

# Selecting Microarchitecture Configuration of Processors for Internet of Things

Prasanna Kansakar, *Student Member, IEEE* and Arslan Munir, *Senior Member, IEEE*

**Abstract**—The Internet of Things (IoT) makes use of ubiquitous internet connectivity to form a network of everyday physical objects for purposes of automation, remote data sensing and centralized management/control. IoT objects need to be embedded with processing capabilities to fulfill these services. The design of processing units for IoT objects is constrained by various stringent requirements, such as performance, power, thermal dissipation etc. In order to meet these diverse requirements, a multitude of processor design parameters need to be tuned accordingly. In this paper, we propose a temporally efficient design space exploration methodology which determines power and performance optimized microarchitecture configurations. We also discuss the possible combinations of these microarchitecture configurations to form an effective two-tiered heterogeneous processor for IoT applications. We evaluate our design space exploration methodology using a cycle-accurate simulator (ESESC) and a standard set of PARSEC and SPLASH2 benchmarks. The results show that our methodology determines microarchitecture configurations which are within 2.23%–3.69% of the configurations obtained from fully exhaustive exploration while only exploring 3%–5% of the design space. Our methodology achieves on average  $24.16\times$  speedup in design space exploration as compared to fully exhaustive exploration in finding power and performance optimized microarchitecture configurations for processors.

**Index Terms**—Internet of Things (IoT), design space exploration, microarchitecture, tunable processor parameters, cycle-accurate simulator (ESESC), PARSEC and SPLASH2 benchmarks

## I. INTRODUCTION AND MOTIVATION

THE internet has grown rapidly in both enterprise and consumer markets. This has given rise to the Internet of Things (IoT) wherein everyday physical objects are interconnected through a communication network for purposes of automation, remote data sensing and centralized management/control. The IoT creates an intelligent, invisible network fabric that can be sensed, controlled and programmed which allows objects in IoT ecosystem to communicate, directly or indirectly, with each other or the Internet [1]. The “things”, in the scope of IoT, are IoT enabled objects containing sensing and actuating elements along with embedded hardware and software components which facilitate data aggregation, network connectivity and security. Each IoT enabled object is designed to perform an application specific task using data gathered by itself or using information made available to it through other objects in the network. There has been widespread deployment of IoT objects in recent years in various applications like healthcare, industry, transportation

etc. It is estimated that 6.4 billion connected end-devices are in use in the year 2016 [2], with the number expected to rise to 26 billion by the year 2020 [1].

The massive deployment of IoT objects results in generation of large volumes of data. Data communication, processing, real-time analysis and security of such large volumes of data are important issues that need to be resolved for efficient growth of the IoT ecosystem in the years to come. In the current IoT model, IoT end-devices are designed to be as simple and as cost effective as possible. Thus, they are designed with limited processing capabilities, just enough to securely connect and offload data to the cloud. Almost all complex data management functionalities such as data filtering and analysis are delegated to cloud datacenters, the core units of the IoT model. With the growth in data volume in the IoT ecosystem, there rises several significant challenges which renders this model infeasible. We list here three such challenges.

- *Network Overload* - Core network bandwidth is a vital resource in the IoT ecosystem which must be used efficiently. With ever increasing number of IoT objects, relaying data over the core network to the cloud, the network is severely overloaded. Network overloads introduce latency in critical data processing operations which impact most IoT applications such as healthcare and transportation that require real time data processing.
- *Data security* - Data communication in the IoT ecosystem mostly occurs over the public network infrastructure. In order to ensure secure data communication, several complex security protocols must be applied to the data. The volume of data requiring security increases as the number of IoT objects deployed in the IoT ecosystem increases. Applying complex security protocols to large volumes of data requires extensive computing operations which cannot be matched by the energy budget of IoT objects.
- *Upgradability* - As the IoT landscape continues to evolve, it becomes necessary to upgrade IoT deployments in frequent periods. IoT objects must be designed to support hassle free addition of new features via remote access. In an ideal IoT model, IoT objects must be able to upgrade to new, more complex features without deployment of new IoT objects and without any direct human involvement. With limited processing ability, addition of new features to existing IoT objects may be challenging or even infeasible.

The challenges posed by the current IoT model can be overcome by adding processing capabilities inside or local

to IoT objects [3]. With the added processing units, data management operations such as filtering and analysis can be carried out within the local network. IoT objects can thus, communicate summaries of information, obtained from filtering the aggregated data, to the cloud. This contributes significantly to freeing up the core network bandwidth. The reduction in data volume also reduces the energy expenditure on data security as less data requires lesser number of computing operations to secure. Having more processing ability also makes IoT deployments more flexible to upgrades as newer features can be added without significantly burdening the system.

Processing units interfaced with IoT objects require an optimal balance between power and performance [4]. Since many IoT objects are battery powered, it is desirable that these objects operate for their entire lifetime with the battery they are deployed with (e.g. medical sensors implanted into a patient's body via invasive surgical process). Although great progress has been made in battery technology, batteries are still not able to keep pace with the demands of modern electronics [5]. So, power optimization must be considered in parallel with performance optimization.

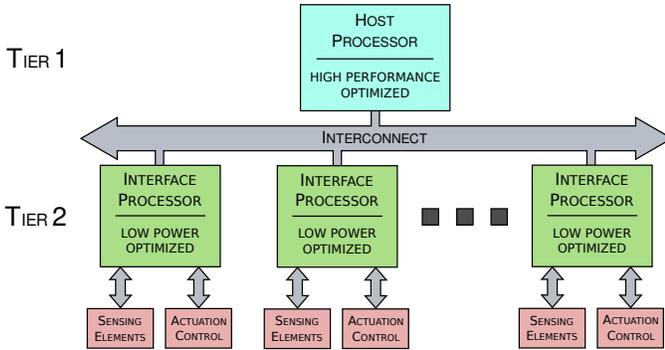


Fig. 1. Two-tiered heterogeneous processor architecture model for IoT

For incorporating higher levels of power optimized performance in IoT deployments, a two-tiered heterogeneous processor architecture is suitable [3] [6]. This two-tiered architecture, shown in Figure 1, consists of a host processor, optimized for high performance, interfaced with a number of interface processors, optimized for low power operation. The interface processors collect data from data-sensing elements and control actuating elements. These processors are always operated in active mode because their low power operation does not severely impact battery life. Higher end function, such as filtering and analysis of data, and, implementation of complex security protocols are performed by the host processor. Since these operations are infrequent, the power hungry host processor is mostly operated in sleep state and only activated intermittently for limited durations.

Designing efficient embedded processors with power-optimized performance, for use in IoT objects, is a tedious process. Preventing high performance processors from violating the power budget requirements dictated by the market is an enormous design challenge [7]. The opportunities for optimizing a processor design for power

are the greatest at the architecture level [7]. Thus, power and performance optimizations should be performed while defining the microarchitecture configuration of processors. The microarchitecture configuration consists of several processor design parameters each of which has to be tuned based on the impact it has on the overall power and performance of the processor. Selecting a microarchitecture configuration involves rigorous design space exploration over a search space consisting of all possible settings for tunable processor design parameters. There are two main challenges that need to be addressed in this process.

Firstly, the design space exploration methodology, employed to select microarchitecture configurations of processors for IoT objects, must be temporally efficient. Long processor design time leads to long time to market which results in lowered profits [8] [9] and shorter product life cycle [9]. The IoT market also lacks accepted industry standards so, those who get to the market first have the greatest opportunity to influence those standards [9].

Secondly, the design space exploration methodology must balance processor power consumption with performance, which are conflicting design metrics [10]. It is not possible to have optimal solutions for optimization problems with conflicting design metrics. The optimization problem should instead be modeled as an Optimal Production Frontier problem also known as Pareto Efficiency [11] problem. Multiple solutions are obtained for such problems where each solution favors one of the conflicting metrics. The design space exploration methodology must intelligently choose the best trade-off solution based on application specific requirements.

In this paper, we propose a temporally efficient design space exploration methodology for determining power and performance optimized microarchitecture configurations of embedded processors used in IoT objects. We use a combination of exhaustive, greedy and one-shot search methods to perform design space exploration. We verify the effectiveness of our methodology by testing it on a cycle accurate simulator using a large set of standard benchmarks with varying workloads.

