IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 16 June 2022, accepted 26 June 2022, date of publication 29 June 2022, date of current version 8 July 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3187140

RESEARCH ARTICLE

Rapid Design-Space Exploration for Low-Power Manycores Under Process Variation Utilizing Machine Learning

SOHAIB MAJZOUB[1], (Senior Member, IEEE), RESVE SALEH[2], (Fellow, IEEE), MOTTAQIALLAH TAOUIL[3], (Member, IEEE), SAID HAMDIOUI[3], (Senior Member, IEEE), AND MOHAMED BAMAKHRAMA[4]

[1]Department of Electrical and Computer Engineering, University of Sharjah, Sharjah, United Arab Emirates
[2]Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC V6T 1Z4, Canada
[3]Laboratory of Computer Engineering, Delft University of Technology, 2628 CD Delft, The Netherlands
[4]Synopsys, 5656 AE Eindhoven, The Netherlands

Corresponding author: Sohaib Majzoub (smajzoub@sharjah.ac.ae)

ABSTRACT Design-space exploration for low-power manycore design is a daunting and time-consuming task which requires some complex tools and frameworks to achieve. In the presence of process variation, the problem becomes even more challenging, especially the time associated with trial-and-error selection of the proper options in the tools to obtain the optimal power dissipation. The key contribution of this work is the novel use of machine learning to speed up the design process by embedding the tool expertise needed for low power design-space exploration for manycores into a trained neural network. To enable this, we first generate a large volume of data for 36000 benchmark applications by running them under all possible configurations to find the optimal one in terms of power. This is done using our own tool called LVSiM, a holistic manycore optimization program including process variations. A neural network is trained with this information to build in the expertise. A second contribution of this work is to define a new set of features, relevant to power and performance optimization, when training the neural network. At design time, the trained neural network is used to select the proper options on behalf of the user based on the features of any new application. However, one problem encountered with this approach is that the database constructed for machine learning has many outliers due to randomness associated with process variation which creates a major headache for classification - the supervised learning task performed by neural networks. The third key contribution of this work is a novel data coercion algorithm used as a corrective measure to handle the outliers. The proposed data coercion scheme produces results that are within 3.9% of the optimal power consumption compared to 7% without data coercion. Furthermore, the proposed method is about an order of magnitude faster than a heuristic approach and two orders of magnitude faster than a brute-force approach for design-space exploration.

INDEX TERMS Neural network, simulator, manycore, low-power, process variation, frequency scaling, voltage scaling, 3D-stack, voltage selection, within-die variation.

I. INTRODUCTION

For the last two decades, technology scaling issues, process variation, time-to-market, and thermal and power densities have been major challenges facing high-performance and low-power designs [1]–[4]. Consequently, processor design has shifted towards the manycore paradigm with a promise of providing higher performance through increased throughput instead of raw performance. In particular, a key area of recent focus is low-power manycore design [5]–[9].

In this new paradigm, a manycore design is constructed out of tiles of identical processors arranged both horizontally

The associate editor coordinating the review of this manuscript and approving it for publication was Shunfeng Cheng.

and vertically in silicon. For example, a 2×2 tile containing 4 processor cores can be used to form a 16×16 array containing 1024 processor cores. However, the excessive power consumption of a thousand-core chip presents one of the key design challenges today. To reduce power, islands of cores can be defined with their own supply voltages and clocking frequencies, called voltage-frequency domains (VFDs), which depend on computational workloads and data traffic of the application to be executed on the array. This approach has been found to be very effective for power reduction.

However, device variability, such as those associated with the threshold voltage, V_{th}, presents yet another problem in small feature technologies and creates an irregular distribution of power and speed among the various processor cores in the array [1], [2], [4], [10]–[17]. A seemingly intractable issue is encountered by the chip designer to find the optimal combinations of VFDs to minimize power for a particular application in the face of such process variations. Some tools have been developed for this purpose but they require a great amount of expertise to use and lots of runtime in order to find the optimal configuration in terms of power.

Machine Learning (ML) techniques have gained a strong foothold across many research fields over the past decade, and the goal here is to apply these techniques to build the needed tool expertise into a trained neural network to quickly find the best configuration to achieve a low-power design. In recent years, manycore power optimization has utilized ML methods to reduce the power [9], [18]–[25]. Voltage and frequency scaling (VFS) is one of the most effective methods for power reduction. Most of the proposed methods use ML techniques along with VFS to estimate the proper cores' voltage and frequency requirements for a given workload [15], [21], [26]–[33].

Reinforcement learning has been applied through selecting dynamic voltage and frequency scaling (DVFS) policies to reach optimal power consumption. More recently, imitation learning, in [21], is used with the support of an Oracle policy (or the teacher) to guide policy selection. An offline estimator is used to develop the Oracle Policy to be used in the imitation learning. These learning techniques are used to dynamically scale the voltage and frequency based on changing workloads. The lack of training data for VFS control was the main justification to move to such techniques. The claim was that there are no datasets generated from manycore commercial chips or existing simulators specifically for voltage and frequency scaling [21].

In this work, we in fact remove this barrier by generating a comprehensive dataset for this purpose using LVSiM (a Low-power and Variation-aware Simulator for Manycores) [4]. LVSiM is a holistic simulation environment for low-power design-space exploration considering process variation. It allows the user to map an application, defined by its task graph with given workloads and data traffic, to a target manycore design while minimizing power in the presence of process variation. Using VFS in LVSiM involves a sequence of steps and a myriad of options for selecting the proper voltage and frequency values from a larger set (usually referred to as the Voltage/Frequency Selection Problem), then estimating the number of cores required for every Voltage/Frequency Domain, and finally assigning the voltage and frequency values to cores within the manycore array [4], [15], [16], [34].

LVSiM is a complex tool that provides many different configuration options to the chip designer to implement the aforementioned low power techniques – in fact, too many for a novice user to select the best options. In this paper, we utilize machine learning to recommend the best configuration based on the features of the application task graph. We first build a massive dataset of applications and corresponding LVSiM options, and then use a combination of unsupervised and supervised machine learning to train a neural network to produce the minimum power design for any given new (previously unseen) application considering process variation.

In the presence of within-die variation, the optimization process might deviate from the assumed optimal option. Within-die process variation creates power and speed discrepancies among cores in the same platform. This affects tasks' start and finish time, total execution time, and the power consumed by the cores executing those tasks. Thus, the optimal choice might be different for the same application executed using different process variation profiles.

In an ML context, we refer to these cases due to process variation as outliers. Having too many outliers (i.e. noise) in the dataset poses a major problem for supervised machine learning. In such cases, it may not be possible to train an ML system such that the resulting model could be used for future predictions with acceptable results. We resolve this issue by identifying a subset of outliers that can be modified – called mutable outliers – and then apply a scheme called data coercion to facilitate the use of supervised machine learning later on. Another potential problem facing the use of ML is the issue of skewed datasets. The issue of skewed data (generally known as class imbalance in classification) is a common problem in real datasets [35]. In this work, we also attempt to handle this problem by balancing different classes.

To summarize, LVSiM is used to generate the data samples. We use self-organizing maps for unsupervised learning to cluster the sample pool. A data coercion step is then performed to handle mutable outliers to create more uniform clusters with less skewed or unbalanced classes. Once the dataset is prepared and labeled, it is ready for use in supervised learning. A multi-layer artificial neural network (ANN) is then trained to predict the proper LVSiM configuration for each application to produce the lowest power design.

The contributions of this paper are as follows:

- **Novel unsupervised/supervised learning combination** for low-power design-space exploration with ML-assist that may be used with many other complex EDA tools.
- **New features for application task graphs** are proposed for use in ML that include workloads and traffic.

- **New data coercion approach** to identify and reduce mutable outliers due to random variations. The proposed approach uses a cost function taking into account power consumption, design-space exploration time, cluster uniformity, and class skewness.

This paper is organized as follows. Section II presents a literature review of previous work. Section III presents the motivation for our work. The proposed algorithm along with demonstration examples are discussed in Section IV. Section V describes the overall ML methodology. Section VI presents the experiments, runtimes and analysis of the results to validate the proposed method.

II. RELATED WORK

In this section, we describe existing research on power optimization for manycore designs using machine learning or artificial intelligence methods to illustrate the breadth and depth of the ongoing work in this field.

Rahmani *et al.* proposed in [11] a multi-objective dynamic power management for a network-on-chip (NoC). The proposed method uses fine-grained voltage and frequency scaling and power gating considering core reliability. The authors claim to minimize the aging effects while extending the core lifetime and boosting the overall throughput.

Wang *et al.* showed in [20] an improved reinforcement learning (RL) method for dynamic voltage and frequency scaling in a multi-core environment. They attempted to solve the problem of focusing on the core's local conditions by using core-level Modular Reinforcement Learning. The proposed method considered the state of multiple cores. The paper claims 28% in energy savings.

In [21], Kim *et al.* presented an imitation learning (IL) algorithm to optimize Voltage/Frequency Islands (VFIs). They claim that their method is the first architecture-independent IL-based methodology for dynamic VFI control for manycores. The paper suggests that IL produces better quality policies compared to reinforcement learning. The results show 5% in energy savings and lower computation time by 3.1x compared to RL. In [22], Chen and Liu proposed a data-driven manycore design. The authors present a manycore design approach from a machine learning requirements perspective.

Gupta *et al.* proposed in [36] an adaptive approach that uses an online learning framework to estimate power and performance under variable workloads. The proposed approach, namely STAFF, does feature selection adaptively and changes the estimation model dynamically. The reported speedup was 6x running on Intel®CoreTM i5 6th generation.

In [37], Choi *et al.* proposed a new paradigm for the communication network of manycores. The new network architecture is meant to improve traffic for a convolutional neural network (CNN). The authors analyzed the traffic in existing platforms, namely LeNet and CDBNet, and used this information to design a CNN-friendly network for manycores. The paper claims 1.8x reduction in network latency.

Islam and Lin proposed in [29] an RL methodology to dynamically select a proper voltage and frequency scaling approach based on workload conditions. They claim better energy savings compared to a single policy approach.

Cai *et al.* proposed in [28] a ML-based method to minimize energy in both nominal and near-threshold computing. The proposed method showed 31.1% energy saving and 11.5% throughput increase. They considered 30% process variation in their work.

In [38], Kim *et al.* used an adaptive Q-learning based method to optimize voltage and frequency scaling along with switching cores on and off. They consider electromigration in their work as well.

In [39], Chen and Marculescu proposed an online distributed reinforcement learning technique. They used a local per-core approach to adjust the frequency and the voltage of each core and then a global power budget reallocation algorithm. They claim 23% in energy savings and 44.3x higher throughput.

Kodaka *et al.* developed in [40] a method to predict the needed number of cores to satisfy workload changes. The method optimizes for low power without any performance degradation. The authors used DVFS and power gating to achieve their goal. They used a maximum of 32 cores to demonstrate their method.

Drego *et al.* used a near-optimal search algorithm to select a proper voltage value (of two available voltages) to mitigate core-to-core speed variation [41]. The authors claim 6 to 16% in energy savings. The paper also addresses the switched-off cores to save on energy while meeting the performance constraints. The proposed methodology assumed manycore platforms with 100 and 1000 cores.

Other papers used similar methods of reinforcement learning, mostly online, to handle power and throughput for multicore and manycores platforms [18], [19], [26]–[28], [42]–[47].

