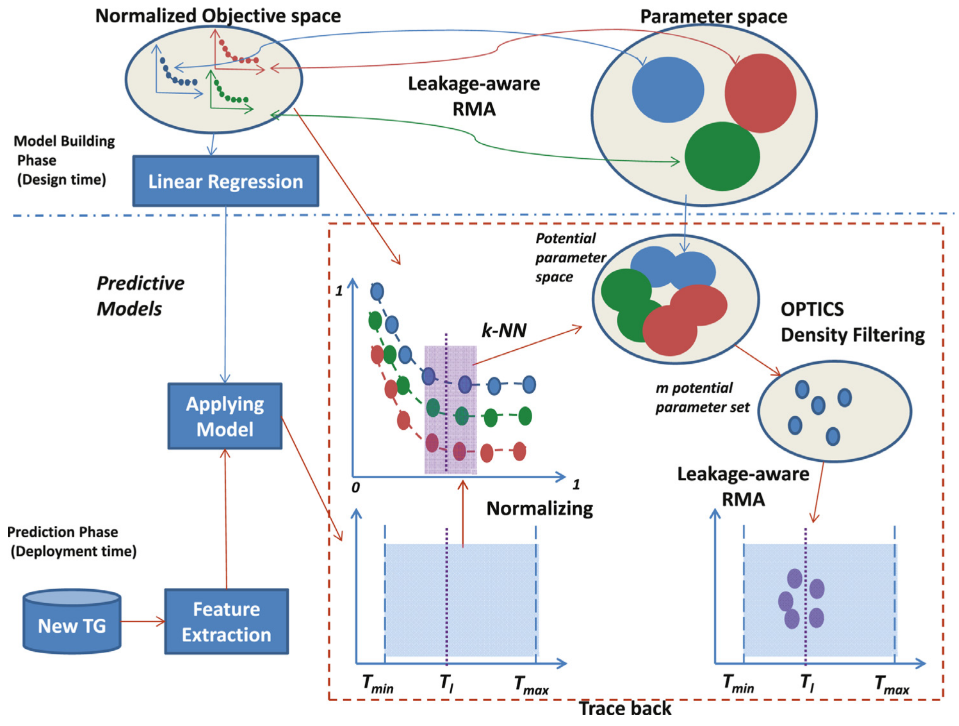


Fig. 4. Details of Traceback module.

from generation to generation. The good parameter sets are transferred through generations by inheritance while the new potential parameter sets are explored through mutation. There are quite a number of different implementations of GA but we use the NSGA II algorithm because of its proven efficiency and popularity [36].

5.1.2. Build linear regression for Pareto's range

From previous training data, we need to build predictive models to capture the dependencies of the range of Pareto curve on the TGs features. That also describes the role of **Linear Regression** (LR) block in our framework. This block takes input from training Pareto curves and associate TG features to generate the LR models for predicting the min, max of Pareto curves. The general formulation of LR model is given in Eq. (5):

$$Y_i = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \epsilon_i \tag{5}$$

where Y_i is the outcome. In our framework it will be the min , *max* of Schedule Length and Leakage Waste of points on Pareto front. X_i refers to the features extracted from TGs such as: number of tasks, maximum bottom level, maximum top level [35], mean of task size, variance of task size, mean of columns, variance of columns. β_i are the coefficients of LR model. In our example, there will be 4 models and coefficient sets in total for estimating the range of Pareto front. The reason behind the choices of TGs features and LR is the trade-off between accuracy and computational complexity. In fact, our simple LR model can well describe the dependency between the TGs features and the outcomes since it can explain more than 95% of the variation in the dataset ($R^2 >= 0.95$).

5.2. Phase 2: applying the ML models for prediction at deployment time

Fig. 4 presents the procedure in the second phase of our framework. On the top, it shows the results from previous phase - the Pareto fronts in the **Normalized Objective space** and the set of parameters in **Parameter space**for obtaining the points on Pareto front, which are necessary input for Traceback Module.

When a new TG comes, its features are extracted and sent to **Applying Model** block to generate an estimated range of potential Pareto curve, which is denoted as $T_{\min}$ and $T_{\max}$ in the objective space. The **Trace back** module is introduced to obtain real Pareto points on the curve. First the estimated range are partitioned into n intervals with $n+1$ knots on the *ScheduleLength* axis (including $T_{\min}$ and $T_{\max}$). Then the procedure presented in the dotted rectangle (**Traceback**) is applied for each knots. Fig. 4 shows the steps implemented for l-th knot T_l ($l = \overline{1, n}$). First, the objective space of *targeted TG* is normalized and put on the same objective space with the normalized Pareto fronts of training TGs.