The main contributions of our paper are:

- We propose a temporally efficient design space exploration methodology to find microarchitecture configurations for low-power and high-performance optimized embedded processors used in IoT objects.
- We include a threshold parameter in the design space exploration methodology which can be manipulated by the system designer to control design time based on time to market constraints.
- We propose exhaustive, greedy and one-shot search algorithms which yield microarchitecture configurations which are 2.23%-3.69% of the microarchitecture configurations obtained from fully exhaustive search.
- We distinguish between different microarchitecture configurations based on the size and type of benchmark used, and, relate them with potential use cases in IoT.

The remainder of the paper is organized as follows. In Section II, we present a review of related work. We describe our design space exploration methodology in Section III and

elaborate on its different phases in Section IV. In Section V we describe the cycle-accurate simulator and benchmarks used to test our methodology. We discuss the results in Section VI and present our conclusions and future research directions in Section VII.

## II. RELATED WORK

Several SoC design companies have released articles on techniques of increasing processing capabilities in IoT objects. Some articles guide the selection of processors for IoT objects while others describe low power optimized processor architectures for IoT deployments.

ARM proposed a processor architecture consisting of multiple homogeneous processors in a single IoT object each serving a different purpose [6]. They defined a system with three Cortex-M processors, one to handle network connectivity, one to manage interface with sensors and actuators and one as a host processor controlling the other two. They stated that multiple processors are better for lowering power consumption in IoT objects since only the processor serving the current task would be in active mode while the rest would be in sleep mode. ARM also proposed a guide to selecting microcontrollers for IoT objects [12]. In this guide, they argued that high-end microcontrollers were suitable for IoT deployments for two reasons. Firstly, high-end microcontrollers complete processing tasks sooner and can enter sleep mode to conserve power and secondly, larger flash and RAM sizes available with high-end microcontrollers facilitate implementation of complex networking protocols without addition of any new processors in the system. These articles clearly demonstrate the need for having more power-optimized performance in IoT deployments.

Synopsys also proposed the use of multiple processors in IoT deployments [13]. They described the use of two-tiered processor architecture in IoT objects – ultra low power embedded processors used to interface with sensing elements to collect, filter and process data and host processor used to manage embedded processors. Their processor architecture lowered power consumption by keeping power hungry host processor mostly in sleep mode, similar to the concept used by ARM. Synopsys also discussed optimization of processors using configurable hardware extensions for sensor applications [13]. They stated that adding custom hardware extensions for executing typical sensor functions reduces the processor cycle count required to execute sensor applications. The reduction in cycle count lowers energy consumption either by lowering the clock frequency and keeping the same execution time, or having the same power but shorter execution time.

Apart from research carried out by SoC design companies, processor design has also been extensively studied in academia [14] [15]. There are many research works in literature involving optimized processor design. Most works employ design space exploration [16] [17] techniques utilizing search methods like exhaustive and greedy search and optimizing algorithms like genetic and evolutionary algorithms. Givargis et al. [18] developed an exploration methodology named PLATUNE (PLATform TUNEr) that carried out exhaustive searches in two stages: first, over

clusters of strongly interconnected parameters to obtain Pareto-optimal configurations local to each cluster, and second, over all the clusters to obtain a global Pareto-optimal solution. The approach could explore design spaces as large as  $10^{14}$  configurations, but it took an order of 1-3 days to complete. Palesi et al. [19] argued that the high exploration time for PLATUNE was due to the formation of large partial search spaces in the clustering process. Palesi et al. improved the PLATUNE exploration methodology by introducing a new threshold value that distinguished between clusters based on the size of their partial search-space. Exhaustive search method was used for clusters with partial search-spaces smaller than the threshold value and a genetic exploration algorithm was used for larger spaces. Through this improvement, they were able to achieve 80% reduction in simulation time while still remaining within 1% of the results obtained from exhaustive search. Genetic algorithms were also used in the system MULTICUBE, by Silvano et al. [20]. The MULTICUBE system defined an automatic design space exploration algorithm that could quickly determine an approximate Pareto front for a given design requirements.

Munir et al. [21] proposed another alternative to overcome the overhead of exhaustive search in their work on dynamic optimization of wireless sensor networks. Their approach was divided into two phases. In the first phase, a one-shot search algorithm selected initial parameter settings and further ordered the parameters based on their significance towards the application requirements. In the second phase, a greedy algorithm was used to search the design space. Their approach yielded a design configuration that was within 8% of the optimal configuration while only exploring 1% of the design space.

In this paper, we improve on the work carried out by Munir et al. [21]. We leverage a similar approach to design space exploration but add two new phases: a set-partitioning phase and an exhaustive search phase. The addition of the exhaustive search phase aims at increasing the degree of closeness to the optimal solution by exploring a larger portion of the design space, as argued by Silvano et al. [20]. The limit on the number of configurations considered in the exhaustive search is determined by the set-partitioning phase that uses a threshold value [19].

## III. METHODOLOGY

Our design space exploration methodology for determining optimal microarchitecture configuration of embedded processors for IoT is shown in Figure 2. Our methodology is implemented in four phases – initial one-shot search configuration tuning and parameter significance, set-partitioning, exhaustive search configuration tuning and greedy search configuration tuning.

The initial one-shot search configuration tuning and parameter significance phase is carried out by the initial one-shot search configuration tuning module and the parameter significance ordering module. The microarchitecture configuration parameter settings set, which consists of all the possible settings for each tunable microarchitecture parameter, is provided as input to the initial one-shot search

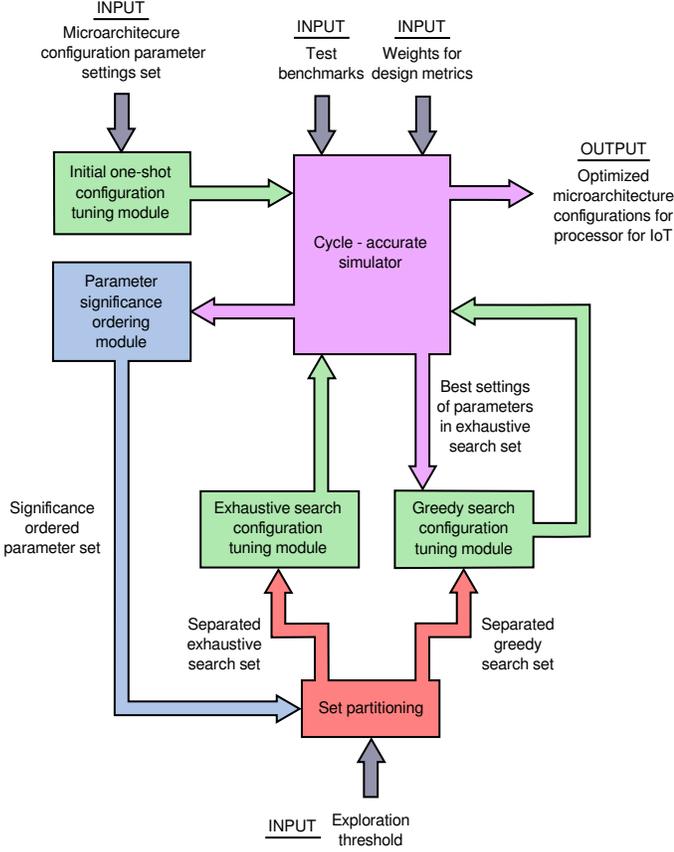


Fig. 2. Design space exploration methodology for determining optimal microarchitecture configuration of embedded processor for IoT

configuration tuning module by the system designer. This module uses the parameter settings set to generate initial test configurations. Each initial configuration is passed to a cycle-accurate simulator. The test benchmarks for evaluating the microarchitecture configurations are provided as input to the simulator by the system designer. The simulator executes each initial test configuration separately for each test benchmark specified. The test benchmarks provide varying workloads for testing the initial test configurations. The system designer also provides the weights for balancing design metrics as input to the simulator. These weights are used to specify the preferred tradeoff between conflicting design metrics.

The simulator module evaluates the initial test configurations supplied by the initial one-shot search configuration tuning module to determine the best initial setting for each tunable microarchitecture parameter. The simulation results are forwarded to the parameter significance ordering module where the tunable microarchitecture parameters are ordered based on their significance to the design metrics considered.