There are some key differences between this work and previous work. As indicated above, publications over the last decade use iterative ML methods based primarily on or related to *reinforcement learning*. The reason given for not using a dataset to train a neural network in some of these papers is that such a dataset is not available [21]. In contrast, we tackle this issue by generating the needed dataset and using it for non-iterative ML methods that use *unsupervised and supervised learning*. Furthermore, most of the previous work focused on a single optimization problem within the power or performance framework, such as VFS control, network traffic, or workload allocation, due to the lack of a holistic tool that can be used to evaluate the overall power and performance numbers in a multi-stage optimization context. Our approach uses LVSiM which addresses all these issues in one tool. Moreover, we introduce a novel data coercion technique as a corrective measure to handle outliers induced by process variation. As a result, a direct comparison with other approaches is difficult. The appropriate comparison here is to study the cases with and without data coercion.

We also compare it with a heuristic approach that would be a natural alternative to our approach.

III. MOTIVATION

Given the wide scope of the work published in the open literature, we now describe the motivation behind our work in this section. The conventional multi-layer optimization flow used in most electronic design automation (EDA) tools is shown in FIGURE 1(a) [5]. It consists of an optimization loop with one or more optimization phases. A possible configuration is selected using an optimization criteria in every phase. The selected configurations are then evaluated in every iteration according to a cost function. Acceptable configurations for all phases are produced once the cost target is reached. The new trend in EDA tools is to use machine learning techniques to assist in the configuration selection process [1], [2], [9]. As discussed in the previous section, most of the work used different flavors of reinforcement learning to optimize for dynamic voltage and frequency scaling. The learning/training part is embedded in the optimization loop. In our view, utilizing machine learning techniques in EDA tools has to take a more holistic approach [9].

In this work, we attempt to apply machine learning over the entire process of the power optimization phases, starting from the application characterization phase all the way down to the mapping onto manycores phase. Design-space exploration of this sort can be application specific and might involve multi-phase optimization that usually takes a long time to optimize [4]. It involves analyzing the application, estimating the required voltage and frequency values, estimating the required number of cores, assigning voltage and frequency values to those cores, scheduling tasks into time slots without conflicts and routing traffic to satisfy the power and performance budgets. Scaling up the problem to manycores with thousands of cores to run thousands of tasks optimizing for high performance and low power is much more challenging [4], [9]. It is especially important to reduce this design-space exploration simulation time.

We postulate that the application task graph features largely determine the options selected in LVSiM. Based on this, we propose the flow shown in FIGURE 1(b) to automatically figure out the best configuration to use in LVSiM. So we begin with the application task graph to be mapped to the manycore design and extract its key features. Instead of using an iterative loop, we use a trained neural network to predict the proper settings of the options in LVSiM for every optimization phase for the given application. The training of this network is carried out using the key features of the task graph (input) and known LVSiM options (output) using a comprehensive set of benchmark applications. This reduces the time needed to find optimized state during design-space exploration since the entire space of options does not need to be examined.

The core problem addressed in this work is a multi-class classification problem. The applications are first roughly grouped into clusters using unsupervised learning, in

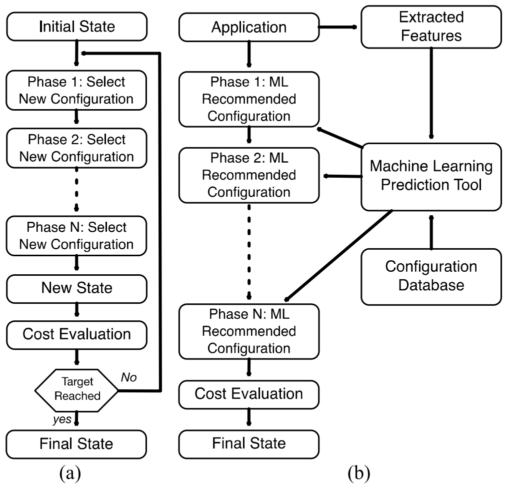

Initial State
Phase 1: Select New Configuration
Phase 2: Select New Configuration
Phase N: Select New Configuration
New State
Cost Evaluation
Target Reached
No
yes
Final State
(a)
Application
Extracted Features
Phase 1: ML Recommended Configuration
Phase 2: ML Recommended Configuration
Machine Learning Prediction Tool
Configuration Database
Phase N: ML Recommended Configuration
Cost Evaluation
Final State
(b)

FIGURE 1. EDA flow (a) using conventional optimization loop to select the configuration, and (b) using machine learning to recommended the configuration.

preparation for a subsequent step of supervised learning. Within each cluster, the same configuration (i.e. set of options in LVSiM) should be used by all applications in that cluster. If some do not, they are referred to as outliers. Supervised learning (used in a later step) works best if most of the applications within each cluster agree on the same configuration. Too much noise in the clusters affects the neural networks' ability to learn and predict. Any randomness in the system produces outliers in all clusters and impacts the uniformity of the clusters. Noisy and unbalanced datasets cannot be easily or reliably trained using supervised machine learning. A typical solution to this problem is to identify and eliminate outliers before training begins.

Outliers arise from many sources. Process variation is one of the factors considered during the power optimization. Process variation injects a randomness factor when selecting the optimal configuration during optimization. Other factors can also be sources of variability during the optimization. For instance, in the case of 3D manycores, we use xyz-routing to pass the data from one core to another. Typically, the path with minimum traffic is used to route the data. If all routes have equal traffic, then one is selected at random. Moreover, task scheduling also has an embedded randomness. That is, tasks with equal scheduling priority are scheduled at random.

In this paper, we propose a data coercion approach to enhance uniformity within the cluster. The proposed approach also handles the issue of skewed classes. Some configurations are more popular than others because they are good default settings of the options. As a result, many of the applications may prefer one of these configurations. This imbalance can create a problem when training artificial neural networks. We attempt to even out the classes with synthetic data, as discussed later in section.

At this point, the LVSiM tool requires a brief description. The tool accepts an application task graph as input and provides users with different possible configuration options in each phase to map the application to the manycore array when it performs the optimization. Thus, such a tool is instrumental to realize the proposed flow shown in FIGURE 1(b). However, it is not straightforward to choose the best options for a given application. A brute-force technique would be to run all combinations to find the optimal settings but this is expensive. Instead, machine learning will be used to recommend the proper options of each of the optimization phases of the LVSiM tool for a given application task graph.

IV. PROPOSED DATA COERCION STRATEGY

In this section, we present the central contribution of this paper in detail. First, we describe *data coercion* as a concept in an abstracted form and then describe the proposed algorithm with a simple example. We start with an analogy to a well-known binary classification problem: cats vs. dogs. If we are starting from scratch, we need to create images of cats and dogs by taking thousands of pictures. The features of each image will be based on the pixels of the image (input) and each one will be labeled as 1 for cats and 0 for dogs (output). If all the images can be properly labeled, then no data coercion is needed because we can readily classify all the data into cats and dogs based on the features. On the other hand, if there are some cats incorrectly labeled as dogs and vice-versa, they are called outliers because they are mislabeled. But what if 50% of the images cannot be easily identified as a dog or cat, or the images are blurry for some reason? Some cat images may have the likeness of dogs, and some dog images may resemble cats. We can assign some initial label to them but if they are clustered into groups and declared to be outliers, they become eligible for relabeling. These are called *mutable outliers*.

We can perform an initial clustering of the images of cats and dogs using unsupervised learning and then perform a data cleaning step. In fact, we want to look into the clusters and identify the mutable outliers and try to re-label them to their majority class. By doing this, we will make the job of the supervised learning using neural networks much easier. Simply stated, data coercion refers to forcing a subset of the outliers to fit into the majority class of elements in that cluster to facilitate learning rather than treat them as true outliers.

Carrying the analogy a bit further, we may also need to generate synthetic data to balance classes if we had far more dog images than cats. For example, one image of a cat can be manipulated to create many cat images with the cat in different locations in the image. One can also flip the image and repeat the same process. The combination of data coercion and synthetic data generation will make the downstream classification problem easier and produce better predictions.

A. DATA COERCION EXAMPLE

As discussed in Section III, the new design flow is meant to use ML to quickly obtain an optimized mapping of an application onto a manycore array. The first step is to take each application, run through all possible options in our LVSiM tool to determine the power dissipated, and then label each application with the configuration that produced the lowest power. We then cluster the applications into groups that used the same configuration. However, the learning process is affected by the inherent noise due to process variation, and randomized decisions during routing and scheduling. This random variability component is referred to here as the **Inherent Variability Factor** (IVF).

Consequently, unsupervised learning produces clusters with lots of outliers sprinkled within the clustered data. To resolve this dilemma, we check each outlier to see if we can change its label to the majority label. The cost of changing a label is a suboptimal option and, hence, more power. If the increase in power is marginal, then it is appropriate to switch the label. However, if the power is too high, then label switching is not permitted. To summarize, after initial clustering, data coercion tries to identify mutable outliers based on the power cost to relabel them to the majority, taking into account if it makes the cluster more uniform.

More formally, after unsupervised learning, a cluster **I** with majority label $\mathbf{L_i}$ has an outlier element $\boldsymbol{S_j}$ if it has a different label $\mathbf{L_{S_j}}$ than the label of the majority $\mathbf{L_i}$ within the cluster. If this outlier is due to the IVF, then it is likely that the cost of changing its label is small, and it is referred to as a *mutable* outlier. Our method uses data coercion on the mutable outliers to try to impose uniformity to enable efficient supervised learning at the cost of slightly more power.

FIGURE 2 shows an example of data coercion. The circles represent clusters formed during unsupervised learning and the symbols therein represent elements of the cluster. Elements with the **x** label in cluster **A**, and **o** label in cluster **B,** are assumed to be outliers and a mutable subset of them should be relabeled into the label of the majority. The relabeling can take place only if the cost of doing so is acceptable (i.e. small power increase); otherwise the element is considered a *true* outlier and it cannot be relabeled (i.e. large power increase). This approach is referred to as data coercion because the mutable outliers are coerced into the label of the cluster majority to improve uniformity.

In our context, a given label refers to the optimal configuration used to minimize the power consumption for a given application. However, these labels are not necessarily fixed so we look for opportunities to enhance classification. But relabeling the element to any other (less optimal) label incurs an increase in the power consumption. For example, assume that the optimal configuration for an element **S** is labeled **x**. If this element is forced to switch to the **o** label (the less optimal case), there will be an increase in the power consumption. If this increase in power is considered acceptable according to a coercion cost function (to be discussed later), then element **S** is considered to be a mutable outlier and it should be forced to be relabeled to the **o** label with a small penalty in power increase. This mutable outlier is assumed to be a result of the IVF problem. However, if the power increase due to the relabeling is too high according to the coercion cost function, then element **S** is assumed to be a true outlier and it cannot be relabeled. Identifying all mutable outliers and relabeling

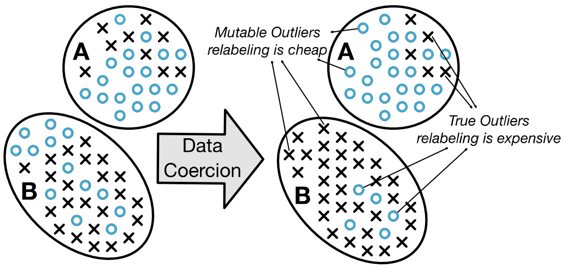

A
B
Data Coercion
Mutable Outliers relabeling is cheap
True Outliers relabeling is expensive
A
B

FIGURE 2. Data Coercion applied at the element level.

them to produce more uniform clusters will reduce the IVF noise effect and improve the classification performance.