Fig. 5 explains the reason behind normalization step. It shows an example of different Pareto fronts of 9 TGs before and after normalizing. As can be observed in Fig. 5a, the scale of the Pareto curves have major differences and they couldnot help to add more information for generating the Pareto front of new TG. To overcome this problem and make the Pareto fronts easier to interpret and more uniformly across the TG dataset, we normalize the curves so that all the Pareto fronts fit in the range of [0, 1] for all dimensions of objectives (Schedule length and Leakage Waste). The formulas used in the normalization process are given in Eqs. (6)–(7).

$$T_i = (T_i - T_{min})/(T_{max} - T_{min}) \tag{6}$$

$$E_i = (E_i - E_{min})/(E_{max} - E_{min}) \tag{7}$$

where T_i and E_i denote the Schedule Length and Leakage Waste of the i-th point on the Pareto front. $T_{\max}$, $T_{\min}$ and $E_{\max}$, $E_{\min}$ represent the range of Pareto curve in two objective dimensions.

As can be seen from Fig. 4, after normalizing, k points in the normalized objective space with smallest distance (k-NN) to the vertical line associated with T_l are extracted. Subsequently, the parameter space of these k points are fed from historical data and merged together to form a *potential parameter space*. A clustering algorithm called *Ordering points to identify the clustering structure* (OPTICS) [37] is applied to *potential parameter space* to filter out *m potential parameter sets*, which have the smallest reachability distance [37]. Finally, our Leakage-aware RMA is called for these *m*

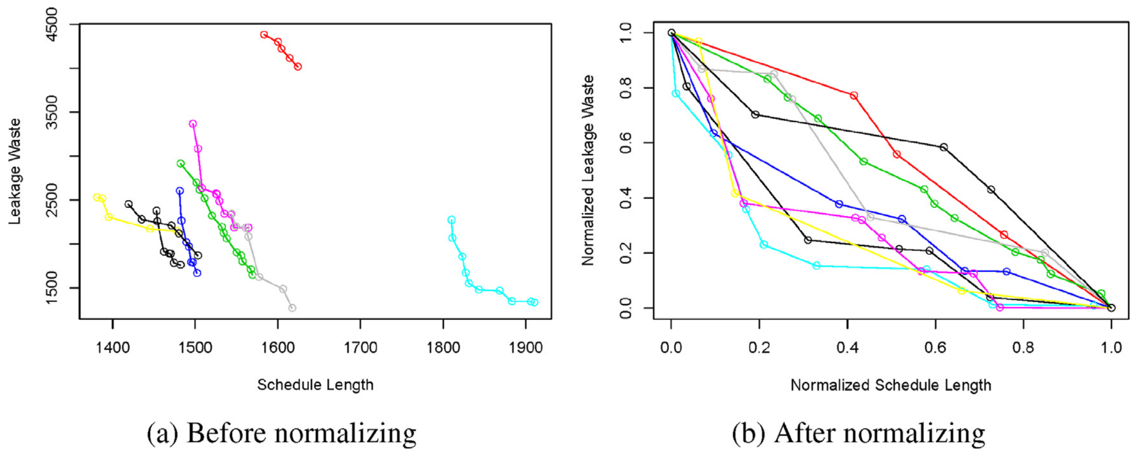


Fig. 5. Pareto fronts of several TGs before and after normalizing.

potential parameter sets to generate the desired points on objective space that are closest to the T_0. The rationale behind the k-NN and OPTICS steps is to extract the most potential parameter sets from the historical parameters space.

The detailed implementation of **Traceback** block is given in Algorithm 4. The first input of this procedure is the Objective Space S_O of all normalized Pareto front $Pareto_i$ of TG i-th in the training set *TrainSet*. The second input is the Parameter Space S_P containing all the Parameter sets P_i associated with Pareto front $Pareto_i$. Another input is the predicted Pareto range $T_{\min}$, $T_{\max}$ of the new coming TG, which is generated by **Applying Model** block. The hyper-parameter n, k, m are respectively the number of knots in interval $[T_{\min}, T_{\max}]$, the number of selected nearest neighbors in *kNN* step and the number of selected potential parameter sets from density filtering step. Especially, the two hyper-parameters n, m have huge decision role on the complexity and performance of the whole framework. Therefore, their impact is examined more thoroughly in the experimental Section. The expected output of this Block is the Pareto front $Pareto_{new}$ of the new TG. The main part of the Algorithm (the For Loop from Line 2–25) is described above and illustrated in the dotted rectangle (**Traceback**) of Fig. 4.

6. Experimental results

6.1. Performance of leakage-aware RSA

A series of experiments are conducted to demonstrate the performance of the proposed Leakage-Aware RSA. Three versions of our scheduling and placement approach with different value of constant a in Eq. (3) ($a = 1$, $a = 2$, $a = 10$) are compared with following existing approaches: performance-driven algorithm (PDA) proposed in [20], Enhanced Leakage Aware Algorithm (ELAA) employed in [7], the ILP and Iterative Refinement (ITE) heuristic approach proposed in [6]. The PDA does not consider the leakage waste in the scheduling process, and has been used as the baseline approach for comparisons. ELAA demonstrates high performance when dealing with the leakage problem [7]. One important target in this work is to examine the trade-off between leakage saving and the schedule length, so no deadline (in terms of schedule length) is set for the trade-off analysis. The results from our post-placement approach are compared to that of [6].