The ordered set of significance values is communicated to the set-partitioning module which separates the parameters into two search sets – exhaustive and greedy. The parameters are separated based on an exploration threshold value provided by the system designer. The exploration threshold value is used to control search space for the exhaustive search phase of our design space exploration methodology. The exhaustive search phase is the longest phase in the design space exploration

methodology and processor design time can be significantly altered by varying this exploration threshold value.

The microarchitecture parameters separated out in the exhaustive search set are communicated to the exhaustive search configuration tuning module. This module generates optimized test configurations using all possible combinations of tunable processor design parameters. The parameters which are not in the exhaustive search set retain their best settings from the initial one-shot search configuration tuning process. These test configurations are evaluated on the cycle-accurate simulator to determine a test configuration possessing the best tradeoff between the conflicting design metrics considered. The best settings for the microarchitecture parameters in the exhaustive search set are then communicated to the greedy search configuration tuning module.

The greedy search configuration tuning module generates test configurations using the processor design parameters separated out in the greedy search set. A greedy search algorithm (refer Section IV-D) is used to generate these test configurations. The microarchitecture parameters in the exhaustive search set retain their best setting obtained from the exhaustive search simulation process. The parameters which are in neither of the two search sets, retain their best settings from the initial one-shot search configuration tuning process. The best configuration obtained at the end of the greedy search configuration tuning process is communicated back to the processor designer as the optimal microarchitecture of the processor with the preferred tradeoff between the conflicting design metrics.

#### A. Defining the Design Space

Consider  $n$  number of tunable parameters are available to describe the microarchitecture configuration of an embedded processor for IoT. Let  $P$  be the list of these tunable parameters defined as the following set:

$$P = \{P_1, P_2, P_3, \dots, P_n\} \quad (1)$$

Each tunable parameter  $P_i$  [where  $i \in \{1, 2, \dots, n\}$ ] in the list  $P$  is the set of possible settings for  $i^{th}$  parameter. Let  $L$  be the set containing the size of the set of possible settings for each parameter in list  $P$ .

$$L = \{L_1, L_2, L_3, \dots, L_n\} \quad (2)$$

such that,

$$L_i = |P_i| \quad \forall i \in 1, 2, \dots, n \quad (3)$$

where  $|P_i|$  is the cardinal value of set  $P_i$ .

So, each parameter setting set  $P_i$  in the list  $P$  is defined as follows:

$$P_i = \{P_{i1}, P_{i2}, P_{i3}, \dots, P_{iL_i}\} \quad \forall i \in \{1, 2, \dots, n\} \quad (4)$$

The values in the set  $P_i$  are arranged in ascending order.

The state space for design space exploration is the collection of all the possible configurations that can be obtained using the  $n$  parameters.

$$S = P_1 \times P_2 \times P_3 \times \dots \times P_n \quad (5)$$

Here,  $\times$  represents the Cartesian product of lists in  $P$ . Throughout this paper, we use the term  $S$  to denote the state space composed of all  $n$  tunable parameters. To maintain generality, when referring to a state space composed of  $a$  tunable parameters where  $a < n$ , we attach a subscript to the term  $S$ .

$$S_a = P_1 \times P_2 \times P_3 \times \cdots \times P_a \quad \forall a < n \quad (6)$$

We note that the state space of  $a$  tunable parameters does not constitute a complete design configuration and is only used as an intermediate when defining our methodology.

We also reserve the use of  $\times$  operator in the following manner:

$$S_a = S_a \times P_i \quad \forall i \in \{1, 2, \dots, n\} \quad (7)$$

This represents the extension of the state space  $S_a$  to include one new set of parameter settings  $P_i$  from the list  $P$ . This operation increases the number of tunable parameters in state space  $a$  by one.

When referring to a design configuration that belongs to the state space  $S$ , we use the term  $s$ . We attach subscripts to  $s$  to refer to specific design configurations. For example, a state  $s_f$  that consists of the first setting of each tunable parameter can be written as:

$$s_f = (P_{11}, P_{21}, P_{31}, \dots, P_{n1}) \quad (8)$$

Similarly, to denote an incomplete/partial design configuration of  $a$  tunable parameters we use the term  $\delta s_a$ .

### B. Benchmarks

Each of the configurations, selected from the state space  $S$  by our methodology, is tested on  $m$  number of test benchmarks. The design metrics for each simulated configuration is collected separately for each benchmark.

### C. Objective Function

In our methodology, design configurations are compared with each other based on their objective functions. The objective function of a design configuration is the weighted sum of the normalized design metrics obtained after simulating that design configuration. Let  $o$  be the number of design metrics and  $V$  be the set of normalized values of design metrics which are obtained from the simulation.

$$V_s^k = \{V_{s1}^k, V_{s2}^k, V_{s3}^k, \dots, V_{so}^k\} \quad \forall k = 1, 2, \dots, m \quad (9)$$

Let  $w$  be the set of weights for the design metrics based on the requirements of the targeted application. These weights are set by the system designer.

$$w = \{w_1, w_2, w_3, \dots, w_o\} \quad (10)$$

such that,

$$0 \leq w_l \leq 1 \quad \forall l = 1, 2, \dots, o \quad (11)$$

and,

$$\sum w_l = 1 \quad \forall l = 1, 2, \dots, o \quad (12)$$

TABLE I  
LIST OF SYMBOLS

Symbol	Description
$n$	Number of tunable microarchitecture parameters
$P$	List of tunable microarchitecture parameters
$P_i$	Set of possible settings for $i^{th}$ tunable microarchitecture parameter
$L$	Size of set of possible settings for each tunable microarchitecture parameter
$L_i$	Cardinal value of set $P_i$
$S$	State space for design space exploration
$S_a$	Partial/Incomplete state space
$s_{tag}$	State in state space $S$ with 'tag' identifier
$\delta s_a$	State in partial state space $S_a$
$m$	Number of test benchmarks
$o$	Number of design metrics
$V_s^k$	Set of normalized values obtained for design metrics from simulation of state $s$ for $k^{th}$ benchmark
$w$	Set of weights for design metrics
$w_l$	Weight for $l^{th}$ design metric
$\mathcal{F}_s^k$	Objective function obtained from simulating state $s$ for $k^{th}$ benchmark

The objective function  $\mathcal{F}$  of a design configuration  $s$  for a test benchmark  $k$  is defined as follows:

$$\mathcal{F}_s^k = \sum w_l V_{sl}^k \quad \forall l = 1, 2, \dots, o \quad (13)$$

The optimization problem, considered in this paper, is to minimize the value of the objective function  $\mathcal{F}$ . The design metrics are chosen such that the minimization of their values is the favorable design choice. For example, when considering the performance metric, the design goal is to maximize performance. To model this into the objective function which we use execution time to measure performance. Minimizing execution time would fit with minimizing the objective function while still modeling the design goal of maximizing performance. The optimization problem for each test benchmark  $k$  is defined as follows:

$$\begin{aligned} \min. & \mathcal{F}_s^k \\ \text{s.t.} & s \in S \end{aligned} \quad (14)$$

Table I presents the symbols established in this section in list form.

## IV. PHASES OF METHODOLOGY

Our proposed design space exploration methodology consists of four distinct phases. In this section, we elaborate on the steps involved in each phase using the notation set up in Section III.

### A. Phase I : Initial One-Shot Search Configuration Tuning and Parameter Significance

In this phase of our methodology, best initial setting for each tunable microarchitecture parameter in set  $P$  is determined by using a one-shot search configuration tuning process. The one-shot search process is based on single factor analysis which is an effective heuristic approach used in design space exploration [22]. Unlike single factor analysis wherein parameters can have only two settings, a zero value and a non-zero value setting, one-shot search works on parameters

with more than two non-zero value settings. In one-shot search process, parameters are evaluated on a one by one basis. Two test configurations are generated for each parameter, one with the first setting and one with the last settings from the list of settings for the current parameter. The remaining parameters are arbitrarily set to their first setting from their corresponding list of settings.