B. DATA COERCION COST FUNCTION

The first step is to assign initial labels to each element in our set for the purposes of clustering using unsupervised learning. The proposed data coercion scheme is then used to identify the true and mutable outliers, and then relabel the mutable outliers to improve cluster uniformity based on its cost. The coercion cost is calculated using a cost function during the relabeling process. It incorporates an evaluation of the cluster uniformity, power penalty, and class imbalance when relabeling a given element. Note that the simulation time is also used in the cost function but neglected here for simplicity.

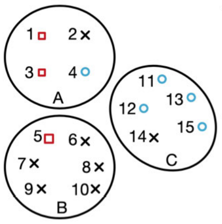
FIGURE 3 shows a detailed toy example of the coercion cost calculation and the relabeling process. Note that the example is simplified to demonstrate the proposed method. Thus, only one stage of the optimization flow of FIGURE 1(b) is assumed. In this example, we assume 15 elements with 3 labels, and 3 clusters. The elements represent the applications and the labels represent the optimization option. Each element can be assigned one of the three labels with a specific power cost. FIGURE 3(a) shows the 15 elements in the dataset and the assumed normalized power cost for each of the three labels (Note that the numbers relevant to this example are set in bold). The power cost is normalized with respect to the optimal value in each column. Thus, the optimal power label for a given element is shown as **1**. The rest of the non-optimal labels are greater than one.

For instance, elements 1, 3, and 5 have the optimal power label given as □. As shown, the normalized power is 1, while the power for the other suboptimal labels are all greater than 1, e.g. element 1's normalized power values for labels **X** and **O** are 1.8 and 1.9, respectively. Similarly, the optimal power label for elements 2, 6, 7, 8, 9, 10, and 14 is **X**, and for elements 4, 11, 12, 13, and 15 is **O**.

The diagram in FIGURE 3(b) shows the elements hypothetically grouped into 3 clusters (supposedly after unsupervised learning based on the elements features) where the elements are labeled with the *minimum power option* (hence the power costs of the elements are considered 1). Elements 1, 2, 3, and 4 are grouped into cluster **A**; elements 5, 6, 7, 8, 9, 10 into cluster **B**; and 11, 12, 13, 14, 15 into cluster **C**.

Label	Normalized Power Cost of the 15 Samples (1 is the optimal label)														
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
□	**1**	**7**	**1**	**1.6**	**1**	1.5	1.3	1.1	1.9	2.1	1.6	1.8	3	1.4	1.2
X	1.8	**1**	2.3	1.5	**6**	**1**	**1**	**1**	**1**	**1**	1.9	1.1	1.3	**1**	5.5
O	2.9	1.8	2.1	**1**	5	7.2	6.5	8.1	2.1	3.4	**1**	**1**	**1**	**5**	**1**

(a)



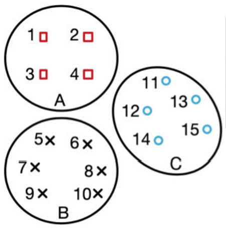
Optimal Power Solution

N#	Cluster Uniformity	Power Cost
A	4/2	4
B	6/5	6
C	5/4	5

N#	Label Skewness
□	5/3
X	5/7
O	5/5

Power Cost = 4+6+5 = 15
Cluster Uniformity Cost = 4/2+6/5+5/4 = 4.45
Skewness Cost = max(5/3,5/7,5/5) = 5/3 = 1.667
Coercion Cost = 15+ 4.45 + 1.667 = 21.1

(b)



Fully Uniform Clusters Solution

N#	Cluster Uniformity	Power Cost
A	4/4 = 1	10.6
B	6/6 = 1	11
C	5/5 = 1	9

N#	Label Skewness
□	5/4
X	5/6
O	5/5

Power Cost = 10.6+11+9 = 30.6
Cluster Uniformity Cost = 4/4+6/6+5/5 = 3.0
Skewness Cost = max(5/4,5/6,5/5) = 5/4 = 1.25
Coercion Cost = 30.6 + 3.0 + 1.25 = 34.85

(c)

FIGURE 3. Coercion demonstration example.

The table in FIGURE 3(b) shows the uniformity and power costs per cluster, and the skewness cost per label. The given clusters are not uniform, but the majority labels are □'s in cluster **A** (2 out of 4), **X**'s in cluster **B** (5 out of 6), and **O**'s in cluster **C** (4 out of 5). Thus, the uniformity cost is calculated by dividing the total number of elements in a cluster by the count of the majority label. For instance, the uniformity cost in cluster **A** is equal to 4/2, i.e. the total number of elements, 4, divided by the count of the majority label, 2 (elements 1 and 3).

The power cost of a cluster is the total sum of the elements' costs. For instance, in cluster **B**, the summation of the total power costs of all elements, based on the table shown in FIGURE 3(a) is 6.

Then, the cluster imbalance is computed. A non-skewed label distribution necessitates 5 elements per label (i.e. 15 elements divided by 3 labels, 5 □'s, 5 **O**, and 5 **X**'s), which is not the case here. As shown in FIGURE 3(b), 3 elements are labeled □ (elements 1, 3, and 5); 5 elements are labeled **O** (elements 4, 11, 12, 13, and 15); and 6 are labeled **X** (elements 2, 6, 7, 8, 9, 10, and 14). Thus, the skewness cost of every label is calculated as the non-skewed label count divided by the current label count. For instance, the skewness costs for the □, **X**, and **O** labels are 5/3, 5/7, and 5/5, respectively.

The total uniformity cost of the given configuration in FIGURE 3(b) is the sum of the individual clusters' uniformity costs, i.e. 4.45. The power cost is the sum of the power cost of the three clusters, i.e. 15. The skewness cost is the maximum label skewness costs, i.e. 1.667. Consequently, the coercion cost is the sum of all costs, i.e. 21.1. These calculations are shown below the table of FIGURE 3(b).

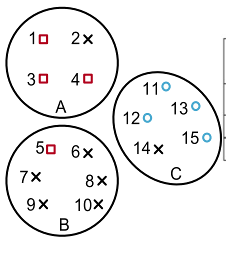
The example shown in FIGURE 3(b) begins with the optimal power labels. The next step is to modify the labels of the elements in the clusters to improve uniformity. FIGURE 3(c) shows an extreme case where all the elements are forced into the majority label to produce fully uniform clusters. In this case, elements 2 and 4 are relabeled □ in cluster **A**. The power cost can be obtained from the table in FIGURE 3(a), which is 7.0 for element 2, and 1.6 for element 4. Unfortunately, the power cost increases from 4 (FIGURE 3(b) minimum power configuration) to 10.6 (fully uniform clusters configuration). The same is applied to clusters **B** and **C**, where the power cost increases to 11.0 and 9.0, respectively. The uniformity cost, on the other hand, is reduced to 1 for all clusters. The total uniformity cost is reduced from 4.45 (FIGURE 3(b)) to 3 (FIGURE 3(c)). The total skewness cost is also reduced as a result of this change to 1.25. However, the full coercion cost is now 34.85 which is much higher than the option of FIGURE 3(b), where the cost is 21.1.

We can observe that there is a large penalty going from FIGURE 3 (b) to (c). The coercion costs of the two cases are 21.1 (optimal power) and 34.85 (fully uniform clusters). This might be too high to be acceptable. We need to minimize the total coercion cost and not just the power, uniformity, or skewness costs. Thus, the proposed algorithm attempts to switch the label of various elements and re-calculate the coercion cost. The initial configuration is assumed to be the one given in FIGURE 3(b), i.e. lowest power cost. Then, the coercion cost is calculated when the label of a given element is switched to any of the existing labels.

FIGURE 4(a) shows the coercion cost covering all scenarios. It shows the coercion cost when an element is labeled □, **X** or **O**. This helps the algorithm decide whether an element should be relabeled or not. For example, if element 1 kept its □ label, then the total coercion cost is going to be 21.1 (as demonstrated in the previous example), but it is going to increase to 22.7 and 23.8 if it is relabeled to **X** or **O**, respectively.

Label	Coercion Cost if a Sample changed label														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
□	21.1	26.0	21.1	20.6	21.1	21.5	21.3	21.1	21.9	22.1	21.7	21.9	23.1	21.1	21.3
X	22.7	21.1	23.2	21.6	26.7	21.1	21.1	21.1	21.1	21.1	22.4	21.6	21.8	21.1	26.0
O	23.8	21.9	23.0	21.1	25.9	27.6	26.9	28.5	22.5	23.8	21.1	21.1	21.1	24.8	21.1

(a)



Minimum Coercion Cost Solution

N#	Cluster Uniformity	Power Cost
A	4/3	4.6
B	6/5	6
C	5/4	5

N#	Label Skewness
□	5/4
X	5/7
O	5/4

Power Cost = 4.6+6+5 = 15.6
Cluster Uniformity Cost = 4/3+6/5+5/4 = 3.78
Skewness Cost = max(5/4,5/7,5/4) = 5/4 = 1.25
Coercion Cost = 15.6 + 3.78 + 1.25 = 20.63

(b)

FIGURE 4. Coercion cost calculation and minimum coercion cost solution.

Cluster **A** is now used to demonstrate the coercion process. There are 4 elements in cluster **A** (1, 2, 3, and 4). The coercion cost of element 1 is 21.1 for the □ label (it is already □ in FIGURE 3(b) so it shows the same cost). If element 1 is switched to **X**, the coercion cost increases to 22.7. Similarly, if the label is switched to **O**, the cost increases to 23.8. So element 1 must remain as □ because this is the label with the minimum coercion cost. Since this label matches the majority label (i.e. □), then *element 1 is not an outlier*. Similarly, element 3 shows the label with the minimum coercion cost to be □, i.e. 21.1, thus it is not an outlier either.

Next, the minimum coercion cost of element 2 is 21.1, i.e. when it is left as **X**, which is different than the label of the cluster majority. This means that *element 2 is a true outlier*. In the case of element 4, the coercion cost is 20.6 for the □ label which is the minimum. Consequently, it should be switched from **O** to □. Hence, *element 4 a mutable outlier*. The same procedure can be done to clusters **B** and **C**, where elements 5 and 14 are deemed to be true outliers, since the minimum coercion costs (21.1 for both as shown in table in FIGURE 4(a)) are associated with labels □ and **X** for the two elements, respectively, so they must remain in place. Thus, clusters **B** and **C** stay the same.

The calculations of the total coercion cost for this case are shown in FIGURE 4(b). As shown, the power cost increases to 15.6, i.e. a 4% increase. This power increase is acceptable to improve the cluster uniformity. The next step would be to apply unsupervised learning and re-clustering again in a 2nd pass and to repeat the whole process. This would continue until the clusters stabilize. In our work, we proceeded directly to supervised learning using the labels resulting from one or two iteration of this process.

C. MULTI-PHASE COST FUNCTION

In the previous example, each element is assumed to have one label. But the flow, discussed earlier in section III, has multiple phases so, in reality, each element should have a set of labels, one for each phase. The goal is to select a configuration option for each of the different phases all at once. Each label represents the optimization method used during one of the phases within the optimization flow shown earlier in FIGURE 1(b). The following demonstration example is used to show the calculation of the coercion cost in the case of a multi-phase optimization flow.