Our algorithm is implemented in Java language and experiments are performed on an Intel Core i7 2.26 GHz CPU with 4GB RAM. The experiments are performed with real-life task graphs and synthetic task sets generated by the TGFF tool [38]. For the synthetic case, five task sets are considered. Each task set contains 10 task graphs with different level of parallelism, which is defined by the *task_degree* option in TGFF tool; each task in the task graph requires 10–50 columns and has the execution time from 1 to 9 time units. The FPGA platform is considered to have a fixed number of columns as 100. For real-life task graphs, JPEG encoder [20], MP3 decoder [39] and MPEG4 decoder [40] are considered with their specifications provided in respective references in order to demonstrate the applicability of our approach for real-life scenarios. The size of the HW implementation is estimated in proportion with their SW implementation in these references.

The criteria of the comparison are schedule length, leakage waste, and the runtime of the algorithms. The schedule length is measured in time unit, while the leakage waste is measured in energy unit, which is the power dissipation of one column during 1 time unit. The leakage waste of a particular task is computed by Eq. (2). The leakage waste of the task graph after scheduling is the sum of leakage waste of all its tasks. For leakage waste of a task set, leakage values of all the contained task graphs are added. Further, as sleep transistors are used to switch-off the unused SRAM cells for each column, the leakage waste for a task before its configuration and after the execution is considered as zero.

6.1.1. Leakage waste and schedule length

Fig. 6 presents the leakage waste and schedule length (in terms of time extension over baseline approach PDA) of all the approaches over the five task sets. The whole bars present the leakage waste obtained after Scheduling and Placement (S&P) stage, while the lower parts of the bars describe the leakage waste after applying Post-Placement (PP) methods. Therefore, for existing approaches, the whole bars describe the leakage waste of PDA methods, and the lower part of each bar is the leakage after post-placement refinement (PDA+ITE or PDA+ILP). The time extension is the extended deadline required for leakage reduction in proportion with the original execution time with no optimization. It is computed by subtracting the schedule length of each approach to the schedule length of the baseline approach (PDA) then dividing to the later to get an relative value in percentage; these values are presented by columns with reversed direction (up to down). The horizontal axis declares notations for different approaches. For example, the first two notations PDA+ILP and PDA+ITE denote two approaches used in [6], where PDA is used in Scheduling and Placement (S&P) phase and either ILP or ITE is used in Post Placement phase.

It can be seen from Fig. 6 that all versions of our approach achieve better leakage saving when compared with the two approaches in [6]. Furthermore, when the number of tasks is large (greater than 10), our approach with $a = 10$ can reach the optimal leakage saving (leakage waste = 0) with smaller extension in time

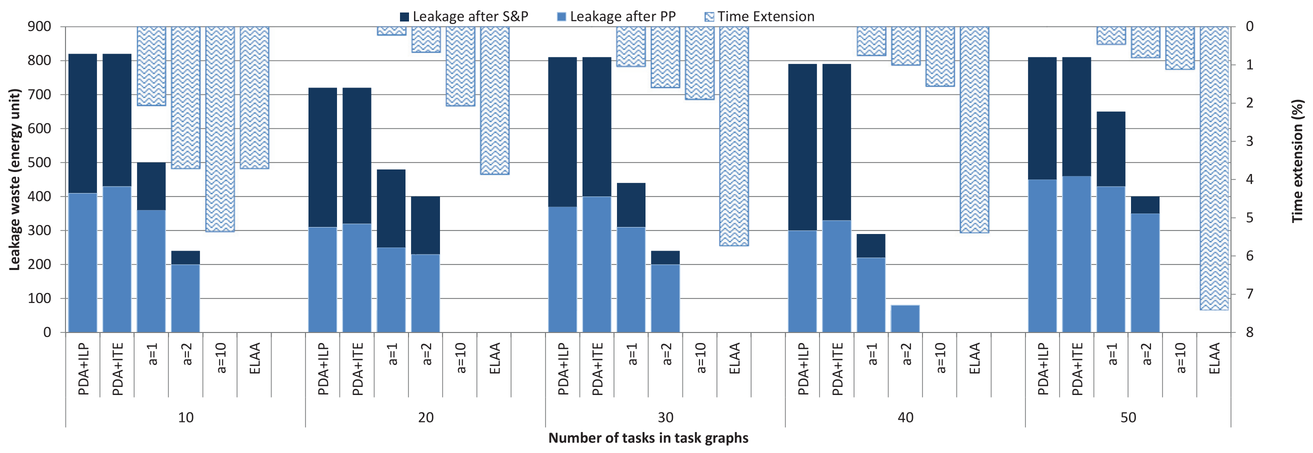


Fig. 6. Leakage and schedule length when employing different approaches.