---

**Algorithm 1:** Initial One-Shot Search Configuration Tuning and Parameter Significance

---

**Input:**  $P$  - List of Tunable Parameters

**Output:**  $B$  - Set of Best Settings;  $D$  - Significance of Parameters with respect to Objective Function

```

1 for  $i \leftarrow 1$  to  $n$  do
2    $s_f = \{P_{i1}\}$ 
3    $s_l = \{P_{iL[i]}\}$ 
4   for  $j \leftarrow 1$  to  $n$  do
5     if  $i \neq j$  then
6        $s_f = s_f \cup \{P_{j1}\}$ 
7        $s_l = s_l \cup \{P_{j1}\}$ 
8     end
9   end
10  for  $k \leftarrow 1$  to  $m$  do
11    Explore  $k^{th}$  benchmark using configuration  $s_f$ 
12    Calculate  $\mathcal{F}_{s_f}^k$ 
13    Explore  $k^{th}$  benchmark using configuration  $s_l$ 
14    Calculate  $\mathcal{F}_{s_l}^k$ 
15     $D_i^k = \mathcal{F}_l^k - \mathcal{F}_f^k$ 
16    if  $D_i^k > 0$  then
17       $B_i^k = P_{i1}$ 
18    else
19       $B_i^k = P_{iL[i]}$ 
20    end
21  end
22 end
```

---

The steps involved in initial one-shot search configuration tuning and determining parameter significance are detailed in Algorithm 1. The first and last test configurations generated for evaluating a tunable microarchitecture parameter,  $P_i$  in set  $P$ , are denoted by  $s_f$  and  $s_l$ , respectively. These configurations are tested on the cycle-accurate simulator. From the results of the simulation, objective functions,  $\mathcal{F}_{s_f}$  and  $\mathcal{F}_{s_l}$  corresponding to  $s_f$  and  $s_l$ , respectively, are determined. The objective function values are used to determine best initial setting as well as significance of each microarchitecture parameter. The magnitude of the difference between  $\mathcal{F}_{s_f}$  and  $\mathcal{F}_{s_l}$ , which is stored in parameter significance set  $D$  (line 15), is used as parameter significance. The higher the magnitude of a difference  $D_i^k$ ,  $i \in \{1, 2, 3, \dots, n\}$  for a benchmark  $k$ ,  $k \in \{1, 2, 3, \dots, m\}$ , the higher is the significance of parameter  $P_i$  to the workload characterized by benchmark  $k$ . The sign of the difference between  $\mathcal{F}_{s_f}$  and  $\mathcal{F}_{s_l}$  is used to pick the best initial setting for parameter  $P_i$ . If the difference is positive, then the first setting of parameter  $P_i$  is chosen as the best setting, otherwise the last setting is chosen. The best settings

for the parameters are stored in the set of best settings  $B_i^k$  (lines 17 and 19).

*B. Phase II : Set-Partitioning*

---

**Algorithm 2:** Set-Partitioning

---

**Input:**  $D$  - Significance of Parameters towards Objective Function;  $I$  - Index Set;  $T$  - Exhaustive Search Threshold Factor

**Output:**  $\mathcal{E}$  - Set of Parameters for Exhaustive Search;  $\mathcal{G}$  - Set of Parameters for Greedy Search

```

1  $\mathcal{E} = \emptyset$  and  $\mathcal{G} = \emptyset$ 
2 for  $k \leftarrow 1$  to  $m$  do
3   sortDescending ( $|D^k|$ ) - s.t. index information of
   the sorted values is preserved in  $I^k$ 
4   sort( $P^k$ ) and sort( $L^k$ ) w.r.t. index information in  $I^k$ 
5    $num_{\mathcal{E}} = 1$  and  $i = 1$ 
6   while  $num_{\mathcal{E}} \leq T$  do
7      $num_{\mathcal{E}} = num_{\mathcal{E}} \times L_i^k$ 
8     if  $num_{\mathcal{E}} \leq T$  then
9        $\mathcal{E}^k = \mathcal{E}^k \cup \{P_i\}$ 
10       $i = i + 1$ 
11    else
12      break
13    end
14  end
15   $num_{\mathcal{G}} = ceil((|P^k| - |\mathcal{E}^k|) / 2)$ 
16  while  $num_{\mathcal{G}} > 0$  do
17     $\mathcal{G}^k = \mathcal{G}^k \cup \{P_i^k\}$ 
18     $num_{\mathcal{G}} = num_{\mathcal{G}} - 1$ 
19     $i = i + 1$ 
20  end
21 end
```

---

The set-partitioning phase, presented in Algorithm 2, shows how the parameter significance values determined in the first phase of our methodology are used to separate the list of tunable microarchitecture parameters into exhaustive and greedy search sets. First, the parameter significance set  $|D^k|$  for each benchmark  $k$ ,  $k \in \{1, 2, 3, \dots, m\}$ , is sorted in descending order of magnitude using the sortDescending( $|D^k|$ ) function. The index information of the sorted values is preserved in a set of indexes  $I^k$  (line 3). For example, if the fifth entry  $D_5^k$  has the greatest value,  $D_5^k$  will become the first entry in the set  $D^k$  and first entry in the set of indexes  $I^k$  will be 5, that is,  $I_1^k = 5$ . The set of indexes,  $I^k$ , is used to sort the list of tunable microarchitecture parameters,  $P^k$ , and list of set sizes,  $L^k$ . After sorting, the parameters with higher significance lie towards the start of the set and the parameters with lower significance lie towards the end of the set. The list of parameters is then divided into three subsets, exhaustive search, greedy search and one-shot search sets. The exhaustive search set gets parameters with the highest significance. The number of parameters separated into the exhaustive search set depends on the exploration threshold value,  $T$ , provided by the system designer. The threshold value

$T$  limits the size of the partial search space of the exhaustive search set,  $num_{\mathcal{E}}$  (line 6).

After separating out exhaustive search set, the parameters remaining in the parameter list are separated into greedy search and one-shot search sets. The list of remaining parameter is divided into two halves (line 15) and the upper half  $ceil((|P| - |\mathcal{E}^k|)/2)$  is separated as the greedy search set and the lower half is separated as one-shot search set. We observe empirically that dividing the list of remaining parameters into halves provides efficient design space exploration without significantly compromising the solution quality. The parameters separated as one-shot search set are not explored further and are left at the best settings determined for them in Algorithm 1.

### C. Phase III : Exhaustive Search Configuration Tuning

---

#### Algorithm 3: Exhaustive Search

---

**Input:**  $P$  - List of Tunable Parameters;  $B$  - Set of Best Settings for One-shot Search;  $\mathcal{E}$  - List of Parameters for Exhaustive Search  
**Output:**  $B$  - List of Best Settings for One-shot and Exhaustive Search

```

1  $s_{\mathcal{E}} = \emptyset$ 
2  $\delta s_{\mathcal{E}} = \emptyset$  and  $\delta s_{\mathcal{E}'} = \emptyset$ 
3 for  $k \leftarrow 1$  to  $m$  do
4    $\mathcal{F}_{s_b}^k = \infty$ 
5   for  $i \leftarrow 1$  to  $n$  do
6     if  $P_i \notin \mathcal{E}^k$  then
7        $\delta s_{\mathcal{E}'}^k = \delta s_{\mathcal{E}'}^k \cup \{B_i^k\}$ 
8     end
9   end
10  for  $i \leftarrow 1$  to  $n$  do
11    if  $P_i \in \mathcal{E}^k$  then
12       $S_{\mathcal{E}}^k = S_{\mathcal{E}}^k \times P_i$ 
13    end
14  end
15  for  $j \leftarrow 1$  to  $|S_{\mathcal{E}}^k|$  do
16     $\delta used s_{\mathcal{E}'}^k$  is a partial configuration in state
      space  $S_{\mathcal{E}}^k$ 
17     $s_{\mathcal{E}}^k = \delta s_{\mathcal{E}'}^k \cup \delta s_{\mathcal{E}}^k$ 
18    Explore  $k^{th}$  benchmark using configuration  $s_{\mathcal{E}}^k$ 
19    Calculate  $\mathcal{F}_{\mathcal{E}}^k$ 
20    if  $\mathcal{F}_{s_{\mathcal{E}}^k}^k < \mathcal{F}_{s_b}^k$  then
21       $\mathcal{F}_{s_b}^k = \mathcal{F}_{s_{\mathcal{E}}^k}^k$ 
22       $B^k = s_{\mathcal{E}}^k$ 
23    end
24  end
25 end
```