Table 1 shows an example of the normalized power values for 15 elements in a two-phase optimization example. Each phase has a label set, namely phase 1 has □, **X**, and **O** labels; and phase 2 has ♦, △, and ∇ labels. Each element can be assigned only one of the three labels in each of the two phases. The table shows the power cost for each label assignment in the two phases. An assignment is assumed to be the minimal power case if it is represented by **1** as the normalized power. Thus, each row represents an element with two **1**s indicating the minimum power labels for each of the two phases.

Similar steps are used to calculate the coercion cost for the multi-phase case. First, we calculate the total coercion cost for the strictly optimal power case. In this case, we consider the normalized power cost to be **1** for all elements. Thus, the power cost is the summation of all 15 elements, i.e. **15**. The average uniformity cost of each phase for all clusters are as follows:

TABLE 1. Normalized Power of 15 elements in a Multi-Phase Optimization Example.

Cluster	Element Number	Normalized Power Cost					
		Phase 1 Labels			Phase 2 Labels		
			X	O	◊	△	▽
A	1	**1**	1.8	2.9	2	**1**	3.1
	2	7	**1**	1.8	**1**	4.1	2.3
	3	**1**	2.3	2.1	**1**	5	2.1
	4	1.6	1.5	**1**	**1**	2	3
B	5	**1**	6	5	1.3	**1**	2
	6	1.5	**1**	7.2	1.6	6	**1**
	7	1.3	**1**	6.5	1.7	**1**	2.5
	8	1.1	**1**	8.1	1.8	**1**	1.2
	9	1.9	**1**	2.1	**1**	2	1.9
	10	2.1	**1**	3.4	1.9	**1**	2.9
C	11	1.6	1.9	**1**	2.4	1.6	**1**
	12	1.8	1.1	**1**	6.1	3.2	**1**
	13	3	1.3	**1**	2.2	**1**	3.2
	14	1.4	**1**	5	3.3	**1**	2.2
	15	1.2	5.5	**1**	2.7	2.5	**1**

- Cluster A uniformity cost is (4/2+4/3)/2 = 1.667
- Cluster B uniformity cost is (6/5+6/4)/2 = 1.350
- Cluster C uniformity cost is (5/4+5/3)/2 = 1.458

In the case of cluster **A**, phase 1 has 3 labels where the □ is the majority label, since it is assigned to two elements (elements 1 and 3). Thus, the number of elements in the cluster, i.e. four, divided by 2 is the uniformity cost of this cluster for phase 1. In the case of phase 2 of cluster **A**, the ♦ is the majority label (assigned to three elements, namely elements 2, 3, and 4), so the uniformity cost is 4/3. Thus, the total uniformity cost of cluster **A** is the average of the two, i.e. (4/2+4/3)/2. In the same way, the cluster uniformity cost is calculated for clusters **B** and **C**. The total cluster uniformity cost is the sum of the uniformity costs of all clusters.

As discussed earlier, the skewness of a given label is calculated by dividing the ideal number of elements per label by the actual number. Since the total number of elements is 15 and there are 3 labels in each phase, the ideal number of elements per label is 15/3 which is 5. For instance, the total number of elements labeled ♦ is 4 in phase 2. Thus, the skewness of the ♦ label is 5/4. Similarly, the skewness cost is calculated for all other labels. Then, the skewness cost per phase is considered to be the maximum skewness cost of its labels. Thus, the skewness cost per phase are calculated as follows:

- Phase 1 skewness cost is max(5/3,5/7,5/5) = 5/3
- Phase 2 skewness cost is max(5/4,5/7,5/4) = 5/4

The total skewness cost is the average of the two phases, i.e. (5/3+5/4)/2. The coercion cost is then the summation of the power, uniformity, and skewness costs, which in this case is given by 15+4.475+1.458 = 20.9.

D. COERCION COST EQUATIONS

Based on the detailed calculations given above, the general equations used to calculate the coercion cost can now be presented. It should be noted that each of the power, simulation time, uniformity, and skewness costs are multiplied by a calibration factor to fine tune the coercion cost.

Henceforth, the term **sample** will be used to refer to an element of a cluster since we are now describing the specifics of our methodology. It refers to the application or benchmark to be mapped to cores of the manycore design using LVSiM. The power cost is the summation of the normalized power for each individual sample within the cluster. The total power cost is the summation of the clusters' power costs multiplied by a calibration weight as shown in Eq. (1).

$$PC = W_P \sum_1^{N_C} \left(\sum_1^{N_S} NP_i \right) \tag{1}$$

W_P: power cost calibration weight.
NP_i: normalized power of a given sample.
N_S: number of samples in the cluster.
N_C: number of clusters

The simulation time cost is calculated in a similar fashion as the power cost. The equation is shown below:

$$TC = W_T \sum_1^{N_C} \left(\sum_1^{N_S} NT_i \right) \tag{2}$$

W_T: time cost calibration weight.
NT_i: normalized simulation time of a given sample.
N_S: number of samples in the cluster.
N_C: number of clusters

The total uniformity cost is the multiplication of the uniformity calibration weight by the summation of the uniformity cost for all clusters. The uniformity cost of each cluster is the average of the uniformity cost per phase for that cluster. The uniformity cost per phase is calculated by dividing the total number of samples in a cluster (N_S) by the number of samples of the majority label (M_S) within that phase. The uniformity cost for a cluster is shown in the equation below:

$$UC = W_U \times \sum_1^{N_C} \left(\frac{\sum_1^{M_{Ph}} \left(\frac{N_S}{M_S} \right)}{M_{Ph}} \right) \tag{3}$$

W_U: uniformity cost calibration weight.
M_{Ph}: number of phases.
N_C: number of clusters.
N_S: number of samples in the cluster.
M_S: number of samples of the majority label in the cluster.

The skewness cost is the skewness calibration weight multiplied by the average phase skewness cost (summation of the skewness costs of all phases divided by the number of phases). The skewness cost per phase is equal to the maximum skewness cost of all labels in that phase. The skewness cost of a given label is equal to the ideal number of samples per label (i.e. total number of samples, N_{TS}, divided by the number of labels in that phase, N_L) divided by the total number of samples for a given label in that phase (L_{TS}).

The skewness cost formula is shown in the equation below:

$$SC = W_S \times \frac{\sum_1^{M_{Ph}} max\left(\frac{N_{TS}}{N_L \times L_{TS}}\right)_j}{M_{Ph}} \tag{4}$$

W_S: skewness cost calibration weight.
M_{Ph}: number of phases.
N_{TS}: total number of samples.
N_L: number of available labels for a given phase.
L_{TS}: total number of samples for a given label for a given phase.

Finally, the coercion cost is the summation of the power, uniformity, and skewness costs. The general formula to calculate the coercion cost is given in the equation below:

$$CC = PC + TC + UC + SC \tag{5}$$

CC: Coercion Cost.
PC: Power Cost.
TC: Simulation Time Cost.
UC: Uniformity Cost.
SC: Skewness Cost.

E. DATA COERCION ALGORITHM

FIGURE 5 shows the proposed data coercion algorithm. The algorithm starts by importing the samples with their features. **Synthetic samples** are then created based on the features of the real samples. The values of the synthetic samples are generated at random, within the range of the real samples. The synthetic samples help to produce well-represented and well-spread-out data set for clustering purposes only.

A demonstration of synthetic data is shown in FIGURE 6. Each sample is represented as a point in the 2D plot (assuming 2 features and 9 clusters) and the labels are represented using different colors. As shown, the real data is usually packed into groups in some corners (colored regions) and scattered in others. Synthetic data generation populates the open areas of all clusters evenly to help create a better distribution. A Self-Organizing Map (SOM) is then used to group the sample pool into clusters. The output of this step is still going to be non-uniform clusters, but then the data coercion algorithm converts them into more uniform clusters in preparation for supervised learning. While it is possible to use the results of unsupervised learning to perform inference, we found it more efficient to use a neural network to produce the options for each phase, as will be discussed later.

The algorithm of FIGURE 5 continues by selecting a cluster, and then calculating the coercion cost for every sample within this cluster, as described earlier. The sample with the minimum coercion cost is selected for re-labeling. Then, data coercion is applied to the sample with the minimum coercion cost. Consequently, if the label of that sample can be changed to the majority label, then it is considered to be a mutable outlier. Otherwise, if it differs from the majority label, it is a true outlier.

Changing the label of one sample increases the power and affects the uniformity of the cluster under examination and the overall skewness. Thus, the algorithm revisits the remaining samples within the cluster and attempts coercion again. All outliers within the cluster have to be identified as either true or mutable. Then, it will be coerced, or relabeled, if it is a mutable outlier. Finally, after visiting all samples in all clusters, the algorithm produces a more uniform set of clusters. The data coercion algorithm is applied to the training set while the test set is used on the trained neural network to recommend the LVSiM options to obtain low power.

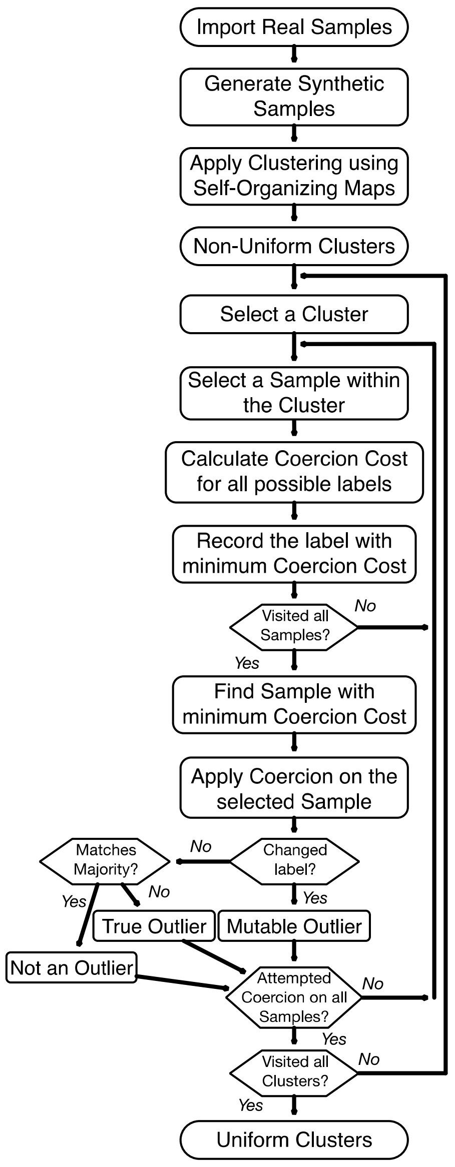

Import Real Samples
Generate Synthetic Samples
Apply Clustering using Self-Organizing Maps
Non-Uniform Clusters
Select a Cluster
Select a Sample within the Cluster
Calculate Coercion Cost for all possible labels
Record the label with minimum Coercion Cost
Visited all Samples?
No
Yes
Find Sample with minimum Coercion Cost
Apply Coercion on the selected Sample
Matches Majority?
No
Changed label?
Yes
No
Yes
True Outlier
Mutable Outlier
Not an Outlier
Attempted Coercion on all Samples?
No
Yes
Visited all Clusters?
No
Yes
Uniform Clusters

FIGURE 5. Proposed Data Coercion Algorithm.