Table 2
Leakage waste and algorithm runtime of post-placement methods.

Algorithms	Number of tasks in task graphs									
	10		20		30		40		50	
	Leakage	Runtime (s)	Leakage	Runtime (s)	Leakage	Runtime (s)	Leakage	Runtime (s)	Leakage	Runtime (s)
PDA+ILP	0	2.278	40	12.451	60	25.812	0	50.24	60	199.24
PDA+ITE	20	2.46E−04	80	4.32E−04	180	8.17E−04	80	1.14E−03	80	3.69E−03
PDA + Our heuristic	20	2.15E−04	80	4.36E−04	100	3.66E−04	80	4.32E−04	80	5.02E−04

when compared to ELAA. On an average, our approach adopted with the parameter $a = 1$ and $a = 2$ shows leakage power savings of 40% and 65% respectively when compared to PDA. Furthermore, when compared with existing approach PDA+ITE, our approach achieves 15% and 43% more leakage savings with parameter $a = 1$ and $a = 2$, respectively. The reason behind superior results by our approach over other approaches is that we consider leakage optimization first in scheduling and placement stages and then in post-placement stage as well. The optimization in scheduling and placement stages results in minimize delays between configurations and executions, and the post-placement stage try to further minimize the left delays in order to reduce the leakage dissipation. However, other approaches tackle the leakage optimization in only one stage (e.g., in placement stage in ELAA [7] and in post-placement stage in [6]).

6.1.2. *Post-placement leakage waste and algorithm runtime*

In this experiment, we examine the leakage saving and runtime of 3 post-placement methods ILP, ITE in [6], and our proposed heuristic. The methods are executed with the same inputs, which are the placement results from PDA. The deadline of all the task graphs are set to the schedule length of our approach when achieving optimal value of leakage saving (i.e., $a = 10$).

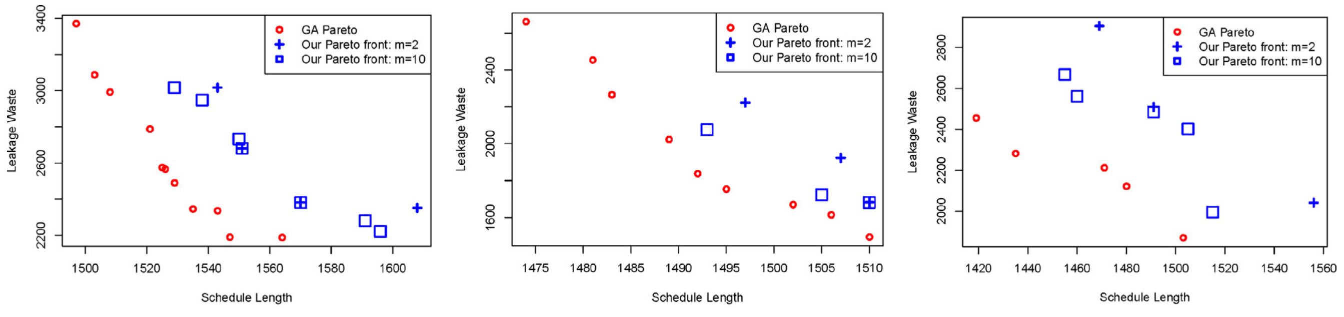
Table 2 shows leakage waste and algorithm runtime for various post-placement methods. As can be seen from Table 2, in many cases, all the post-placement methods are unable to totally eliminate the leakage dissipation over the PDA placement. However, for the same deadline, our multi-stage approach can achieve the optimal solution (leakage waste = 0) as described earlier. This signifies the advantages of our comprehensive strategy that addresses the leakage problem throughout the resource management process. Although our scheduling and placement stages achieve high leakage savings, they still can leave spaces between reconfiguration and execution parts of many tasks. Our post-placement stage tries to reallocate reconfigurations and executions so that the spaces between them are minimized in order to achieve further leakage savings. Table 2 shows that our post-placement heuristic can produce better leakage results than ITE. Additionally, our heuristic obtains the results in a smaller runtime.

Table 3
Leakage waste and schedule length for real-life applications.