---

Algorithm 3 details the steps involved in the exhaustive search process. The exhaustive search process determines the best settings for the parameters in the exhaustive search set  $\mathcal{E}$ . First, the settings for the parameters that are not in the exhaustive search set  $\mathcal{E}$  are assigned (line 7). These parameters are assigned their best settings from the set of

best settings  $B_i^k$  as determined in the initial one-shot search configuration tuning process described in Algorithm 1. These settings make up the partial test design configuration  $\delta s_{\mathcal{E}'}$ . Next, a partial state space  $S_{\mathcal{E}}$  is formed for the parameters in the exhaustive search set  $\mathcal{E}$  (line 12). Every possible partial test design configuration,  $\delta s_{\mathcal{E}j}$  (line 16), in the partial state space  $S_{\mathcal{E}}$ , is combined with the partial test design configuration  $\delta s_{\mathcal{E}'}$  to form complete simulatable test design configurations. Each complete test design configuration is evaluated on the simulator. An objective function value,  $\mathcal{F}_{s_{\mathcal{E}}}$ , is obtained for each complete test design configuration,  $s_{\mathcal{E}}$ , from the simulator. The algorithm keeps track of the smallest objective function value encountered in the search process in  $\mathcal{F}_{s_b}$ , which represents the best objective function value. When a design configuration results in an objective function that has a value less than  $\mathcal{F}_{s_b}$  (line 20), then  $\mathcal{F}_{s_b}$  is changed to the new minimum value and the set of best settings  $B$  is updated with the corresponding design configuration.

### D. Phase IV : Greedy Search Configuration Tuning

In the final phase of our methodology, described in Algorithm 4, the best settings for the parameters in greedy search set  $\mathcal{G}$  are determined. For each parameter in the set  $\mathcal{G}$ , the sign of the parameter significance is checked to determine whether the first setting or last setting was chosen as the best setting in the first phase of our methodology. If the sign of parameter significance is positive, then it indicates that first setting for that parameter yields a smaller objective function as compared to the last. If the sign is negative then it indicates that the last setting for that parameter yields a smaller objective function as compared to the first. We assume that the setting that yields the smallest objective function lies closer towards the setting that yields the smallest objective function in the initial one-shot search configuration tuning process. To ensure that the search process starts from the setting that yielded the smallest objective function in the initial one-shot search configuration tuning process, we sort the set of parameter settings  $P_i$  in descending order (for last setting as best setting) or left unchanged in default ascending order (for first setting as best setting) (line 8).

In the greedy search process, the parameters in the greedy search set are considered one at a time. First, a partial test design configuration  $\delta s_{\mathcal{G}p'}$  is formed using the exhaustive search set, the one-shot search set and the non-current parameters in greedy search set. The parameters in the exhaustive search set,  $\mathcal{E}$ , are assigned their best values as determined in the exhaustive search configuration tuning process. The parameters in the one-shot search set retain the best settings determined in the initial one-shot configuration tuning process. The non-current parameters in the greedy search set,  $\mathcal{G}$ , are assigned best settings in one of two ways. If the non-current parameter has already been processed by the greedy search optimization process, then the parameter is assigned the best setting obtained from that process. If the non-current parameter has not been processed yet, then the parameter is assigned the best setting obtained from the initial one-shot search configuration tuning process.

**Algorithm 4: Greedy Search**

**Input:**  $P$  - List of Tunable Parameters,  $D$  - Significance of Parameters towards Objective Function,  $B$  - Set of Best Settings for One-shot and Exhaustive Search,  $\mathcal{E}$  - Set of Parameters for Exhaustive Search,  $\mathcal{G}$  - Set of Parameters for Greedy Search

**Output:**  $B$  - Complete set of Best Settings

```

1  $s_{\mathcal{G}} = \emptyset$ 
2  $\delta s_{\mathcal{G}'} = \emptyset$ 
3  $\mathcal{G}_P = \emptyset$ 
4 for  $k = 1$  to  $m$  do
5    $\mathcal{F}_{s_b}^k = \infty$ 
6   for  $i \leftarrow 1$  to  $n$  do
7     if  $P_i \in \mathcal{G}^k$  then
8       if  $D_i^k < 0$  then
9          $\mathcal{G}_P = \text{sortDescending}(P_i)$ 
10      end
11     for  $j \leftarrow 1$  to  $n$  do
12       if  $P_j \notin \mathcal{G}_P$  then
13          $\delta s_{\mathcal{G}'}^k = \delta s_{\mathcal{G}'}^k \cup \{B_j^k\}$ 
14       end
15     end
16     for  $l \leftarrow 1$  to  $L_i$  do
17        $s_{\mathcal{G}}^k = \delta s_{\mathcal{G}'}^k \cup \{\mathcal{G}_{Pl}\}$ 
18       Explore  $k^{\text{th}}$  benchmark using
19       configuration  $s_{\mathcal{G}}^k$ 
20       Calculate  $\mathcal{F}_{s_{\mathcal{G}}}^k$ 
21       if  $\mathcal{F}_{s_{\mathcal{G}}}^k < \mathcal{F}_{s_b}^k$  then
22          $\mathcal{F}_{s_b}^k = \mathcal{F}_{s_{\mathcal{G}}}^k$ 
23          $B_i^k = \mathcal{G}_{Pj}$ 
24       else
25         break
26       end
27     end
28   end
29 end

```

The partial test design configuration  $\delta s_{\mathcal{G}'}$  is then combined with the settings for the current parameter being processed to form the complete simulatable test design configuration  $s_{\mathcal{G}}$  (line 17). This configuration is evaluated on the cycle-accurate simulator. The resulting objective function,  $\mathcal{F}_{s_{\mathcal{G}}}$ , is compared with the best objective function  $\mathcal{F}_{s_b}$ , which holds the smallest value objective function encountered thus far in the search process. Similar to the exhaustive search process, when a design configuration results in an objective function that has a value less than  $\mathcal{F}_{s_b}$  (line 20), then  $\mathcal{F}_{s_b}$  is changed to the new minimum value and the set of best settings  $B_i^k$  is updated with the corresponding design configuration. However, when the search process encounters a design configuration that results in an objective function that has a value greater than  $\mathcal{F}_{s_b}$ , then the search process for the current parameter is terminated and the next parameter in the parameter list  $\mathcal{G}$  is explored.

## V. EXPERIMENTAL SETUP

We used the ESESC [23] (Enhanced Super EScalar) simulator to simulate all the test microarchitecture configurations generated by our methodology. The ESESC simulator is a fast cycle-accurate chip multiprocessor simulator. It models an out-of-order RISC (Reduced Instruction Set Computing) processor running ARM instruction set.

We used benchmarks from the PARSEC and SPLASH2 [24], [25] benchmark suite to test our methodology. The PARSEC and SPLASH2 benchmark suite is a collection of standardized benchmarks which provides a diverse range of workloads for evaluation of processors.

We used the following benchmarks from the PARSEC and SPLASH2 suite to test our methodology.

**PARSEC Benchmarks:** Blackscholes, Canneal, Facesim, Fluidanimate, Freqmine, x264

**SPLASH2 Benchmarks:** Cholesky, FFT, LU\_cb, LU\_ncb, Ocean\_cp, Ocean\_ncp, Radiosity, Radix, Raytrace

The methodology phases were implemented using PERL [26]. The results from the simulation processes were collected in MS Excel using Excel-Writer-XLSX [27] tool for PERL.

We tested our design space exploration methodology separately for low-power and high-performance processor design. We combined the microarchitecture configurations obtained from these tests to form a two-tiered heterogeneous processor architecture. The microarchitecture configuration obtained from the low-power processor design tests were used to implement the low-power optimized interface processors, the lower tier of the two-tiered architecture. The microarchitecture configuration obtained from the high-performance processor design tests were used to implement the high-performance optimized host processor, the upper tier of the two-tiered architecture.