V. MACHINE LEARNING METHODOLOGY

In this section, we describe the complete methodology for low-power manycore design-space exploration under process variation using machine learning. We have already described how we clean the data so that it is suitable for supervised training. In order for the ANN to be trained, a set of features

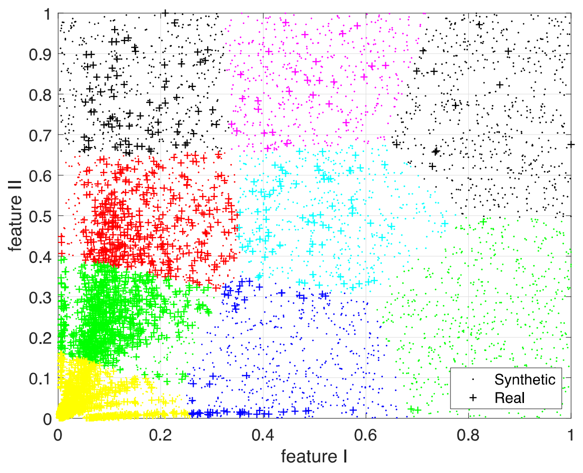


FIGURE 6. Real and Synthetic samples within the clusters.

must be defined and used for each application as input to the ANN and an optimal configuration must be specified with a binary code at the output. Then, a suitable architecture of the ANN must be identified, which is more of an art than science. After training, the machine learning flow is used to predict the options for LVSiM for any new application, and then applied to LVSiM produce the optimized array in a single run.

A. LVSiM OPTIMIZATION PLATFORM

For the interested reader, the details of the simulator have been described in [4] but we provide pertinent information here for completeness. The LVSiM optimization flow is shown in FIGURE 7. The application is first analyzed to determine the initial power and speed characteristics. The tasks and the number of cores for each Voltage and Frequency Domain (VFD) are estimated. Then, a core reduction step is used to optimize the number of cores needed per VFD. A VFD reduction step is then performed to reduce the number of domains in the design to a pre-defined number. The core reduction step within VFDs is performed again to further reduce the number of cores. Next, the layout of VFD onto cores is performed followed by the mapping of the application onto cores.

As mentioned earlier, the term **Sample** is used to refer to the application or benchmark to be mapped into manycore using LVSiM. Each optimization stage is referred to as a **Phase**. Selecting one option in each phase defines the set of **Labels** for a sample and constitutes a **configuration** in LVSiM. The five optimization phases in LVSiM are the core reduction phase, the VFD reduction phase, VFD layout shape phase, VFD layout order phase, and task mapping to cores phase.

The *core reduction phase* optimizes the number of needed cores by attempting to remove some cores and migrate their tasks onto other cores within other VFDs. There are four different options to consider in this phase [9]:

1) ***Core to Multiple Domains (C2MD):*** tasks of removed core can be scheduled on any destination core of any destination VFD.

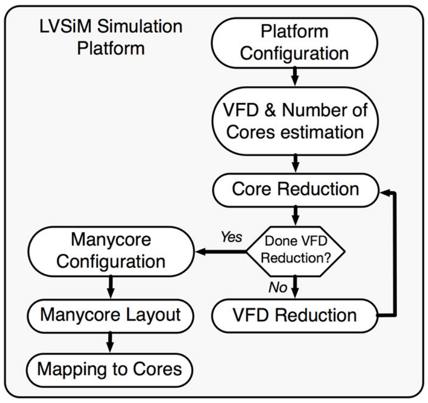


FIGURE 7. LVSiM simulation platform.

2) ***Core to Single Domain (C2SD):*** tasks of removed core must be scheduled on any other core but all within the same destination VFD.
3) ***Core to Single Core (C2SC):*** tasks of the removed core must be scheduled on any other core but all within the same destination core in the same destination VFD.
4) ***VFD Nominal Number of Cores (VFDNC):*** uses the nominal number of cores without any further reduction.

The *VFD reduction phase* reduces the number of voltages and frequencies in a design from a larger given set. The Removal-Cost Method is used in the reduction [4], [15], [34]. The available options by LVSiM are as follows:

1- Reducing voltages and then frequencies (**VFR**).
2- Reducing frequencies then voltages (**FVR**)
3- Reducing both voltage and frequencies simultaneously (**SVFR**).

The *VFD layout shape phase* assigns the voltage and frequency values onto actual cores. LVSiM provides three options for VFD layout shape as follows:

1- **Stacked Domain Layout** (**SDL**), VFDs are stacked one after the other starting with the one with the highest voltage and frequency values.
2- **Alternating Domain Layout** (**ADL**), the voltage and frequency values of the VFDs are alternated.
3- **Square Domain Layout** (**SQDL**), a square shaped island encompassing a group of cores of the same VFD are going to be placed in an alternating fashion. The size of the square is user defined.

The *VFD layout order phase* in which the VFD are placed can be of two user-defined options as follows:

1- **Number Ordered VFD Layout** (**NOVFL**), VFDs are placed from higher to lower Vdd/Frequency values.
2- **Traffic Ordered VFD Layout** (**TOVFL**), the VFD are placed based on the traffic between VFDs.

The *task scheduling phase* is a modified As-Soon-As-Possible (ASAP) scheduler. The scheduler prioritizes ready tasks based on which one should go first. The following are two prioritization options to be used during scheduling:

1- ***Child-based Task Priority (CTP):*** after random shuffling, ready tasks with more children are scheduled first. Prioritizing with respect to the number-of-children assumes that finishing a task (with more children) would allow more successor tasks to be scheduled.

2- ***Slack-based Task Priority (STP):*** after random shuffling, ready tasks with smaller slack times are scheduled first.

In summary, the given **Phases** and **Labels** are as follows:

- Core Reduction **Phase 1**: given **Labels** {***C2MD***, ***C2SD***, ***C2SC***, ***VFDNC***}.
- VFD Selection **Phase 2**: given **Labels** {***FVR***, ***VFR***, ***SVFR***}.
- VFD Layout Shape **Phase 3**: given **Labels** {***SDL***, ***ADL***, ***SQDL***}.
- VFD Layout Order **Phase 4**: given **Labels** {***NOVFL***, ***TOVEL***}.
- Task Scheduling **Phase 5**: given **Labels** {***CTP***, ***STP***}.

We can now determine the total number of options. There are 4 × 3×3 × 2×2+1=145 possible configurations to try in LVSiM so it may be difficult for the user to easily determine which one to use to obtain the optimal power. The extra added option is the nominal case with only one nominal VFD. Note that, although LVSiM provides more options for the task priority and core selection during scheduling, we limited the methods to those mentioned above. This is because the selected methods produced the best power and performance numbers for a wide set of applications in our initial experiments with the tool. Furthermore, limiting the available configuration options helps reduce the number of simulation cases while still demonstrating the proposed idea with a reasonable overall dataset generation time. We built a database of designs and options that is more representative of the expected applications to be used in manycore designs.

B. FEATURE SELECTION

The feature selection issue is perhaps the most important and difficult part of the machine learning task. Since the problem at hand is very complex with multi-phase optimization, the important features of an application must be selected carefully to improve the overall performance of the neural network. Here we are looking for the application features particular to the optimization steps. We believe that the optimal configuration is tied to the characteristics of the task graph of the application. For instance, any selected features have to capture properties such as slack, execution time, task graph width, etc.

Table 2 shows the 13 features under initial consideration. The first 3 are conventional features used in similar optimization cases, namely traffic, critical path time, and parallelism. The traffic is considered as the number of edges or communication links between tasks. The critical path time is the processing time of the tasks that lie on the critical path of the task graph. The parallelism is the sum of all task computation times divided by the critical path time.

In this work, we propose 10 more features specifically for power optimization. These features can be extracted after analyzing the task graph information only. As shown in Table 2, the Total Slack feature is the sum of the slack of all tasks within the application. The Total Execution Time is the sum of execution time of all the tasks of the application. The Task-Graph Width is measured by the maximum number of tasks executed simultaneously within the task graph. The Core-Task Density is the task graph width divided by the total number of tasks (i.e. the average number of tasks executed per core).

TABLE 2. Features ranked in each Optimization Stage.

N#	*Feature Name*	*Core Reduction*	*VFD Reduction*	*VFD Layout*		*Task Scheduling*
				Layout Shape	Layout Order	Task Priority
A	Critical Path	7	5	11	6	11
B	Parallelism	1	1	1	1	3
C	Traffic	13	11	12	12	6
D	Total Slack	4	4	4	4	1
E	Total Execution Time	10	12	5	7	8
F	Task-Graph Width	3	3	3	3	2
G	Core-Task Density	12	13	13	13	13
H	VFD Density	2	2	2	2	4
I	Power-Task Density	11	6	6	5	12
J	Maximum number of parents	6	9	8	11	5
K	Average number of parents	8	7	9	8	9
L	Maximum number of children	9	8	10	10	10
M	Average number of children	5	10	7	9	7

Given that the only available information is from the task graph at this point, the VFD is defined by a group of tasks with the same slack. The density for the VFD is defined as the ratio of the number of cores divided by the total number of tasks in that VFD. In other words, it is the maximum number of tasks (within the VFD) that can run simultaneously divided by the total number of tasks in that VFD. The VFD Density feature is calculated as the average densities of all VFDs as shown below:

$$VFD\ Density = mean\left(\frac{Number\ of\ Cores}{Number\ of\ Tasks}\right) \tag{6}$$

This feature encapsulates the number of required cores, and number of tasks for each VFD, i.e. tasks with similar slack (since tasks with similar slack belong to the same VFD).

The Power-Task Density is the total dynamic power of a task graph divided by the total number of tasks. The dynamic power of a task is assumed to be the dynamic power of a core running this task given the voltage and frequency values to eliminate its slack. The idle power is unknown at this point because the application is not mapped to the cores yet.

The maximum and average number of parents of a task are assumed to be the number of times a given task is a parent of another task. The maximum and average number of children are assumed to be the number of times a given task is a child task. The averages in both cases, i.e. features K and M, are calculated excluding zero values.

Table 2 also shows the features ranked by their relative importance, from 1 to 13, for each optimization stage. The Infinite Latent Feature Selection algorithm by Roffo *et al.*, [48], is used to prioritize the features with respect to each other. As shown, Parallelism, VFD Density, and Task-Graph Width are the most important features for most cases. The Core-Task Density seems to be the least important. FIGURE 8 however shows the correlation matrix of the given features and J, K, L, and M have high correlations. Hence, the K, L, and M features were removed during classification after many trials. The top 10 features A-J were used in our flow.

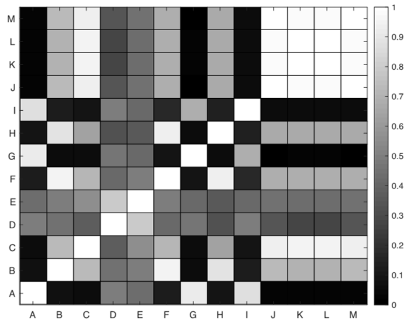


FIGURE 8. Correlation matrix of the selected features.

C. SUPERVISED LEARNING FLOW

Supervised learning is used to train the artificial neural network using the database generated as described in previous sections. For the training set, 10 features are used as input to the ANN. The outputs are binary-encoded values that specify which of the 145 possible LVSiM configurations was used to obtain the optimum power. After training, the ANN is used to predict the proper LVSiM configuration for the applications in the test set.