		PDA+ITE	a = 1	a = 10	ELAA
MPEG	Schedule length	44	44	53	57
	Leakage S&P	140	80	0	0
	Leakage PP	0	0	NA	NA
JPEG	Schedule length	22	23	24	29
	Leakage S&P	60	20	0	0
	Leakage PP	20	20	NA	NA
MP3 decoder	Schedule length	50	57	61	63
	Leakage S&P	270	30	0	0
	Leakage PP	270	30	NA	NA

6.1.3. *Case-study: real-life applications*

We applied different scheduling approaches on real-life applications: JPEG encoder [20], MP3 decoder [39] and MPEG4 decoder [40] as mentioned earlier. Table 3 shows leakage waste and schedule length for real-life applications. The notations used in this experiment are the same as those in previous experiments. The ELAA and our approach with $a = 10$ always achieve the optimal value of leakage waste (zero) with some extension in schedule length. Therefore, leakage in these cases does not need any improvement by Post-placement methods and not applicable (NA) has been mentioned for the same. As can be seen from the table, for MPEG and JPEG, our approach with $a = 1$ can obtain the same results as that of approach *PDA+ITE*. However, when it comes to MP3 decoder, the advantage of our comprehensive strategy becomes obvious. Due to low quality solution in the first two phases, the ITE approach cannot remove all the leakage from initial placement of previous phases. In contrast, all stages of our approach still work well to get maximum leakage saving.



(a) TG from Fat group (b) TG from Medium group (c) TG from Slim group

Fig. 7. Pareto fronts generated from GA and our framework.

Algorithm 4 Trace back procedure.

```
Require: Normalized Pareto sets on Objective Space: $S_O = \{Pareto_i, i \in TrainSet\}$ ,
    Parameter sets associated with Pareto points on Parameter Spaces: $S_P = \{P_i, i \in TrainSet\}$,
    Predicted Pareto Range of new TG: $T_{\min}$, $T_{\max}$ ,
    Hyper-parameter: $n, k, m$ .
Ensure: Pareto set of new TG: $Pareto_{new}$
 1: $O_{new} = \emptyset$ /* initializing the Objective Space of new TG */
 2: for each knot $l$-th ($l = \overline{1, n}$) in interval $[T_{\min}, T_{\max}]$ do
 3:    $T_l = T_{\min} + l * (T_{\max} - T_{\min})/N$
 4:    $t_l = (T_l - T_{\min})/(T_{\max} - T_{\min})$

 5:    /* find k- nearest neighbours points from training set */
 6:    for each TG $i$-th in training set $TrainSet$ do
 7:       for each point $j$-th $Pareto_{ij} = (T_{ij}, E_{ij})$ on Normalized Pareto front of TG $i$-th do
 8:          Compute distance: $d_{ij} = T_{ij} - t_l$
 9:       end for
10:    end for
11:    Sort distance array $d$
12:    Add $k$ points with smallest distance to $kNN$ set

13:    /* find the potential parameter space for Pareto front of new TG */
14:    $P_{new} = \emptyset$ /* initializing the Parameter Space of new TG */
15:    for each points $k$-th in $kNN$ set do
16:       extract the parameter space $P_k$ of point $k$-th
17:       $P_{new} = P_{new} \cup P_k$
18:    end for
19:    $reach\_dist = OPTICS(P_{new})$
20:    Sort $reach\_dist$ and extract $m$ parameter set with smallest reachability distance to potential parameter space $P_{potential}$
21:    for each parameter set $m$-th in $P_{potential}$ do
22:       $O_{temp}$ = Leakage-aware RMA(parameter set $m$-th)
23:       $O_{new} = O_{new} \cup O_{temp}$
24:    end for
25: end for
26: Extract the Pareto set $Pareto_{new}$ of new TG from its Objective Space $O_{new}$
```

6.2. Performance of machine learning optimization framework

A number of experiments are conducted to evaluate the performance and efficiency of our ML optimization framework. The GA optimization is implemented with the NSGA II algorithm from NGPM package [36] in Matlab 2013 and run with a configuration of 50 population size and 10 generations. The ML techniques are developed with R 3.2 and OPTICS package [41]. Our ML framework is compared directly with the **GA method**. The criteria for comparison are quality of generated Pareto fronts and the number of evaluations that each method need to call the Leakage-aware RMA.

In this experiment, three groups of TGs, each with 50 TGs, are generated from TGFF tool [38] with different levels of parallelism: *fat, medium, slim*. As discussed earlier, the parallelism is dependent on the *task$_d$egree* option from TGFF tool, which represents maximum number of edges coming out from a task. When this parameter is small, the task has low parallelism and a *thin* shape. Whereas, a larger parameter gives TG with higher level of parallelism and a *fat* shape. Out of 50 TGs in each group, 40 TGs are used as training set in **Phase 1** of our framework. The predictive models are built with 10-fold Leave One Out Cross Validation process [42] to assure the generalization capability of the models. The other 10 TGs are used as new TGs to test the accuracy of the ML techniques. All the results shown in this Section are from the test set.