TABLE II  
MIRCOARCHITECTURE CONFIGURATION PARAMETER SETTINGS SET

Parameter Name	Set of Settings	
	Low-Power	High-Performance
Cores	1, 2, 4	2, 4, 8
Frequency (MHz)	75, 100, 125, 150	1700, 2200, 2800, 3200
L1-I Cache Size (kB)	8, 16, 32, 64	8, 16, 32, 64, 128
L1-D Cache Size (kB)	8, 16, 32, 64	8, 16, 32, 64, 128
L2 Cache Size (kB)	256, 512, 1024	256, 512, 1024
L3 Cache Size (kB)	2048, 4096	2048, 4096, 8192

The list of microarchitecture parameters considered for testing our methodology along with the set of possible settings for each parameter is listed in Table II. We used different range of settings for low-power and high-performance processor design. The range of settings listed in Table II under low-power design were used for the design of low-power optimized interface processors. The design space cardinality for low-power processor design was 1,152 configurations. The range of settings listed in Table II under high-performance design were used for the design of high-performance optimized host processor. The design space cardinality for high-performance processor design was 2,700 configurations.

TABLE III  
WEIGHTS FOR DESIGN METRICS

Configuration	Power	Performance
Low-Power	0.9	0.1
High-Performance	0.1	0.9

We used power and performance as design metrics to evaluate the microarchitecture configurations for both low-power and high-performance optimized processors. We used normalized value of total dynamic power and leakage power [28] across all the cores in the processor as the power metric and the normalized value of total execution time as the performance metric. We used the weights presented in Table III to specify the preference for the conflicting design metrics of power and performance. The linear objective function used for the evaluation of the test microarchitecture configurations was:

$$\mathcal{F} = w_P \cdot \mathcal{P} + w_E \cdot E \quad (15)$$

where,

$$\begin{aligned} \mathcal{P} &= \text{Dynamic Power} + \text{Leaked Power} \\ E &= \text{Total Execution Time} \end{aligned} \quad (16)$$

## VI. RESULTS

In this section, we present the results obtained while testing our methodology. This section is divided into two subsections. In the first subsection, we present results to validate our design space exploration methodology and in the second subsection, we discuss some of the applicability of some of the microarchitecture configurations to important IoT use cases.

### A. Evaluation of design space exploration methodology

For evaluating our methodology, we compared our microarchitecture configuration results with those obtained from a fully exhaustive search of the design space. We tested our methodology with an exploration threshold of  $T = 150$ . This threshold value is an upper bound which limits the partial state space for the exhaustive search phase of our methodology.

1) *Parameter significance*: Figure 3 shows the normalized values of parameter significance for different PARSEC benchmarks. The normalization is carried out using the maximum values for total power and total execution time obtained in the initial one-shot search configuration tuning process. The parameter significance values are calculated in the first phase of our methodology, initial one-shot search configuration tuning. We observe that the significance of each of the tunable processor design parameters varies based on the type of workload offered by the test benchmarks. For each of the test benchmarks, there are at most three significant processor design parameters. We note that the operating frequency is the processor design parameter with the highest significance for most of the test benchmarks followed by core count, which is the second most significant design parameter. For certain test benchmarks, the size of the L1-I cache and L1-D cache are also highly significant to overall design. The large significance in cache sizes is a result of large working sets with fine data-parallel granularity offered by those test benchmarks.

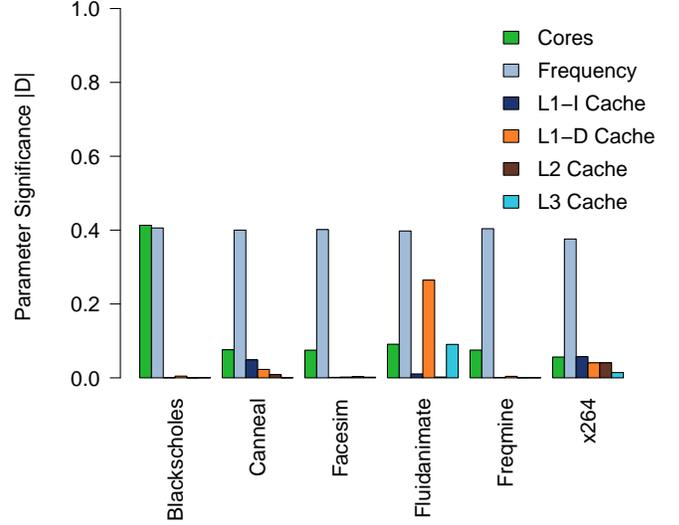


Fig. 3. Significance of microarchitecture configuration parameters for PARSEC benchmarks for high-performance optimized processor for IoT

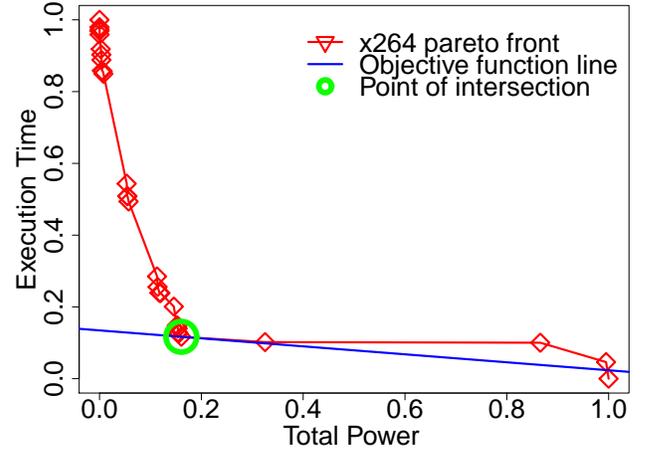


Fig. 4. Linear objective function plotted with Pareto front for x264 (PARSEC) benchmark for high-performance optimized processor for IoT

2) *Selecting a favorable tradeoff solution*: Figure 4 shows the Pareto front obtained for x264 (PARSEC) benchmark for high-performance optimization requirement. The Pareto front is generated using the normalized values of total power and execution time design metrics. The front represents the conflicting interdependency between power and performance in a processor. It shows that increasing the performance of a processor degrades its power efficiency whereas increasing power efficiency degrades performance. It is thus impossible to determine a microarchitecture configuration which results in both these metrics having optimal values. The goal of the design space exploration methodology is to determine a balance between these conflicting design metrics. A suitable tradeoff between these metrics is selected by using the preference specified using the weights assigned to each metric. In our experiments, we specified  $w_P$  and  $w_E$  as the weights for power and performance metrics respectively to define a linear objective function (Equation 15). Figure 4 shows the objective function plotted along with the Pareto front. We note that the objective function forms a straight line in the power-

TABLE IV  
COMPARISON OF MICROARCHITECTURE CONFIGURATIONS OBTAINED FOR X264 (PARSEC) BENCHMARK FOR HIGH-PERFORMANCE OPTIMIZED PROCESSOR FOR IOT

Parameter Name	Microarchitecture Configuration	
	Proposed Methodology	Fully Exhaustive Search
Cores	2	2
Frequency (MHz)	3200	3200
L1-I Cache Size (kB)	64	128
L1-D Cache Size (kB)	64	128
L2 Cache Size (kB)	1024	256
L3 Cache Size (kB)	2048	8192
Total Power (W)	1.597	1.600
Execution Time (ms)	35.142	34.152

performance graph with the slope  $-w_P/w_E$ . We observe that the objective function is tangent to the Pareto front at the power-performance value pair of the microarchitecture configuration obtained as solution by our methodology.

3) *Comparison with fully exhaustive search*: We verified the microarchitecture configuration obtained as solution from our methodology by comparing it against the solution obtained by running a fully exhaustive search of the design space. We present a comparison of the x264 (PARSEC) benchmark as an example in Table IV. The table shows a side-by-side comparison of the microarchitecture configurations obtained from our proposed methodology with the same obtained from fully exhaustive search. Comparing these values, we see that significant parameters like operating frequency and core-count match exactly while other parameters only differ slightly. The table also contains the values of the total power and execution time obtained for both configurations. Comparing the values of these design metrics, we see that the total power and execution time values obtained from our methodology are within -0.18% and 2.89% respectively of the total power and execution time values obtained from fully exhaustive exploration.