We now review the key steps shown in FIGURE 9. For a given application (or sample), LVSiM is used to generate the power consumption of the mapped application onto a many-core array. All possible combinations of the configuration options are used in a brute-force approach to find the power consumption numbers for all configurations. Once the power numbers are obtained, the sample's features along with the power numbers are passed to the data coercion algorithm. The new coerced datasets are then used to train the ANN.

Finding the proper ANN architecture is another difficult task. After many trials, we selected a 5-layer ANN with 10 input features, 50 hidden units in layer 2, 30 hidden units in layer 3, 15 hidden units in layer 4 and 15 binary-encoded outputs. This is depicted in FIGURE 10. Note that the outputs of the neural network are associated with Phases 1 to 5 with an additional output for the default case of a uniform voltage and frequency for all cores. The outputs for each phase can be generated by individual neural networks but we found it to be more efficient using one neural network. The outputs can be decoded to produce 1-out-of-145 possible configurations associated with the optimal settings, if desired.

The optimization platform is given in FIGURE 11. As shown, the machine learning prediction platform produces

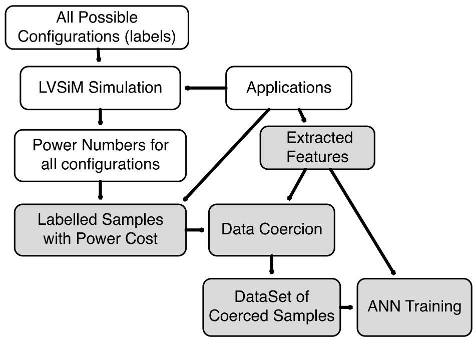

All Possible Configurations (labels)
LVSiM Simulation
Applications
Power Numbers for all configurations
Extracted Features
Labelled Samples with Power Cost
Data Coercion
DataSet of Coerced Samples
ANN Training

FIGURE 9. Steps used for the training Phase of the Artificial Neural Network.

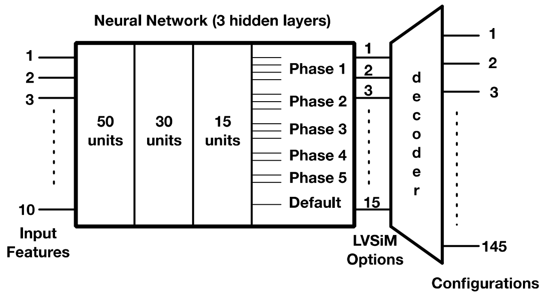


FIGURE 10. Neural Network Architecture.

the recommended options to be used in every step of the LVSiM optimization flow. One ANN is trained to produce all LVSiM options at once to be used for optimization.

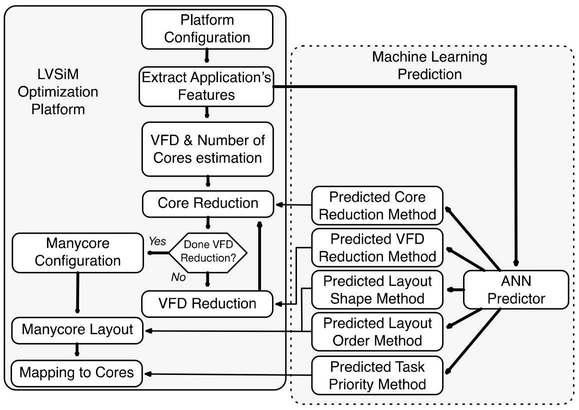
The main advantage of the proposed system is that the user does not have to follow a brute-force approach and try all 145 different permutations to reach the one that produces the minimum power. Thus, the design exploration time is reduced drastically. All the hard work is done in the ML portion of the design flow to build the needed expertise into the trained neural network. The art of selecting the proper architecture for the ANN is non-trivial but once a workable architecture is found, it can be re-used as new data is added to the database. Despite the fact that it takes a long time to generate the needed database and properly train the network, the new platform can predict a reasonable configuration in a single run in a shorter amount of time.

VI. EXPERIMENTS AND ANALYSIS

In this section, we describe the experimental setup. We present the benchmark characteristics, and runtimes, and finally analyze and compare the results of with and without the proposed coercion algorithm.

A. BENCHMARK CHARACTERISTICS AND PLATFORM SETUP

We used 36,000 Standard Task Graph (STG) benchmarks created by combining STGs generated by Tobita and Kasahara [49]. FIGURE 12 shows two features of the used benchmarks (the two top-rated features of Table 2, namely B and H). As it is intended to show, the benchmarks are selected



Platform Configuration
LVSiM Optimization Platform
Extract Application's Features
Machine Learning Prediction
VFD & Number of Cores estimation
Core Reduction
Predicted Core Reduction Method
Manycore Configuration
Yes
Done VFD Reduction?
No
Predicted VFD Reduction Method
VFD Reduction
Predicted Layout Shape Method
ANN Predictor
Manycore Layout
Predicted Layout Order Method
Mapping to Cores
Predicted Task Priority Method

FIGURE 11. Proposed manycore optimization platform using ML.

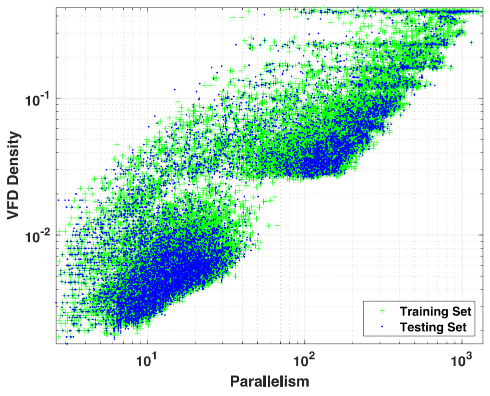


FIGURE 12. Characteristics of 36000 Standard Task Graphs benchmarks generated by Tobita et. al. [49].[1] Parallelism versus VFD Density (highest ranked features), showing the training and the test sets.

to cover a wide range of values for all the given features. The figure also shows the random nature of the selected training and test sets.

The dimensions of a manycore design is defined by the number of columns, rows, and layers or dies. We consider a 3D manycore architecture with 2 dies stacked on top of each other. Each layer has a two-dimensional router mesh. The routers are also connected through a vertical link to the upper layer to create the xyz Network-on-Chip (NoC) used by the simulator. The manycore array size is assumed to be 40 × 40×2=3200 cores. The values of the voltage levels vary from 0.6V to 1.4V in 0.05V increments, and the normalized frequencies from 0.15 to 1.0 with steps of 0.05. Finally, the number of voltages and frequencies permitted in the design are 2 and 4 respectively, and any unused cores are switched off completely. The NoC routers and links are assumed to run at the nominal voltage and frequency without any voltage/frequency scaling.

B. RESULTS AND ANALYSIS

Our research objectives required an exhaustive set of data to demonstrate the viability of our new approach. Therefore,

[1]http://www.kasahara.elec.waseda.ac.jp/schedule/

TABLE 3. Outliers Numbers using the Proposed Method.

Number of Outliers in the Training Set (24000 samples)	
without coercion (mutable + true outliers)	12447 (51.9%)
with coercion using proposed method (true outliers)	1900 (7.9%)

TABLE 4. Prediction Results.

Training Set:	ANN without data coercion	ANN with data coercion
Correlation Accuracy	61.6%	90%
Precision	0.55	0.84
Recall	0.46	0.72
F1 score	0.44	0.75
Test Set: Average power increase above optimal	7%	3.9%

we carried out a comprehensive set of simulations to create a large data pool. Each STG benchmark was simulated 145 times to select the best configuration. The 36000 benchmarks required a total of 36000 × 145=5,220,000 simulations. To carry out these simulations, we deployed a high-performance computing (HPC) environment.

Although this scale of simulation is needed in this work to demonstrate the proposed approach over a wide set of applications, it may not be needed in general. The data generation time can be reduced to a manageable level when focusing on applications within a single domain. The benchmarks can be selected carefully targeting a specific domain so that the total number of simulations is kept to a minimum (especially if the runtime for each configuration is high). This requires *domain knowledge* about the application and the tool itself. In addition, the tool should be mature and stable (such is the case for LVSiM) so that the data set generation process need only be performed once. The HPC environment and runtime characteristics are provided in a section to follow.

As mentioned, the sample pool was divided into two sets: the training set with 24000 samples (66.7%), and the test set with the remaining 12000 (33.3%) samples. The data coercion algorithm was performed on the training set to identify the true and mutable outliers. Table 3 shows the total number of outliers with and without coercion. As shown, the number of outliers is almost 52% of the data, which is a very high number due to process variation. Applying the proposed method reduced the number of outliers to a very reasonable percentage in the single digits (7.9%) as shown in the table. Several iterations of unsupervised learning and data coercion were performed until a consistent set of clusters was obtained.

Two ANNs were trained using the training data with and without data coercion. Table 4 shows the results of the prediction step on the training set. The standard figures of merit such as the correlation accuracy, precision, recall, and F1 scores are listed, all using their corresponding multi-class versions. As shown, the numbers improve considerably using the new method. When the ANNs were used for inference on the test set, we computed only the resulting power from LVSiM and compared it to the optimal case. The average power increase across all samples is 3.9% for the proposed method, while it is 7% for the approach without data coercion (see Table 4). The runtime cost is the sum of the neural network forward pass plus the optimization time which is 1.4x longer on average.

The 3.9% power increase is expected and hard to mitigate because of process variation. As discussed in earlier sections, consider an application mapped to the cores with process variation profiles 1 and 2. The configurations that produce the optimal power are going to be different for the two process variation profiles. In fact, the power consumption difference between the two profiles is going to be within the given margin, i.e. 3.9%. Furthermore, having two different applications as opposed to one (even with similar features) adds another complexity dimension to the problem at hand. This was observed and confirmed during experimentation.

C. HEURISTIC ALTERNATIVE TO ML

An alternate way to quickly find an optimal configuration is to use a *heuristic approach*. We describe this approach and compare it to our method. The heuristic approach is defined by selecting the top power saving configurations as a subset of the total number of 145 possible configurations. For example, the top 10 power saving configurations can be identified based on the training set, and then used on the test set. Which one to use is not clear so all 10 must be attempted.

Two more figures of merit are introduced to better capture the efficacy of the heuristic method, namely the Average Power Increase (API), and the Optimization-Time Ratio (OTR). The API and OTR refer to the average power increase and the optimization time compared to the optimal. The acceptable but suboptimal case obtained using the heuristic approach is to try all optimization configurations in the subset (i.e. 10 in this case) for a given application and then selecting the one that produces the highest power savings. The optimization time ratio is the ratio of the total sum of the times taken to explore all optimization configurations in the subset compared to the optimal method. The API and OTR values are calculated considering the samples in the test set.