Fig. 7 shows the result for a random TG three different TG sets. The red circles are the Pareto front generated by GA. All the points generated from **Trace back** step are marked with blue color, where the plus signs and square signs represent the result with $m = 2$ and $m = 10$ respectively. As can be seen, the Pareto fronts generated by our framework are close to the ones from GA method. The figure also shows how the quality of our Pareto fronts improve with the increase in m.

Since the most time consuming process in both GA and our framework is executing the Leakage-aware RSA to get the design point on objective space, we designed the experiments around two hyper-parameters: *n-interval* and *m-potential candidates*, which directly decide on the number of evaluation points in our framework. After varying the value of m and n, we quantify the quality of the Pareto fronts using popular metrics in the MOA domain: R2-indicator (R2I) [43] and Hyper-volume indicator (HVI) [44]. Basically, the Hyper-volume indicators characterize the area in objective space covered by the Pareto front and a reference point in the top-right corner. Therefore, bigger HVI usually indicates better quality of Pareto front. The R2-indicator is a widely used metric to measure the difference between a targeted Pareto front and a base-line Pareto front. In our set up, the base-line is the origin of the objective space. It means Pareto front with smaller the R2-indicator is closer to the origin and has better quality. The quality degradation of the Pareto fronts generated by our framework when compared with GA's Pareto fronts are measure relatively by the Quality Trade-off (QT) in percentage of the GA's R2-indicator and HVI. The measurements are averaged over all the TGs of the test set and reported a long with the number of evaluations in Table 4. As can be observed from both the Table and the Figure, the quality of our framework is approached to the one generated by the GA method when increasing the number of evaluations (by increasing m or n) and that is also the pay-off in execution time. However, to achieve the comparable quality to the

Table 4
Execution time and quality comparison.

		Our approach					GA
TGs	m	5	10	5	5	5	10
	n	10	10	20	40	80	50
	Evaluations	50	100	200	200	400	500
	R2I	0.6966	0.6974	0.6924	0.6946	0.6845	0.6412
Fat	QT-R2 (%)	8.64	8.77	7.98	8.33	6.76	
	HVI	4,651,619	4,597,379	4,672,818	4,625,446	4,747,431	5,101,319
	QT-HV (%)	8.82	9.87	8.4	9.33	6.94	
	R2I	0.6866	0.6827	0.6777	0.6719	0.6719	0.6235
Medium	QT-R2 (%)	10.13	9.5	8.69	7.77	7.77	
	HVI	3,839,951	3,668,927	3,890,741	3,887,740	3,887,740	4,089,676
	QT-HV (%)	6.11	10.29	4.86	4.94	4.94	
	R2I	0.6822	0.6726	0.6822	0.6735	0.6739	0.6278
Slim	QT-R2 (%)	8.67	7.15	8.67	7.29	7.35	
	HVI	4,213,634	4,371,432	4,213,634	4,335,209	4,324,271	4,411,757
	QT-HV (%)	4.49	0.91	4.49	1.74	1.98	

result from GA we need only a fraction of time. With $m = 5$ and $n = 10$, we can achieve 10x speed-up over the GA with around 10% deficiency in the quality of the Pareto front for all types of task graph. Such an achievement is due to the fact that all the heavy computation is moved to the training phase and we take advantage of the ML models built upon the historical data.

7. Conclusion

This work presents a multi-stage resource management approach with ML Optimization framework to tackle the leakage power problem in Partially Reconfigurable FPGAs. Our multi-stage approach employs leakage-aware priority function in scheduling stage, leakage-performance trade-off function in placement stage and a heuristic in post-placement stage. Meanwhile, the ML Framework uses Linear Regression and Density-based Clustering Algorithm to estimate the Pareto front with only 10% degradation in quality and $10\times$ faster compared with an well-established MOA (NSGM II). A series of experiments are performed to highlight the advantages of the proposed approach over existing works. The results demonstrate that the proposed approach dominates the existing approaches when the application task graph contains higher number of tasks. Additionally, experiments show that our approach can always achieve the smallest leakage waste as a comprehensive strategy is adopted, whereas other single-stage methods may not achieve the same level of leakage saving. Furthermore, through the GA integration and ML Framework, our approach also provides the flexibility to the designers to explore the trade-off values between leakage saving and performance.