Using our methodology, on average we achieve microarchitecture configurations with total power values within 2.23% for low-power optimized processor and execution time within 3.69% for high-performance optimized processors as compared to fully exhaustive search. These configurations are obtained by exploring only 3%–5% of the processor design space which results in our methodology having an average speedup of  $24.16\times$  as compared to fully exhaustive exploration of the design space.

### B. Application scopes in IoT

Based on the type and size of workload offered by the test benchmarks, we separate them into four different categories each of which relates to an IoT application or process. Table V shows the categorization of some of the key test benchmarks. The Cholesky and Radix benchmarks from the SPLASH2 benchmark suite are categorized under data sensing and aggregation. The Cholesky benchmark is a sparse matrix factorization kernel and the Radix benchmark is an integer sort kernel [29]. The Cholesky benchmark is representative of data sensing in IoT applications, where data is acquired from multiple sensor sources and transformed into

TABLE V  
CATEGORIZATION OF TEST BENCHMARKS ACCORDING TO IOT APPLICATION

IoT Application	Benchmarks
Data sensing and aggregation	Cholesky, Radix
Data analysis and Data mining	Blackscholes, Freqmine
Graphics	Facesim, Fluidanimate
Signal processing and Communication	FFT

TABLE VI  
MICROARCHITECTURE CONFIGURATIONS FOR LOW-POWER OPTIMIZED PROCESSORS FOR IOT

Parameter Name	Microarchitecture Configuration	
	Cholesky	Radix
Cores	1	1
Frequency (MHz)	75	75
L1-I Cache Size (kB)	8	8
L1-D Cache Size (kB)	32	64
L2 Cache Size (kB)	256	256
L3 Cache Size (kB)	2048	4096
Total Power (W)	0.0934	0.0935
Execution Time (ms)	327.958	332.535

a more useful format. The Radix benchmark is representative of data aggregation, where indexing, sorting and storing operations are carried out on sensed data. These benchmarks are useful in determining the microarchitecture configurations of low-power optimized interface processors for the two-tiered heterogeneous processor architecture.

The remaining categories all model more complex applications requiring high level of processing capabilities. The Blackscholes and Freqmine benchmarks from the PARSEC benchmark suite are listed under data analysis and data mining. The Blackscholes benchmark is a financial analysis benchmark that analytically solves large sets of partial differential equations [24]. The Freqmine benchmark is a data mining kernel which implements Frequent Itemset Mining [24]. These benchmarks are representative of data analysis and filtering operations that need to be carried out on large volumes of sensor data in an IoT network.

The Facesim and Fluidanimate benchmarks from the PARSEC benchmark suite are listed under graphics. The Facesim benchmark generates a visually realistic model of a human face and the Fluidanimate benchmark simulates an incompressible fluid for interactive animation purposes [24]. Graphical applications are important in IoT objects which need to interact with users via graphical user interfaces.

The FFT benchmark from the SPLASH2 benchmark suite is listed under signal processing and communication. The FFT benchmark is an implementation of Fast Fourier Transform algorithm which is optimized to minimize interprocess communication [29]. Signal processing and communication is one of the most common applications in an IoT network. FFT is an important Digital Signal Processing (DSP) algorithm which is required in communication of data over Software Defined Radios (SDR) [14].

These benchmarks, which require higher processing capabilities, are useful in determining the microarchitecture configurations of high-performance optimized host processor for the two-tiered heterogeneous processor architecture.

TABLE VII  
MIRCOARCHITECTURE CONFIGURATION FOR HIGH-PERFORMANCE OPTIMIZED PROCESSORS FOR IOT

Parameter Name	Microarchitecture Configuration				
	Blackscholes	Freqmine	Facesim	Fluidanimate	FFT
Cores	8	2	2	2	4
Frequency (MHz)	3200	3200	3200	3200	3200
L1-I Cache Size (kB)	64	32	8	8	128
L1-D Cache Size (kB)	128	128	64	64	32
L2 Cache Size (kB)	256	1024	1024	1024	512
L3 Cache Size (kB)	8192	2048	8192	4096	2048
<b>Total Power (W)</b>	4.549	1.565	1.546	1.546	2.563
<b>Execution Time (ms)</b>	28.1239	67.319	60.072	55.605	29.986

1) *Microarchitecture configurations for low-power optimized processors for IoT*: Table VI shows the microarchitecture configurations obtained for Cholesky and Radix benchmarks from the SPLASH2 benchmark suite. In these configurations, we note that for low-power optimized processor, the lowest operating frequency and core count are selected. This result can be interpreted intuitively, because high operating frequency and high number of cores in the processor increases the power consumption of the processor. We also note that these configurations have large L1-D cache sizes. This is because of the large workload offered by the test benchmarks. This is representative of the growing IoT ecosystem in which large volumes of data are gathered from a large number of sensing elements. The values of total power and execution times for microarchitecture configurations are also shown in Table VI. We observe that the power values are in the range of a hundred milliwatts and the execution time is in the range of a few hundred milliseconds. These values are within the operational requirements in most IoT deployments. These configurations implement the interface processors in the two-tiered heterogeneous processor architecture. With low-power requirements, these processors can always be operated in active mode, without impacting the power budget of IoT deployments

2) *Microarchitecture configuration for high-performance optimized processors for IoT*: Table VII shows the microarchitecture configurations obtained for Blackscholes, Freqmine, Facesim and Fluidanimate benchmarks from the PARSEC benchmark suite and the FFT benchmark from the SPLASH2 benchmark suite. We analyze the microarchitecture configurations obtained for these test benchmarks according to the categorization discussed in subsection VI-B. We observe that for data analysis and data mining applications, represented by the Blackscholes and Freqmine benchmarks, higher performance is achieved primarily by the increase in operating frequency. We note that the size of the L1-D cache for these applications is also high, which is because both are highly data-parallel benchmarks. The size of the L2 cache, for Blackscholes, and, L3 cache, for Freqmine, is also high which is also a result of data-parallelism in these benchmarks. For graphics applications, represented by Facesim and Fluidanimate benchmarks, higher performance can again be attributed to increase in operating frequency. These benchmarks are also highly data-parallel which explains the large L1-D cache, L2 cache and L3 cache in the resulting microarchitecture configurations. In signal processing and

communication applications, represented by FFT benchmark, performance improvement, similar to other applications, is attained by increase in operating frequency. However, FFT requires a larger instruction cache as compared to larger data caches for other applications. Higher L1-I cache could be a result of the FFT benchmark being optimized for low interprocess communication.

The total power and execution time of each microarchitecture configuration is also listed in Table VII. These configurations have high total power values in the range of one to a few watts but significantly low execution time values in the range of few tens of milliseconds. These configurations implement the host processor in the two-tiered heterogeneous processor architecture. Due to their high-power requirement, these processors are mostly kept in sleep mode and are activated intermittently for short durations to save energy and prolong battery life. Because these processors have shorter execution times, they can execute their tasks quickly and go to sleep thus, decreasing the duration that they are active.

## VII. CONCLUSION AND FUTURE WORK

In this paper, we proposed a temporally efficient design space exploration methodology for selecting microarchitecture configurations of processors for IoT. Our exploration methodology consisted of four phases. In the first phase, we determined best initial settings for tunable processor design parameters using initial one-shot search method. We also calculated the significance of each design parameter on the overall design in this phase. The results of this phase were used in the second phase to separate the processor design parameters into distinct search sets using an exploration threshold value supplied by the system designer. The third and the fourth phase of the methodology implemented exhaustive and greedy search methods to prune these search sets to determine the best microarchitecture configuration of the processor.

We tested our methodology over two design spaces, one for determining low-power optimized and the other for determining high-performance optimized processors for IoT. We validated the results obtained from our methodology by comparing with solutions obtained from fully exhaustive exploration of the design spaces. Our results revealed that our methodology obtained microarchitecture configurations close to within 2.23%–3.69% of the configurations obtained from

fully exhaustive search. Our methodology only explored 3%–5% of the overall design space to determine these high quality solutions. This resulted in  $24.16\times$  average speedup on design space exploration as compared to the time required for fully exhaustive exploration.