FIGURE 13 shows the API and OTR values considering the top power configurations ranging from NC = 1 to 10. As explained earlier, all configurations in the selected subset are tried to see which one obtained the best power number. This is going to increase the design space exploration time. For instance, if the selected subset has ten possible methods, then, ideally, the optimization time should be ten times. Thus as shown in FIGURE 13, as NC increases, the optimization time increases but the resulting power decreases. That is, increasing the number of possible methods improves the chances of finding a lower power consumption method. In fact, including all possible methods (i.e. 145 cases) is going to take very long time to finish the design exploration, but it is

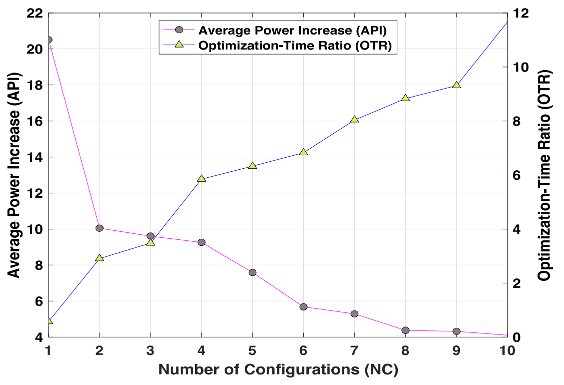


FIGURE 13. The API and OTR results of the heuristic method using a number of the top power saving configurations (NC), starting from 1 to 10 different configurations.

going to obtain the optimal method. This behavior is clearly visible in FIGURE 13, as the number of possible methods (i.e. NC) increases, the optimization time increases and the power consumption number is reduced.

If the API values shown in FIGURE 13 are compared to the proposed method, as shown in Table 4, the NC = 10 seems to produce similar results (3.9% for the proposed method versus 4.1% for the heuristic method with NC = 10). The OTR in this case is around 11.6x for the heuristic method compared to 1.4x for our ANN method. This means that the speedup is about 8.4x to reach the same power saving numbers. Further, the speedup of the ANN approach over brute-force is about 145x, i.e. trying all 145 possible methods.

D. COMPUTATIONAL ENVIRONMENT AND RUNTIMES

The HPC facilities shown in Table 5 were used to create the dataset. As mentioned earlier, our dataset contained 36,000 applications. Each application was simulated using LVSiM on all 145 configurations. This amounted to a total of 5,220,000 LVSiM simulations. The total time taken to generate the dataset was roughly 1-2 months, using the available HPC facilities, which allowed up to roughly 700 processors to run in parallel. For each of the 36,000 applications, the best configuration in terms of power was extracted.

TABLE 5. HPC Facilities Used to Generate the Dataset.

Provider	HPC processing details
University Of Sharjah	**FAHAD HPC: 432 cores** = 12 compute nodes each with 2 Intel Xeon Gold 6140 (18 Cores, 2.3GHz, 192 GB RAM, Centos OS). **SAQR HPC: 128 cores** = 4 compute nodes each with 2 Intel Xeon E5 (16 core, 2.3 GHz, Ubuntu). **MAHA HPC: 96 cores** = 4 HP Z840 WSs each with 2 Intel Xeon E5 (12-core, 2.6GHz, Ubuntu)
TUDelft	**40 cores** = 2 compute nodes each with 20 cores, Intel(R) Xeon(R) E5-2670 (2.50GHz, Ubuntu.)

The MATLAB software tool was used as the ML development environment. The extracted relevant data was passed to MATLAB. The commands *patternnet* and *train* were used to create and train the ML model. The time taken to perform unsupervised learning and data coercion on the 24,000 data points of the training set was about 20 hours to generate the clusters and clean the data. Supervised training to build the ML model required about 1 hour. The prediction time performed to generate the 12,000 configurations of the test set took only few minutes. All these steps were performed on a Macbook pro with 2.5GHz processor and 16GB of RAM. From an end-user's perspective, the time required to predict one configuration and execute the LVSiM code once is equal to the prediction time, i.e. few minutes, plus the optimization time taken by the predicted method (i.e. anywhere from 1 to 24 hours of simulation time depending on the configuration). Clearly, there is a significant savings in time for the user due to ML.

VII. CONCLUSION

In this paper, we proposed a methodology for manycore design space exploration under process variation. In contrast to reinforcement learning used in previous approaches, we use a combination of unsupervised and supervised machine learning procedures to automatically determine the best optimization methods to be used in LVSiM based on the features of the application. A key contribution of this work is a novel data coercion approach used along with unsupervised learning to manage outliers. The proposed algorithm attempts to reduce outliers in the dataset in order to improve the performance of supervised learning.

A large database was generated for this purpose but over 50% of the data were deemed to be outliers. The proposed data coercion algorithm was used to separate true outliers from other mutable outliers. The mutable outliers were coerced into the label of the cluster majority to make these clusters more uniform at the cost of slightly more power. The outlier percentage was reduced to 8% thereby producing a more homogenous dataset. In other words, the proposed method is an outlier identification and reduction approach to improve the ANN performance in the case of noisy datasets. This proposed approach can be applied in any situation where lots of outliers exist in the data and a cost function can be developed to decide whether or not coercion should be carried out. The overall proposed methodology can be applied in EDA to improve the ease-of-use of complex tools by providing the expertise in terms of a trained neural network.

There are some limitations of the ML approach described herein. It is suitable for tools with a large but finite set of options. The configurations should be limited in number and the same applies to the number of benchmarks. Otherwise, the data generation time may be significant. Of course, the data is generated only once but the total runtime must still be kept to an acceptable level. Other possible limitations of the approach are that the data must have mutable outliers for which choosing a different label is possible and the cost of choosing a different label is small. If the degree of process variations is significant, the approach may not be as effective since the number of outliers may dominate the data set. Aside from these issues, using the proposed ML flow and data coercion algorithm produced better power numbers than without data coercion, and a much faster optimization time compared to brute-force methods ($\sim$100x) or a more heuristic approach ($\sim$10x).

REFERENCES

[1] M. Shafique, S. Garg, J. Henkel, and D. Marculescu, "The EDA challenges in the dark silicon era," in *Proc. 51st Annu. Design Autom. Conf. Design Autom. Conf.*, vol. 14, 2014, pp. 1–6.

[2] D. Marculescu, "The quest for energy aware computing," in *Proc. 6th Int. Green Sustain. Comput. Conf. (IGSC)*, Dec. 2015, p. 1.

[3] T. Kim, Z. Sun, C. Cook, J. Gaddipati, H. Wang, H. Chen, and S. X.-D. Tan, "Dynamic reliability management for near-threshold dark silicon processors," in *Proc. 35th Int. Conf. Comput.-Aided Design*, Nov. 2016, p. 70.

[4] S. Majzoub, R. A. Saleh, I. Ashraf, M. Taouil, and S. Hamdioui, "Energy optimization for large-scale 3D manycores in the dark-silicon era," *IEEE Access*, vol. 7, pp. 33115–33129, 2019.

[5] P. A. Beerel and M. Pedram, "Opportunities for machine learning in electronic design automation," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2018, pp. 1–5.

[6] B. De Salvo, "Brain-inspired technologies: Towards chips that think?" in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 12–18.

[7] R. G. Kim, J. R. Doppa, and P. P. Pande, "Machine learning for design space exploration and optimization of manycore systems," in *Proc. Int. Conf. Comput.-Aided Design*, Nov. 2018, pp. 1–6.

[8] D. Biswas, V. Balagopal, R. Shafik, B. M. Al-Hashimi, and G. V. Merrett, "Machine learning for run-time energy optimisation in many-core systems," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 1588–1592.

[9] R. G. Kim, J. R. Doppa, P. P. Pande, D. Marculescu, and R. Marculescu, "Machine learning and manycore systems design: A serendipitous symbiosis," *Computer*, vol. 51, no. 7, pp. 66–77, Jul. 2018.

[10] I. Loi, A. Capotondi, D. Rossi, A. Marongiu, and L. Benini, "The quest for energy-efficient I$ design in ultra-low-power clustered many-cores," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 4, no. 2, pp. 99–112, Jun. 2018.

[11] A. M. Rahmani, M.-H. Haghbayan, A. Miele, P. Liljeberg, A. Jantsch, and H. Tenhunen, "Reliability-aware runtime power management for manycore systems in the dark silicon era," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 2, pp. 427–440, Feb. 2017.

[12] S. Majzoub, "Reducing random-dopant fluctuation impact using footer transistors in many-core systems," *Integration*, vol. 48, no. 1, pp. 46–54, Jan. 2015.

[13] S. Majzoub, R. Saleh, and R. Ward, "PVT variation impact on voltage island formation in MPSoC design," in *Proc. 10th Int. Symp. Quality Electron. Design*, Mar. 2009, pp. 814–819.

[14] S. Majzoub, R. Saleh, S. J. E. Wilton, and R. Ward, "Simultaneous PVT-tolerant voltage-island formation and core placement for thousand-core platforms," in *Proc. Int. Symp. System Chip*, Oct. 2009, pp. 1–4.

[15] S. S. Majzoub, R. A. Saleh, S. J. E. Wilton, and R. K. Ward, "Energy optimization for many-core platforms: Communication and PVT aware voltage-island formation and voltage selection algorithm," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 29, no. 5, pp. 816–829, May 2010.

[16] S. Majzoub, "Instruction-based voltage scaling for power reduction in SIMD MPSoCs," *J. Low Power Electron.*, vol. 7, no. 2, pp. 141–147, Apr. 2011.

[17] S. Majzoub, R. Saleh, and H. Diab, "Reconfigurable platform evaluation through application mapping and performance analysis," in *Proc. IEEE Int. Symp. Signal Process. Inf. Technol.*, Aug. 2006, pp. 496–501.

[18] H. Jung and M. Pedram, "Supervised learning based power management for multicore processors," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 29, no. 9, pp. 1395–1408, Sep. 2010.

[19] H. Shen, Y. Tan, J. Lu, Q. Wu, and Q. Qiu, "Achieving autonomous power management using reinforcement learning," *ACM Trans. Design Autom. Electron. Syst.*, vol. 18, no. 2, pp. 1–32, Mar. 2013.

[20] Z. Wang, Z. Tian, J. Xu, R. K. V. Maeda, H. Li, P. Yang, Z. Wang, L. H. K. Duong, Z. Wang, and X. Chen, "Modular reinforcement learning for self-adaptive energy efficiency optimization in multicore system," in *Proc. 22nd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2017, pp. 684–689.

[21] R. G. Kim, W. Choi, Z. Chen, J. R. Doppa, P. Pande, D. Marculescu, and R. Marculescu, "Imitation learning for dynamic VFI control in large-scale manycore systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 9, pp. 2458–2471, Sep. 2017.

[22] C.-H. Chen and C.-B. Liu, "Reinforcement learning-based differential evolution with cooperative coevolution for a compensatory neuro-fuzzy controller," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 10, pp. 4719–4729, Oct. 2018.

[23] R. G. Kim, W. Choi, Z. Chen, P. P. Pande, D. Marculescu, and R. Marculescu, "Wireless NoC and dynamic VFI codesign: Energy efficiency without performance penalty," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 7, pp. 2488–2501, Jul. 2016.

[24] R. Ye and Q. Xu, "Learning-based power management for multi-core processors via idle period manipulation," in *Proc. 17th Asia South Pacific Design Autom. Conf.*, Jan. 2012, pp. 115–120.

[25] M. F. Reza, "Machine learning for design and optimization challenges in multi/many-core network-on-chip," in *Proc. 14th Int. Workshop Netw. Chip Architectures*, Oct. 2021, pp. 29–34.

[26] D.-C. Juan, S. Garg, J. Park, and D. Marculescu, "Learning the optimal operating point for many-core systems with extended range voltage/frequency scaling," in *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES+ISSS)*, Sep. 2013, p. 8.