References

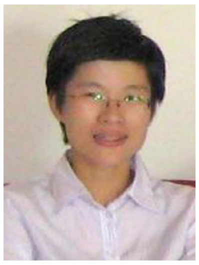
[1] I. Kuon, J. Rose, Measuring the gap between fpgas and asics, TCAD 26 (2007) 203–215.
[2] M. Shafique, L. Bauer, J. Henkel, Remis: run-time energy minimization scheme in a reconfigurable processor with dynamic power-gated instruction set, in: ICCAD, 2009, pp. 55–62.
[3] J. Kao, S. Narendra, A. Chandrakasan, Subthreshold leakage modeling and reduction techniques, in: ICCAD, 2002, pp. 141–148.
[4] S. Hauck, Configuration prefetch for single context reconfigurable coprocessors, in: FPGA, 1998, pp. 65–74.
[5] T. Tuan, B. Lai, Leakage power analysis of a 90 nm fpga, in: CICC, 2003, pp. 57–60.
[6] P. Yuh, et al., Leakage-aware task scheduling for partially dynamically reconfigurable fpgas, TODAES 14 (2009) 52.
[7] J. Hsieh, et al., An enhanced leakage-aware scheduler for dynamically reconfigurable fpgas, in: ASP-DAC, 2011, pp. 661–667.
[8] B. Calhoun, F. Honore, A. Chandrakasan, Design methodology for fine-grained leakage control in mtcmos, in: ISLPED, 2003, pp. 104–109.
[9] A. Sathanur, B. Luca, M. Alberto, M. Enrico, P. Massimo, Row-based power-gating: a novel sleep transistor insertion methodology for leakage power optimization in nanometer CMOS circuits, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 19 (3) (2011) 469–482.
[10] A. Gayasen, et al., Reducing leakage energy in fpgas using region-constrained placement, in: FPGA, 2004, pp. 51–58.
[11] F. Li, Y. Lin, L. He, Field programmability of supply voltages for fpga power reduction, TCAD 26 (2007) 752–764.
[12] L. Ciccarelli, A. Lodi, R. Canegallo, Low leakage circuit design for fpgas, in: CICC, 2004, pp. 715–718.
[13] A. Calimera, et al., Design techniques and architectures for low-leakage srams, Circuits Syst. I: Regul. Pap. IEEE Trans. (2012) 1992–2007.
[14] M. Hoyer, D. Helms, W. Nebel, Modelling the impact of high level leakage optimization techniques on the delay of rt-components, in: ICSD, 2007, pp. 171–180.
[15] A. Raghunathan, N.K. Jha, S. Dey, High-Level Power Analysis and Optimization, 1998.
[16] R. Bharadwaj, et al., Exploiting temporal idleness to reduce leakage power in programmable architectures, in: ASP-DAC, 2005, pp. 651–656.
[17] M. Zapater, et al., Leakage and temperature aware server control for improving energy efficiency in data centers, in: DATE, 2013, pp. 266–269.
[18] A. Ahmadinia, C. Bobda, J. Teich, A dynamic scheduling and placement algorithm for reconfigurable hardware, in: Organic and Pervasive Computing–ARCS 2004, 2004, pp. 443–465.
[19] C. Steiger, H. Walder, M. Platzner, Heuristics for online scheduling real-time tasks to partially reconfigurable devices, in: FPGA, 2003, pp. 575–584.
[20] S. Banerjee, E. Bozorgzadeh, N. Dutt, Physically-aware hw-sw partitioning for reconfigurable architectures with partial dynamic reconfiguration, in: DAC, 2005, pp. 335–340.
[21] F. Ferrandi, P.L. Lanzi, C. Pilato, D. Sciuto, A. Tumeo, Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems, Comput.-Aided Des. Integr. CircuitsSyst. IEEE Trans. 29 (6) (2010) 911–924.
[22] M. Guzek, et al., A survey of evolutionary computation for resource management of processing in cloud computing [review article], Comput. Intell. Mag. IEEE 10 (2) (2015) 53–67.
[23] S. Sutar, et al., Task scheduling for multiprocessor systems using memetic algorithms, in: 4th International Working Conference Performance Modeling and Evaluation of Heterogeneous Networks, 2006.
[24] G. Zarinzad, et al., A novel intelligent algorithm for fault-tolerant task scheduling in real-time multiprocessor systems, in: Convergence and Hybrid Information Technology, 2, IEEE, 2008, pp. 816–821.
[25] A.K. Samal, et al., Fault tolerant scheduling of hard real-time tasks on multiprocessor system using a hybrid genetic algorithm, Swarm Evol. Comput. 14 (2014) 92–105, doi:10.1016/j.swevo.2013.10.002.
[26] E. Hormozi, et al., Using of machine learning into cloud environment (a survey): managing and scheduling of resources in cloud systems, in: Proceedings - 3PGCIC, 2012, pp. 363–368, doi:10.1109/3PGCIC.2012.69.
[27] J. Berral, et al., Toward energy-aware scheduling using machine learning, Energy Effic. Distrib. Comput. Syst. (2012) 215–244, doi:10.1002/9781118342015.ch8.
[28] N.H. Kapadia, et al., Predictive application-performance modeling in a computational grid environment, in: International Symposium on High Performance Distributed Computing, IEEE, 1999, pp. 47–54.
[29] J.L. Berral, et al., Power-aware multi-data center management using machine learning, in: ICPP, 2013, pp. 858–867, doi:10.1109/ICPP.2013.102.
[30] H. Eom, et al., MALMOS: machine learning-based mobile offloading scheduler with online training, in: International Conference on Mobile Cloud Computing, Services, and Engineering, 2015, pp. 51–60, doi:10.1109/MobileCloud.2015.19.
[31] J. Perez, et al., Utility-based reinforcement learning for reactive grids, in: International Conference on Autonomic Computing, IEEE, 2008, pp. 205–206.
[32] A.K. Das, et al., Reinforcement learning-based inter-and intra-application thermal optimization for lifetime improvement of multicore systems, in: Design Automation Conference, IEEE, 2014, pp. 1–6.
[33] Xilinx, Partial Reconfiguration User Guide, Technical Report.
[34] O. Sinnen, Task Scheduling for Parallel Systems, 60, 2007.