We also described a two-tiered heterogeneous processor architecture for incorporating power-optimized performance in IoT objects. We used the results obtained from the evaluation of our design space exploration methodology to describe the two-tiered architecture. We categorized the test benchmarks into four different categories, relating them with possible IoT use cases and analyze microarchitecture configurations determined for these benchmarks to make our assertions on processors for IoT objects. We determined that for low-power optimization, microarchitecture configurations with lower core count and lower operating frequency are more suitable. For high-performance optimization, improvement in performance primarily results from increase in operating frequency. We also analyzed the cache hierarchy for different microarchitecture configurations and related them with the type and size of workloads offered by the test benchmarks.

In the future, we plan to investigate microarchitecture configurations of ultra-low power processors for IoT. We also intend to test our design space exploration methodology using standard IoT benchmarks. We also aim to improve our methodology by incorporating better optimization techniques like genetic and evolutionary algorithms and machine-learning. We also plan to study the practical applicability of the two-tiered heterogeneous processor model for processors for IoT objects, and, compare the model with processor architecture models currently in use in the IoT market.

## REFERENCES

- [1] J. Chase, “The evolution of the internet of things - from connected things to living in the data, preparing for challenges and IoT readiness,” Texas Instruments, Tech. Rep., Sep 2013.
- [2] (2015, Nov) Gartner says 6.4 billion connected “things” will be in use in 2016, up 30 percent from 2015. [Online]. Available: <http://www.gartner.com/newsroom/id/3165317>
- [3] S. Bath. (2016, Aug) Developing solutions for the internet of things. [Online]. Available: <https://www.intrinsyc.com/increasing-solution-differentiation-edge-based-heterogeneous-computing/>
- [4] “Developing solutions for the internet of things,” Intel, Tech. Rep., 2014.
- [5] S. Matalon, R. Klein, and C. Walls, “Embedded system power consumption: A software or hardware issue?” Mentor Graphics, Tech. Rep., Jun 2011.
- [6] “Intelligent flexible IoT nodes,” ARM, Tech. Rep., Oct 2015.
- [7] Y. Veller and S. Matalon, “Why you should optimize power at the electronic system level,” Mentor Graphics, Tech. Rep., Aug 2010.
- [8] C. Rommel, “Architecting success with heterogenous systems,” VDC Research, Mentor Graphics, Tech. Rep., 2016.
- [9] “IoT opportunity demands new approach to mcu-based embedded designs - rapidly moving market requires integrated silicon/software platform,” Renesas and Synergy, Tech. Rep., Oct 2015.
- [10] J. Branke, K. Deb, K. Miettinen, and R. Slowinski, *Multiobjective Optimization - Interactive and Evolutionary Approaches*. Verlag Berlin Heidelberg: Springer, 2008.
- [11] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.
- [12] K. Char, “Internet of things system design with integrated wireless MCUs,” Silicon Labs, ARM, Tech. Rep., Oct 2015.
- [13] J. Geuzebroek and A. Vaassen, “Building an efficient, tightly coupled embedded system using an extensible processor,” Synopsys, Tech. Rep., Jun 2014.
- [14] T. Adegbija, A. Rogacs, C. Patel, and A. Gordon-Ross, “Enabling right-provisioned microprocessor architectures for the internet of things,” in *ASME Proceedings of International Mechanical Engineering Congress and Exposition*, Houston, Texas, USA, Nov 2015.
- [15] J. Michanan, R. Dewri, and M. J. Rutherford, “Understanding the power-performance tradeoff through pareto analysis of live performance data,” in *Proceedings of International Green Computing Conference (IGCC)*, Dallas, Texas, USA, Nov 2014.
- [16] Q. Guo, T. Chen, Y. Chen, Z.-H. Zhou, W. Hu, and Z. Xu, “Effective and efficient microprocessor design space exploration using unlabeled design configurations,” *ACM Transactions on Intelligent Systems and Technology*, vol. 5, no. 1, pp. 20:1–20:18, Jan 2014.
- [17] M. Monchiero, R. Canal, and A. Gonzalez, “Power/performance/thermal design-space exploration for multicore architectures,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 5, pp. 666–681, May 2008.
- [18] T. Givargis and F. Vahid, “Platune: A tuning framework for system-on-a-chip platforms,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 11, pp. 1317–1327, Nov 2002.
- [19] M. Palesi and T. Givargis, “Multi-objective design space exploration using genetic algorithms,” in *Proceedings of the 10th International Symposium on Hardware/Software Codesign (CODES)*, Estes Park, CO, USA, May 2002.
- [20] C. Silvano, W. Fornaciari, G. Palermo, V. Zaccaria, F. Castro, M. Martinez, S. Bocchio, R. Zafalon, P. Avasare, G. Vanmeerbeeck, C. Ykman-Couvreur, M. Wouters, C. Kavka, L. Onesti, A. Turco, U. Bondi, G. Mariani, H. Posadas, E. Villar, C. Wu, F. Dongrui, Z. Hao, and T. Shibin, “MULTICUBE: Multi-objective design space exploration of multi-core architectures,” in *Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Lixouri, Kefalonia, Jul 2010.
- [21] A. Munir, A. Gordon-Ross, S. Lysecky, and R. Lysecky, “A lightweight dynamic optimization methodology and application metrics estimation model for wireless sensor networks,” *Sustainable Computing: Informatics and Systems*, vol. 3, no. 2, pp. 94 – 108, Jun 2013.
- [22] D. Sheldon, “Design space exploration of parameterized systems using design of experiments,” Ph.D. dissertation, Department of Computer Science, Dec 2011.
- [23] E. K. Ardestani and J. Renau, “ESESC: A fast multicore simulator using time-based sampling,” in *Proceedings of IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, Washington, DC, USA, Feb 2013.
- [24] C. Bienia, “Benchmarking modern multiprocessors,” Ph.D. dissertation, Department of Computer Science, Jan 2011.
- [25] Y. Bao, C. Bienia, and K. Li, *The PARSEC Benchmark Suite Tutorial - PARSEC 3.0*, San Jose, CA, USA, Jun 2011.
- [26] (2015) Perl reference. [Online]. Available: <http://perlmaven.com/>
- [27] J. McNamara. (2015, Apr) Excel-writer-XLSX. [Online]. Available: <http://search.cpan.org/dist/Excel-Writer-XLSX/>
- [28] A. F. Lorenzon, M. C. Cera, and A. C. S. Beck, “On the influence of static power consumption in multicore embedded systems,” in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, Lisbon, Portugal, May 2015.
- [29] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, “The SPLASH-2 programs: Characterization and methodological considerations,” in *Proceedings of 22nd Annual International Symposium on Computer Architecture (ISCA)*, Santa Margherita Ligure, Italy, Jun 1995.



**Prasanna Kansakar** is a PhD student in the Department of Computer Science (CS) at Kansas State University (K-State), Manhattan, KS. His research interests include Internet of Things, embedded and cyber-physical systems, computer architecture, multicore, secure and trustworthy systems, and hardware-based security. Kansakar has an MS degree in computer science and engineering from the University of Nevada, Reno (UNR). He is a student member of the IEEE.



**Arslan Munir** is currently an Assistant Professor in the Department of Computer Science (CS) at Kansas State University (K-State). He holds a Michelle Munson-Serban Simu Keystone Research Faculty Scholarship from the College of Engineering. He was a postdoctoral research associate in the Electrical and Computer Engineering (ECE) department at Rice University, Houston, Texas, USA from May 2012 to June 2014. He received his M.A.Sc. in ECE from the University of British Columbia (UBC), Vancouver, Canada, in 2007 and

his Ph.D. in ECE from the University of Florida (UF), Gainesville, Florida, USA, in 2012. From 2007 to 2008, he worked as a software development engineer at Mentor Graphics in the Embedded Systems Division.

Munir's current research interests include embedded and cyber-physical systems, secure and trustworthy systems, hardware-based security, computer architecture, multicore, parallel computing, distributed computing, reconfigurable computing, artificial intelligence (AI) safety and security, data analytics, and fault tolerance. Munir received many academic awards including the doctoral fellowship from Natural Sciences and Engineering Research Council (NSERC) of Canada. He earned gold medals for best performance in electrical engineering, gold medals and academic roll of honor for securing rank one in pre-engineering provincial examinations (out of approximately 300,000 candidates). He is a Senior Member of IEEE.