[27] U. A. Khan and B. Rinner, "Online learning of timeout policies for dynamic power management," *ACM Trans. Embedded Comput. Syst.*, vol. 13, no. 4, pp. 1–25, Dec. 2014.

[28] E. Cai, D.-C. Juan, S. Garg, J. Park, and D. Marculescu, "Learning-based power/performance optimization for many-core systems with extended-range voltage/frequency scaling," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 35, no. 8, pp. 1318–1331, Aug. 2016.

[29] F. M. M. U. Islam and M. Lin, "Hybrid DVFS scheduling for real-time systems based on reinforcement learning," *IEEE Syst. J.*, vol. 11, no. 2, pp. 931–940, Jun. 2017.

[30] R. G. Kim, W. Choi, G. Liu, E. Mohandesi, P. P. Pande, D. Marculescu, and R. Marculescu, "Wireless NoC for VFI-enabled multicore chip design: Performance evaluation and design trade-offs," *IEEE Trans. Comput.*, vol. 65, no. 4, pp. 1323–1336, Apr. 2016.

[31] F. M. M. ul Islam, M. Lin, L. T. Yang, and K.-K.-R. Choo, "Task aware hybrid DVFS for multi-core real-time systems using machine learning," *Inf. Sci.*, vols. 433–434, pp. 315–332, Apr. 2018.

[32] X. Chen, Z. Xu, H. Kim, P. Gratz, J. Hu, M. Kishinevsky, and U. Ogras, "In-network monitoring and control policy for DVFS of CMP networks-on-chip and last level caches," in *Proc. IEEE/ACM 6th Int. Symp. Netw. Chip*, May 2012, pp. 43–50.

[33] Y.-L. Chen, M.-F. Chang, C.-W. Yu, X.-Z. Chen, and W.-Y. Liang, "Learning-directed dynamic voltage and frequency scaling scheme with adjustable performance for single-core and multi-core embedded and mobile systems," *Sensors*, vol. 18, no. 9, p. 3068, Sep. 2018.

[34] S. Majzoub, R. Saleh, S. J. E. Wilton, and R. Ward, "Removal-cost method: An efficient voltage selection algorithm for multi-core platforms under PVT," in *Proc. IEEE Int. SOC Conf. (SOCC)*, Sep. 2009, pp. 357–360.

[35] M. Buda, A. Maki, and M. A. Mazurowski, "A systematic study of the class imbalance problem in convolutional neural networks," *Neural Netw.*, vol. 106, pp. 249–259, Oct. 2018.

[36] U. Gupta, M. Babu, R. Ayoub, M. Kishinevsky, F. Paterna, and U. Y. Ogras, "STAFF: Online learning with stabilized adaptive forgetting factor and feature selection algorithm," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6.

[37] W. Choi, K. Duraisamy, R. G. Kim, J. R. Doppa, P. P. Pande, D. Marculescu, and R. Marculescu, "On-chip communication network for efficient training of deep convolutional networks on heterogeneous manycore systems," *IEEE Trans. Comput.*, vol. 67, no. 5, pp. 672–686, May 2018.

[38] T. Kim, X. Huang, H.-B. Chen, V. Sukharev, and S. X.-D. Tan, "Learning-based dynamic reliability management for dark silicon processor considering EM effects," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2016, pp. 463–468.

[39] Z. Chen and D. Marculescu, "Distributed reinforcement learning for power limited many-core system performance optimization," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2015, pp. 1521–1526.

[40] T. Kodaka, A. Takeda, S. Sasaki, A. Yokosawa, T. Kizu, T. Tokuyoshi, H. Xu, T. Sano, H. Usui, J. Tanabe, T. Miyamori, and N. Matsumoto, "A near-future prediction method for low power consumption on a many-core processor," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2013, pp. 1058–1059.

[41] N. Drego, A. Chandrakasan, D. Boning, and D. Shah, "Reduction of variation-induced energy overhead in multi-core processors," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 30, no. 6, pp. 891–904, Jun. 2011.

[42] Y. You, S. L. Song, H. Fu, A. Marquez, M. M. Dehnavi, K. Barker, K. W. Cameron, A. P. Randles, and G. Yang, "MIC-SVM: Designing a highly efficient support vector machine for advanced modern multi-core and many-core architectures," in *Proc. IEEE 28th Int. Parallel Distrib. Process. Symp.*, May 2014, pp. 809–818.

[43] P. Mercati, F. Paterna, A. Bartolini, L. Benini, and T. S. Rosing, "Dynamic variability management in mobile multicore processors under lifetime constraints," in *Proc. IEEE 32nd Int. Conf. Comput. Design (ICCD)*, Oct. 2014, pp. 448–455.

[44] M. M. Hasan, M. S. Rahman, and A. Bell, "Deep reinforcement learning for optimization," in *IEEE 33rd Int. System Chip Conf. (SOCC)*, Dec. 2019, vol. 68, no. 10, pp. 180–196.

[45] M. F. Reza, "Reinforcement learning based dynamic link configuration for energy-efficient NoC," in *Proc. IEEE 63rd Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2020, pp. 468–473.

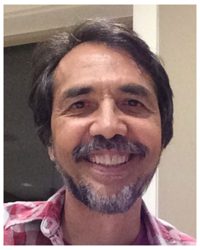
[46] N. Wu, L. Deng, G. Li, and Y. Xie, "Core placement optimization for multi-chip many-core neural network systems with reinforcement learning," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 26, no. 2, pp. 1–27, Feb. 2021.

[47] B. Zimmer, R. Venkatesan, Y. S. Shao, J. Clemons, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. S. Emer, C. T. Gray, S. W. Keckler, and B. Khailany, "A 0.11 pJ/op, 0.32–128 TOPS, scalable multi-chip-module-based deep neural network accelerator with ground-reference signaling in 16 nm," in *Proc. Symp. VLSI Circuits*, Jun. 2019, pp. C300–C301.

[48] G. Roffo, S. Melzi, U. Castellani, and A. Vinciarelli, "Infinite latent feature selection: A probabilistic latent graph-based ranking approach," in *Proc. IEEE Int. Conf. Comput. Vis.*, Jul. 2017, pp. 1398–1406.

[49] T. Tobita and H. Kasahara, "A standard task graph set for fair evaluation of multiprocessor scheduling algorithms," *J. Scheduling*, vol. 394, no. 5, pp. 379–394, 2002.

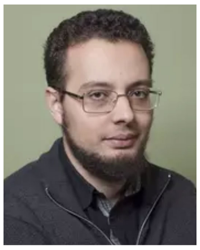
SOHAIB MAJZOUB (Senior Member, IEEE) received the B.E. degree in electrical engineering from the Computer Section, BAU, in 2000, the M.E. degree from AUB, Lebanon, in 2003, and the Ph.D. degree from the System-on-Chip Research Laboratory, The University of British Columbia, Canada. Then, he worked for one year at the Processor Architecture Laboratory, Swiss Federal Institute of Technology, Lausanne, Switzerland. He worked for two years as an Assistant Professor at American University in Dubai, Dubai, United Arab Emirates. Then, he joined King Saud University, Saudi Arabia, in 2012 for three years. In 2015, he joined the Department of Electrical and Computer Engineering, University of Sharjah, United Arab Emirates, as a Faculty Member. He was promoted to an Associate Professor, in June 2020. His research interests include low power digital electronics and embedded systems. He is a member of the IEEE Computer Society, Solid State Society, and Circuits and Systems.



RESVE SALEH (Fellow, IEEE) received the B.S. degree in electrical engineering from Carleton University, Ottawa, Canada, and the M.S. and Ph.D. degrees in electrical engineering from the University of California, Berkeley, CA, USA.

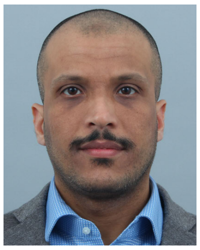
He was a Professor and the NSERC/PMC-Sierra Chairholder with the Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, Canada, in the field of system-on-chip design and test. He was a Founder of Simplex Solutions which developed CAD software for deep submicron digital design verification. Prior to starting Simplex, he spent nine years as a Professor with the Department of Electrical and Computer Engineering, University of Illinois Urbana–Champaign. He also taught for one year at Stanford University. He has worked for Mitel Corporation, Ottawa; Toshiba Corporation, Japan; Tektronix, Beaverton, OR, USA; and Nortel, Ottawa. He has published over 100 journal articles and conference papers. He coauthored a book entitled "*Design and Analysis of Digital Integrated Circuit Design: In deep submicron technology*."

Dr. Saleh is a Professional Engineer of British Columbia. He received the Presidential Young Investigator Award from the National Science Foundation, USA, in 1990. He served as General Chair, in 1995, a Conference Chair, in 1994, and a Technical Program Chair, in 1993 for the Custom Integrated Circuits Conference. He held the positions of a Technical Program Chair, a Conference Chair, and a Vice-General Chair of the International Symposium on Quality in Electronic Design, in 2001. He has served as an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN (CAD).



MOTTAQIALLAH TAOUIL (Member, IEEE) received the M.Sc. and Ph.D. degrees (Hons.) in computer engineering from the Delft University of Technology, Delft, The Netherlands. He is currently a Postdoctoral Researcher with the Dependable Nano-Computing Group, Delft University of Technology. His current research interests include reconfigurable computing, embedded systems, very large scale integration design and test, built-in-self-test, and 3D stacked integrated circuits, architectures, design for testability, yield analysis, and memory test structures.

SAID HAMDIOUI (Senior Member, IEEE) is currently a Chair Professor of dependable and emerging computer technologies at the Computer Engineering Laboratory, Delft University of Technology (TUDelft), The Netherlands. Prior to joining TUDelft, he worked for Intel Corporation, Santa Clara, CA, USA; Philips Semiconductors Research and Development, Crolles, France; and Philips/ NXP Semiconductors, Nijmegen, The Netherlands. He owns one patent and has published one book and coauthored over 170 conference and journal papers. He delivered dozens of keynote speeches, distinguished lectures, and invited presentations and tutorial at major international forums/conferences/schools and at leading semiconductor companies. His research interests include two domains: dependable CMOS nano-computing (including reliability, testability, and hardware security) and emerging technologies and computing paradigms (including 3D stacked ICs, memristors for logic and storage, and in-memory-computing). He is a member of AENEAS/ENIAC Scientific Committee Council (AENEAS =Association for European NanoElectronics Activities). He is an Associate Editor of IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION SYSTEMS (TVLSI), and he serves on the Editorial Board for IEEE DESIGN AND TEST, and of the *Journal of Electronic Testing: Theory and Applications* (JETTA).



MOHAMED BAMAKHRAMA received the B.Sc. degree in computer engineering from the University of Sharjah, United Arab Emirates, in 2005, the M.Sc. degree from the Technical University of Munich, Germany, in 2007, and the Ph.D. degree in computer science from Leiden University, The Netherlands, in 2014.

He is currently a Staff Engineer at Synopsys Corporation, Eindhoven, The Netherlands working on processor modeling, simulation, and architecture. Prior to Synopsys, he worked for several years at NXP Semiconductors, Intel Corporation, and ASML. His research interests include embedded multicore systems design and programming, real-time systems, computer architecture, and computer communication protocols and security.

Dr. Bamakhrama is a Senior Member of the Institute of Electrical and Electronics Engineers (IEEE). He was a recipient of the ACM Special Interest Group in Embedded Systems (SIGBED) Frank Anger Memorial Award, in 2011.