[35] Y.-K. Kwok, et al., Benchmarking and comparison of the task graph scheduling algorithms, J. Parallel Distrib. Comput. 59 (3) (1999) 381–422.
[36] L. Song, Ngpm–a nsga-ii program in matlab, (2011).
[37] M. Ankerst, M.M. Breunig, H.-P. Kriegel, J. Sander, Optics: ordering points to identify the clustering structure, in: ACM Sigmod Record, 28, ACM, 1999, pp. 49–60.
[38] R.P. Dick, et al., Tgff: task graphs for free, in: Proceedings of the 6th International Workshop on Hardware/Software Codesign, IEEE Computer Society, 1998, pp. 97–101.
[39] P. Kumar, L. Thiele, Thermally optimal stop-go scheduling of task graphs with real-time constraints, in: ASP-DAC, IEEE Press, 2011, pp. 123–128.
[40] J. Cong, K. Gururaj, Energy efficient multiprocessor task scheduling under input-dependent variation, in: DATE, 2009, pp. 411–416.
[41] R Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, 2015.
[42] J. Friedman, et al., The Elements of Statistical Learning, Springer series in statistics, 1, Springer, Berlin, 2001.
[43] D. Brockhoff, T. Wagner, H. Trautmann, On the properties of the r2 indicator, in: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation, ACM, 2012, pp. 465–472.
[44] E. Zitzler, L. Thiele, Multiobjective optimization using evolutionary algorithms—a comparative case study, in: Parallel Problem Solving from Nature—PPSN V, Springer, 1998, pp. 292–301.

Pham Khanh received the B.Tech. degree in South Russian State University of Technology, Russia, in 2012. Since then, he has been working toward his Ph.D. degree at Department of Electrical and Computer Engineering, National University of Singapore. At the same time, he is a Research Scholar at Data Storage Institute, A*STAR, Singapore, under SINGA scholarship. His research interests include mapping and scheduling for MPSoC, High Level Synthesis, Design Space Exploration and Machine Learning in Resource Management for Embedded System.

Akash Kumar received the B.E. degree in computer engineering from the National University of Singapore (NUS), Singapore, in 2002. He received the joint Master of Technological Design degree in embedded systems from NUS and the Eindhoven University of Technology (TUe), Eindhoven, The Netherlands, in 2004, and received the joint Ph.D. degree in electrical engineering in the area of embedded systems from TUe and NUS, in 2009. He is currently the Chair for Processor Design at the Center for Advancing Electronics Dresden, Technische Universität Dresden, Germany. His research interests include design, analysis and resource management of low-power and fault tolerant embedded multiprocessor systems. He has published over 100 papers in leading international electronic design automation journals and conferences on these topics. He is also a member of technical program committees of major conferences in the design automation area like, DAC, DATE, ASPDAC, etc.

Amit Kumar Singh received the B.Tech. degree in Electronics Engineering from Indian Institute of Technology (Indian School of Mines), Dhanbad, India, in 2006. Thereafter, he worked with HCL Technologies, India for year and half before starting his PhD at School of Computer Engineering, Nanyang Technological University (NTU), Singapore, in 2008. He completed his PhD in 2012. He worked as a post-doctoral researcher at National University of Singapore (NUS) from 2012 to 2014 and at University of York, UK from 2014 to 2016. Currently, he is working as senior research fellow at University of Southampton, UK. His research interests include system level design-time and run-time optimizations of 2D and 3D multi-core systems with focus on performance, energy, temperature, and reliability. He has published over 45 papers in these areas in leading international journals/conferences. He was the receipt of ISORC 2016 Best Paper Award, PDP 2015 Best Paper Award, HiPEAC Paper Award, and GLSVLSI 2014 Best Paper Candidate. He has served on the TPC of IEEE/ACM conferences like ISED, MES, NoCArc and ESTIMedia.



Khin M. M. Aung received a Ph.D. degree of Computer Engineering from Korea Aerospace University, in 2006. She is currently a Senior Scientist with A*STAR, Data Storage Institute, Singapore. Her research interests include Data Security, Data Center and Network Storage Technologies.